日期: /

线性回归 linear regression.

$y = w * x + b + \varepsilon$ $\varepsilon \in N(0,1)$ 噪声. 正态分布.

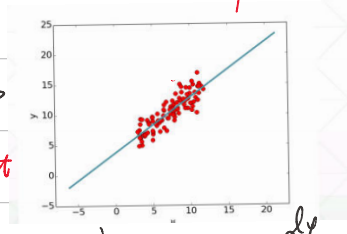
$loss = \Sigma_i (w * x_i + b - y_i)^2$

(误差函数). minimize loss

error

→ 梯度下降. Gradient Descent

$w' = w - lr * \frac{dy}{dw}$



exp. $x' = x - 0.05 * \frac{dy}{dx}$

w in 值向最低点 (导数为负) 方向移动.

learningrate ↑ 就. 衰减因子.

- $loss = \Sigma_i (w * x_i + b - y_i)^2$
- $w' = w - lr * \frac{\partial loss}{\partial w}$ 梯度: 同梯度为负的方向行进.
- $b' = b - lr * \frac{\partial loss}{\partial b}$
- $w' * x + b' \rightarrow y$

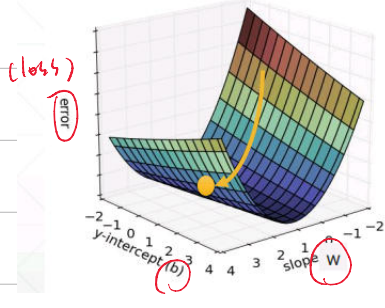
得到误差最小 (最优) w, b.

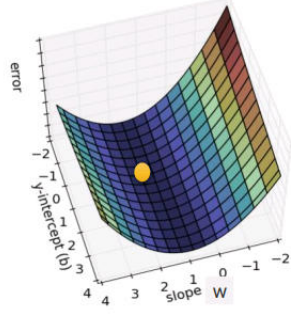
surface

$$loss = \sum_{i=1} (w * x_i + b - y_i)^2$$

convex Optimize 凸优化.

(loss) error





日期: /

Linear Regression.

Logistic Regression 上述

classification 离散.

点集: $[(x_0, y_0) \cdots (x_{99}, y_{99})]$

随机初始化. w_0, b_0 对于 $w_0 = b_0 = 0$

Step 1. compute Loss.

$loss = \frac{1}{N} \sum_i (w x_i + b - y_i)^2$.

Step 2. Compute Gradient and update.

计算梯度.

- $w' = w - lr * \frac{\partial loss}{\partial w}$
- $b' = b - lr * \frac{\partial loss}{\partial b}$

$\sum_i 2(w x_i + b - y_i) x_i$

$\frac{\partial loss}{\partial w} = 2(w x_0 + b - y_0) x_0 + 2(w x_1 + b - y_1) x_1 \cdots$

$\frac{\partial loss}{\partial b} = 2(w x_0 + b - y_0) \cdot 1 + \cdots$

$\sum_i 2(w x_i + b - y_i)$

日期: /

Step 3. set w = w' and loop

循环更新 w, b.

w ← w' b ← b'

离散预测. Discrete Prediction.

计算机视觉, 分类. Image Classification.

手写数字体识别. Hand-written Digits Recognition.

mnist → 数据集.

Image. 大小: [28, 28, 1] 灰度图.

打平: 每行相连. ↓ [784] 28×28.

Input x = [b, 784] ↑ b张图.

prediction.

- 1. dog = 0, cat = 1, fish = 2, ...
- 2. dog = [1, 0, 0, ...]
  - cat = [0, 1, 0, ...]
  - fish = [0, 0, 1, ...]

日期: /

regression vs. classification.

$y = w*x + b$ $\quad y \in R^d$.

out = x @ w + b. 排序，输出向量

out = [0.1, 0.8, 0.02, 0.08] 置信度.

pred = argmax(out) 预测 → 置信度最大

pred = 1

label = 2

computation Graph

out = x @ w + b

x = [b, 784] → 1个batch.

w = [784, 10] 10: 0~9.

b = [10]

out = [b, 10]

[b, 784] @ [784, 10] + [10]

[b, 10] + [10] → [b, 10]

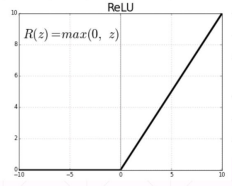
out

日期: /

It's linear !

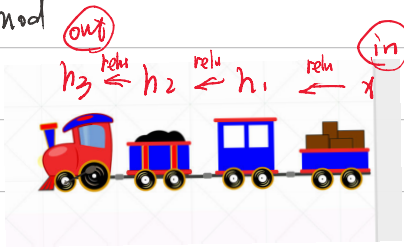
添加非线性因子.

out = relu (x @ w + b)



激活函数 = relu , sigmod

h: hidden 隐藏层



依述多道工序

out = relu (x @ w + b)

↓

h_1 = relu ($x @ w_1 + b_1$)

h_2 = relu ($h_1 @ w_2 + b_2$)

out = relu ($h_2 @ w_3 + b_3$)

(relu 激活)

particularly 单一道工序

- X = [v1, v2, ..., v784]
  - X: [1, 784]
- h_1 = relu($X @ W_1 + b_1$)
  - W_1 : [784, 512]
  - b_1 : [1, 512]

→ [1, 512] (h_1)

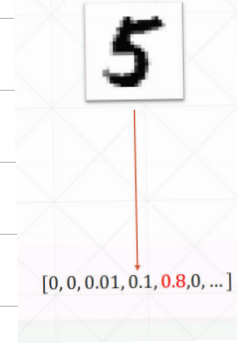
- h_2 = relu($h_1 @ W_2 + b_2$)
  - W_2 : [512, 256]
  - b_2 : [256]

→ [1, 256] (h_2)

- out = relu($h_2 @ W_3 + b_3$)
  - W_3 : [256, 10]
  - b_3 : [10]

→ [1, 10] (out)

(降维)



日期: /

- out: [1, 10]

数据集 → 确定值. 打标签.

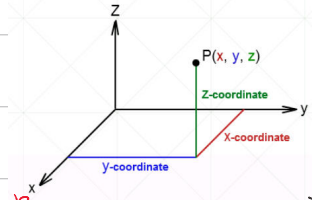
- Y/label: 0~9

one-hoting

  - eg.: 1 → [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  - eg.: 3 → [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

将标签与数据集计算(误差比较)

- Euclidean Distance: *out* → Label
  - MSE



MSE 欧式距离

(矩阵 → 多维度)

$$\sum_i (y_i - out)^2$$

$$out = relu\{relu\{relu[x @ W_1 + b_1] @ W_2 + b_2\} @ W_3 + b_3\}$$

$$pred = \arg\max(out)$$

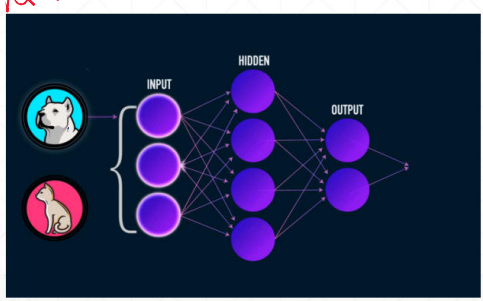
$$loss = MSE(out, label)$$

minimize loss.

$$\{W_1', b_1', W_2', b_2', W_3', b_3'\}$$

预计 → 不 (out)

与训练集符合度最高.



日期: /

用训练集 → 找分类器.

classification procedure.

step 1. compute [h1, h2, out]

step 2. compute loss.

step 3. compute gradient and update.

[w1, b1, w2, b2, w3, b3] → 优化后的参数.

step 4. loop.

学习过程:

数据集 → 打好标签(label)的图象.

label = [1, 10] 矩阵.

计算 → 通过神经网络 (relu)

输入 图象. [b, 784]

输出 [1, 10] 矩阵.

out

比较. 更新参数 loop

* 每次loop都计算 ~~所有的误差~~ (b张图) 的loss总和.

$\sum_i (y_i - out_i)^2$

(loop) 训练 n次, 得到一组相对最优参数 (loss 最小)

1个batch (使训练结果与标签相差最小)

即得到训练好的模型.

日期: /

Data container.

list 数字. 字母. 数组. 类.

np.array.

tf.Tensor. 默认数据类型.

Tensor?

scalar = 1.1 标量. dim=0

vector: [1.1], [1.1, x2, ...] 向量. dim=1.

matrix: [[1.1, x2], [3.3, 4.4]]

tensor: rank > 2 维度 > 2

tensorflow 数据 运算 流动.

TF is a computing lib.

int. float, double.

bool 布尔型. equal T/F?

string

日期: /

Create:

整形: tf.constant(num, dtype=)

(True, False)

(Hello world)

Tensor Property.

① tf.device. 设备名. ("cpu") ("gpu")

转换: a.gpu / a.cpu.

转换.

② b.numpy. tensor → nump

b.shape. 查看一个 tensor 的 shape.

rank / ndim numpy=1 → 维度

name

check Tensor Type.

① isinstance(a, tf.tensor).

② tf.is_tensor(b) → 返回 true / false.

③ a.dtype.

日期: /

convert

numpy → 默认 int 64.

① aa = tf.convert_to_tensor(a) / (a, dtype=int32).

② tf.cast(aa, dtype=tf.float32).

tf.double.

tf.int32.

如果返回的是[0,1] → tf.bool

③ bool → int

tf.cast(aa, dtype=tf.int32). false → 0, true → 1

b = tf.Variable.(a) ← tensor 自动包装 b.

b 具有 { b.name
b.trainable. 求导. 可训练. 精度信息.

To numpy:

a.numpy.

a.tf.one([1]). → int(a) / float(a) 必须是 scalar.

日期: /

创建 Tensor:

1. from numpy, list → data.

tf.convert_to_tensor(np.ones([2,3])) ← size.

① tf.zeros([]) → shape. [] 空.

[2,2] [2,3,3]

tf.zeros_like(a) 和 a is shap 一样. 元素全为0.

② tf.ones. 全为1. 同样传入 shape.

③ tf.fill([2,2], 0) → shap, 值.

Normal 正态

正态分布.

tf.radom.nomal([2,2], mean=0, stddev=1)

tf.random.truncated_normal([2,2], mean=0, stddev=1)

截取正态分布的一部分

用于 sigmoid

日期: /

Uniform . 均匀 → shape

tf.random.uniform([2,2], minval, maxval)

Random Permutation. 随机打散.

以数据集 (有时应关系的)

idx = tf.range(10) [0,9]

idx = tf.random.shuffle(idx)

两种方法

```
In [24]: idx=tf.range(10)                    得到一个乱数据的顺序
In [26]: idx=tf.random.shuffle(idx)
Out[27]: <tf.Tensor: id=67, shape=(10,), dtype=int32, numpy=array([2, 1, 9, 3, 8,
7, 0, 5, 4, 6])>

In [29]: a=tf.random.normal([10,784])
In [31]: b=tf.random.uniform([10], maxval=10, dtype=tf.int32)
Out[32]: <tf.Tensor: id=78, shape=(10,), dtype=int32, numpy=array([1, 4, 6, 9, 4,
9, 3, 2, 0, 5])>

In [33]: a=tf.gather(a,idx)                  同样的顺序(idx) 对 a,b 中乱数据
In [34]: b=tf.gather(b,idx)
Out[35]: <tf.Tensor: id=83, shape=(10,), dtype=int32, numpy=array([6, 4, 5, 9, 0,
2, 1, 9, 4, 3])>
```

tf.constant = tf.convert_to_tensor.

`每行的维度一致`

日期: /

Typical Dim Data.

1. Scalar : 标量

[]

loss = mse(out, y)

accuracy

loss

out

y.

```
In [24]: out=tf.random.uniform([4,10])
<tf.Tensor: id=78, shape=(4, 10), dtype=float32, numpy=
array([[0.11917067, 0.4999225 , 0.5349126 , 0.12400055, 0.5961182 , ...
        0.48180282, 0.47066116, 0.96908057, 0.6354896 , 0.44619405]],
      dtype=float32)>

In [28]: y=tf.range(4)
In [29]: y=tf.one_hot(y,depth=10) 编码. 独一热.
<tf.Tensor: id=95, shape=(4, 10), dtype=float32, numpy=
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], ...
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]], dtype=float32)>

In [31]: loss=tf.keras.losses.mse(y, out)   Σ(y-out)²
Out[32]: <tf.Tensor: id=99, shape=(4,), dtype=float32, numpy=array([0.37688   , 0.27573627,
0.5074741 , 0.3111775 ], dtype=float32)>

In [33]: loss=tf.reduce_mean(loss)
Out[34]: <tf.Tensor: id=102, shape=(), dtype=float32, numpy=0.36781698>
```

→ scalar

Dim = 1.

Vector

Bias: $x @ w + b$ → bias

[out_dim] 与 out 维度一致.

```
In [20]: net=layers.Dense(10)  全连接层.  → 打印. 8→10
In [21]: net.build((4,8))
In [22]: net.kernel  → w
<tf.Variable 'kernel:0' shape=(8, 10) dtype=float32, numpy=
array([[ 0.19603091, -0.2846143 ,  0.14747244, -0.03926206, -0.02218038,
         0.02417278, -0.31929022, -0.07324755,  0.17448658,  0.06289428]],
      dtype=float32)>

In [23]: net.bias   b: dim = 10
Out[23]: <tf.Variable 'bias:0' shape=(10,) dtype=float32, numpy=array([0., 0., 0., 0., 0.,
0., 0., 0., 0., 0.], dtype=float32)>
```

日期: /

Dim=2.

Matrix 矩阵.

input x: [b, vec_dim]

weight

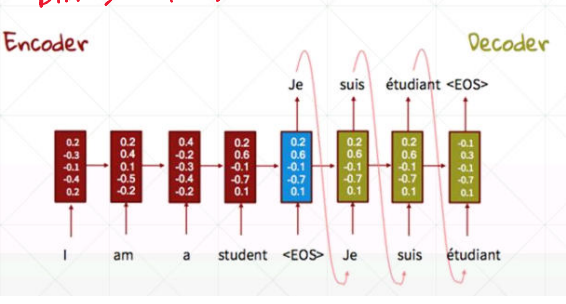
x:[b. seq_len, word_dim]

Dim=3 Tensor

```
In [35]: x = tf.random.normal([4,784])
Out[36]: TensorShape([4, 784])

In [38]: net=layers.Dense(10)
In [39]: net.build((4,784))

In [40]: net(x).shape
Out[40]: TensorShape([4, 10])

In [41]: net.kernel.shape
Out[41]: TensorShape([784, 10])

In [42]: net.bias.shape
Out[42]: TensorShape([10])
```

784 → 10

[4. 10]




```
In [4]: (X_train, y_train), (X_test, y_test) =
keras.datasets.imdb.load_data(num_words=10000)
In [5]: x_train = keras.preprocessing.sequence.pad_sequences(X_train, maxlen=80)
In [14]: x_train.shape
Out[14]: TensorShape([25000, 80])

In [12]: emb = embedding(x_train)
In [13]: emb.shape
Out[13]: TensorShape([25000, 80, 100])

In [19]: out=rnn(emb[:4])
In [20]: out.shape
Out[20]: TensorShape([4, 256])
```

num.

words.

编码.

emb

100 维.

日期: /

Dim=4 Tensor.

image. [b, h, w, 3] → R.G.B.

feature maps. [b, h, w, c]

卷积神经网络.

```
In [4]: x=tf.random.normal((4,32,32,3))

In [5]: net=layers.Conv2D(16,kernel_size=3)

In [6]: net(x)
# [4, 32, 32, 16]
```

随机.

卷积.

Dim=5 Tensor.

Single task = [b, h, w, 3]

Meta-learning. (多任务)

任务数 [4, 64, 28, 28, 1]

通俗含义: task.

日期: /

索引与切片.

Basic indexing. 给定每一个维度.

```
In [4]: a=tf.ones([1,5,5,3])
In [5]: a[0][0]
<tf.Tensor: id=16, shape=(5, 3), dtype=float32, numpy=
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]], dtype=float32)>

In [6]: a[0][0][0]
Out[6]: <tf.Tensor: id=29, shape=(3,), dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>

In [7]: a[0][0][0][2]
Out[7]: <tf.Tensor: id=46, shape=(), dtype=float32, numpy=1.0>
```

可读性差、取样简单一.

Numpy-style indexing.

```
In [8]: a=tf.random.normal([4,28,28,3])
In [9]: a[1].shape
Out[9]: TensorShape([28, 28, 3])

In [10]: a[1,2].shape
Out[10]: TensorShape([28, 3])

In [11]: a[1,2,3].shape
Out[11]: TensorShape([3])

In [12]: a[1,2,3,2].shape
Out[12]: TensorShape([])
```

日期: /

start : end　　　起　末　[A:B).

切片　冒号 A:B

```
In [8]: a=tf.range(10)
Out[9]: <tf.Tensor: numpy=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])>

In [14]: a[-1:]
Out[14]: <tf.Tensor: id=48, shape=(1,), dtype=int32, numpy=array([9])>
In [15]: a[-2:]
Out[15]: <tf.Tensor: id=53, shape=(2,), dtype=int32, numpy=array([8, 9])>
In [16]: a[:2]
Out[16]: <tf.Tensor: id=58, shape=(2,), dtype=int32, numpy=array([0, 1])>
In [17]: a[:-1]
Out[17]: <tf.Tensor: id=63, shape=(9,), dtype=int32, numpy=array([0, 1, 2, 3, 4,
5, 6, 7, 8])>
```

对索引进行编号.　[-1:0) → 9 最末　-1　vector

Indexing by:

*逗号隔开!!!

多种取样方式

```
In [14]: a.shape #TensorShape([4, 28, 28, 3])

In [15]: a[0].shape #TensorShape([28, 28, 3])

In [16]: a[0,:,:,:].shape
Out[16]: TensorShape([28, 28, 3])

In [17]: a[0,1,:,:].shape
Out[17]: TensorShape([28, 3])

In [18]: a[:,:,:,0].shape
Out[18]: TensorShape([4, 28, 28])

In [19]: a[:,:,:,2].shape
Out[19]: TensorShape([4, 28, 28])

In [20]: a[:,0,:,:].shape
Out[20]: TensorShape([4, 28, 3])
```

步长 : step.

- start:end:step
- ::step

```
In [21]: a.shape
Out[21]: TensorShape([4, 28, 28, 3])

In [22]: a[0:2,:,:,:].shape
Out[22]: TensorShape([2, 28, 28, 3])

In [23]: a[:,0:28:2,0:28:2,:].shape
Out[23]: TensorShape([4, 14, 14, 3])

In [24]: a[:,:14,:14,:].shape
Out[24]: TensorShape([4, 14, 14, 3])

In [25]: a[:,14:,14:,:].shape
Out[25]: TensorShape([4, 14, 14, 3])

In [26]: a[:,::2,::2,:].shape
Out[26]: TensorShape([4, 14, 14, 3])
```

前14　后14.

日期: /

```
In [27]: a=tf.range(4)
Out[28]: <tf.Tensor: id=118, shape=(4,), dtype=int32, numpy=array([0, 1, 2, 3],
dtype=int32)>

In [29]: a[::-1]
Out[29]: <tf.Tensor: id=123, shape=(4,), dtype=int32, numpy=array([3, 2, 1, 0],
dtype=int32)>

In [30]: a[::-2]
Out[30]: <tf.Tensor: id=128, shape=(2,), dtype=int32, numpy=array([3, 1], dtype=int32)>

In [31]: a[2::-2]
Out[31]: <tf.Tensor: id=133, shape=(2,), dtype=int32, numpy=array([2, 0], dtype=int32)>
```

Selective Indexing

- tf.gather
- tf.gather_nd
- tf.boolean_mask

```
In [46]: tf.gather(a, axis=0, indices=[2,3]).shape
Out[46]: TensorShape([2, 35, 8])

In [47]: a[2:4].shape
Out[47]: TensorShape([2, 35, 8])

In [48]: tf.gather(a, axis=0, indices=[2,1,3,0]).shape
Out[48]: TensorShape([4, 35, 8])

In [49]: tf.gather(a, axis=1, indices=[2,3,7,9,16]).shape
Out[49]: TensorShape([4, 5, 8])

In [50]: tf.gather(a, axis=2, indices=[2,3,7]).shape
Out[50]: TensorShape([4, 35, 3])
```

日期:

tf.gather_nd

- data: [classes, students, subjects]
- What if sample several students and their several subjects?
- $aa = tf.gather(a, axis, [several\ students])$
- $aaa = tf.gather(aa, axis, [several\ subjects])$

tf.gather_nd

- data: [classes, students, subjects]
- What if sample several (classes and students)?
- *for instance*:
  - [class1_student1, class2_studnet2, class3_student3, class4_student4]
  - → [4, 8]

```
In [55]: a.shape
Out[55]: TensorShape([4, 35, 8])

In [60]: tf.gather_nd(a, [0]).shape
Out[60]: TensorShape([35, 8])

In [61]: tf.gather_nd(a, [0,1]).shape
Out[61]: TensorShape([8])

In [62]: tf.gather_nd(a, [0,1,2]).shape
Out[62]: TensorShape([])

In [63]: tf.gather_nd(a, [[0,1,2]]).shape
Out[63]: TensorShape([1])
```

日期:

```
In [55]: a.shape
Out[55]: TensorShape([4, 35, 8])

In [56]: tf.gather_nd(a, [[0,0],[1,1]]).shape
Out[56]: TensorShape([2, 8])

In [57]: tf.gather_nd(a, [[0,0],[1,1],[2,2]]).shape
Out[57]: TensorShape([3, 8])

In [58]: tf.gather_nd(a, [[0,0,0],[1,1,1],[2,2,2]]).shape
Out[58]: TensorShape([3])

In [59]: tf.gather_nd(a, [[[0,0,0],[1,1,1],[2,2,2]]]).shape
Out[59]: TensorShape([1, 3])
```

tf.boolean_mask

```
In [75]: a.shape
Out[75]: TensorShape([4, 28, 28, 3])

In [76]: tf.boolean_mask(a, mask=[True,True,False,False]).shape
Out[76]: TensorShape([2, 28, 28, 3])

In [77]: tf.boolean_mask(a, mask=[True,True,False],axis=3).shape
Out[77]: TensorShape([4, 28, 28, 2])

In [78]: a=tf.ones([2,3,4])

In [79]: tf.boolean_mask(a,mask=[[True,False,False],[False,True,True]])
<tf.Tensor: id=354, shape=(3, 4), dtype=float32, numpy=
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]], dtype=float32)>
```

日期:　　　/

维度变换

Outline

batch row column channel
[b, 28, 28, 1]
轴 axis　hight　width

- shape, ndim
- reshape 变换
- expand_dims/squeeze 增加，减少
- transpose 顺序
- broadcast_to

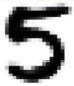
View

0000000000001000000000000000101000 1111111111101000111111111110100 1111111111101110000000000001010 11111111111101000000000000011110

row 1　row 2　row 3

content [b, h, w] 内存中的格式

- 每一张 [b, 28, 28] view 1 (行+列) 数据无破坏
- →[b, 28*28] 混一在一起 (无行列) 整体 view 2
- →[b, 2, 14*28] view 3 上下两部分
- →[b, 28, 28, 1] channel 和 view 1 类似 view 4



Reshape

```
In [80]: a=tf.random.normal([4,28,28,3])
Out[81]: TensorShape([4, 28, 28, 3])

In [82]: a.shape, a.ndim
Out[82]: (TensorShape([4, 28, 28, 3]), 4)

In [83]: tf.reshape(a,[4,784,3]).shape
Out[83]: TensorShape([4, 784, 3])

In [84]: tf.reshape(a,[4,-1,3]).shape
Out[84]: TensorShape([4, 784, 3])

In [85]: tf.reshape(a,[4,784*3]).shape
Out[85]: TensorShape([4, 2352])

In [86]: tf.reshape(a,[4,-1]).shape
Out[86]: TensorShape([4, 2352])
```

b h w c　view 1　view 2 → [b, pixel, c]　自动计算　view 3 data point

将行、列、channel 合体 个数 = 784×3. 无法恢复。

Reshape is flexible

```
In [80]: a=tf.random.normal([4,28,28,3])
Out[81]: TensorShape([4, 28, 28, 3])

In [87]: tf.reshape(tf.reshape(a,[4,-1]),[4,28,28,3]).shape
Out[87]: TensorShape([4, 28, 28, 3])

In [88]: tf.reshape(tf.reshape(a,[4,-1]),[4,14,56,3]).shape
Out[88]: TensorShape([4, 14, 56, 3])

In [89]: tf.reshape(tf.reshape(a,[4,-1]),[4,1,784,3]).shape
Out[89]: TensorShape([4, 1, 784, 3])
```

Size 相等即可.
乘积相等
意义?

Reshape could lead to potential bugs!

- images: [4, 28, 28, 3]
  - [b, h, w, 3]
- reshape to: [4, 784, 3]
  - [b, pixel, 3]

恢复? 需要 content 信息
如: 图片的长宽. 顺序

- [4, 784, 3] $\overrightarrow{height: 28, width: 28}$ [4, 28, 28, 3]
- [4, 784, 3] $\overrightarrow{height: 14, width: 56}$ [4, 14, 56, 3]
- [4, 784, 3] $\overrightarrow{width: 28, height: 28}$ [4, 28, 28, 3]

tf.transpose

改变 content. 转置.

[h,w] → [w,h]

```
In [93]: a=tf.random.normal((4,3,2,1))

In [94]: a.shape
Out[94]: TensorShape([4, 3, 2, 1])

In [95]: tf.transpose(a).shape
Out[95]: TensorShape([1, 2, 3, 4])

In [97]: tf.transpose(a,perm=[0,1,3,2]).shape
Out[97]: TensorShape([4, 3, 1, 2])
```

全部转置.

New 按需: 0,1,3,2 →

顺序.

日期: /

→[b, 3, h, w]

```
In [98]: a=tf.random.normal([4,28,28,3])

In [99]: tf.transpose(a,[0,2,1,3]).shape
Out[99]: TensorShape([4, 28, 28, 3])

In [101]: tf.transpose(a,[0,3,2,1]).shape
Out[101]: TensorShape([4, 3, 28, 28])

In [102]: tf.transpose(a,[0,3,1,2]).shape
Out[102]: TensorShape([4, 3, 28, 28])
```

Expand dim

- a: [classes, students, classes]
  - [4, 35, 8]
- add school dim
- [1, 4, 35, 8] + [1, 4, 35, 8]
  - [2, 4, 35, 8]

```
In [103]: a=tf.random.normal([4,35,8])

In [105]: tf.expand_dims(a,axis=0).shape
Out[105]: TensorShape([1, 4, 35, 8])

In [106]: tf.expand_dims(a,axis=3).shape
Out[106]: TensorShape([4, 35, 8, 1])
```

axis

0	1	2

```
In [103]: a=tf.random.normal([4,35,8])

In [105]: tf.expand_dims(a,axis=0).shape
Out[105]: TensorShape([1, 4, 35, 8])

In [106]: tf.expand_dims(a,axis=3).shape
Out[106]: TensorShape([4, 35, 8, 1])

In [107]: tf.expand_dims(a,axis=-1).shape
Out[107]: TensorShape([4, 35, 8, 1])

In [108]: tf.expand_dims(a,axis=-4).shape
Out[108]: TensorShape([1, 4, 35, 8])
```

日期: /

Squeeze dim

- Only squeeze for shape=1 dim 只能用于 shape = 1
- [4, 35, 8, 1]
- [1, 4, 35, 8]
- [1, 4, 35, 1]

Broadcasting:

- expand
- without copying data
- tf.broadcast_to

key idea. Insert 1 dim if needed 加维度，再扩张。

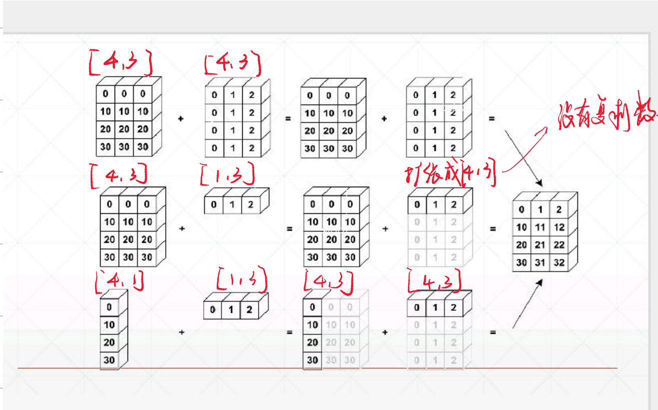
① 小维度先对齐。

② 插入1.维度 → 再扩张。

a [4, 32, 32, 3]

b. [3] → [1, 1, 1, 32] → [4, 32, 32, 3]

日期:　　　/



意义?

- When it has no axis
  - Create a new concept 创造新概念
  - [classes, students, scores] + [scores] 加分

默认对所有概念适用

无班级 / 学生概念

大维度 4　35　8　小维度

- When it has dim of size 1
  - Treat it shared by all
  - [classes, students, scores] + [students, 1]

加偏置.

对某些学生加分,

[35, 1] 前多少学生加分

日期: /

Why broading casting?

对. tile 会扩张内存.

- 1. for real demanding 简洁
  - [classes, students, scores]
  - Add bias for every student: +5 score 每学生加分.
  - [4, 32, 8] + [4, 32, 8]
  - [4, 32, 8] + [5.0] [1] → [4, 32, 8]

默认广播.

内存消耗

- 2. memory consumption
  - [4, 32, 8] → 1024 字节数. 10×4×4
  - bias=[8]: [5.0,5.0,5.0,...] → 8 用 bias 8×4 节省内存.

Broadcastable:

[1] √

a [4] ×.

b [1,3]

- Match from Last dim! 底层维度.
  - If current dim=1, expand to same
  - If either has no dim, insert one dim and expand to same
  - otherwise, NOT broadcastable

右对齐. 最后维度相等

1 可以扩张.

```
In [25]: x=tf.random.normal([4,32,32,3])

In [27]: (x+tf.random.normal([3])).shape
Out[27]: TensorShape([4, 32, 32, 3])

In [28]: (x+tf.random.normal([32,32,1])).shape
Out[28]: TensorShape([4, 32, 32, 3])

In [29]: (x+tf.random.normal([4,1,1,1])).shape
Out[29]: TensorShape([4, 32, 32, 3])

In [31]: (x+tf.random.normal([1,4,1,1])).shape
InvalidArgumentError: Incompatible shapes: [4,32,32,3] vs. [1,4,1,1] [Op:Add]
name: add/
```

```
In [35]: x.shape
Out[35]: TensorShape([4, 32, 32, 3])

In [36]: (x+tf.random.normal([4,1,1,1])).shape
Out[36]: TensorShape([4, 32, 32, 3])

In [37]: b=tf.broadcast_to(tf.random.normal([4,1,1,1]), [4,32,32,3])

In [38]: b.shape
Out[38]: TensorShape([4, 32, 32, 3])
```

```
In [4]: a=tf.ones([3,4])
In [5]: a1=tf.broadcast_to(a, [2,3,4])
<tf.Tensor: id=7, shape=(2, 3, 4), dtype=float32, numpy=
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]], dtype=float32)>

In [7]: a2=tf.expand_dims(a,axis=0)
Out[8]: TensorShape([1, 3, 4])
In [10]: a2=tf.tile(a2, [2,1,1])
<tf.Tensor: id=12, shape=(2, 3, 4), dtype=float32,
numpy=array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]], dtype=float32)>
```

日期: /

数学运算.

- +-*/ 加减乘除
- **, pow, square 幂方
- sqrt
- //, % 取余. 整除
- exp, log 指数.
- @, matmul 矩阵.
- linear layer

- element-wise 对应元素
  - +-*/
- matrix-wise 矩阵相乘.
  - @, matmul [b,3,4] @ [b,4,5]
- dim-wise 对某一个维度.
  - reduce_mean/max/min/sum

```
In [134]: b=tf.fill([2,2],2.)
In [135]: a=tf.ones([2,2])

In [136]: a+b,a-b,a*b,a/b
(<tf.Tensor: id=462, shape=(2, 2), dtype=float32, numpy=
 array([[3., 3.],
        [3., 3.]], dtype=float32)>,
 <tf.Tensor: id=463, shape=(2, 2), dtype=float32, numpy=
 array([[-1., -1.],
        [-1., -1.]], dtype=float32)>,
 <tf.Tensor: id=464, shape=(2, 2), dtype=float32, numpy=
 array([[2., 2.],
        [2., 2.]], dtype=float32)>,
 <tf.Tensor: id=465, shape=(2, 2), dtype=float32, numpy=
 array([[0.5, 0.5],
        [0.5, 0.5]], dtype=float32)>)

In [137]: b//a,b%a
(<tf.Tensor: id=470, shape=(2, 2), dtype=float32, numpy=
 array([[2., 2.],
        [2., 2.]], dtype=float32)>,
 <tf.Tensor: id=471, shape=(2, 2), dtype=float32, numpy=
 array([[0., 0.],
        [0., 0.]], dtype=float32)>)
```

+ - * / 合法√. 维度一样. 可以广播

```
In [138]: a
<tf.Tensor: id=461, shape=(2, 2), dtype=float32, numpy=
array([[1., 1.],
       [1., 1.]], dtype=float32)>

In [140]: tf.math.log(a)
<tf.Tensor: id=475, shape=(2, 2), dtype=float32, numpy=
array([[0., 0.],
       [0., 0.]], dtype=float32)>

In [141]: tf.exp(a)
<tf.Tensor: id=477, shape=(2, 2), dtype=float32, numpy=
array([[2.7182817, 2.7182817],
       [2.7182817, 2.7182817]], dtype=float32)>
```

log_e 无法使用 log_2, log_10

日期:

使用 log2 , log10 ?

$\frac{\log_e b}{\log_e 2}$ $\log_2 b$.

```
In [22]: tf.math.log(8.)/tf.math.log(2.)
Out[22]: <tf.Tensor: id=54, shape=(), dtype=float32,
numpy=3.0>
In [23]: tf.math.log(100.)/tf.math.log(10.)
Out[23]: <tf.Tensor: id=60, shape=(), dtype=float32,
numpy=2.0>
```

pow sqrt

```
In [142]: b
<tf.Tensor: id=458, shape=(2, 2), dtype=float32, numpy=
array([[2., 2.],
       [2., 2.]], dtype=float32)>

In [143]: tf.pow(b,3)
<tf.Tensor: id=481, shape=(2, 2), dtype=float32, numpy=
array([[8., 8.],
       [8., 8.]], dtype=float32)>

In [144]: b**3
<tf.Tensor: id=484, shape=(2, 2), dtype=float32, numpy=
array([[8., 8.],
       [8., 8.]], dtype=float32)>

In [145]: tf.sqrt(b)
<tf.Tensor: id=486, shape=(2, 2), dtype=float32, numpy=
array([[1.4142135, 1.4142135],
       [1.4142135, 1.4142135]], dtype=float32)>
```

矩阵 @ matmul

```
In [146]: a,b
(<tf.Tensor: id=461, shape=(2, 2), dtype=float32, numpy=
 array([[1., 1.],
        [1., 1.]], dtype=float32)>,
 <tf.Tensor: id=458, shape=(2, 2), dtype=float32, numpy=
 array([[2., 2.],
        [2., 2.]], dtype=float32)>)

In [147]: a@b
<tf.Tensor: id=490, shape=(2, 2), dtype=float32, numpy=
array([[4., 4.],
       [4., 4.]], dtype=float32)>

In [148]: tf.matmul(a,b)
<tf.Tensor: id=492, shape=(2, 2), dtype=float32, numpy=
array([[4., 4.],
       [4., 4.]], dtype=float32)>
```

日期:

```
In [150]: a=tf.ones([4,2,3])
In [151]: b=tf.fill([4,3,5], 2.)

In [152]: a@b
<tf.Tensor: id=503, shape=(4, 2, 5), dtype=float32, numpy=
array([[[6., 6., 6., 6., 6.],
        ...
        [6., 6., 6., 6., 6.]]], dtype=float32)>

In [153]: tf.matmul(a,b)
<tf.Tensor: id=505, shape=(4, 2, 5), dtype=float32, numpy=
array([[[6., 6., 6., 6., 6.],
        ...
        [6., 6., 6., 6., 6.]]], dtype=float32)>
```

batch. 重复4次. [2,3] [3,5] b

```
In [164]: a.shape # TensorShape([4, 2, 3])

In [165]: b.shape # TensorShape([3, 5])

In [166]: bb=tf.broadcast_to(b, [4,3,5])
In [167]: a@bb
<tf.Tensor: id=516, shape=(4, 2, 5), dtype=float32, numpy=
array([[[6., 6., 6., 6., 6.],
        [6., 6., 6., 6., 6.]], ...

       [[6., 6., 6., 6., 6.],
        [6., 6., 6., 6., 6.]]], dtype=float32)>
```

b

Recap

- $y = w * x + b$ 标量、relu. 运算
- $Y = X@W + b$ 矩阵
- $\begin{bmatrix} x_0^0 & x_0^1 \\ x_1^0 & x_1^1 \end{bmatrix} \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{bmatrix} + [b_0, b_1, b_2] \rightarrow \begin{bmatrix} y_0^0 & y_0^1 & y_0^2 \\ y_1^0 & y_1^1 & y_1^2 \end{bmatrix}$
- [b, 2] → [b, 3]

$y_0^0 = x_0^0 w_{00} + x_0^1 w_{10} + b_0$

$$Y = X@W + b$$

日期:

```
In [168]: x=tf.ones([4,2])

In [169]: W=tf.ones([2,1])

In [170]: b=tf.constant(0.1)

In [171]: x@W+b
<tf.Tensor: id=526, shape=(4, 1), dtype=float32, numpy=
array([[2.1],
       [2.1],
       [2.1],
       [2.1]], dtype=float32)>
```

$$out = relu(X@W + b)$$

```
In [171]: x@W+b
<tf.Tensor: id=526, shape=(4, 1), dtype=float32, numpy=
array([[2.1],
       [2.1],
       [2.1],
       [2.1]], dtype=float32)>

In [172]: out=x@W+b
In [173]: out=tf.nn.relu(out)
<tf.Tensor: id=530, shape=(4, 1), dtype=float32, numpy=
array([[2.1],
       [2.1],
       [2.1],
       [2.1]], dtype=float32)>
```

日期: /

前向传播（张量）实战

Recap:

- $out = relu\{relu\{relu[X@W_1 + b_1]@W_2 + b_2\}@W_3 + b_3\}$
- $pred = argmax(out)$
- $loss = MSE(out, label)$
- minimize *loss*
  - $[W_1', b_1', W_2', b_2', W_3', b_3']$

非线性 层 误差 使误差最小 增加复杂度

张量的合并与分割 Merge and split

- tf.concat 连接
- tf.split 分割
- tf.stack 堆叠
- tf.unstack 减(分割)

关键在维度!!!

concat

- Statistics about scores
  - [class1-4, students, scores] [4,35,8]
  - [class5-6, students, scores] [2,35,8]

4+2=6 不对

[6, 35, 8]

```
In [3]: a=tf.ones([4,35,8])
In [4]: b=tf.ones([2,35,8])

In [6]: c=tf.concat([a,b],axis=0)
In [7]: c.shape
Out[7]: TensorShape([6, 35, 8])

In [8]: a=tf.ones([4,32,8])
In [9]: b=tf.ones([4,3,8])

In [10]: tf.concat([a,b],axis=1).shape
Out[10]: TensorShape([4, 35, 8])
```

非拼接维度(4+2)

拼接维度(32+3)

注：其他维度一致。 合并增加维度。

日期: /

Stake.

- Statistics about scores
  - School1:[classes, students, scores]
  - School2:[classes, students, scores]
  - [schools, classes, students, scores]

注：所有已有维度都相等.

```
In [19]: a.shape
Out[19]: TensorShape([4, 35, 8])
In [23]: b.shape
Out[23]: TensorShape([4, 35, 8])

In [20]: tf.concat([a,b],axis=-1).shape
Out[20]: TensorShape([4, 35, 16])

In [21]: tf.stack([a,b],axis=0).shape
Out[21]: TensorShape([2, 4, 35, 8])

In [22]: tf.stack([a,b],axis=3).shape
Out[22]: TensorShape([4, 35, 8, 2])
```

Un Stack.

```
In [30]: a.shape # TensorShape([4, 35, 8])
In [32]: b=tf.ones([4,35,8])

In [33]: c=tf.stack([a,b])
In [34]: c.shape
Out[34]: TensorShape([2, 4, 35, 8])

In [35]: aa,bb=tf.unstack(c,axis=0)
In [36]: aa.shape,bb.shape
Out[36]: (TensorShape([4, 35, 8]), TensorShape([4, 35, 8]))

# [2, 4, 35, 8]
In [41]: res=tf.unstack(c,axis=3)

In [42]: res[0].shape, res[7].shape
Out[42]: (TensorShape([2, 4, 35]), TensorShape([2, 4, 35]))
```

Split

指定了数的数量.

```
# [2, 4, 35, 8]
In [43]: res=tf.unstack(c,axis=3)
In [44]: len(res)
Out[44]: 8

In [45]: res=tf.split(c,axis=3, num_or_size_splits=2)
In [46]: len(res)
Out[46]: 2

In [47]: res[0].shape
Out[47]: TensorShape([2, 4, 35, 4])

In [48]: res=tf.split(c,axis=3, num_or_size_splits=[2,2,4])

In [49]: res[0].shape, res[2].shape
Out[49]: (TensorShape([2, 4, 35, 2]), TensorShape([2, 4, 35, 4]))
```

日期: /

数据统计.

- tf.norm 范数.
- tf.reduce_min/max
- tf.argmax/argmin
- tf.equal
- tf.unique

Vector Norm

Eukl. Norm	$\|x\|_2 = [\sum_k x_k^2]^{1/2}$ 二范数.
Max.norm	$\|x\|_\infty = \max_k \|x_k\|$
L_1-Norm	$\|x\|_1 = \sum_k \|x_k\|$ 一范数.

- Here talks about Vector Norm

tf.norm( ).

```
In [50]: a=tf.ones([2,2])

In [52]: tf.norm(a)
Out[52]: <tf.Tensor: id=192, shape=(), dtype=float32, numpy=2.0>

In [54]: tf.sqrt(tf.reduce_sum(tf.square(a)))
Out[54]: <tf.Tensor: id=197, shape=(), dtype=float32, numpy=2.0>

In [55]: a=tf.ones([4,28,28,3])

In [56]: tf.norm(a)
Out[56]: <tf.Tensor: id=206, shape=(), dtype=float32, numpy=96.99484>

In [57]: tf.sqrt(tf.reduce_sum(tf.square(a)))
Out[57]: <tf.Tensor: id=211, shape=(), dtype=float32, numpy=96.99484>
```

```
In [68]: b = tf.ones([2,2])

In [69]: tf.norm(b)
Out[69]: <tf.Tensor: id=250, shape=(), dtype=float32, numpy=2.0>

In [73]: tf.norm(b,ord=2,axis=1)
Out[73]: <tf.Tensor: id=271, shape=(2,), dtype=float32, numpy=array([1.4142135,
1.4142135], dtype=float32)>

In [70]: tf.norm(b,ord=1)
Out[70]: <tf.Tensor: id=255, shape=(), dtype=float32, numpy=4.0>

In [71]: tf.norm(b,ord=1,axis=0)
Out[71]: <tf.Tensor: id=260, shape=(2,), dtype=float32, numpy=array([2., 2.],
dtype=float32)>

In [72]: tf.norm(b,ord=1,axis=1)
Out[72]: <tf.Tensor: id=265, shape=(2,), dtype=float32, numpy=array([2., 2.],
dtype=float32)>
```

日期: /

reduce_min/max/mean 减维

```
In [76]: a=tf.random.normal([4,10])

In [78]: tf.reduce_min(a),tf.reduce_max(a),tf.reduce_mean(a)
Out[78]:
(<tf.Tensor: id=283, shape=(), dtype=float32, numpy=-1.1872448>,
 <tf.Tensor: id=285, shape=(), dtype=float32, numpy=2.1353827>,
 <tf.Tensor: id=287, shape=(), dtype=float32, numpy=0.3523524>)

In [79]: tf.reduce_min(a,axis=1),tf.reduce_max(a,axis=1),tf.reduce_mean(a,axis=1)
Out[79]:
(<tf.Tensor: id=292, shape=(4,), dtype=float32, numpy=array([-0.3937837,
-1.1872448, -1.0798895, -1.1366792], dtype=float32)>,
 <tf.Tensor: id=294, shape=(4,), dtype=float32, numpy=array([1.9718986, 1.1612172,
2.1353827, 2.0984378], dtype=float32)>,
 <tf.Tensor: id=296, shape=(4,), dtype=float32, numpy=array([ 0.61504304,
-0.01389184,  0.606747  ,  0.20151143], dtype=float32)>)
```

argmax/argmin

得到位置.

```
In [80]: a.shape
Out[80]: TensorShape([4, 10])

In [81]: tf.argmax(a).shape
Out[81]: TensorShape([10])

In [83]: tf.argmax(a)
Out[83]: <tf.Tensor: id=305, shape=(10,), dtype=int64, numpy=array([0, 0, 2, 3, 1,
3, 0, 1, 2, 0])>

In [82]: tf.argmin(a).shape
Out[82]: TensorShape([10])
```

tf.equal

比较.

```
In [44]: a=tf.constant([1,2,3,2,5])

In [45]: b=tf.range(5)

In [46]: tf.equal(a,b)
Out[46]: <tf.Tensor: id=170, shape=(5,), dtype=bool, numpy=array([False, False,
False, False, False])>

In [47]: res=tf.equal(a,b)

In [48]: tf.reduce_sum(tf.cast(res, dtype=tf.int32))
Out[48]: <tf.Tensor: id=175, shape=(), dtype=int32, numpy=0>
```

```
In [99]: a
<tf.Tensor: id=308, shape=(2, 3), dtype=float32, numpy=
array([[0.1 , 0.2 , 0.7 ],
       [0.9 , 0.05, 0.05]], dtype=float32)>

In [100]: pred=tf.cast(tf.argmax(a,axis=1), dtype=tf.int32)
Out[101]: <tf.Tensor: id=324, shape=(2,), dtype=int32, numpy=array([2, 0])>

In [110]: y
Out[110]: <tf.Tensor: id=328, shape=(2,), dtype=int32, numpy=array([2, 1],
dtype=int32)>

In [112]: tf.equal(y,pred)
Out[112]: <tf.Tensor: id=335, shape=(2,), dtype=bool, numpy=array([ True, False])>

In [113]: correct=tf.reduce_sum(tf.cast(tf.equal(y,pred),dtype=tf.int32))

In [114]: correct
Out[114]: <tf.Tensor: id=340, shape=(), dtype=int32, numpy=1>

In [115]: correct/2
Out[115]: <tf.Tensor: id=345, shape=(), dtype=float64, numpy=0.5>
```

日期:

tf.unique, 去重. 去掉重复元素

```
In [116]: a=tf.range(5)

In [117]: tf.unique(a)
Out[117]: Unique(y=<tf.Tensor: id=351, shape=(5,), dtype=int32, numpy=array([0, 1,
2, 3, 4], dtype=int32)>, idx=<tf.Tensor: id=352, shape=(5,), dtype=int32,
numpy=array([0, 1, 2, 3, 4], dtype=int32)>)

In [118]: a=tf.constant([4,2,2,4,3])

In [119]: tf.unique(a)
Out[119]: Unique(y=<tf.Tensor: id=356, shape=(3,), dtype=int32, numpy=array([4, 2,
3], dtype=int32)>, idx=<tf.Tensor: id=357, shape=(5,), dtype=int32,
numpy=array([0, 1, 1, 0, 2], dtype=int32)>)
```

有重复 [0,1,2] 去重. Dilor.

在不重复中的索引号.

很普遍.

- Sort/argsort (顺序 序号)
- Topk (前一个, 后一个)
- Top-5 Acc.

Sort/argsort

```
In [86]: a=tf.random.shuffle(tf.range(5)) #numpy=array([2, 0, 3, 4, 1])

In [98]: tf.sort(a,direction='DESCENDING') 降序.
Out[98]: <tf.Tensor: id=397, shape=(5,), dtype=int32, numpy=array([4, 3, 2, 1,
0])>

In [91]: tf.argsort(a,direction='DESCENDING')
Out[91]: <tf.Tensor: id=409, shape=(5,), dtype=int32, numpy=array([3, 2, 0, 4,
1])>

In [92]: idx=tf.argsort(a,direction='DESCENDING')

In [93]: tf.gather(a,idx)
Out[93]: <tf.Tensor: id=422, shape=(5,), dtype=int32, numpy=array([4, 3, 2, 1,
0])>
```

idx. 最大的有效位置.

最大的元素值.

高维 → 对某一个维度.

```
In [95]: a=tf.random.uniform([3,3],maxval=10, dtype=tf.int32)
array([[4, 6, 8],
       [9, 4, 7],
       [4, 5, 1]])
In [97]: tf.sort(a)
array([[4, 6, 8],
       [4, 7, 9],
       [1, 4, 5]])
In [98]: tf.sort(a, direction='DESCENDING')
array([[8, 6, 4],
       [9, 7, 4],
       [5, 4, 1]])
In [99]: idx=tf.argsort(a)
array([[0, 1, 2],
       [1, 2, 0],
       [2, 0, 1]])
```

0~9

行

日期:

Top_k

indices

values.

```
In [104]: a
array([[4, 6, 8],
       [9, 4, 7],
       [4, 5, 1]])
In [101]: res=tf.math.top_k(a,2)

In [102]: res.indices
<tf.Tensor: id=467, shape=(3, 2), dtype=int32, numpy=
array([[2, 1],
       [0, 2],
       [1, 0]])>

In [103]: res.values
<tf.Tensor: id=466, shape=(3, 2), dtype=int32, numpy=
array([[8, 6],
       [9, 7],
       [5, 4]])>
```

accuracy.

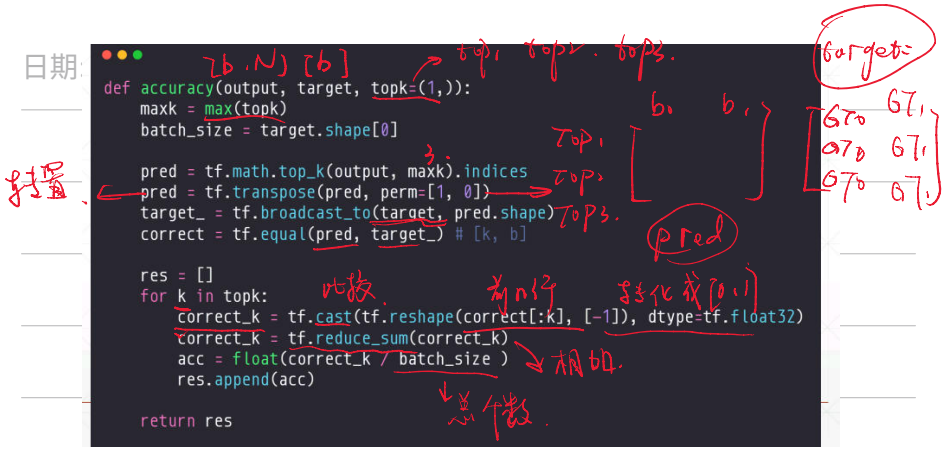
- Prob:[0.1, 0.2, 0.3, 0.4]
- Label:[2] 真实值.
- Only consider top-1 prediction: [3]
- Only consider top-2 prediction: [3, 2]
- Only consider top-3 prediction: [3, 2, 1]

Top-5

```
In [59]: prob=tf.constant([[0.1,0.2,0.7], [0.2,0.7,0.1]])
In [60]: target=tf.constant([2,0])

In [61]: k_b=tf.math.top_k(prob,3).indices
array([[2, 1, 0],
       [1, 0, 2]])
In [63]: k_b=tf.transpose(k_b, [1,0])
array([[2, 1],
       [1, 0],
       [0, 2]])

In [65]: target=tf.broadcast_to(target, [3,2])
<tf.Tensor: id=214, shape=(3, 2), dtype=int32, numpy=
array([[2, 0],
       [2, 0],
       [2, 0]])>
```

日期

```python
def accuracy(output, target, topk=(1,)):
    maxk = max(topk)
    batch_size = target.shape[0]

    pred = tf.math.top_k(output, maxk).indices
    pred = tf.transpose(pred, perm=[1, 0])
    target_ = tf.broadcast_to(target, pred.shape)
    correct = tf.equal(pred, target_) # [k, b]

    res = []
    for k in topk:
        correct_k = tf.cast(tf.reshape(correct[:k], [-1]), dtype=tf.float32)
        correct_k = tf.reduce_sum(correct_k)
        acc = float(correct_k / batch_size )
        res.append(acc)

    return res
```



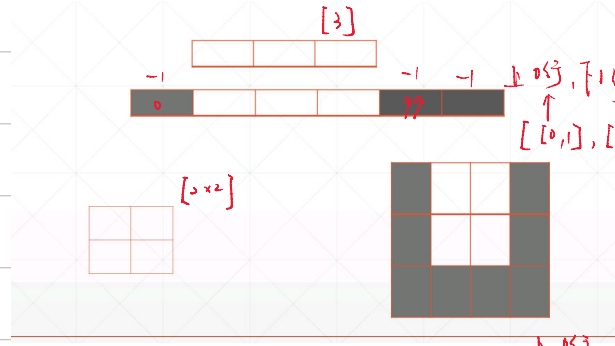
日期: /

填充与复制

pad

tile

broadcast_to




```
In [120]: a=tf.reshape(tf.range(9), [3,3])
<tf.Tensor: id=365, shape=(3, 3), dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=int32)>

In [123]: tf.pad(a,[[0,0],[0,0]])
Out[123]:
<tf.Tensor: id=369, shape=(3, 3), dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=int32)>

In [124]: tf.pad(a,[[1,0],[0,0]])
Out[124]:
<tf.Tensor: id=372, shape=(4, 3), dtype=int32, numpy=
array([[0, 0, 0],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=int32)>
```

```
In [125]: tf.pad(a,[[1,1],[0,0]])
<tf.Tensor: id=375, shape=(5, 3), dtype=int32, numpy=
array([[0, 0, 0],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [0, 0, 0]], dtype=int32)>

In [126]: tf.pad(a,[[1,1],[1,0]])
<tf.Tensor: id=378, shape=(5, 4), dtype=int32, numpy=
array([[0, 0, 0, 0],
       [0, 0, 1, 2],
       [0, 3, 4, 5],
       [0, 6, 7, 8],
       [0, 0, 0, 0]], dtype=int32)>

In [127]: tf.pad(a,[[1,1],[1,1]])
<tf.Tensor: id=381, shape=(5, 5), dtype=int32, numpy=
array([[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]], dtype=int32)>
```

日期: /

image padding.

4张 28×28

```
In [128]: a=tf.random.normal([4,28,28,3])

In [129]: b=tf.pad(a,[[0,0],[2,2],[2,2],[0,0]])

In [130]: b.shape
Out[130]: TensorShape([4, 32, 32, 3])
```

batch — 4个 batch.

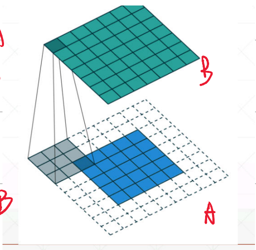
行 列

B>A

A→A'

A'>A

使 A'=B



[1,5,5,1]

[[0,0],[2,2],[2,2],[0,0]]

↓

[1,9,9,1]

tile.

repeat data along dim n times.

[a,b,c], 2 ⟶ [a,b,c,a,b,c]

选择维度不同的倍数

```
In [132]: a
<tf.Tensor: id=396, shape=(3, 3), dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=int32)>

In [133]: tf.tile(a,[1,2])
<tf.Tensor: id=399, shape=(3, 6), dtype=int32, numpy=
array([[0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5],
       [6, 7, 8, 6, 7, 8]], dtype=int32)>

In [134]: tf.tile(a,[2,1])
<tf.Tensor: id=402, shape=(6, 3), dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=int32)>
```

[1,2]: 行不变，新列是原来的2倍 → w\[1,2] 排列原数据

[2,1]: 行2倍 → 列不变.

都复制成2倍,

```
In [135]: tf.tile(a,[2,2])
<tf.Tensor: id=405, shape=(6, 6), dtype=int32, numpy
array([[0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5],
       [6, 7, 8, 6, 7, 8],
       [0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5],
       [6, 7, 8, 6, 7, 8]], dtype=int32)>
```

日期: /

张量限幅.

- clip_by_value 把指值来裁剪
- relu
- clip_by_norm
- gradient clipping

```
In [149]: a
Out[149]: <tf.Tensor: id=422, shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3,
4, 5, 6, 7, 8, 9], dtype=int32)>

In [150]: tf.maximum(a,2)
Out[150]: <tf.Tensor: id=426, shape=(10,), dtype=int32, numpy=array([2, 2, 2, 3,
4, 5, 6, 7, 8, 9], dtype=int32)>

In [151]: tf.minimum(a,8)
Out[151]: <tf.Tensor: id=429, shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3,
4, 5, 6, 7, 8, 8], dtype=int32)>

In [152]: tf.clip_by_value(a,2,8)
Out[152]: <tf.Tensor: id=434, shape=(10,), dtype=int32, numpy=array([2, 2, 2, 3,
4, 5, 6, 7, 8, 8], dtype=int32)>
```

max(x,y) if x<y, res=y. else x.

min(x,y) if x>y res=y else x.

[2,8] 2~8. [min,max]

relu.

```
In [153]: a=a-5
Out[154]: <tf.Tensor: id=437, shape=(10,), dtype=int32, numpy=array([-5, -4, -3,
-2, -1,  0,  1,  2,  3,  4], dtype=int32)>

In [155]: tf.nn.relu(a)
Out[155]: <tf.Tensor: id=439, shape=(10,), dtype=int32, numpy=array([0, 0, 0, 0,
0, 0, 1, 2, 3, 4], dtype=int32)>

In [156]: tf.maximum(a,0)
Out[156]: <tf.Tensor: id=442, shape=(10,), dtype=int32, numpy=array([0, 0, 0, 0,
0, 0, 1, 2, 3, 4], dtype=int32)>
```

日期: /

$\sqrt{\Sigma x^2}$ 等比放缩。

clip_by_norm 范数。 不改变 gradient 方向。

```
In [157]: a=tf.random.normal([2,2],mean=10)
<tf.Tensor: id=449, shape=(2, 2), dtype=float32, numpy=
array([[12.217459 , 10.1498375],
       [10.84643  , 10.972536 ]], dtype=float32)>

In [159]: tf.norm(a)
Out[159]: <tf.Tensor: id=455, shape=(), dtype=float32, numpy=22.14333>

In [161]: aa=tf.clip_by_norm(a,15)
<tf.Tensor: id=473, shape=(2, 2), dtype=float32, numpy=
array([[8.276167 , 6.8755493],
       [7.3474245, 7.43285  ]], dtype=float32)>

In [162]: tf.norm(aa)
Out[162]: <tf.Tensor: id=496, shape=(), dtype=float32, numpy=15.000001>
```

归一到 [0,1] 再乘以 x. → 到一个范围

$\sqrt{\Sigma x^2}$.

15: new norm.

Gradient clipping

- Gradient Exploding or vanishing
  过大 过小
- set lr=1
  learning rate.
- new_grads, total_norm = tf.clip_by_global_norm(grads,25)
  缩放后的grad. 没有clipping的norm.

对所有的进行缩放. → 整体的norm. → 所有grads.

$\Sigma ||W_1|| + ||W_2|| ....$

一个网络包含了很多个维度方向 $[w_1, w_2, w_3 ...]$

每个方向都存在梯度 gradient_w_1 gradient_w_2

保持整体缩放的方向不变.

```
(x, y), _ = datasets.mnist.load_data()
x = tf.convert_to_tensor(x, dtype=tf.float32) / 50.

i@z68:~/TutorialsCN/code_TensorFlow2.0/lesson18-数据限幅$ python main.py
2.0.0-dev20190225
x: (60000, 28, 28) y: (60000, 10)
sample: (128, 28, 28) (128, 10)
==before==
tf.Tensor(89.03711, shape=(), dtype=float32)
tf.Tensor(2.6175494, shape=(), dtype=float32)
tf.Tensor(118.17449, shape=(), dtype=float32)
tf.Tensor(2.1617627, shape=(), dtype=float32)
tf.Tensor(134.27968, shape=(), dtype=float32)
tf.Tensor(2.5254946, shape=(), dtype=float32)
0 loss: 28.88848876953125
==before==
tf.Tensor(1143.292, shape=(), dtype=float32)
tf.Tensor(35.841225, shape=(), dtype=float32)
tf.Tensor(1279.236, shape=(), dtype=float32)
tf.Tensor(24.312374, shape=(), dtype=float32)
tf.Tensor(1185.6311, shape=(), dtype=float32)
tf.Tensor(17.80448, shape=(), dtype=float32)
```

```
        print('==before==')
        for g in grads:
            print(tf.norm(g))
        grads, _ = tf.clip_by_global_norm(grads, 15)

        print('==after==')
        for g in grads:
            print(tf.norm(g))
```

```
i@z68:~/TutorialsCN/code_TensorFlow2.0/lesson18-数据限幅$ python main.py
2.0.0-dev20190225
x: (60000, 28, 28) y: (60000, 10)
sample: (128, 28, 28) (128, 10)
==before==
tf.Tensor(118.00854, shape=(), dtype=float32)
tf.Tensor(3.5821552, shape=(), dtype=float32)
tf.Tensor(146.76697, shape=(), dtype=float32)
tf.Tensor(2.830059, shape=(), dtype=float32)
tf.Tensor(183.28879, shape=(), dtype=float32)
tf.Tensor(3.4088597, shape=(), dtype=float32)
==after==
tf.Tensor(6.734187, shape=(), dtype=float32)
tf.Tensor(0.20441659, shape=(), dtype=float32)
tf.Tensor(8.375294, shape=(), dtype=float32)
tf.Tensor(0.16149803, shape=(), dtype=float32)
tf.Tensor(10.45942, shape=(), dtype=float32)
tf.Tensor(0.19452743, shape=(), dtype=float32)
0 loss: 41.25679016113281
```

日期: /

高阶操作.

where.

scatter

meshgrid

布尔型.

where.(tensor)

返回true的坐标.

[3,3]

True	False	False
False	True	False
False	False	True

[3,2] → [[0,0] [1,1] [2,2]]

shape.

```
In [3]: a=tf.random.normal([3,3])
<tf.Tensor: id=11, shape=(3, 3), dtype=float32, numpy=
array([[ 1.6420907 ,  0.43938753, -0.31872085],
       [ 1.144599  , -0.02425919, -0.9576591 ],
       [ 1.5931814 ,  0.1182256 , -0.39948994]], dtype=float32)>

In [5]: mask=a>0  返回布尔值.
<tf.Tensor: id=14, shape=(3, 3), dtype=bool, numpy=
array([[ True,  True, False],
       [ True, False, False],
       [ True,  True, False]])>

In [7]: tf.boolean_mask(a,mask)  得到为true的具体值.
<tf.Tensor: id=42, shape=(5,), dtype=float32, numpy=
array([1.6420907 , 0.43938753, 1.144599  , 1.5931814 , 0.1182256 ],
      dtype=float32)>  (具体值)

In [8]: indices=tf.where(mask)
<tf.Tensor: id=44, shape=(5, 2), dtype=int64, numpy=
array([[0, 0],
       [0, 1],
       [1, 0],
       [2, 0],
       [2, 1]])>

In [10]: tf.gather_nd(a,indices)  将原值取回来. 二步合起来可实现 boolearn.
<tf.Tensor: id=46, shape=(5,), dtype=float32, numpy=
array([1.6420907 , 0.43938753, 1.144599  , 1.5931814 , 0.1182256 ],
      dtype=float32)>
```

```
In [11]: mask
<tf.Tensor: id=14, shape=(3, 3), dtype=bool, numpy=
array([[ True,  True, False],
       [ True, False, False],
       [ True,  True, False]])>

In [12]: A=tf.ones([3,3])

In [13]: B=tf.zeros([3,3])

In [14]: tf.where(mask, A, B)
<tf.Tensor: id=55, shape=(3, 3), dtype=float32, numpy=
array([[1., 1., 0.],
       [1., 0., 0.],
       [1., 1., 0.]], dtype=float32)>
```

where (cond, A, B)

(mask) T 从 A 上选.

F 从 B 上选.

scatter_nd

- tf.scatter_nd(
  - indices,
  - updates,
  - shape

根据 indices. 将 updates 中对的数据.

依次更新到 shape 中. 返回 output.

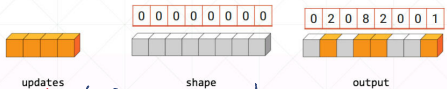
底的是全 0.

更新现有 tensor = A

A − A' → clear

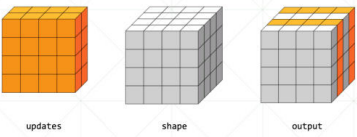
A'' → scatter_nd (保留必要 nd 数据).

将两个 output 相累加




```
In [17]: indices = tf.constant([[4], [3], [1], [7]])
In [18]: updates = tf.constant([9, 10, 11, 12])
In [19]: shape = tf.constant([8])

In [20]: tf.scatter_nd(indices, updates, shape)
Out[20]: <tf.Tensor: id=60, shape=(8,), dtype=int32, numpy=array([ 0, 11,  0, 10,
 9,  0,  0, 12], dtype=int32)>
```

日期: 二维.



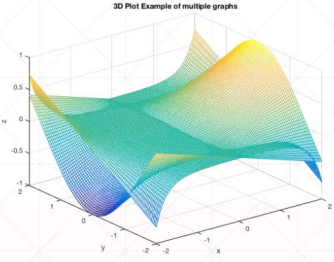

```
In [21]: indices = tf.constant([[0], [2]])  # 指定

In [22]: updates = tf.constant([   [[5, 5, 5, 5], [6, 6, 6, 6],
    ...:                            [7, 7, 7, 7], [8, 8, 8, 8]],
    ...:
    ...:                           [[5, 5, 5, 5], [6, 6, 6, 6],
    ...:                            [7, 7, 7, 7], [8, 8, 8, 8]]])
    ...:
In [24]: updates.shape
Out[24]: TensorShape([2, 4, 4])

In [23]: shape = tf.constant([4, 4, 4])

In [25]: tf.scatter_nd(indices, updates, shape)
<tf.Tensor: id=65, shape=(4, 4, 4), dtype=int32, numpy=
array([[[5, 5, 5, 5],
        [6, 6, 6, 6],
        [7, 7, 7, 7],
        [8, 8, 8, 8]],

       [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]],

       [[5, 5, 5, 5],
        [6, 6, 6, 6],
        [7, 7, 7, 7],
        [8, 8, 8, 8]],

       [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]]], dtype=int32)>
```

→更新

→底板.

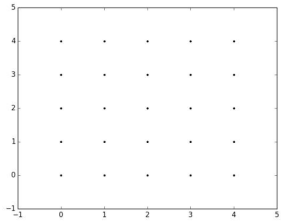
meshgrid.

- [-2, -2]
- [-1, -2]
- [0, -2]
- [-2,-1]
- [-1,-1]
- ...
- [2,2]



日期: /

- [y, x, 2]
  - [5, 5, 2]
- [N, 2]



Points.

取样.

采样.

```
points = []                                  -2~2 :5
for y in np.linspace(-2,2,5):
  for x in np.linspace(-2,2,5):
    points.append([x,y])

return np.array(points)
```

X[-2,2]

Y[-2,2]

points

[N,2]

如何通过 GPU acceleration 实现

```
In [39]: y=tf.linspace(-2.,2,5)
In [40]: y
Out[40]: <tf.Tensor: id=136, shape=(5,), dtype=float32, numpy=array([-2., -1.,
 0.,  1.,  2.], dtype=float32)>

In [41]: x=tf.linspace(-2.,2,5)

In [42]: points_x, points_y=tf.meshgrid(x,y)

In [43]: points_x.shape
Out[43]: TensorShape([5, 5])
```

```
In [44]: points_x
<tf.Tensor: id=162, shape=(5, 5), dtype=float32, numpy=
array([[-2., -1.,  0.,  1.,  2.],
       [-2., -1.,  0.,  1.,  2.],
       [-2., -1.,  0.,  1.,  2.],
       [-2., -1.,  0.,  1.,  2.],
       [-2., -1.,  0.,  1.,  2.]], dtype=float32)>

In [45]: points_y
<tf.Tensor: id=163, shape=(5, 5), dtype=float32, numpy=
array([[-2., -2., -2., -2., -2.],
       [-1., -1., -1., -1., -1.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.,  2.]], dtype=float32)>
```

堆在一起 [5,5,2] 坐标对分.

x坐标.

y坐标.

```
 [51]: points_x.shape
Out[51]: TensorShape([5, 5])

In [52]: points=tf.stack([points_x,points_y], axis=2)
<tf.Tensor: id=176, shape=(5, 5, 2), dtype=float32, numpy=
array([[[-2., -2.],
        [-1., -2.],
        [ 0., -2.],
        [ 1., -2.],
        [ 2., -2.]],

       [[-2., -1.],
        [-1., -1.],
        [ 0., -1.],
        [ 1., -1.],
        [ 2., -1.]],
```

stack 还原成 [5,5,2]

reshape [N,2]

日期: /

数据集加载

小型数据集.　　keras.datasets

- keras.datasets
  - 变为 tensor
- tf.data.Dataset.from_tensor_slices
  - shuffle
  - map
  - batch
  - repeat
- we will talk *Input Pipeline* later

- boston housing
  - Boston housing price regression dataset.
- mnist/fashion mnist
  - MNIST/Fashion-MNIST dataset. 手写字.
- cifar10/100
  - small images classification dataset.
- imdb
  - sentiment classification dataset.

Mnist

numpy格式

```
In [5]: (x,y), (x_test,y_test)=keras.datasets.mnist.load_data()
In [6]: x.shape # (60000, 28, 28)
In [7]: y.shape # (60000,)
In [8]: x.min(),x.max(),x.mean()
Out[8]: (0, 255, 33.318421449829934)

In [9]: x_test.shape, y_test.shape
Out[9]: ((10000, 28, 28), (10000,))

In [10]: y[:4]
Out[10]: array([5, 0, 4, 1], dtype=uint8)

In [11]: y_onehot=tf.one_hot(y, depth=10)
In [12]: y_onehot[:2]
<tf.Tensor: id=8, shape=(2, 10), dtype=float32, numpy=
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)>
```

60k train　10k test

0~7 → one-hot (label)

0~255

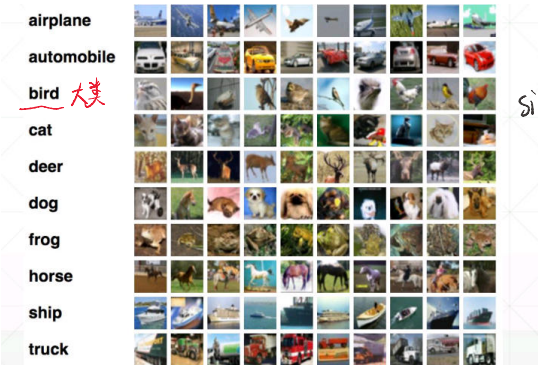
10k

label

10类

日期:

CIFAR10/100.



大类 → 10小类

size: [32, 32, 3]

```
In [14]: (x,y),(x_test,y_test)=keras.datasets.cifar10.load_data()

In [15]: x.shape,y.shape,x_test.shape,y_test.shape
Out[15]: ((50000, 32, 32, 3), (50000, 1), (10000, 32, 32, 3), (10000, 1))

In [16]: x.min(),x.max()
Out[16]: (0, 255)

In [17]: y[:4]
Out[17]:
array([[6],
       [9],
       [9],
       [4]], dtype=uint8)
```

5ok 10K. cifar100.

tf.data.Dataset

- from_tensor_slices()

x: → x~tensor

```
In [5]: (x,y),(x_test,y_test)=keras.datasets.cifar10.load_data()

In [6]: db=tf.data.Dataset.from_tensor_slices(x_test)

In [7]: next(iter(db)).shape
Out[7]: TensorShape([32, 32, 3])

# can not use [x_test,y_test]
In [9]: db=tf.data.Dataset.from_tensor_slices((x_test,y_test))

In [11]: next(iter(db))[0].shape
Out[11]: TensorShape([32, 32, 3])
```

it=iter(db).

next(it) 迭代器

image + label

日期: /

label 顺序.

,Shuffle, suffle 将图片顺序打散

(DataSet)

```
In [12]: db=tf.data.Dataset.from_tensor_slices((x_test,y_test))

In [13]: db=db.shuffle(10000)
```

缓冲区

数据预处理. x→Tensor → float32 → one_hot

Map.

```
In [16]: def preprocess(x,y):
    ...:     x=tf.cast(x, dtype=tf.float32)/255.
    ...:     y=tf.cast(y, dtype=tf.int32)
    ...:     y=tf.one_hot(y,depth=10)
    ...:     return x,y

In [17]: db2=db.map(preprocess)

In [18]: res=next(iter(db2))
In [19]: res[0].shape, res[1].shape
Out[19]: (TensorShape([32, 32, 3]), TensorShape([1, 10]))
In [20]: res[1][:2]
Out[20]: <tf.Tensor: id=58, shape=(1, 10), dtype=float32, numpy=array([[1., 0.,
0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)>
```

[32,32,3] 归一化. 处理后. one_hot

.batch.

```
In [21]: db3=db2.batch(32)

In [25]: res=next(iter(db3))

In [26]: res[0].shape, res[1].shape
Out[26]: (TensorShape([32, 32, 32, 3]), TensorShape([32, 1, 10]))
```

batch 大小 [b,32,32,3] 预处理 [60k,1] squeeze. 压缩.

stop Iteration.

```
In [27]: db_iter = iter(db3)

In [28]: while True:
    ...:     next(db_iter)
    ...:
---------------------------------------------------------------------------
OutOfRangeError                           Traceback (most recent call last)

StopIteration:
```

for x,y in db4

.repeat().

```
In [29]: db4=db3.repeat()

In [30]: db4=db3.repeat(2)
```

停止.

这代次.

日期: /

For example:

```python
def prepare_mnist_features_and_labels(x, y):
    x = tf.cast(x, tf.float32) / 255.0
    y = tf.cast(y, tf.int64)
    return x, y

def mnist_dataset():
    (x, y), (x_val, y_val) = datasets.fashion_mnist.load_data()
    y = tf.one_hot(y, depth=10)
    y_val = tf.one_hot(y_val, depth=10)

    ds = tf.data.Dataset.from_tensor_slices((x, y))
    ds = ds.map(prepare_mnist_features_and_labels)
    ds = ds.shuffle(60000).batch(100)
    ds_val = tf.data.Dataset.from_tensor_slices((x_val, y_val))
    ds_val = ds_val.map(prepare_mnist_features_and_labels)
    ds_val = ds_val.shuffle(10000).batch(100)
    return ds,ds_val
```

日期: /

测试张量. 实战

- Matmul
- Neural Network
- Deep Learning
- Multi-Layer

全连接层

多次线性.

$out = f(X @ W + b)$

$out = relu(X @ W + b)$

降维. 高维.

- $h = relu(X@W + b)$
- $\begin{bmatrix} h_0^0 & h_1^0 \\ h_0^1 & h_1^1 \end{bmatrix} = relu(\begin{bmatrix} x_0^0 & x_1^0 & x_2^0 \\ x_0^1 & x_1^1 & x_2^1 \end{bmatrix} @ \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} + [b_0 \quad b_1])$

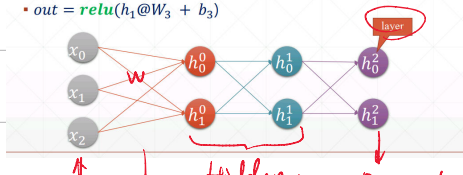
[h0, h1] [1,2]

输入 权值 输出

权值

X @ W + b 即神经网络一层.

- $h_0 = relu(X@W_1 + b_1)$
- $h_1 = relu(h_0@W_2 + b_2)$
- $out = relu(h_1@W_3 + b_3)$

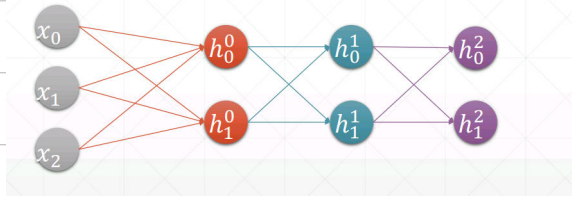


input 每条 weight. Hidden. Out put

权值通过 deep learning 得到. loss 函数. → 梯度下降.

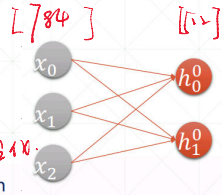
日期: /

- Neural Network in the 1980s
  - 3~5 layers



deeplearning now: n≈1200 layers.

- BigDATA
- ReLU relu.
- Dropout
- BatchNorm
- ResNet
- Xavier Initialization 初始化
- Caffe/TensorFlow/PyTorch



tf.keras.layer.Dense.

```
In [49]: x=tf.random.normal([4,784])

In [48]: net=tf.keras.layers.Dense(512)
In [50]: out=net(x)

In [51]: out.shape
Out[51]: TensorShape([4, 512])

In [52]: net.kernel.shape, net.bias.shape
Out[52]: (TensorShape([784, 512]), TensorShape([512]))
```

自动生成 w 和 b [784, 512] [512]

```
In [3]: net=tf.keras.layers.Dense(10)

In [4]: net.bias
# AttributeError: 'Dense' object has no attribute 'bias'

In [5]: net.get_weights()
Out[5]: []
In [6]: net.weights
Out[6]: []

In [13]: net.build(input_shape=(None,4))
In [14]: net.kernel.shape,net.bias.shape
Out[14]: (TensorShape([4, 10]), TensorShape([10]))

In [15]: net.build(input_shape=(None,20))
In [16]: net.kernel.shape,net.bias.shape
Out[16]: (TensorShape([20, 10]), TensorShape([10]))

In [10]: net.build(input_shape=(2,4))
In [11]: net.kernel
<tf.Variable 'kernel:0' shape=(4, 10) dtype=float32, numpy=
array([[-0.28106192, -0.2522246 ,  0.16050524,  0.43587887, -0.50773597,
```

使用 build 生成 (w, b)

① 自动生成 w

② 使用 build 生成 w

```
In [15]: net.build(input_shape=(None,20))

In [16]: net.kernel.shape,net.bias.shape
Out[16]: (TensorShape([20, 10]), TensorShape([10]))

In [17]: out=net(tf.random.randn((4,12)))

InvalidArgumentError: Matrix size-incompatible: In[0]: [4,12], In[1]: [20,10]
[Op:MatMul]
In [19]: out=net(tf.random.normal((4,20)))

In [20]: out.shape
Out[20]: TensorShape([4, 10])
```

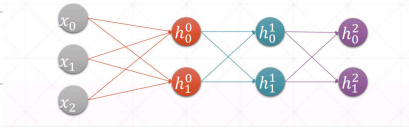
要相等

日期: /

设定好 Dense.

- keras.Sequential([layer1, layer2, layer3])

Dense: 1, 2, 3



创建网络结构

```
x = tf.random.normal([2, 3])

model = keras.Sequential([
    keras.layers.Dense(2, activation='relu'),
    keras.layers.Dense(2, activation='relu'),
    keras.layers.Dense(2)
])
model.build(input_shape=[None, 3])
model.summary()

for p in model.trainable_variables:
    print(p.name, p.shape)
```

3特征. 2输出. print 查. [w1, b1, w2, b2, w3, b3]

输出方式

- $y \in R^d$ 实数
- $y_i \in [0,1],\ i = 0,1,\dots,y_d - 1$ 概率.
- $y_i \in [0,1],\ \sum_{i=0}^{y_d} y_i = 1,\ i = 0,1,\dots,y_d - 1$ 概率和=1
- $y_i \in [-1,1], i = 0,1,\dots,y_d - 1$

1. $y \in R^d$ 实数.

- linear regression 线性回归.
- naïve classification with MSE 误差.
- other general prediction
- $out = relu(X@W + b)$ (正数 的 relu)
  - logits

日期: /

2. $y_i \in [0,1]$

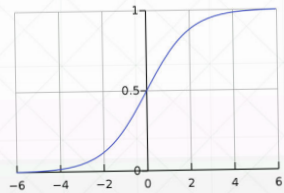
- binary classification
  - y>0.5, →1
  - y<0.5, →0

二分类

- Image Generation
  - rgb 0~255 → [0,1] 归一化.

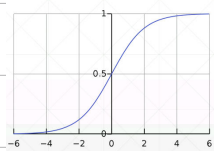
如何实现将数据压缩

- $out = relu(X@W + b)$
- sigmoid
- out' =sigmoid(out)



Sigmoid

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



tf.sigmoid

```
In [10]: a=tf.linspace(-6.,6,10)
<tf.Tensor: id=19, shape=(10,), dtype=float32, numpy=
array([-6.        , -4.6666665, -3.3333333, -2.        , -0.6666665,
        0.666667 ,  2.        ,  3.333334 ,  4.666667 ,  6.       ])

In [12]: tf.sigmoid(a)
<tf.Tensor: id=21, shape=(10,), dtype=float32, numpy=
array([0.00247264, 0.00931597, 0.03444517, 0.11920291, 0.33924365,
       0.6607564 , 0.8807971 , 0.96555483, 0.99068403, 0.9975274 ]

In [13]: x=tf.random.normal([1,28,28])*5
In [15]: tf.reduce_min(x), tf.reduce_max(x)
(<tf.Tensor: id=32, shape=(), dtype=float32, numpy=-18.78872>,
 <tf.Tensor: id=34, shape=(), dtype=float32, numpy=15.466431>)

In [16]: x=tf.sigmoid(x)
In [17]: tf.reduce_min(x), tf.reduce_max(x)
(<tf.Tensor: id=39, shape=(), dtype=float32, numpy=0.0>,
 <tf.Tensor: id=41, shape=(), dtype=float32, numpy=0.99999976>)
```

日期: /

和为 1.

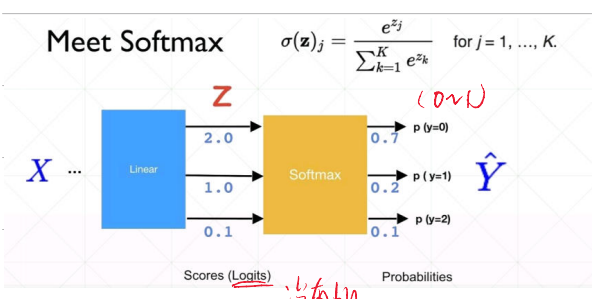
3. $y_i \in [0,1]$ $\sum y_i = 1$

- sigmoid

```
In [21]: a=tf.linspace(-2.,2,5)
In [22]: tf.sigmoid(a)
<tf.Tensor: id=54, shape=(5,), dtype=float32, numpy=
array([0.11920291, 0.26894143, 0.5       , 0.7310586 , 0.880797  ],
      dtype=float32)>
```

- softmax 概率和为 1

```
In [23]: tf.nn.softmax(a)
<tf.Tensor: id=56, shape=(5,), dtype=float32, numpy=
array([0.01165623, 0.03168492, 0.08612854, 0.23412167, 0.6364086 ],
      dtype=float32)>
```



多类问题 (0~1)

没有加和 概率通过 softmax 输出

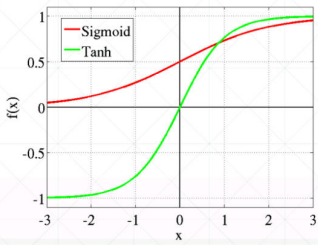
```
In [24]: logits=tf.random.uniform([1,10],minval=-2,maxval=2)
<tf.Tensor: id=64, shape=(1, 10), dtype=float32, numpy=
array([[-1.5842109 ,  1.6475668 , -0.71567106,  1.5819931 ,  0.35972595,
         0.12536812,  1.0662012 , -0.70131207, -1.5194197 ,  1.5553613 ]],
      dtype=float32)>

In [27]: prob=tf.nn.softmax(logits,axis=1)
<tf.Tensor: id=67, shape=(1, 10), dtype=float32, numpy=
array([[0.00946281, 0.23964217, 0.02255392, 0.22443208, 0.06610908,
        0.05229748, 0.13399215, 0.02288011, 0.01009621, 0.21853393]],
      dtype=float32)>

In [29]: tf.reduce_sum(prob,axis=1)
Out[29]: <tf.Tensor: id=70, shape=(1,), dtype=float32, numpy=array([0.99999994],
dtype=float32)>
```

日期: /

4. $y_i \in [-1, 1]$

- tanh

$Sum = 1$



$\tanh(x) = \sinh(x)/\cosh(x) = (e^x - e^{-x})/(e^x + e^{-x})$

sigmoid 激活函数
↓ 下移
tanh.

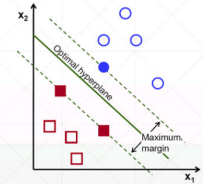
```
In [30]: a
Out[30]: <tf.Tensor: id=53, shape=(5,), dtype=float32, numpy=array([-2., -1.,  0.,
 1.,  2.], dtype=float32)>

In [33]: tf.tanh(a)
<tf.Tensor: id=73, shape=(5,), dtype=float32, numpy=
array([-0.9640276, -0.7615942,  0.       ,  0.7615942,  0.9640276],
      dtype=float32)>
```

日期: /

损失函数.

- MSE 均方差.
- Cross Entropy Loss 分类
- Hinge Loss 支持向量机.

$$\sum_i max(0, 1 - y_i * h_\theta(x_i))$$



MSE

- $loss = \frac{1}{N}\sum(y - out)^2$ N = batch.
- $L_{2-norm} = \sqrt{\sum(y - out)^2}$ 二范数.

```
y = tf.constant([1, 2, 3, 0, 2])
y = tf.one_hot(y, depth=4)  0,1,2,3
y = tf.cast(y, dtype=tf.float32)

out = tf.random.normal([5, 4])  logits.

= MSE.  均值.
loss1 = tf.reduce_mean(tf.square(y-out))

loss2 = tf.square(tf.norm(y-out))/(5*4)  范数.

loss3 = tf.reduce_mean(tf.losses.MSE(y, out)) # VS MeanSquaredError is a class

tf.Tensor(1.0918376, shape=(), dtype=float32)
tf.Tensor(1.0918376, shape=(), dtype=float32)
tf.Tensor(1.0918376, shape=(), dtype=float32)
```

日期: /

Entropy. 熵.

- Uncertainty
- measure of surprise

低熵. 信息量大 不稳定

- lower entropy → more info.

离散.

$$Entropy = -\sum_i P(i)\log P(i)$$

熵.

Claude Shannon

Lottery.

概率均等 → 高熵. (精细度小)

```
In [3]: a=tf.fill([4],0.25)                                          4类. [0.25, 0.25, 0.25, 0.25]
In [6]: a*tf.math.log(a)/tf.math.log(2.)                             a log2 a.
Out[6]: <tf.Tensor: id=13, shape=(4,), dtype=float32, numpy=array([-0.5, -0.5,
-0.5, -0.5], dtype=float32)>
In [7]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))             求和.
Out[7]: <tf.Tensor: id=22, shape=(), dtype=float32, numpy=2.0>       → entropy.  高熵

In [8]: a=tf.constant([0.1,0.1,0.1,0.7])                             概率不均等.
In [9]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))
Out[9]: <tf.Tensor: id=32, shape=(), dtype=float32, numpy=1.3567796>  精细度变大

In [10]: a=tf.constant([0.01,0.01,0.01,0.97])
In [11]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))
Out[11]: <tf.Tensor: id=42, shape=(), dtype=float32, numpy=0.24194068>  精细度高
```

Cross Entropy.

两个分布 交叉熵 分类问题.

$$H(p,q) = -\sum p(x)\log q(x)$$

$$H(p,q) = H(p) + D_{KL}(p|q).$$

相似度 / 距度.

- for $p = q$
  - Minima: $H(p,q) = H(p)$
- for p: one-hot encoding
  - $h(p: [0,1,0]) = -1log1 = 0$
  - $H([0,1,0],[p_0,p_1,p_2]) = 0 + D_{KL}(p|q) = -\mathbf{1}log q_1$

p

当 $p=q$ 时, loss=0

p 预测 q: prob.

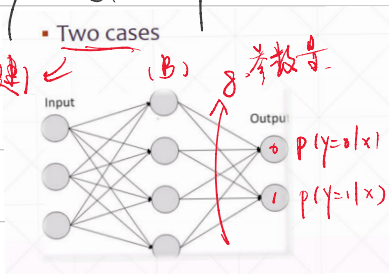
p: one-hot. → label.

日期: /

Binary Classification.

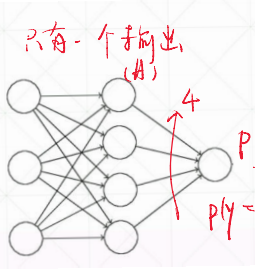
- Two cases

(不关键)



参数多

只有一个输出



Single Output

loss = H(p, q) One_hot

= 0 + log q_i

是cat

$$H(P,Q) = -P(cat)\log Q(cat) - (1-P(cat))\log(1-Q(cat))$$

∵ = 缺 ∴ $P(dog) = (1-P(cat))$ (非cat) 非cat

使概率最大。

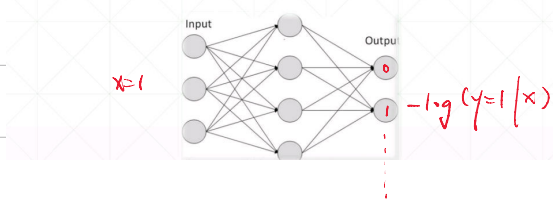
$$H(P,Q) = -\sum_{i=(cat,dog)} P(i)\log Q(i)$$

$$= -P(cat)\log Q(cat) - P(dog)\log Q(dog)$$

loss = $-(y)\log(p) + (1-y)\log(1-p)$

p(cat) 真实分布 概率值.

- $H([0,1,0],[p_0,p_1,p_2]) = 0 + D_{KL}(p|q) = -\mathbf{1}\log q_1$



Classification

label $P_1 = [1 \quad 0 \quad 0 \quad 0 \quad 0]$

预测值. 置信度

$Q_1 = [0.4 \quad 0.3 \quad 0.05 \quad 0.05 \quad 0.2]$

loss.

$$
\begin{aligned}
H(P_1, Q_1) &= -\sum_i P_1(i)\log Q_1(i) \\
&= -(1\log 0.4 + 0\log 0.3 + 0\log 0.05 + 0\log 0.05 + 0\log 0.2) \\
&= -\log 0.4 \\
&\approx 0.916
\end{aligned}
$$

-(label) log(prob).

优化.

$Q_1 = [0.98 \quad 0.01 \quad 0 \quad 0 \quad 0.01]$

$$
\begin{aligned}
H(P_1, Q_1) &= -\sum_i P_1(i)\log Q_1(i) \\
&= -(1\log 0.98 + 0\log 0.01 + 0\log 0 + 0\log 0 + 0\log 0.01) \\
&= -\log 0.98 \\
&\approx 0.02
\end{aligned}
$$

Categorical Cross Entropy. 传两个值.

(y-true.) (prob)

```
In [15]: tf.losses.categorical_crossentropy([0,1,0,0],[0.25,0.25,0.25,0.25])
Out[15]: <tf.Tensor: id=98, shape=(), dtype=float32, numpy=1.3862944>

In [16]: tf.losses.categorical_crossentropy([0,1,0,0],[0.1,0.1,0.8,0.1])
Out[16]: <tf.Tensor: id=117, shape=(), dtype=float32, numpy=2.3978953>

In [17]: tf.losses.categorical_crossentropy([0,1,0,0],[0.1,0.7,0.1,0.1])
Out[17]: <tf.Tensor: id=136, shape=(), dtype=float32, numpy=0.35667497>

In [18]: tf.losses.categorical_crossentropy([0,1,0,0],[0.01,0.97,0.01,0.01])
Out[18]: <tf.Tensor: id=155, shape=(), dtype=float32, numpy=0.030459179>
```

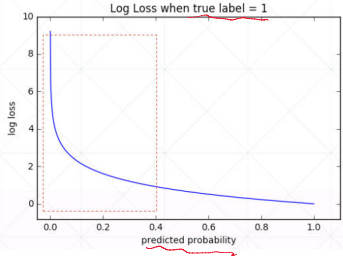
↓ = CrossEntropy.

```
In [20]: criteon([0,1,0,0],[0.1,0.7,0.1,0.1])
Out[20]: <tf.Tensor: id=186, shape=(), dtype=float32, numpy=0.35667497>

In [21]: criteon([0,1],[0.9,0.1])
Out[21]: <tf.Tensor: id=216, shape=(), dtype=float32, numpy=2.3025851>

In [22]: tf.losses.BinaryCrossentropy()([1],[0.1])
Out[22]: <tf.Tensor: id=254, shape=(), dtype=float32, numpy=2.3025842>

In [23]: tf.losses.binary_crossentropy([1],[0.1])
Out[23]: <tf.Tensor: id=281, shape=(), dtype=float32, numpy=2.3025842>
```

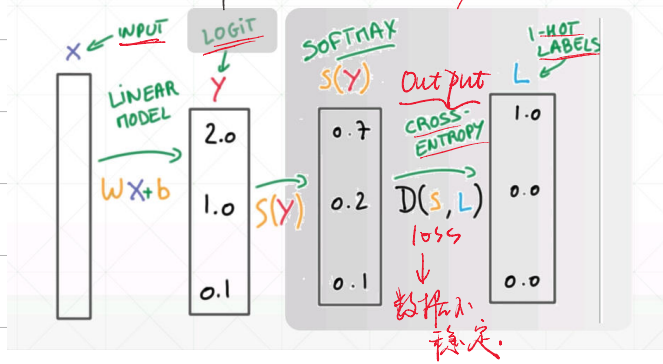
单输入 二分类. 输一个类

日期: /

- sigmoid + MSE
  - gradient vanish
- converge slower
- However
  - e.g. meta-learning



logits → CrossEntropy → loss.




```
In [24]: x=tf.random.normal([1,784])
In [25]: w=tf.random.normal([784,2])
In [26]: b=tf.zeros([2])

In [27]: logits=x@w+b
Out[29]: <tf.Tensor: id=299, shape=(1, 2), dtype=float32, numpy=array([[-26.27812,
 28.63038]], dtype=float32)>

In [30]: prob=tf.math.softmax(logits,axis=1)
Out[31]: <tf.Tensor: id=301, shape=(1, 2), dtype=float32,
numpy=array([[1.4241021e-24, 1.0000000e+00]], dtype=float32)>

In [34]: tf.losses.categorical_crossentropy([0,1],logits, from_logits=True)
Out[34]: <tf.Tensor: id=393, shape=(1,), dtype=float32, numpy=array([0.],
dtype=float32)>

In [35]: tf.losses.categorical_crossentropy([0,1],prob)
Out[35]: <tf.Tensor: id=411, shape=(1,), dtype=float32, numpy=array([1.192093e-
07], dtype=float32)>
```

one-hot

自行softmax

用logit.

直接计算loss

日期: /

梯度下降. Gradient Descent.

What's Gradient?

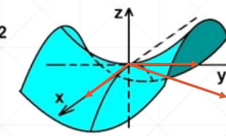
通用 → (标量)具体

- 导数, derivative $\lim_{x \to} \frac{\Delta y}{\Delta x}$ 变动率. 一维. 方向
- 偏微分, partial derivative 沿着某一方向. (x/y.)
- 梯度, gradient 所有方向的偏微分. 合在一起.

$$z = y^2 - x^2$$

$$\frac{\partial z}{\partial x} = -2x$$

$$\frac{\partial z}{\partial y} = 2y$$

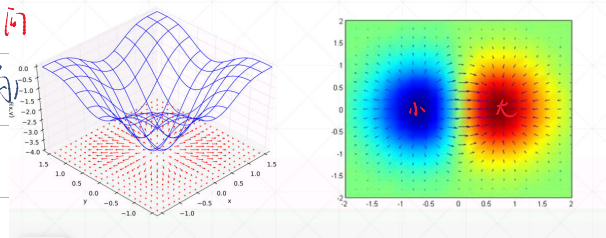


$$\nabla f = \left(\frac{\partial f}{\partial x_1}; \frac{\partial f}{\partial x_2}; \ldots; \frac{\partial f}{\partial x_n}\right)$$

形成一个向量. (有方向)

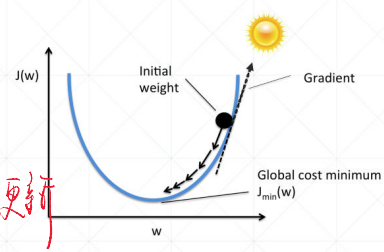
箭头: 梯度方向 (增大方向)

模(速率)



How to search?

- $\nabla f(\theta) \rightarrow larger\ value$



向梯度减小的方向更新

- Search for minima:
  - lr α η

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t).$$

梯度.

lr

For instance

$\theta_1 = 0$

$\theta_2 = 0$

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t).$$

偏微分

$\theta_1 = 4 \quad \theta_2 = -4$

$\nabla\theta_1 = 8 \quad \nabla\theta_2 = -8$

$\theta_1' = 4 - 0.01 \times 8$

$\theta_2' = -4 - 0.01 \times -8$

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

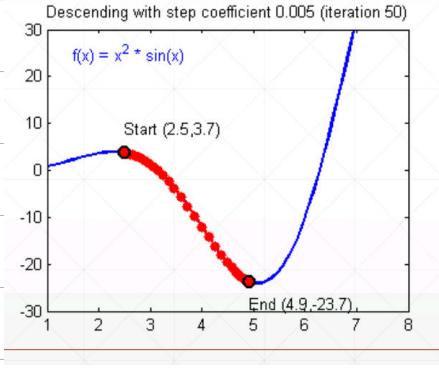
偏微分.

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

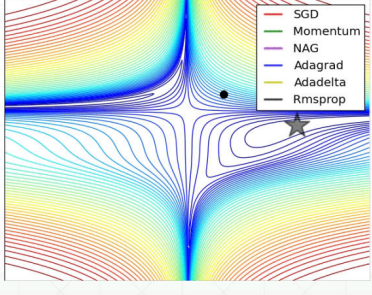
单维

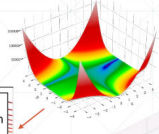


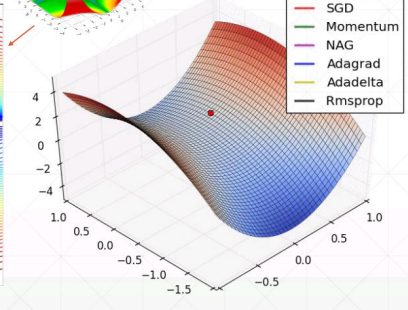
多维.

Learning Process-2

优化对象







日期: /

利用 tensorflow

AutoGrad

在一个环境中

- With Tf.GradientTape() as tape:
  - Build computation graph
  - $loss = f_\theta(x)$

对应的降数.

参数

- [w_grad] = tape.gradient(loss, [w])

```
In [3]: w=tf.constant(1.)
In [4]: x=tf.constant(2.)
In [5]: y=x*w

In [8]: with tf.GradientTape() as tape:
   ...:     tape.watch([w])
   ...:     y2=x*w
In [11]: grad1=tape.gradient(y,[w])
Out[12]: [None]

In [18]: with tf.GradientTape() as tape:
    ...:     tape.watch([w])
    ...:     y2=x*w
In [19]: grad2=tape.gradient(y2,[w])
Out[16]: [<tf.Tensor: id=8, shape=(), dtype=float32, numpy=2.0>]
```

定义参数.

loss

创建环境.

因为 y 没有在环境中.

释放资源

```
In [3]: w=tf.constant(1.)
In [4]: x=tf.constant(2.)
In [5]: y=x*w

In [18]: with tf.GradientTape() as tape:
    ...:     tape.watch([w])
    ...:     y2=x*w
In [19]: grad2=tape.gradient(y2,[w])
Out[16]: [<tf.Tensor: id=8, shape=(), dtype=float32, numpy=2.0>]

In [19]: grad2=tape.gradient(y2,[w])
RuntimeError: GradientTape.gradient can only be called once on non-persistent
tapes.

In [18]: with tf.GradientTape(persistent=True) as tape:
    ...:     tape.watch([w])
      ...
```

调用一次. 可以计算多个参数

日期: /

二阶梯度.

2nd- order

- $y = xw + b$
- $\frac{\partial y}{\partial w} = x$ x和w无关
- $\frac{\partial^2 y}{\partial w^2} = \frac{\partial y'}{\partial w} = \frac{\partial x}{\partial w} = None$

```
with tf.GradientTape() as t1:
  with tf.GradientTape() as t2:
    y = x * w + b
  dy_dw, dy_db = t2.gradient(y, [w, b])
d2y_dw2 = t1.gradient(dy_dw, w)
```

嵌套

None.

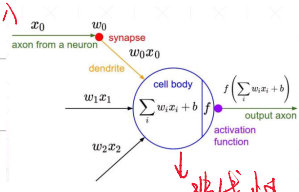
t2梯度1、

t1梯度1

日期: /

激活函数及其梯度.

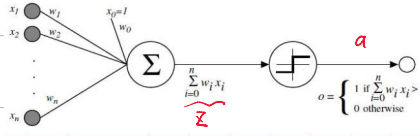
sigmoid tanh. relu.



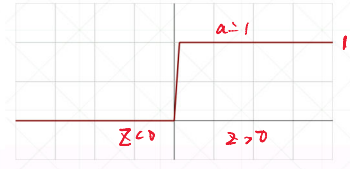
非线性.

阈值响应机制

"激活" 本质



z



Sigmoid / logistic.

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

值域: [0, 1]

"压缩" 本质

prob, RGB...

缺陷 $x \to \infty$ 时, loss 太更新

导数:

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})-1}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}}{(1+e^{-x})^2} - \left(\frac{1}{1+e^{-x}}\right)^2 \\
&= \sigma(x) - \sigma(x)^2 \\
\sigma' &= \sigma(1-\sigma)
\end{aligned}$$

$\sigma' = \sigma(1-\sigma)$

```
a = tf.linspace(-10., 10., 10)

with tf.GradientTape() as tape:
    tape.watch(a)
    y = tf.sigmoid(a)

grads = tape.gradient(y, [a])  梯度
```

```
x: [-10.         -7.7777777  -5.5555553  -3.333333   -1.1111107   1.1111116
   3.333334    5.5555563   7.7777786  10.        ]
y: [4.5388937e-05 4.1878223e-04 3.8510561e-03 3.4445226e-02 2.4766389e-01
 7.5233626e-01 9.6555483e-01 9.9614894e-01 9.9958128e-01 9.9995458e-01]
grad: [4.5386874e-05 4.1860685e-04 3.8362255e-03 3.3258751e-02 1.8632649e-01
 1.8632641e-01 3.3258699e-02 3.8362255e-03 4.1854731e-04 4.5416677e-05]
```

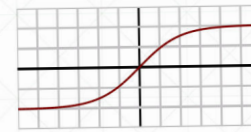
日期: /

Tanh 在 RNN 中使用

sigmoid 缩放 + 平移 → tanh.

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

$$= 2sigmoid(2x) - 1$$



$$\frac{d}{dx}\tanh(x) = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

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

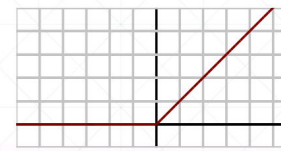
```
In [5]: a=tf.linspace(-5.,5.,10)

In [6]: tf.tanh(a)
Out[6]:
<tf.Tensor: id=10, shape=(10,), dtype=float32, numpy=
array([-0.9999092 , -0.9991625 , -0.99229795, -0.9311096 , -0.50467217,
        0.5046725 ,  0.93110967,  0.99229795,  0.9991625 ,  0.9999092 ],
      dtype=float32)>
```

Rectified Linear Unit (Relu)

优势：梯度不消失.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

没有极值.



$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

```
In [11]: a=tf.linspace(-1.,1.,10)

In [12]: tf.nn.relu(a
<tf.Tensor: id=24, shape=(10,), dtype=float32, numpy=
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.11111116, 0.33333337, 0.5555556 , 0.7777778 , 1.        ],
      dtype=float32)>

In [13]: tf.nn.leaky_relu(a)
<tf.Tensor: id=26, shape=(10,), dtype=float32, numpy=
array([-0.2       , -0.15555556, -0.11111112, -0.06666666, -0.02222222,
        0.11111116,  0.33333337,  0.5555556 ,  0.7777778 ,  1.        ],
      dtype=float32)>
```

[-1, 1]

< 0 时 = kx.

k 通常取较小.

x > 0 时 k = 1.

日期: /

损失函数及其梯度.

Typical Loss.

Mean Square Error $(y-\hat{y})^2$

Cross Entropy Loss $-p\log q$.

MSE

$loss = \sum [y - (xw+b)]^2$ $f(x; w, b)$.

- $loss = \sum[y - f_\theta(x)]^2$
- $\frac{\nabla loss}{\nabla \theta} = 2\sum[y - f_\theta(x)] * \frac{\nabla f_\theta(x)}{\nabla \theta}$

$L_2\text{-}norm = ||y-(xw+b)||_2$ y_1 y_2 $\sqrt{\sum(y_1 - y_2)^2}$

$loss = norm(y-(xw+b))^2$ $x@w+b \rightarrow logit$.

```
In [3]: x=tf.random.normal([2,4])
In [4]: w=tf.random.normal([4,3])
In [5]: b=tf.zeros([3])
In [6]: y=tf.constant([2,0])

In [9]: with tf.GradientTape() as tape:
   ...:     tape.watch([w,b])
   ...:     prob = tf.nn.softmax(x@w+b, axis=1)
   ...:     loss = tf.reduce_mean(tf.losses.MSE(tf.one_hot(y,depth=3), prob))

In [10]: grads = tape.gradient(loss, [w,b])
In [11]: grads[0]
[[-0.00967887, -0.00335512,  0.01303399],
       [-0.04446869,  0.06194263, -0.01747394],
       [-0.04530644,  0.01043231,  0.03487412],
       [ 0.02006017, -0.03638988,  0.0163297 ]]
In [12]: grads[1] # [-0.02585024,  0.06217915, -0.03632889]
```

输出 [2]

label

好读 tf.virable.

使用 y 构建 one_hot

预测

$\frac{\partial L}{\partial w}$ [4,3] 与w一致

$\frac{\partial L}{\partial b}$ [3] 与b一致

日期: /

- soft version of max

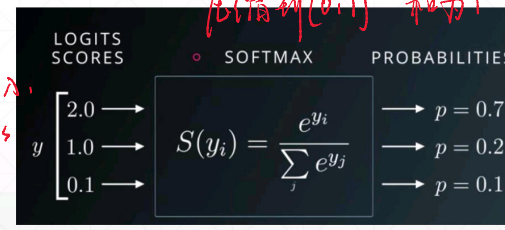
Softmax.

a_0 0 → 0 P_0

a_1 0 → 0 P_1

a_2 0 → 0 P_2

a_3 0 → 0 P_3



Derivative

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

$$f(x) = \frac{g(x)}{h(x)}$$

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}$$

$$g(x) = e^{a_i}$$

$$h(x) = \sum_{k=1}^{N} e^{a_k}$$

when $i = j$

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{e^{a_i}\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}}$$

$$= p_i(1 - p_j)$$

aj 对输出输入.

Derivative

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

$$f(x) = \frac{g(x)}{h(x)}$$

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}$$

$$g(x) = e^{a_i}$$

$$h(x) = \sum_{k=1}^{N} e^{a_k}$$

when $i \neq j$

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{0 - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$= -p_j . p_i$$

$p_i / p_j \in [0,1]$

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j) & if \quad i = j \\ -p_j . p_i & if \quad i \neq j \end{cases}$$

正负.

Or using Kronecker delta $\delta ij = \begin{cases} 1 & if \quad i = j \\ 0 & if \quad i \neq j \end{cases}$

$$\frac{\partial p_i}{\partial a_j} = p_i(\delta_{ij} - p_j)$$

日期: /

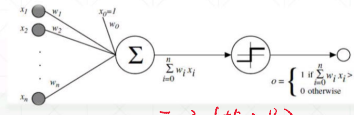
```
In [3]: x=tf.random.normal([2,4])
In [4]: w=tf.random.normal([4,3])
In [5]: b=tf.zeros([3])
In [6]: y=tf.constant([2,0])

In [14]: with tf.GradientTape() as tape:
    ...:     tape.watch([w,b])
    ...:     logits = x@w+b
    ...:     loss =
 tf.reduce_mean(tf.losses.categorical_crossentropy(tf.one_hot(y,depth=3), logits,
                from_logits=True))

In [15]: grads = tape.gradient(loss, [w,b])
<tf.Tensor: id=163, shape=(4, 3), dtype=float32, numpy=
array([[-0.08729011, -0.10937974,  0.19666985],
       [-0.22951077,  0.36995798, -0.14044718],
       [-0.3506433 , -0.2172048 ,  0.56784815],
       [ 0.08480322, -0.26216313,  0.17735994]], dtype=float32)>
In [17]: grads[1] # [-0.07538486,  0.51023775, -0.4348529 ]
```

日期: /

单输出感知机及其梯度

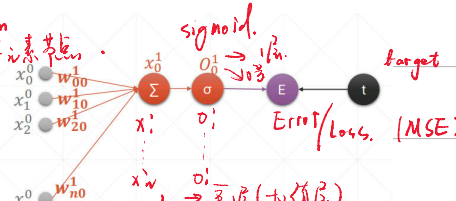
- $y = XW + b$
- $y = \sum x_i * w_i + b$

单层 ↓ 预测 标值 bias.



o → 第0层 (输入层).

输入层 X dim N个元素点.



Derivative

$$E = \frac{1}{2}(O_0^1 - t)^2$$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)\frac{\partial O_0}{\partial w_{j0}}$$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)\frac{\partial \sigma(x_0)}{\partial w_{j0}}$$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)\,\sigma(x_0)(1-\sigma(x_0))\frac{\partial x_0^1}{\partial w_{j0}}$$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)O_0(1-O_0)\frac{\partial x_0^1}{\partial w_{j0}}$$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)O_0(1-O_0)\,x_j^0$$

$O_0 = \sigma(x_0)$

$$\frac{\partial E}{\partial w_{j0}} = (O_0 - t)O_0(1-O_0)\,x_j^0$$

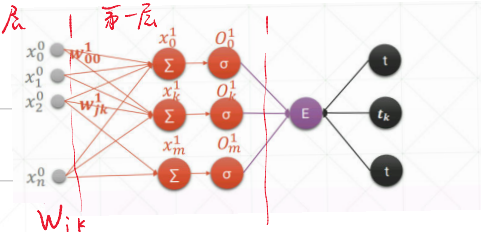
时间相关 i号节点

总误差 O_0, t 和输入 x_j^0 有关.

```
x=tf.random.normal([1,3])  3维.
w=tf.ones([3,1])
b=tf.ones([1])
y = tf.constant([1])  label.

with tf.GradientTape() as tape:
    tape.watch([w, b])
    logits = tf.sigmoid(x@w+b)  归一化.
    loss = tf.reduce_mean(tf.losses.MSE(y, logits))  prob.

grads = tape.gradient(loss, [w, b])
w grad: tf.Tensor(
[[-0.00478814]
 [-0.00588211]
 [ 0.00186196]], shape=(3, 1), dtype=float32)
b grad: tf.Tensor([-0.00444918], shape=(1,), dtype=float32)
```

日期: /

多输出感知机及其梯度.



$$E = \frac{1}{2}\sum (O_i^1 - t_i)^2$$

W_{jk}

x_n^0 $j \to k$ 前一层 后一层

$O_k = \sigma(x_k)$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)\frac{\partial O_k}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)\frac{\partial \sigma(x_k)}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)\,\sigma(x_k)(1-\sigma(x_k))\frac{\partial x_k^1}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\frac{\partial x_k^1}{\partial w_{jk}}$$

只和输入 x_j 以及与输出对应的 t_k, O_k 有关

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\,x_j^0$$

可以拓成矩阵

```
In [3]: x=tf.random.normal([2,4])
In [4]: w=tf.random.normal([4,3])
In [5]: b=tf.zeros([3])
In [6]: y=tf.constant([2,0])

In [9]: with tf.GradientTape() as tape:
   ...:     tape.watch([w,b])
   ...:     prob = tf.nn.softmax(x@w+b, axis=1)
   ...:     loss = tf.reduce_mean(tf.losses.MSE(tf.one_hot(y,depth=3), prob))

In [10]: grads = tape.gradient(loss, [w,b])
In [11]: grads[0]
[[-0.00967887, -0.00335512,  0.01303399],
       [-0.04446869,  0.06194263, -0.01747394],
       [-0.04530644,  0.01043231,  0.03487412],
       [ 0.02006017, -0.03638988,  0.0163297 ]]
In [12]: grads[1] # [-0.02585024,  0.06217915, -0.03632889]
```

日期: /

链式法则.

3门中间复杂.

Rules	Function	Derivative
Multiplication by constant	cf	cf′
Power Rule	x^n	nx^{n-1}
Sum Rule	f + g	f′ + g′
Difference Rule	f - g	f′ − g′
Product Rule	fg	f g′ + f′ g
Quotient Rule	f/g	$(f' g - g' f)/g^2$
Reciprocal Rule	1/f	$-f'/f^2$
Chain Rule (as "Composition of Functions")	f ∘ g	(f′ ∘ g) × g′
Chain Rule (using ′)	f(g(x))	f′(g(x))g′(x)
Chain Rule (using $\frac{d}{dx}$)	$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$	

Basic Rule.

f+g.

f-g.

Product rule.

乘积.

- $(fg)' = f'g + fg'$
- $x^{4\prime} = (x^2 * x^2)' = 2x * x^2 + x^2 * 2x = 4x^3$

Quotient Rule

- $(\frac{f}{g})' = \frac{f'g - fg'}{g^2}$
- e.g. Softmax

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

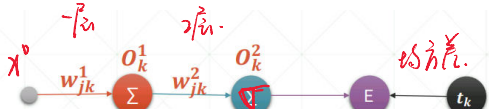
$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{e^{a_i}\sum_{k=1}^{N} e^{a_k} - e^{a_j}e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} = \frac{e^{a_i}\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}} = p_i(1 - p_j)$$

日期: /

Chain rule

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

- $y_2 = y_1 w_2 + b_2$
- $y_1 = x w_1 + b_1$
- $\frac{\partial y_2}{\partial w_1} = \frac{\partial f(y_1)}{\partial w_1} = \frac{\partial f(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w_1} = w_2 * x$
  - $y_2 = (x w_1 + b_1) * w_2 + b_2$



- $\frac{\partial E}{\partial w^1_{jk}} = \frac{\partial E}{\partial O^1_k}\frac{\partial O^1_k}{\partial x} = \frac{\partial E}{\partial O^2_k}\frac{\partial O^2_k}{\partial O^1_k}\frac{\partial O^1_k}{\partial x}$

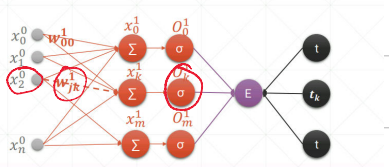
```
x = tf.constant(1.)
w1 = tf.constant(2.)
b1 = tf.constant(1.)
w2 = tf.constant(2.)
b2 = tf.constant(1.)

with tf.GradientTape(persistent=True) as tape:
    tape.watch([w1, b1, w2, b2])

    y1 = x * w1 + b1
    y2 = y1 * w2 + b2

dy2_dy1 = tape.gradient(y2, [y1])[0]
dy1_dw1 = tape.gradient(y1, [w1])[0]
dy2_dw1 = tape.gradient(y2, [w1])[0]
tf.Tensor(2.0, shape=(), dtype=float32)
tf.Tensor(2.0, shape=(), dtype=float32)
```

日期: /

多层感知机梯度

链式

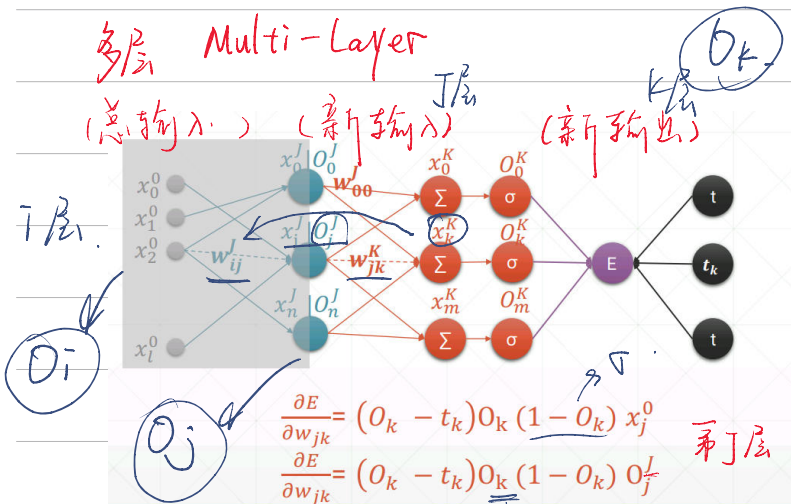
$$\frac{\partial E}{\partial w_{jk}^1} = \frac{\partial E}{\partial o_k^1}\frac{\partial o_k^1}{\partial x} = \frac{\partial E}{\partial o_k^2}\frac{\partial o_k^2}{\partial o_k^1}\frac{\partial o_k^1}{\partial x}$$



$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\,x_j^0$$

多层 Multi-Layer

(总输入) (新输入) (新输出)



$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\,x_j^0$$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\,O_j^J$$

再J层 (k-1)

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)\,O_j^J$$

$$\frac{\partial E}{\partial w_{jk}} = \delta_k^K \quad O_j^J$$

上一层输入.

O_j^J: 中间变量

只跟 O_k 节点有关 (J-1层)

$W_{ij} \to O_j \to X_k$.

日期：

$$\frac{\partial E}{\partial W_{ij}} = O_j(1-O_j)O_i \sum_{k\in K}(O_k - t_k)O_k(1-O_k)W_{jk}$$

$$\frac{\partial E}{\partial W_{ij}} = O_i O_j(1-O_j) \sum_{k\in K}\delta_k W_{jk}$$

$\delta_k = (O_k - t_k)O_k(1-O_k)$.

δ_k 与 k 层有关

For an output layer node $k \in K$

$$\frac{\partial E}{\partial W_{jk}} = O_j \delta_k$$

where

$$\delta_k = O_k(1-O_k)(O_k - t_k)$$

For a hidden layer node $j \in J$

$$\frac{\partial E}{\partial W_{ij}} = O_i \delta_j$$

where

$$\delta_j = O_j(1-O_j) \sum_{k\in K} \delta_k W_{jk}$$

$$\frac{\partial E}{\partial w_{jk}} = (O_k - t_k)O_k(1-O_k)O_j^J$$

$$\frac{\partial E}{\partial w_{jk}} = \delta_k^K O_j^J$$

$$\frac{\partial E}{\partial W_{ij}} = O_j(1-O_j)O_i \sum_{k\in K}(O_k - t_k)O_k(1-O_k)W_{jk}$$

$$\frac{\partial E}{\partial W_{ij}} = O_i O_j(1-O_j) \sum_{k\in K}\delta_k W_{jk}$$

迭代更新

- δ_k^K
- $\frac{\partial E}{\partial w_{jk}}$
- δ_j^J
- $\frac{\partial E}{\partial w_{ij}}$
- δ_i^I
- $\frac{\partial E}{\partial w_{ni}}$

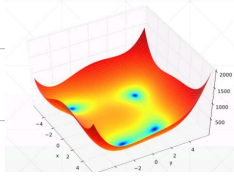
意义：任何一层的输出，都可以由上一层输出 O 和这一层的节点信息 δ 相乘得到

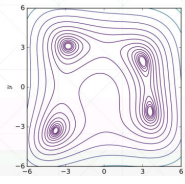
日期: /

Himmelblau 函数优化

Himmelblau function

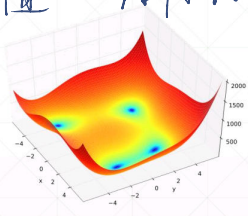
$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2.$$





有四个角 四个局部最小值. 全局最小值.

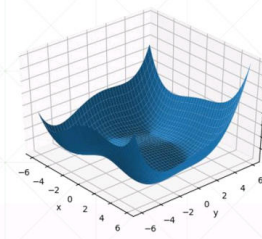
- $f(3.0, 2.0) = 0.0,$
- $f(-2.805118, 3.131312) = 0.0,$
- $f(-3.779310, -3.283186) = 0.0$
- $f(3.584428, -1.848126) = 0.0.$



初始点是会影响结果之

绘制

```python
def himmelblau(x):                                  # 构造函数
    return (x[0] ** 2 + x[1] - 11) ** 2 + (x[0] + x[1] ** 2 - 7) ** 2

x = np.arange(-6, 6, 0.1)                           # 步进
y = np.arange(-6, 6, 0.1)                           # 生成一系列点
print('x,y range:', x.shape, y.shape)
X, Y = np.meshgrid(x, y)                            # 对 meshgrid 生成点
print('X,Y maps:', X.shape, Y.shape)
Z = himmelblau([X, Y])                              # loss = y

fig = plt.figure('himmelblau')
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z)
ax.view_init(60, -30)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```



初始状态.

```
# [1., 0.], [-4, 0.], [4, 0.]
x = tf.constant([-4., 0.])

for step in range(200):

    with tf.GradientTape() as tape:
        tape.watch([x])
        y = himmelblau(x)

    grads = tape.gradient(y, [x])[0]
    x -= 0.01*grads
```

constant → Tensor 需要 watch

dense layer 构建 → variable.

跟踪梯度

求导 得到 gradients

lr $x' = x - lr * grad$ 更新参数

Mnist 手写问题实战

Epoch (时期)

当一个完整的数据集通过了神经网络一次.

(所有样本在神经网络中,进行一次正向,一次反向)

都返回了一次,这个过程称为一次 epoch.

及一个 epoch 就是将所有训练样本训练一次

然而一个 epoch 的样本数量过大,所以需要

将它成 batch 进行训练

日期: /

Batch (批. 样本)

将数据集分成若干个 batch.

Batch_Size 每批样本的大小

Iteration (一次迭代)

训练一个 Batch 就是一次迭代 及一个 step.

① 下载数据. 预处理 → preprocess.

转化成 Tensor. 定义 batch shuffle 打乱

训练集. 测试集 迭代器 *reshape.

② 构建神经网络

model 输入: 输出: [b, 10]

28×28=784

[b, 784] → [b, 256] → [b, 128] → [b, 64] → [b, 32] → [b, 10]

Dense1 Dense2 Dense3 Dense4

使用 layer.Dense. 输出维度. 激活函数.

*** layer.Dense(256, activation=tf.nn.relu)

前连接. 中间层输出可以随便取.

权值 param: 330 = 32×10 + 10 → bias.

上层 这一层

日期: /

③完成前向传播

定义：多少个 eporch. 对每个 eporch 中 in batch 循环

x reshape 28×28 = 784

将 reshape 之后的 x 输入到网络中

★ logit = model(x) 得到 logit

y → one_hot (形状与 x 输出一致)

计算 loss, 计算 grads

④完成更新.

keras 自带.

★ 定义优化器 optimizer. 自动完成更新过程.

使用 optimizer 完成更新

⑤测试：使用 test 集.

logit —softmax 概率最大→ prob [b,10] → [b] int64→int32

(y 也用 one_hot) 和 label 进行比较. True=+1

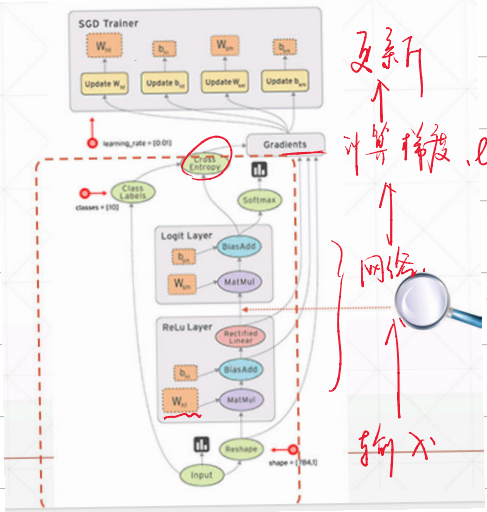
correct / total_num.

日期: /

Tensor Board 可视化.

如何查看数值？

可视化：



更新

计算梯度，loss

网络

输入

如何查，accuracy?

Tensor Board / Visdom (pytorch)

- Installation 安装
- Curves 监听.
- Image Visualization
- Listen logdir 监听目录
- build summary instance 新建日志
- fed data into summary instance

Step1.run listener

- open URL: http://localhost:6006

cmd → 进入路径 → dir → tensorboard

```
i@z68:~/TutorialsCN/code_TensorFlow2.0/lesson28-可视化$ tensorboard --logdir logs
TensorBoard 1.13.0 at http://z68:6006 (Press CTRL+C to quit)
c: , total_correct/total)
```

日期:

Step2.build summary

在程序中，创建 writer.

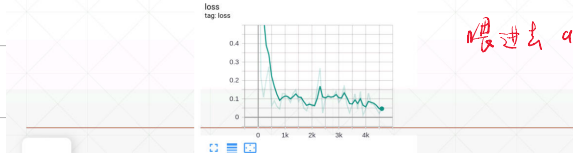
```
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
log_dir = 'logs/' + current_time
summary_writer = tf.summary.create_file_writer(log_dir)
```

路径.

Step3.fed scalar

标量.

```
with summary_writer.as_default():
    tf.summary.scalar('loss', float(loss), step=epoch)
    tf.summary.scalar('accuracy', float(train_accuracy), step=epoch)
```



喂进去 acc和loss.

Step3.fed single Image

```
# get x from (x,y)
sample_img = next(iter(db))[0]
# get first image instance
sample_img = sample_img[0]
sample_img = tf.reshape(sample_img, [1, 28, 28, 1])
with summary_writer.as_default():
    tf.summary.image("Training sample:", sample_img, step=0)
```

一张图.

b

(通道数)

Step3.fed multi-images

显示时会分单元.

```
val_images = x[:25]
val_images = tf.reshape(val_images, [-1, 28, 28, 1])
with summary_writer.as_default():
    tf.summary.scalar('test-acc', float(loss), step=step)
    tf.summary.image("val-onebyone-images:", val_images, max_outputs=25,
step=step)
```

b

将多张图片组合成一张.

```
val_images = tf.reshape(val_images, [-1, 28, 28])
figure = image_grid(val_images)
tf.summary.image('val-images:', plot_to_image(figure), step=step)
```

日期: /

- datasets
- layers
- losses
- metrics

optimizers.

keras 高层 API.

Keras. Metrics.

默认是 tf. keras.

- Metrics 测量器. 新建
- update_state 加数据.
- result().numpy() 计算
- reset_states 清空

Step1.Build a meter

```
acc_meter = metrics.Accuracy()
loss_meter = metrics.Mean()
```

Step2.Update data

添加数据.

```
loss_meter.update_state(loss)
acc_meter.update_state(y, pred)
```

label

Accuracy.

Clear buffer 清理缓存. reset

Step3.Get Average data

```
print(step, 'loss:', loss_meter.result().numpy())
    ...
print(step, 'Evaluate Acc:', total_correct/total, acc_meter.result().numpy())
```

打印 + 清除

```
if step % 100 == 0:
    print(step, 'loss:', loss_meter.result().numpy())
    loss_meter.reset_states()

...
  # evaluate
if step % 500 == 0:
    total, total_correct = 0., 0
    acc_meter.reset_states()
```

日期: /

Compile & Fit.

- Compile 装载 loss和评估器 的选择
- Fit
- Evaluate 评估 测试
- Predict 预测

固定流程.

```
for x,y in db:
    with tf.GradientTape() as tape:
        # [b, 28, 28] => [b, 784]
        x = tf.reshape(x, (-1, 28*28))
        # [b, 784] => [b, 10]
        out = network(x)
        # [b] => [b, 10]
        y_onehot = tf.one_hot(y, depth=10)
        # [b]
        loss = tf.reduce_mean(tf.losses.categorical_crossentropy(y_onehot, out, from_logits=True))

    grads = tape.gradient(loss, network.trainable_variables)
    optimizer.apply_gradients(zip(grads, network.trainable_variables))
```

封成 compile - fit

使用后:

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
```

指定: lr loss. Accuracy (指标)

```
for epoch in range(epochs):

  for step, (x, y) in enumerate(db):
    ....
```

⇒

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10)
```

指定数据集和epoch数等

```
ully opened dynamic library libcublas.so.10.0
4690/4690 [==============================] - 41s 9ms/step - loss: 0.1085
Epoch 2/10
4690/4690 [==============================] - 41s 9ms/step - loss: 0.0597
Epoch 3/10
4690/4690 [==============================] - 41s 9ms/step - loss: 0.0448
Epoch 4/10
4690/4690 [==============================] - 41s 9ms/step - loss: 0.0417
Epoch 5/10
4690/4690 [==============================] - 40s 8ms/step - loss: 0.0513
Epoch 6/10
4690/4690 [==============================] - 40s 9ms/step - loss: 0.1354
Epoch 7/10
4690/4690 [==============================] - 41s 9ms/step - loss: 0.1730
Epoch 8/10
4690/4690 [==============================] - 41s 9ms/step - loss: 0.2567
Epoch 9/10
4690/4690 [==============================] - 39s 8ms/step - loss: 0.3887
```

日期: /

```
    # evaluate
    if step % 500 == 0:
        total, total_correct = 0., 0

        for step, (x, y) in enumerate(ds_val):
            # [b, 28, 28] => [b, 784]
            x = tf.reshape(x, (-1, 28*28))
            # [b, 784] => [b, 10]
            out = network(x)
            # [b, 10] => [b]
            pred = tf.argmax(out, axis=1)
            pred = tf.cast(pred, dtype=tf.int32)
            # bool type
            correct = tf.equal(pred, y)
            # bool tensor => int tensor => numpy
            total_correct += tf.reduce_sum(tf.cast(correct,
dtype=tf.int32)).numpy()
            total += x.shape[0]

        print(step, 'Evaluate Acc:', total_correct/total)
```

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10, validation_data=ds_val,
            validation_steps=2)
```

```
469/469 [==============================] - 5s 10ms/step - loss: 0.2794 - accuracy: 0.8408 - val_loss
: 0.0864 - val_accuracy: 0.9805
Epoch 2/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1378 - accuracy: 0.9590 - val_loss:
 0.0637 - val_accuracy: 0.9805
Epoch 3/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1077 - accuracy: 0.9693 - val_loss:
 0.0801 - val_accuracy: 0.9805
Epoch 4/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0941 - accuracy: 0.9729 - val_loss:
 0.0915 - val_accuracy: 0.9688
Epoch 5/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0833 - accuracy: 0.9775 - val_loss:
 0.1232 - val_accuracy: 0.9609
Epoch 6/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0793 - accuracy: 0.9789 - val_loss:
 0.0678 - val_accuracy: 0.9766
Epoch 7/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0741 - accuracy: 0.9810 - val_loss:
 0.0360 - val_accuracy: 0.9883
Epoch 8/10
```

日期: /

在训练后进行.

Test

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10, validation_data=ds_val,
                validation_steps=2)

network.evaluate(ds_val)
```

if test_acc > 0.97 Save break.

test ← network.evaluate(ds_val) → ds_test

```
Epoch 8/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0710 - accuracy: 0.9807 - val_loss
 0.1060 - val_accuracy: 0.9805
Epoch 9/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0680 - accuracy: 0.9821 - val_loss
 0.0927 - val_accuracy: 0.9766
Epoch 10/10
469/469 [==============================] - 4s 9ms/step - loss: 0.0594 - accuracy: 0.9845 - val_loss
 0.0428 - val_accuracy: 0.9922
79/79 [==============================] - 0s 6ms/step - loss: 0.1205 - accuracy: 0.9757
```

test ←

predict

预测一组数据.

```
sample = next(iter(ds_val))
x = sample[0]
y = sample[1] # one-hot
pred = network.predict(x) # [b, 10]
# convert back to number
y = tf.argmax(y, axis=1)
pred = tf.argmax(pred, axis=1)

print(pred)
print(y)
```

network(x).

日期: /

自定义网络

- keras.Sequential 容器 (层) 串联在一起.
- keras.layers.Layer 层 生成层.
- keras.Model

```
network = Sequential([layers.Dense(256, activation='relu'),
                      layers.Dense(128, activation='relu'),
                      layers.Dense(64, activation='relu'),
                      layers.Dense(32, activation='relu'),
                      layers.Dense(10)])
network.build(input_shape=(None, 28*28))
network.summary()
```

[ ] 生成 list　生成5层.　激活函数

创建 w.b.　参数　network (x).　打印信息.

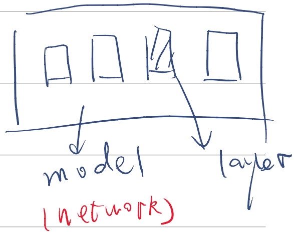
参数管理
- model.trainable_variables　[w1, b1, w2, b2 ...] 传给 optimizer
  - Dense1　Dense2

前向传播
- model.call() forward → model(x) 调用网络

Layer/Model (母类)

(sequential).

初始化.
- Inherit from keras.layers.Layer keras.Model
  自定义层
- ① __init__
- ② call　母类.　对于类的调用　model(x).
- Model: compile/fit/evaluate　predict.



5 of 10

日期: /

layer.Dense(512)
build([None, 784])
W:[784, 512]
b[512]

调用类

自定义层

```python
class MyDense(layers.Layer):

    def __init__(self, inp_dim, outp_dim):
        super(MyDense, self).__init__()

        self.kernel = self.add_variable('w', [inp_dim, outp_dim])
        self.bias = self.add_variable('b', [outp_dim])

    def call(self, inputs, training=None):

        out = inputs @ self.kernel + self.bias

        return out
```

在layers中实现.

784 512

w: b:

自定义.

被model调用.

__call__

w b.

构建网络.

```python
class MyModel(keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = MyDense(28*28, 256)
        self.fc2 = MyDense(256, 128)
        self.fc3 = MyDense(128, 64)
        self.fc4 = MyDense(64, 32)
        self.fc5 = MyDense(32, 10)

    def call(self, inputs, training=None):
        x = self.fc1(inputs)
        x = tf.nn.relu(x)
        x = self.fc2(x)
        x = tf.nn.relu(x)
        x = self.fc3(x)
        x = tf.nn.relu(x)
        x = self.fc4(x)
        x = tf.nn.relu(x)
        x = self.fc5(x)
        return x
```

前向传播的逻辑.

Dense.Layer!

可以加激活 如: x=x-1

激活函数

logits.

最后一层不加激活

日期: /

模型保存与加载

Save / load weights 参数

Save / load entire model

saved_model ONNX.

保存参数

```python
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')

# Restore the weights
model = create_model()
model.load_weights('./checkpoints/my_checkpoint')

loss,acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

train

```python
network.save_weights('weights.ckpt')
print('saved weights.')
del network

network = Sequential([layers.Dense(256, activation='relu'),
                      layers.Dense(128, activation='relu'),
                      layers.Dense(64, activation='relu'),
                      layers.Dense(32, activation='relu'),
                      layers.Dense(10)])
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
network.load_weights('weights.ckpt')
network.evaluate(ds_val)
```

日期: /

所有状态 + 结构 + 参数

```
network.save('model.h5')
print('saved total model.')
del network

print('load model from file')
network = tf.keras.models.load_model('model.h5')

network.evaluate(x_val, y_val)
```

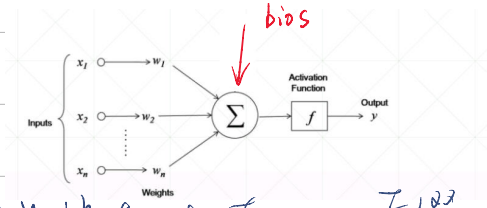
也用创建！

路径.

```
tf.saved_model.save(m, '/tmp/saved_model/')

imported = tf.saved_model.load(path)
f = imported.signatures["serving_default"]
print(f(x=tf.ones([1, 28, 28, 3])))
```

Keras 实战 CIFAR10

使用 my dense 层.



保存：优先保存权值. 后缀名随便取.

日期: /

过拟合和欠拟合.

factor: noise 噪声、误差. $loss=(wx+b-y)^2$.

$y = w*x+b+\varepsilon$ (eps).

$\varepsilon \sim N(0.01, 1)$ 正态分布.

model capacity 容量. (参数量) 多层.

简单 复杂程度. Estimated Ground Truth

under-fitting 模型复杂度 < 真实复杂度

train acc & loss are bad

复杂了

Over-fitting Estimated > Ground Truth

train acc & loss are very good

即训练集在曲线上, but test acc is bad.

Generalization Performance.

日期: /

交叉验证.

Splitting. { train (train) / test

x = [60k] → train 50k / val 10k.

先shuffle!

tf.split(x, num_or_size_splits=[50000, 10000])

network.fit(train, epochs=5, validation_data=val, validation_freq=2)

训练. 循环. 验证. 频率.

network.evaluate(test) → 测试

先验知识 参数.

kaggle. train / val / Test.

K-fold cross-validation.

60k. 10k

train val | test

k 随机取一部分作val.

动态的. 依次取一份.

```
for epoch in range(500):
    idx = tf.range(60000)
    idx = tf.random.shuffle(idx)
    x_train, y_train = tf.gather(x, idx[:50000]), tf.gather(y, idx[:50000])
    x_val, y_val = tf.gather(x, idx[-10000:]) , tf.gather(y, idx[-10000:])

    db_train = tf.data.Dataset.from_tensor_slices((x_train,y_train))
    db_train = db_train.map(preprocess).shuffle(50000).batch(batchsz)

    db_val = tf.data.Dataset.from_tensor_slices((x_val,y_val))
    db_val = db_val.map(preprocess).shuffle(10000).batch(batchsz)

    # training...

    # evalutation...
```

重新切割

index 索引

每个 epoch 都会调

日期: /

```
network.fit(db_train_val, epochs=6, validation_split=0.1, validation_freq=2)
```

Regularization.

Occam's Razor 如无必要，勿增实体

Reduce Overfitting

More data

Constraint model complexity. Droput

① Shallow Data argumentation.

② regularization. 去噪；模糊化 Early stopping

(weight Decay) Acc.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right] + \lambda \sum_{i=1}^{n} |\theta_i|$$

$|\theta| \to 0$ $J(w; x, y) + \frac{1}{2}\lambda \cdot \|w\|^2$

通使参数以获数为0

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \varepsilon.$$

高维参数 $\to 0$

经过 regularization → 退化. → 低维.

日期: /

```python
l2_model = keras.models.Sequential([
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation=tf.nn.relu, input_shape=(NUM_WORDS,)),
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation=tf.nn.relu),
    keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

Flexible regularization

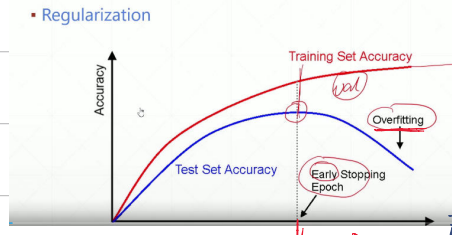
网易云课堂

```python
for step, (x,y) in enumerate(db):
    with tf.GradientTape() as tape:
        # ...
        loss = tf.reduce_mean(tf.losses.categorical_crossentropy(y_onehot, out,
from_logits=True))

        loss_regularization = []
        for p in network.trainable_variables:
            loss_regularization.append(tf.nn.l2_loss(p))
        loss_regularization = tf.reduce_sum(tf.stack(loss_regularization))

        loss = loss + 0.0001 * loss_regularization

    grads = tape.gradient(loss, network.trainable_variables)
    optimizer.apply_gradients(zip(grads, network.trainable_variables))
```

日期: /

Early stop. Drop out. Stochastic Graient Decent.

early stop 设定阈值. (val).

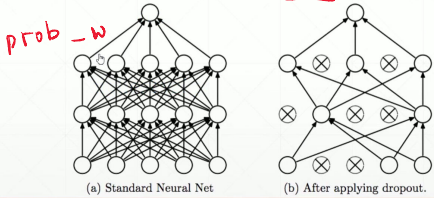


Monitor

Stop at the highest val prof.

Dropout

- *Learning less to learn better*
- Each connection has $p = [0, 1]$ to lose



设 $\sum |w|$ 变小

connection.

当 prob_w=0 时.

将连接断掉.

加一层 dropout.

```
network = Sequential([layers.Dense(256, activation='relu'),
                      layers.Dropout(0.5), # 0.5 rate to drop
                      layers.Dense(128, activation='relu'),
                      layers.Dropout(0.5), # 0.5 rate to drop
                      layers.Dense(64, activation='relu'),
                      layers.Dense(32, activation='relu'),
                      layers.Dense(10)])
```

随机.

加一层, 50% drop.

$\frac{\partial L}{\partial w} = \sum_{i=0}^{i<bok} \frac{\partial L}{\partial w}$

概率一定的. 随机取一个 batch 样本 in 样本.

Stochastic Gradient Descent SGD

Deterministic 一一对应 $x \to f(x)$

所有样本信息平均

日期:

Convolution 局部关联，滑动，共享权值。

卷积神经网络。

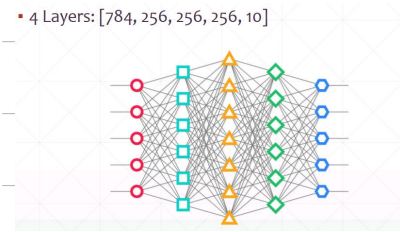
什么是卷积？

Feature map / image [b,h,w,c] channel

位置信息 特征 RGB.

[1,28,28,3] → [3,3,512] → [b,10]

为什么用全连接层？ 参数量： 335k or 1.3MB.



Layer (type)	Output Shape	Param #
dense (Dense)	multiple	200960
dense_1 (Dense)	multiple	65792
dense_2 (Dense)	multiple	65792
dense_3 (Dense)	multiple	2570

Total params: 335,114
Trainable params: 335,114
Non-trainable params: 0

内存占用巨大。 Batch X. Gradient Cache.

小部点能使用 Dense 层。

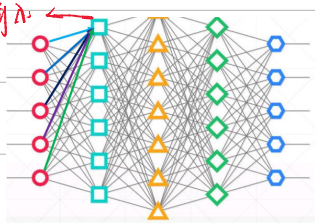
整张图 观察窗口 滑动

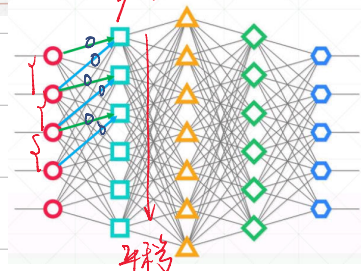
★ Receptive Field 感知域 局部

局部关联

部分输入 观察方式一样。

所有输入



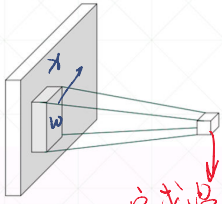


参数权值一样

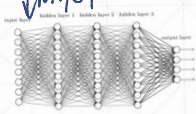
Sharing

平移

使用相同3x3滑动



输入

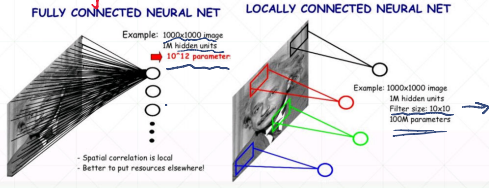


输出

完成滑动

形成一幅新的信息

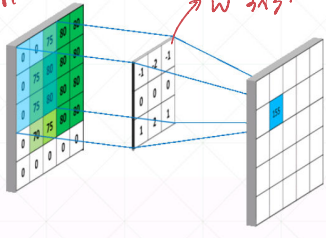
参数减少



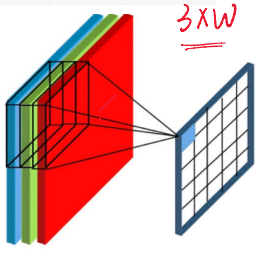
100 x 1M

→ 100个连接线

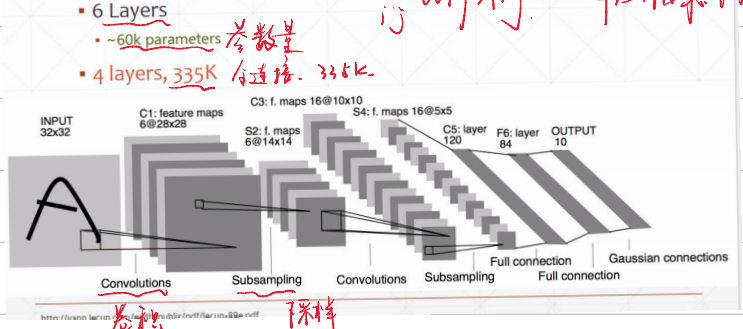
x(窗口) 4x4



3xw 输出



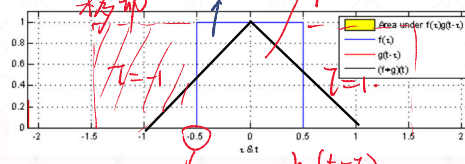
$\sum w_{ij} x_{ij}$ 对应相乘再相加



卷积 下采样

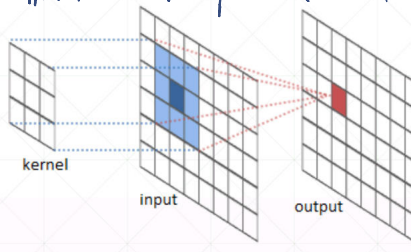
日期:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$



2D Convolution

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$



$g(x,y)$ $f(x,y)$ $o(\Delta x, \Delta y)$

Computer Vison.

Sharpen:

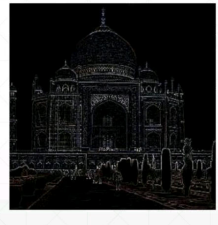
0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Blur:

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

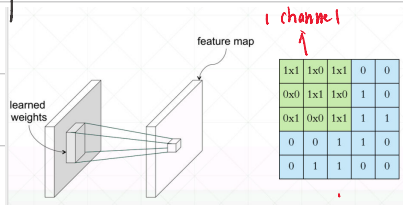
Edge Detect:

0	1	0
1	-4	1
0	1	0



日期: /

卷积神经网络.

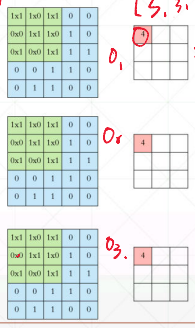


[b,5,5,1] w=[3,3] 9个 weight

[b,5,5,1] → [3,3] → [b,3,3,1] (降维)

输入.

[b,5,5,3]



[3,3,3] kernal 3个[3,3] 或 w

[b,5,5,c]

[c,3,3] ↑通道数一样.

参数量 = 9×c.

[c,3×3] → 1个核

c个3×3矩阵.

与通道数无关.

输出: [b,3,3,1]

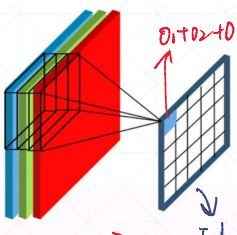
4+4+4	3+3+3	4+4+4
6	12	9
6	9	12

→ $O_1+O_2+O_3$

相乘、相加、累加.

Padding & Stride.

降维.



4+4+4	3+3+3	4+4+4
6	12	9
6	9	12

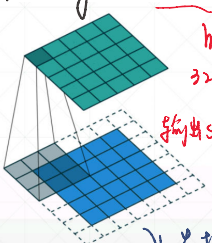
[b,h,w,c]

w=[c,3,3]

[b,h',w',1]

h' 略小于 h.

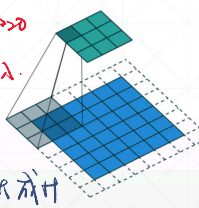
w' 略小于 w. (边界).



h'<h

32→28→14

输出 shape=输入.



h'=h-2.

先把h扩大成H

padding

h → H → h' 使 h'=h.

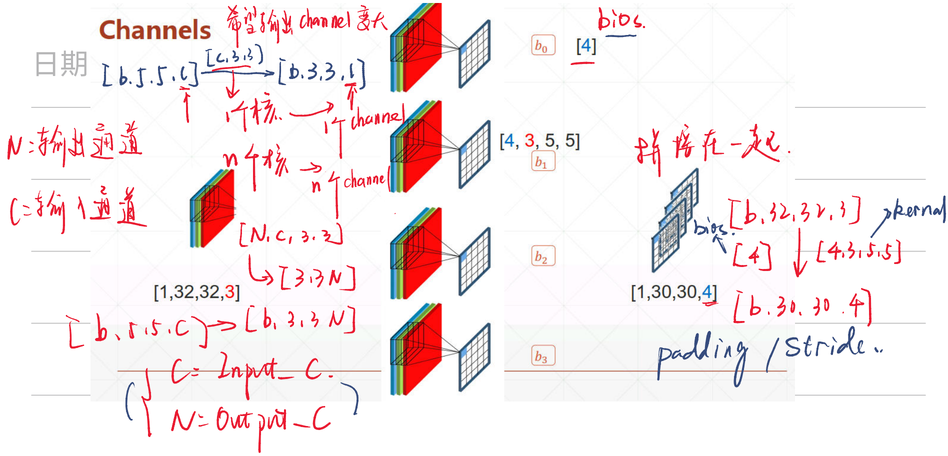
Stride 步长.

改变步长使输出维度减小

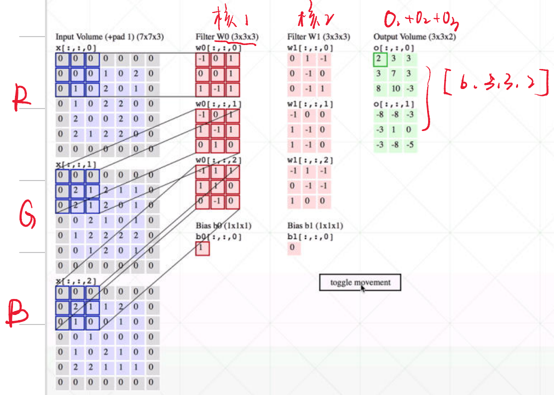
降维. 减得减少. $\frac{1}{2}$. $\frac{1}{3}$

日期

Channels



For instance



x: [b, 28, 28, 3]

one k: [3, 3, 3]

multi-k: [16, 3, 3, 3]

stride: 1

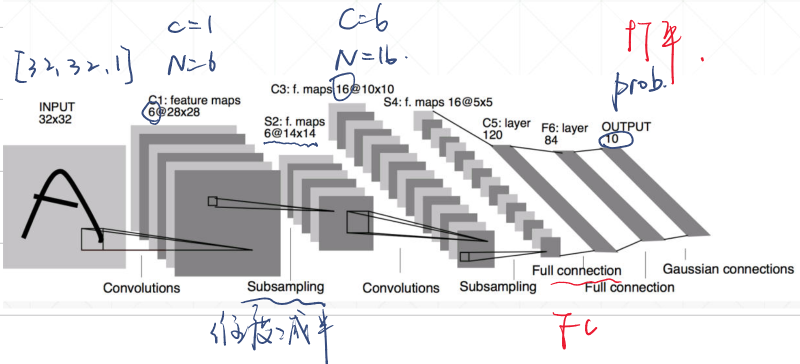
padding: [1,1,1,1]

bias: [16]

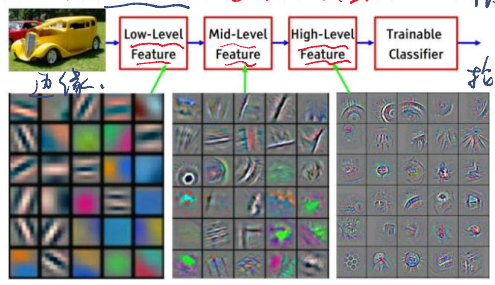
out: [b, 28, 28, 16]

https://skymind.ai/wiki/convolutional-network

LeNet-5



日期: /



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

```
In [1]: import tensorflow as tf
In [2]: from tensorflow.keras import layers

In [6]: layer=layers.Conv2D(4, kernel_size=5,strides=1,padding='valid')
In [8]: out=layer(x)
Out[9]: TensorShape([1, 28, 28, 4])

In [10]: layer=layers.Conv2D(4, kernel_size=5,strides=1,padding='same')
In [11]: out=layer(x)
Out[12]: TensorShape([1, 32, 32, 4])

In [13]: layer=layers.Conv2D(4, kernel_size=5,strides=2,padding='same')
In [14]: out=layer(x)
Out[15]: TensorShape([1, 16, 16, 4])

In [16]: layer.call(x).shape
Out[16]: TensorShape([1, 16, 16, 4])
```

```
In [13]: layer=layers.Conv2D(4, kernel_size=5,strides=2,padding='same')
In [14]: out=layer(x)
Out[15]: TensorShape([1, 16, 16, 4])

In [17]: layer.kernel
<tf.Variable 'conv2d_3/kernel:0' shape=(5, 5, 3, 4) dtype=float32, numpy=
array([[[[-0.16160963,  0.04107726, -0.09828208, -0.00601757],
         [-0.02003701,  0.01415607, -0.07604317, -0.12557343],
         [-0.11157566,  0.1328298 ,  0.14624669, -0.04775226]], ...

In [18]: layer.bias
Out[18]: <tf.Variable 'conv2d_3/bias:0' shape=(4,) dtype=float32, numpy=array([0.,
0., 0., 0.], dtype=float32)>
```

日期:

```
In [21]: w=tf.random.normal([5,5,3,4])
In [22]: b=tf.zeros([4])
In [23]: x.shape
Out[23]: TensorShape([1, 32, 32, 3])

In [29]: out=tf.nn.conv2d(x, w, strides=1, padding='VALID')
Out[30]: TensorShape([1, 28, 28, 4])

In [31]: out = out + b
Out[32]: TensorShape([1, 28, 28, 4])

In [33]: out=tf.nn.conv2d(x,w,strides=2,padding='VALID')
Out[34]: TensorShape([1, 14, 14, 4])
```

自己建 w,b 建 kernal conv2d kernal

Gradient ?

$\frac{\partial Loss}{\partial w}$

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Input

X [b, 3, 3, 1]

weight kernal

[2, 2, C, N]

相乘、相加.

x_{00}	x_{01}	x_{02}
x_{10}	x_{11}	x_{12}
x_{20}	x_{21}	x_{22}

w_{00}	w_{01}
w_{10}	w_{11}

$$O_{00} = x_{00}*w_{00} + x_{01}*w_{01} + x_{10}*w_{10} + x_{11}*w_{11} + b$$

$$O_{01} = x_{01}*w_{00} + x_{02}*w_{01} + x_{11}*w_{10} + x_{12}*w_{11} + b$$

$$O_{10} = x_{10}*w_{00} + x_{11}*w_{01} + x_{20}*w_{10} + x_{21}*w_{11} + b$$

$$O_{11} = x_{11}*w_{00} + x_{12}*w_{01} + x_{21}*w_{10} + x_{22}*w_{11} + b$$

Output

输出:

O_{00}	O_{01}
O_{10}	O_{11}

FC/conv

prob.

0
1
0
0

[b, 2, 2, 1]

打开.

$$\frac{\partial Loss}{\partial w_{00}} = \sum_{i\in\{00,01,10,11\}} \frac{\partial Loss}{\partial O_i}\frac{\partial O_i}{\partial w_i}$$

O_i 中间变量.

$$\frac{\partial O_{00}}{\partial w_{00}} = \frac{\partial(x_{00}*w_{00} + x_{01}*w_{01} + x_{10}*w_{10} + x_{11}*w_{11} + b)}{w_{00}} = x_{00}$$

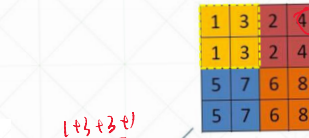
日期: /

池化与采样.

Max/Avg pooling.

下采样 平均

- stride=2



4x4 → 2x2

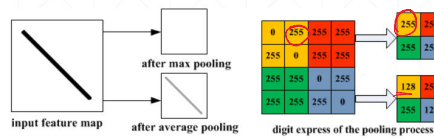
1+3+3+4 / 4

- stride=1

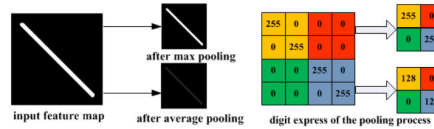
4x4

3x3

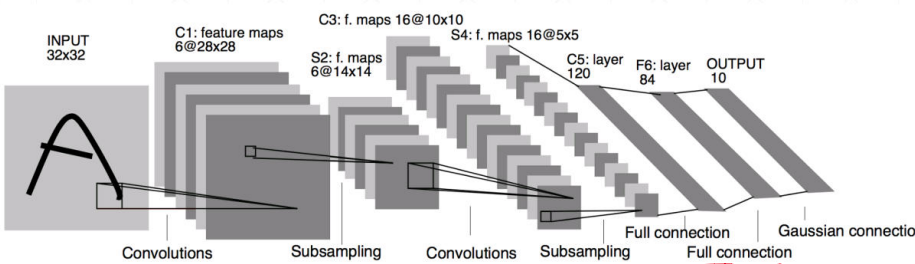
3	3	4
7	7	8
7	7	8



(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback



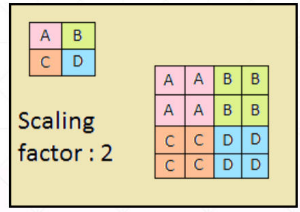
全连接

日期:

```
In [36]: x # TensorShape([1, 14, 14, 4])

In [37]: pool=layers.MaxPool2D(2,strides=2)
In [38]: out=pool(x)
Out[39]: TensorShape([1, 7, 7, 4])

In [40]: pool=layers.MaxPool2D(3,strides=2)
In [41]: out=pool(x)
Out[42]: TensorShape([1, 6, 6, 4])

In [44]: out=tf.nn.max_pool2d(x, 2,strides=2,padding='VALID')
Out[45]: TensorShape([1, 7, 7, 4])
```

UpSampling 2D

- nearest
- bilinear




```
In [47]: x=tf.random.normal([1,7,7,4])

In [48]: layer=layers.UpSampling2D(size=3)
In [49]: out=layer(x)
Out[50]: TensorShape([1, 21, 21, 4])

In [51]: layer=layers.UpSampling2D(size=2)
In [52]: out=layer(x)
Out[53]: TensorShape([1, 14, 14, 4])
```

Relu.

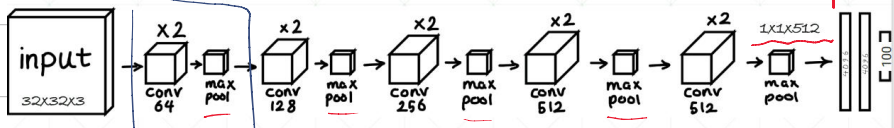
```
In [55]: x=tf.random.normal([2,3])
<tf.Tensor: id=154, shape=(2, 3), dtype=float32, numpy=
array([[-1.533682  , -2.7053335 ,  0.36354962],
       [ 0.00713745,  0.69756126,  0.8053344 ]], dtype=float32)>

In [57]: tf.nn.relu(x)
<tf.Tensor: id=156, shape=(2, 3), dtype=float32, numpy=
array([[0.        , 0.        , 0.36354962],
       [0.00713745, 0.69756126, 0.8053344 ]], dtype=float32)>

In [59]: layers.ReLU()(x)
<tf.Tensor: id=158, shape=(2, 3), dtype=float32, numpy=
array([[0.        , 0.        , 0.36354962],
       [0.00713745, 0.69756126, 0.8053344 ]], dtype=float32)>
```

日期: /

- Load datasets
- Build Network
- Train
- Test

CIFAR100 5 VGG 实战

13 Layers. 5块



一块

prob. [100]

list.

① conv_layer [ ]

生成卷积层. N自定(输出)

layer.Conv2D(64, kernal_size=[3,3], padding="same", activation=tf.nn.relu), 激活函数

channel

strides

padding

生成池化层.

layer.MaxPool2D(pool_size=[2,2], strides=2, padding=same)

size

一般 channel (N) 逐层增大.

每张尺寸 (w, h) 逐层减少

日期: /

2

main 函数:

conv_net = Sequential([conv_layers])

[b, 32, 32, 3] ⇒ [b, 1, 1, 512]

fc_net = Sequential([- - - - - -])

conv_net.build(input_shape=[None, 32, 32, 3]) → 不必须.

fc_net.build(input_shape=[None, 512])

3

preprocess. 预处理. ④数据加载 ⑤创建数据集

out = conv_net(x) [b, 32, 32, 3] ⇒ [b, 1, 1, 512]

打平 out = tf.reshape(out, [-1, 512]) flatten ⇒ [b, 512] ☆

logits = fc_net(out) → prob. [b, 512] ⇒ [b, 100]

one_hot → loss → gradient.

参数. variables = conv_net.trainable_variables + fc_net.tra...

Optimizer = Optimizer.Adam(lr=1e-4)

test.

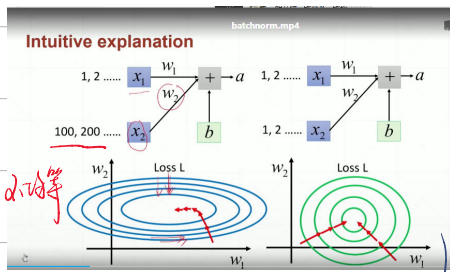
logit —softmax→ prob. —argmax→ pred —和y对比→ correct.

日期: /

Batch Norm (BN)

Sigmoid → 梯度离散.

Normalization. 均匀化. 小会很大很小



小的等

Output都在0周围分布.

缩放+平移

image normalization

x = x - mean

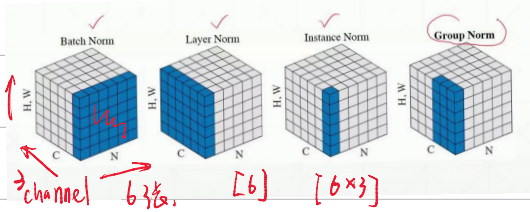
x = x / std

return x.

(RGB)

Batch Normalization Dynamic mean

batch [N, C, H, W] [6, 3, 784]



784

channel 6张. [6] [6×3]

[6, 3, 784] [18]

3个channel

ch 0 → mean → 1个数.

ch 1 → mean

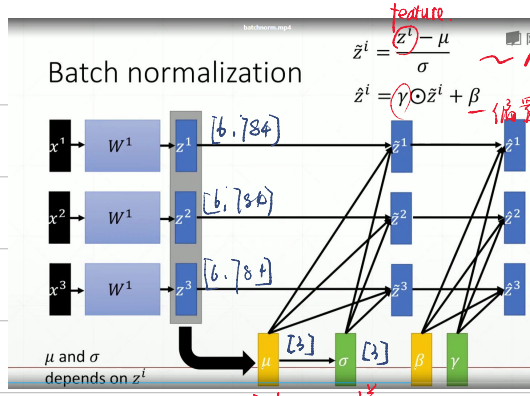
ch 2 → mean

batch=

输出维度 = C.

每个 channel feature 均值. [3]

日期: /



feature.

~N(0,1) 均值 偏置 → N(β, γ).

[6, 3, 784]

running β
running γ
(统计信息)

γ和β 需要学习 (梯度信息)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ // normalize

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

β, γ 自动更新 梯度.

小的值.

N(β, γ).

归一, 平移+缩放.

缩放.

[3, 3, N, C] → -1.

- net = layers.BatchNormalization()
  - axis=-1, 通用. 指定维度 (轴, axis)
  - center=True, → 对应 β. γ.
  - scale=True
  - trainable=True
- net(x, training=None)

偏移 缩

true. 模式. training / test

日期

```
In [3]: net=layers.BatchNormalization()
In [5]: x=tf.random.normal([2,3])
In [6]: out=net(x)

In [7]: net.trainable_variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>]

In [8]: net.variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_mean:0' shape=(3,) dtype=float32, numpy=array([0., 0.,
0.], dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_variance:0' shape=(3,) dtype=float32, numpy=array([1.,
1., 1.], dtype=float32)>]
```

```
for i in range(10):
    with tf.GradientTape() as tape:
        out = net(x, training=True)
        loss = tf.reduce_mean(tf.pow(out,2)) - 1
    grads = tape.gradient(loss, net.trainable_variables)
    optimizer.apply_gradients(zip(grads, net.trainable_variables))

backward(10 steps):
  [<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([0.93549937,
0.9356556 , 0.9355564 ], dtype=float32)>,
   <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([ 1.3411044e-
09,  1.3411045e-08, -4.0978188e-10], dtype=float32)>...]
```

偏差变小，收敛变快

Better performance Converge faster

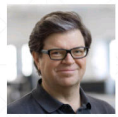
Robust (Stable, larger learning rate)

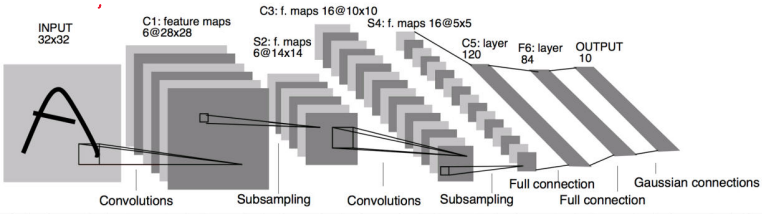
日期: /

经典卷积神经网络. VGG. Google net. Inception.

LeNet-5 手写数字识别. Mnist

- 99.2% acc.
- 5/6 layers





AlexNet

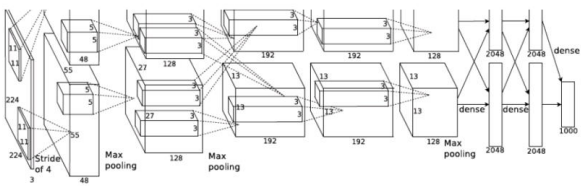
- GTX 580
  - 3GBx2
- 11X11
- 8 layers

5+3.

[224, 224, 3]

Kernal = [11X11]

AlexNet: ILSVRC 2012 winner



- Similar framework to LeNet but:
  - Max pooling, ReLU nonlinearity
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

VGG

- 3x3
- 1x1
- 11-19 layers

VGGNet: ILSVRC 2014 2nd place

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	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	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 conv1-256	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 conv1-512	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 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

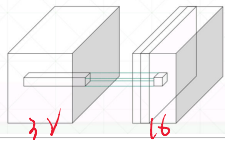
- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3x3 convolutions (with ReLU in between)

- One 7x7 conv layer with C feature maps needs $49C^2$ weights, three 3x3 conv layers need only $27C^2$ weights
- Experimented with 1x1 convolutions

K. Simonyan and A. Zisserman,
Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

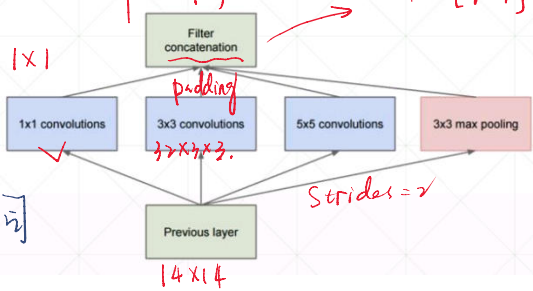
1x1 Convolution

- less computation
- c_in => c_out



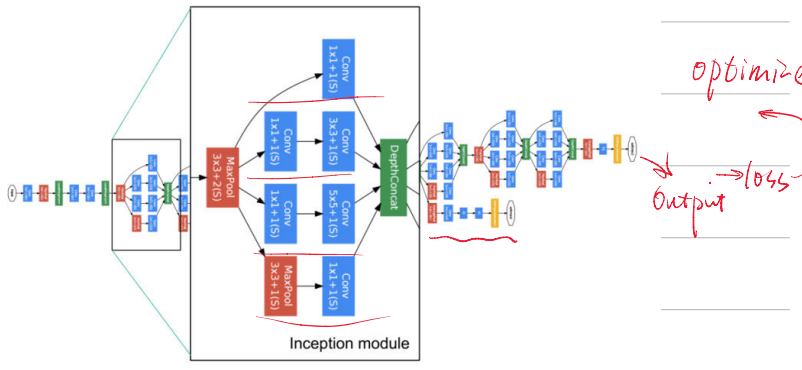
GoogLeNet

- 1st in 2014 ILSVRC
- 22 layers



日期:

GoogLeNet



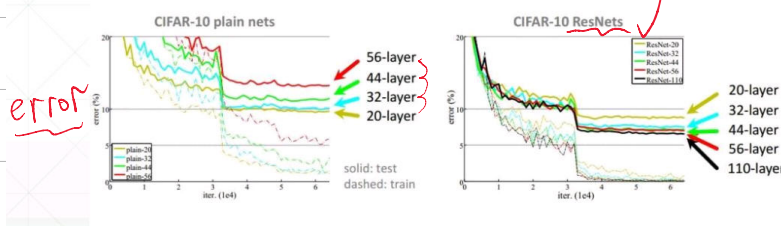
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

Stack more layers? No!!!

- 1000 layers?

But ResNet

CIFAR-10 experiments

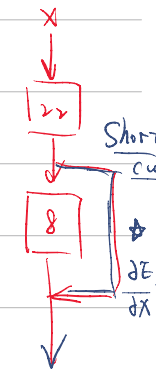


日期: /

深度残差网络

ResNet 与 DenseNet

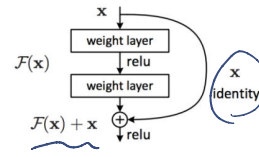
30 → 22



ResNet

- The residual module
  - Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping
  - For some reason, need to skip at least two layers

深度增加
误差增加
梯度离散
梯度爆炸



shortcut

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

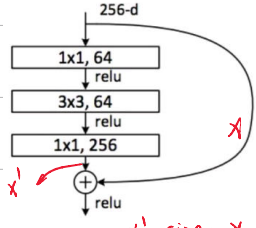
conv + relu + conv + relu + ...

256 → 64 → 64 → 256

ResNet

每个 unit 输入、输出 size 一致

Deeper residual module (bottleneck)



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

直接用

[256, 256, 3, 3]

[64, 256, 1, 1]

↓

[64, 64, 3, 3]

↓

[256, 64, 3, 3]

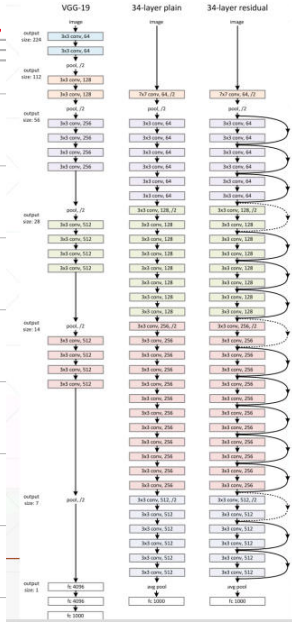
Unit

kernal 很小

x'_size = x_size

参数变少

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)



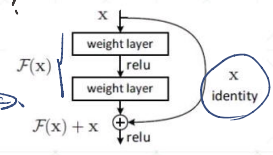
19

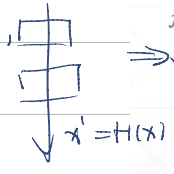
19层+skip

不好优化.

Why call Residual ?

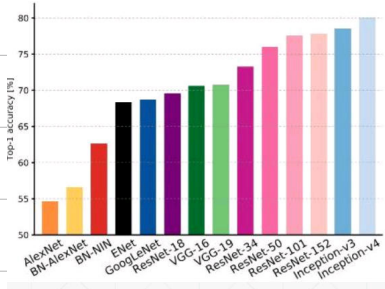
short cut

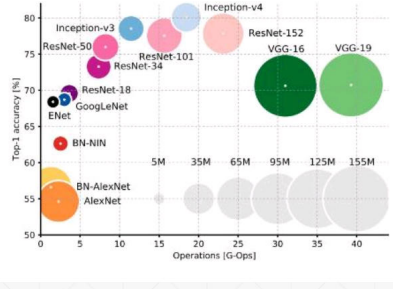




$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ → 学习残差

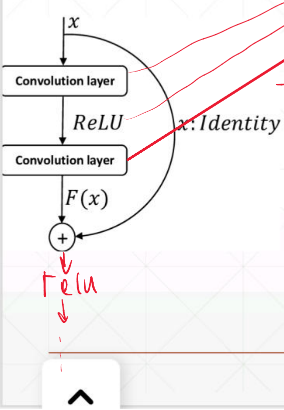
VGG太深了!





Basic Block

conv — BN — relu



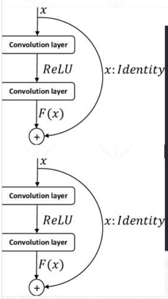

```python
class BasicBlock(layers.Layer):
    def __init__(self, filter_num, stride=1):
        super(BasicBlock, self).__init__()

        self.conv1 = layers.Conv2D(filter_num, (3, 3), strides=stride, padding='same')
        self.bn1 = layers.BatchNormalization()
        self.relu = layers.Activation('relu')
        self.conv2 = layers.Conv2D(filter_num, (3, 3), strides=1, padding='same')
        self.bn2 = layers.BatchNormalization()
        if stride != 1:
            self.downsample = Sequential()
            self.downsample.add(layers.Conv2D(filter_num, (1, 1), strides=stride))
            self.downsample.add(layers.BatchNormalization())
        else:
            self.downsample = lambda x: x

        self.stride = stride

    def call(self, inputs, training=None):
        residual = self.downsample(inputs)

        conv1 = self.conv1(inputs)
        bn1 = self.bn1(conv1)
        relu1 = self.relu(bn1)
        conv2 = self.conv2(relu1)
        bn2 = self.bn2(conv2)

        add = layers.add([bn2, residual])
        out = self.relu(add)
        return out
```

x' = x　　使维度可相加.

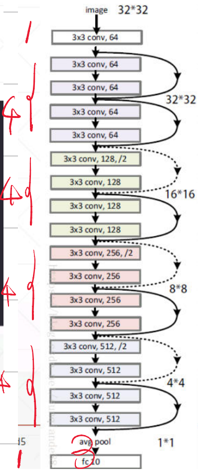
bn2+res ⇒ out —relu→

Res Block

串联 resblock.

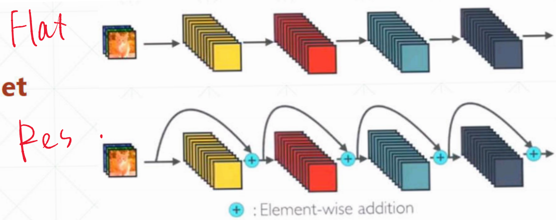



```python
def _build_resblock(self, block, filter_num, blocks, stride=1):

    res_blocks = keras.Sequential()
    res_blocks.add(block(filter_num, stride))

    for _ in range(1, blocks):
        res_blocks.add(block(filter_num, stride=1))

    return res_blocks
```

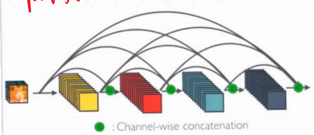


1+4×4+1=18

DenseNet



反馈?



concat　　channel 变大

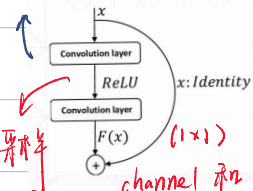
日期: /

class Basic Block

ResNet 实战

① Basic Block

conv + BN + relu.



下采样 (size变小)

channel 和 size 匹配.

结构定义

self.conv1 = layer.Conv2D(

固定变量

stride = stride

self.bn1 = layer.BatchNomalization(

self.relu = layer.Activation('relu')

self.conv2 = layer.Conv2D(stride=1)

self.bn2 = layer.BatchNomalization(

if: stride != 1 (输出维度发生变化,

self.downsample = sequential( )

self.downsample.add(layers.Conv2D((1,1))

传播过程.

def call( ).

out = self.conv1(input)

out = self.bn1(out)

out = self.relu(out)

out = self.conv2(input)

out = self.bn2(out)

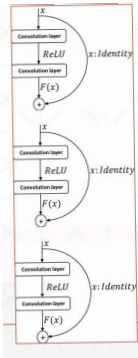
(short cut)

identity = self.downsample(inputs)

output = layers.add([out, identity]) 两层相加

output = self.relu(output) relu层没有参数 当作函数

return output



Res Block (多个 basic block)

class Res Block

def build_resblock ( ).

res_block = Sequential ( )

res_blocks.add (BasicBlock (N, stride))

for _ in range (1, blocks) = 下采样

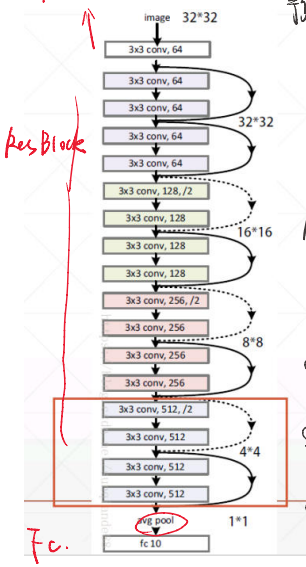
res_blocks.add (BasicBlock (stride=1)

return res_block.

ResNet, 由多个 Res Block 组成 由4个 Res Block

结构: def _init_ (self, layer_dim): [2, 2, 2, 2]

预处理



预处理: self.stem = Sequential ([layer.Conv2D(64, (3,3), Stride=(1,1), layers.BatchNormalization, layers.Activation('relu'), layers.MaxPool2D(...))

ResBlock

fc.

创建四个 Res Block. 0号元素

self.layer1 = self.build_resblock (64, layer_dim[0])

self.layer2 = self.build_resblock (128, layer_dim[1], stride=2) 降维.

self.layer3 = self.build_resblock (256, layer_dim[2], stride=2)

self.layer4 = self.build_resblock (512, layer_dim[3], stride=2)

日期: / output = [b, 512, h, w]

自适应 512 vector [512 × 1 × 1]

降维: self.avgpool = layers.GlobalAveragePooling2D().

Self.fc = layers.Dense(num_classes).

输出分类个数 100

前向传播. def call

x = self.stem(input)

x = self.layer1(x) x = self.layer2(x)

x = self.layer3(x) x = self.layer4(x)

x = self.avgpool(x) [b, c]

x = self.fc(x) [b, 100]

return x.

定义 ResNet 18 resnet 34.

[3, 4, 6, 3]

def ResNet18():

return resnet([2, 2, 2, 2])

日期: /

ResNet_train

from resnet import resnet18 导入 resnet

preprocess 预处理，归一化.

↓ [b, 32, 32, 3] → [b, 1, 1, 512]

main model = resnet18() (参数)

model.build(input_shape=[None, 32, 32, 3])

weight_decate. model.summary() (参数量)

optimizer = optimizer.Adam(lr=1e-3)

for epoch in range(...):

for step, ... in enumerate(train_db):

logit → loss → grad → optimize.

- 1. decrease batch size
- 2. tune resnet [2, 2, 2, 2]
- 3. try Google CoLab → K80 11GB 显存.
- 4. buy new NVIDIA GPU Card

参数)

convolution

日期: /

预训练网络（迁移学习）

保存好的在大型数据集上训练好的卷积神经网络。

可移植的。

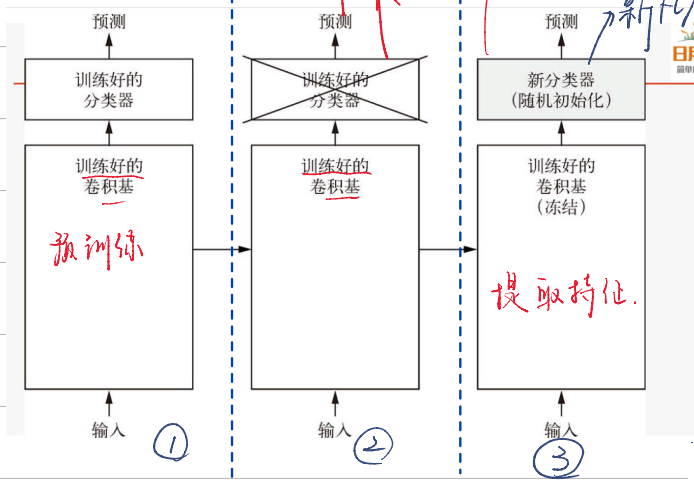
Keras库中包含
VGG16、VGG19、
ResNet50、
Inception v3、
Xception等经典的模型架构。

直接用于 Keras。

在 Image Net 上
进行训练。

网络架构：

3步。



去掉 FC 层

FC，通过提取的 特征 进行 映射

新 FC 层

预训练

提取特征

卷积：提取特征

全连接：综合（组合所有特征）进行分类，映射输出

日期: /

预训练微调

卷积基 { 底层 → 细节、特征
卷积基 { 顶层 → 偏向分类任务

步骤
① 在预训练卷积基上添加自定义层 (fc)
② 冻结卷积基所有层 (conv)
③ 训练分类层 (fc)
④ 解冻顶部卷积基 (conv)
⑤ 联合训练解冻层和分类层 (fc+conv)

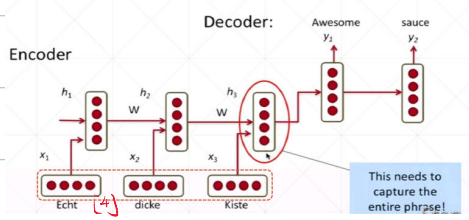
日期: /

位置 & 时间 (语言、文字)

时间序列表示方法

Sequence 序列

文本 → 数据



Sequence embedding 嵌入

采样

[b, 100, 1]

句, 10个词, 4个特征.

- [b, seq_len, feature_len]

[b, 10, 4]

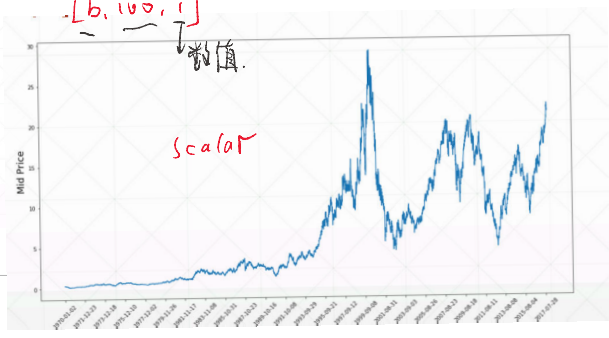
- e.g. I like it.

[000] 张量.

```
<tf.Tensor: id=3, shape=(3, 3), dtype=float32, numpy=
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 1.]], dtype=float32)>
```

数值.

scalar



时间戳 (采样) √

[b, 28, 28] 像素 ×

↑ feature 扫描. √

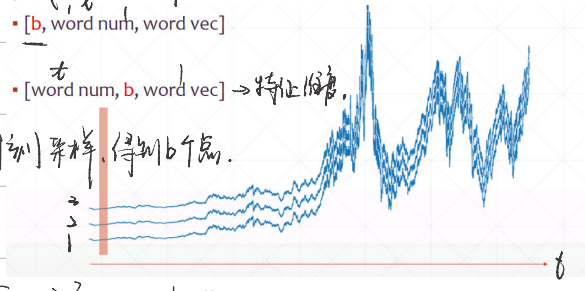
b个曲线, t, 1

- [b, word num, word vec]

t, b, 1

- [word num, b, word vec] → 特征维度.

自-时刻采样, 得到b个点.



主要维度: 时间.

[words, word vec]

日期:

- How to represent a word
  - [Rome, Italy, ...]
- one-hot

Rome, Paris, word V

```
Rome   = [1, 0, 0, 0, 0, 0, ..., 0]
Paris  = [0, 1, 0, 0, 0, 0, ..., 0]
Italy  = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]
```

- sparse
- high-dim
- semantic similarity
- trainable

"语意相关性" similarity

model.most_similar('king', topn=10) (word, similarity with 'king')	model.most_similar('queen', topn=10) (word, similarity with 'queen')
('kings', 0.897245)	('cersei', 0.942618)
('baratheon', 0.809675)	('joffrey', 0.933756)
('son', 0.763614)	('margaery', 0.931099)
('robert', 0.708522)	('sister', 0.928902)
('lords', 0.698684)	('prince', 0.927364)
('joffrey', 0.696455)	('uncle', 0.922507)
('prince', 0.695699)	('varys', 0.918421)
('brother', 0.685239)	('ned', 0.917492)
('aerys', 0.684527)	('melisandre', 0.915403)
('stannis', 0.682932)	('robb', 0.915272)

index	Embedding
0	[1.2, 3.1]
1	[0.1, 4.2]
2	[1.0, 3.1]
3	[0.3, 2.1]
4	[2.2, 1.4]
5	[0.7, 1.7]
6	[4.1, 2.0]

现成的语意表达方法

- Word2Vec vs GloVe

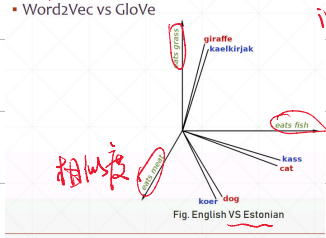


Fig. English VS Estonian

- Random initialized embedding

```
In [8]: from tensorflow.keras import layers
In [3]: x=tf.range(5)
In [5]: x=tf.random.shuffle(x)
Out[6]: <tf.Tensor: id=9, shape=(5,), dtype=int32, numpy=array([0, 4, 1, 3, 2])>

In [9]: net=layers.Embedding(10, 4)
In [10]: net(x)
<tf.Tensor: id=25, shape=(5, 4), dtype=float32, numpy=
array([[-0.03535435,  0.01710499,  0.024379  , -0.010867  ],
       [-0.03977622,  0.01753286,  0.00805125,  0.00836002],
       [-0.0119989 ,  0.01030685, -0.01133521,  0.02052242],
       [ 0.02230989, -0.02186236,  0.01720804, -0.03888531],
       [-0.04997355,  0.00911248,  0.01886252,  0.01570504]],
      dtype=float32)>
```

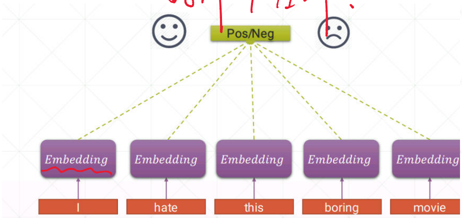
```
In [11]: net.trainable
Out[11]: True

In [12]: net.trainable_variables
[<tf.Variable 'embedding/embeddings:0' shape=(10, 4) dtype=float32, numpy=
 array([[-0.03535435,  0.01710499,  0.024379  , -0.010867  ],
        [-0.0119989 ,  0.01030685, -0.01133521,  0.02052242],
        [-0.04997355,  0.00911248,  0.01886252,  0.01570504],
        [ 0.02230989, -0.02186236,  0.01720804, -0.03888531],
        [-0.03977622,  0.01753286,  0.00805125,  0.00836002],
        [-0.03328228,  0.04350821, -0.03760867, -0.02835478],
        [ 0.03658437, -0.00686272, -0.00069226,  0.02232203],
        [-0.02224611, -0.00517293, -0.00249453,  0.00809779],
        [ 0.00234641, -0.01876179, -0.03804705, -0.02393213],
        [-0.01690916, -0.04798424, -0.04477951, -0.03771662]],
       dtype=float32)>]
```

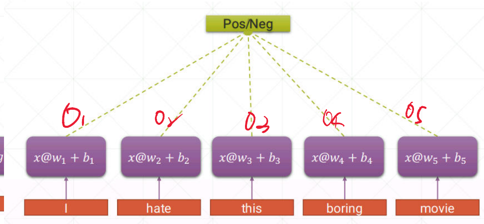
日期:　　/

循环神经网络 (RNN)

多少个词，多少个全连接层

好评 / 差评.





① ▪ Long sentence 的
- 100+ words
- too much parameters $[w_N, b_N]$ 参数量巨大

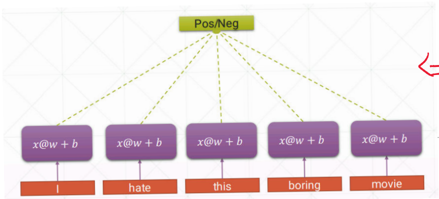
→ weight sharing. 采用同一个线性层

没有 背景/语意 相关性. "顺序"

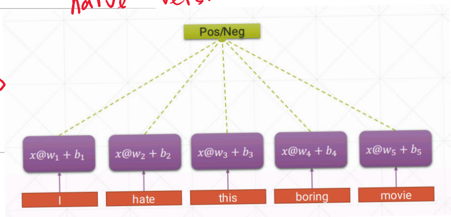
② ▪ No context information
- consistent tensor

→ 单独理解.

naive version



⟺

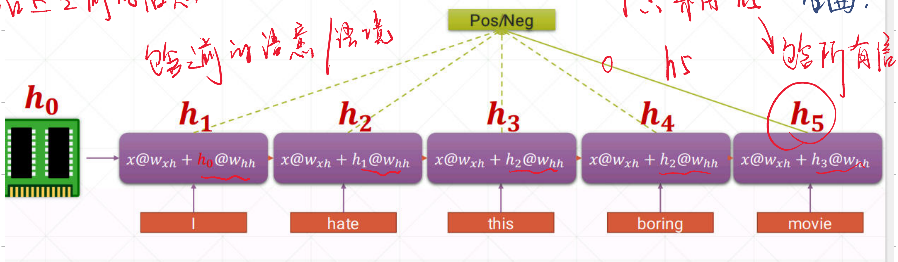


Consistent memory. (概括信息)

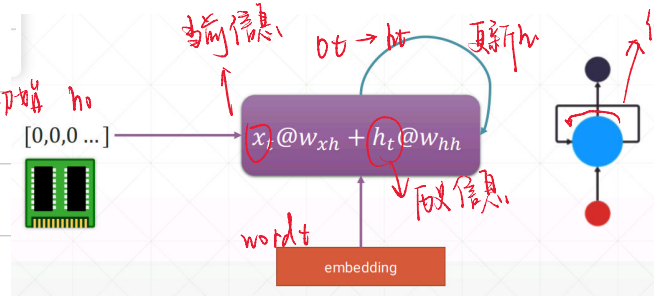
加上之前的信息

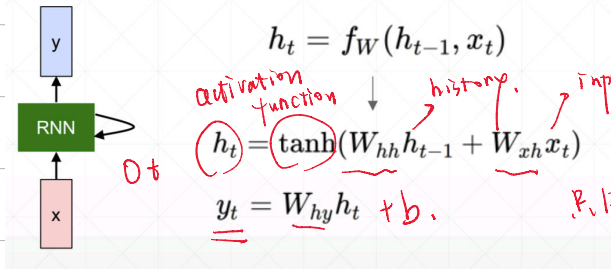
包含前面的语意/语境

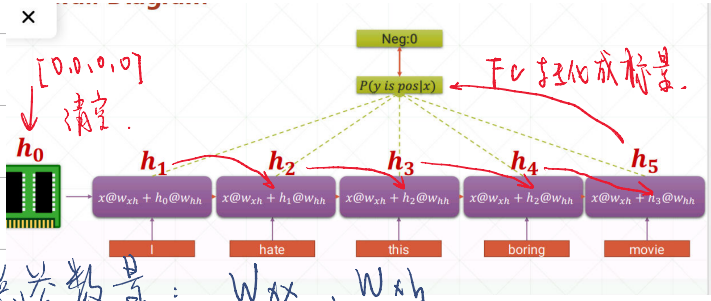
) 所有 h 相加.
| 只采用 ht 全面.
→ 包含所有信息.

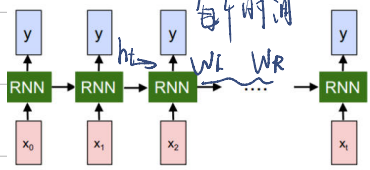


日期:









h_0, w_{ih}, w_{hh}

$$h_t = tanh(W_I x_t + W_R h_{t-1})$$
$$y_t = W_O h_t + b$$

$$\frac{\partial E_t}{\partial W_R} = \sum_{i=0}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_i}\frac{\partial h_i}{\partial W_R}$$

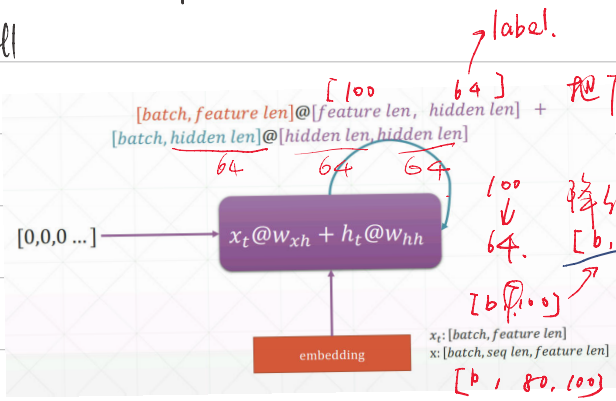
$$\frac{\partial h_t}{\partial h_i} = \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_{i+1}}{\partial h_i} = \prod_{k=i}^{t-1}\frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial h_{k+1}}{\partial h_k} = diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

$$\frac{\partial h_k}{\partial h_1} = \prod_{i}^{k} diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

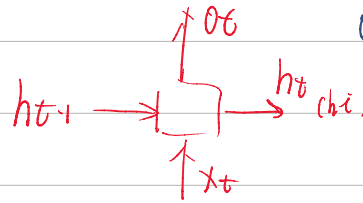
日期: / RNN layer 使用.

RNN cell



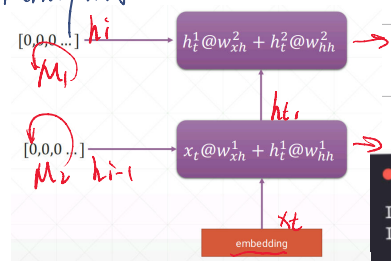

```
In [17]: cell=layers.SimpleRNNCell(3)
In [18]: cell.build(input_shape=(None,4))

In [19]: cell.trainable_variables
[<tf.Variable 'kernel:0' shape=(4, 3) dtype=float32, numpy=
 array([[ 0.23682523,  0.24167228,  0.19834113],
        [ 0.58464265, -0.44347632, -0.23693317],
        [ 0.5130104 , -0.86219984,  0.16108215],
        [-0.37421566, -0.6311711 ,  0.46914995]], dtype=float32)>,
 <tf.Variable 'recurrent_kernel:0' shape=(3, 3) dtype=float32, numpy=
 array([[ 0.7126031 ,  0.39248434,  0.5815092 ],
        [-0.34029973,  0.9182056 , -0.2027184 ],
        [ 0.61350864,  0.05342963, -0.7878784 ]], dtype=float32)>,
 <tf.Variable 'bias:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.], dtype=float32)>]
```

- out, h_1 = call(x, h_0)
  - x: [b, seq len, word vec]
  - h_0/h_1: [b, h dim]
  - out: [b, h dim]



```
In [4]: x=tf.random.normal([4,80,100])
In [5]: xt0=x[:,0,:]

In [6]: cell=tf.keras.layers.SimpleRNNCell(64)

In [9]: out,xt1=cell(xt0, [tf.zeros([4,64])])

In [12]: out.shape,xt1[0].shape
Out[12]: (TensorShape([4, 64]), TensorShape([4, 64]))

In [13]: id(out), id(xt1[0])
Out[13]: (1724126403608, 1724126403608)
```

```
In [6]:  cell=tf.keras.layers.SimpleRNNCell(64)

In [17]: cell.trainable_variables
[<tf.Variable 'simple_rnn_cell/kernel:0' shape=(100, 64) dtype=float32, numpy=
 array([[-0.14681616, -0.18293515,  0.01196897, ..., -0.15498586,
         -0.09611963,  0.1502908 ],
        [ 0.05362315, -0.05283108,  0.1457149 , ..., -0.11584248,
        [-0.10094493, -0.10836099,  0.18547966, ...,  0.15367018,
         -0.05795772, -0.11807813]], dtype=float32)>,

 <tf.Variable 'simple_rnn_cell/recurrent_kernel:0' shape=(64, 64) dtype=float32, numpy=
 array([[ 1.01237297e-01,  8.35651010e-02, -1.47625804e-04, ...,
         -8.24909434e-02, -1.28735945e-01,  4.13244553e-02],
        [-1.57783236e-02,  6.91108108e-02,  5.74547313e-02, ...,
         -1.16356298e-01, -2.58456618e-02, -1.97714210e-01]], dtype=float32)>,

 <tf.Variable 'simple_rnn_cell/bias:0' shape=(64,) dtype=float32, numpy=
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)>]
```




```
In [4]: x=tf.random.normal([4,80,100])
In [5]: xt0=x[:,0,:]

In [6]:  cell=tf.keras.layers.SimpleRNNCell(64)
In [14]: cell2=tf.keras.layers.SimpleRNNCell(64)
In [6]: state0 = [tf.zeros([4,64])]
In [6]: state1 = [tf.zeros([4,64])]

In [9]: out0, state0=cell(xt0, state0)
In [15]: out2, state2=cell2(out, state0)

In [16]: out2.shape, state2[0].shape
Out[16]: (TensorShape([4, 64]), TensorShape([4, 64]))
```

日期:　　　/

Out [b, units

hidden_len.

```
state0 = [tf.zeros([batchsz, units])]
state1 = [tf.zeros([batchsz, units])]
for word in tf.unstack(x, axis=1): # word: [b, 100]
    # h1 = x*wxh+h0*whh
    # out0: [b, 64]
    out0, state0 = self.rnn_cell0(word, state0, training)
    # out1: [b, 64]
    out1, state1 = self.rnn_cell1(out0, state1, training)
```

→ Memory. 更新 x_t h_{t-1} 下个cell

设定网络数量.

```
self.rnn = keras.Sequential([
    layers.SimpleRNN(units, dropout=0.5, return_sequences=True, unroll=True),
    layers.SimpleRNN(units, dropout=0.5, unroll=True)
])

  # x: [b, 80, 100] => [b, 64]
x = self.rnn(x)
```

返回状态

t=80 时的输出 $h^1_{t=80}$

- SimpleRNNCell
  - single layer
  - multi-layers
- RNNCell

情感分类实战

定义: unit=64, eporch=4

加载 IMDB 电影评价数据集.

构建网络　class MyRNN (keras.Model)

def __int__(self),　[b, 80] ⇒ [b, 80, 100] 拓展

self.embedding = layers.embedding(total_words,

embedding_len, input_length=max_review_len.

100　80

日期: /

self.state0 = [tf.zeros([b, units])] → 初始化状态 → [b, 64]

[b, 80, 100] 表示 h_dim = 64

定义 cell

self.rnn_cell0 = layers.SimpleRNNCell(units, dropout=0.5)

self.rnn_cell1 = layers.SimpleRNNCell(units, dropout=0.5)

防止过拟合

每一时刻 → fc.

fc: [b, 80, 100] → [b, 64] → [b, 1]

self.outlayer = layer.Dense(1)

def call(self, inputs, training=None):

net(x) , net(x, training=True)

input [b, 80]

x = input [b, 80]

x = self.embedding(x). [b, 80, 100]

embedding

input

state

out

for word in tf.unstack(x, axis=1): (每个word)

t 维度展开

[b, 64] 在时间上展开

初始化状态 state.

out0, state1 = self.rnn_cell0(word, state0) out0 = [b, 64]

state0 = state1 out1 = [b, 64]

out1, state1 = self.rnn_cell1(word, state1)

日期: /

out
↓ Fc
x
↓
prob
↓
loss → optimize
↓
gradient

x = self.out_layer(out1)

prob = tf.sigmoid(x)

main:

unit = 64 (hidden_dim) epoch = 4

model = MyRNN(units) → 初始类(模型)

model.compile(optimizer = ___

定义参数: train, test loss = ___

epoch, loss, metrics = ____)

model.fit(db_train, epoch = epochs, validation_data=

model.evaluate(db_train)

日期: /

梯度弥散与梯度爆炸.

元祖声pb? 元祖屠开!

Nothing is straight forword.

$$h_t = tanh(W_I x_t + W_R h_{t-1})$$
$$y_t = W_O h_t$$

$$\frac{\partial E_t}{\partial W_R} = \sum_{i=0}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_i}\frac{\partial h_i}{\partial W_R}$$

ht → hi

$$\frac{\partial h_t}{\partial h_i} = \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_{i+1}}{\partial h_i} = \prod_{k=i}^{t-1}\frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial h_{k+1}}{\partial h_k} = diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

$W_R^k \cdot N \rightarrow$ gradient

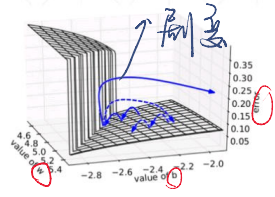
$$\frac{\partial h_k}{\partial h_1} = \prod_{i}^{k} diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

W_h^k

当 W_h 略大于 1 时很大 $W_h^k \rightarrow \infty$

略小于 1 时很大 $W_h^k \rightarrow 0$

查看 W. gradient, 对某一阈值时. clipping. (对 gradient)



Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

$\hat{g} = \frac{g}{\|g\|} \cdot threshold$ 方向不变

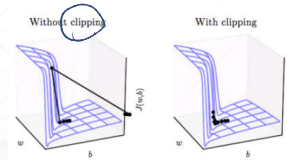


Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depicts standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a fixed size when its norm is above a threshold.

— Goodfellow et al., *Deep Learning*

http://proceedings.mlr.press/v28/pascanu13.pdf

日期:

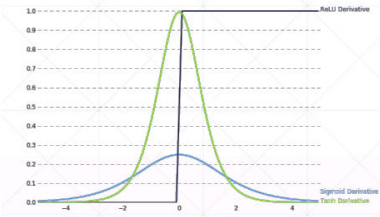
```python
with tf.GradientTape() as tape:
    logits = model(x)
    loss = criteon(y, logits)

grads = tape.gradient(loss, model.trainable_variables)
# MUST clip gradient here or it will disconverge!

optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

grads = [tf.clip_by_norm(g, 15) for g in grads]

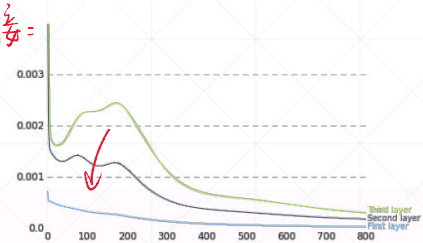
clipping　　threshold 经验值.

反向传播，累计误差 $\delta_k \to 0$ 梯度 w_j

Gradient Vanishing



梯度值:

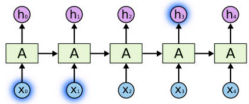


日期: /

LSTM. 解决 gradient vanishing 长时间记忆

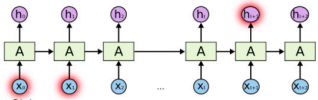
The problem of long-term dependencies

(Vanilla) RNNs connect previous information to present task:
- enough for predicting the next word for "the clouds are in the *sky*"



short-term-memory.

- may not be enough when more context is needed
"I grew up in France... I speak fluent *French*."



Long-term

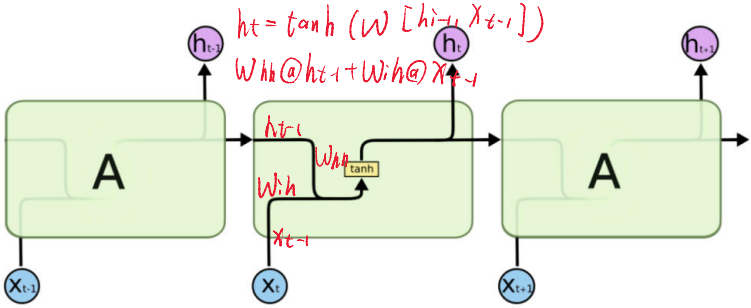
Adapted from Christopher Olah

RNN:

All recurrent neural networks have the form of a chain of repeating modules of neural network

$h_t = \tanh(W[h_{t-1}, x_{t-1}])$

$W_{hh} h_{t-1} + W_{ih} x_{t-1}$



The repeating module in a standard RNN contains a single layer.

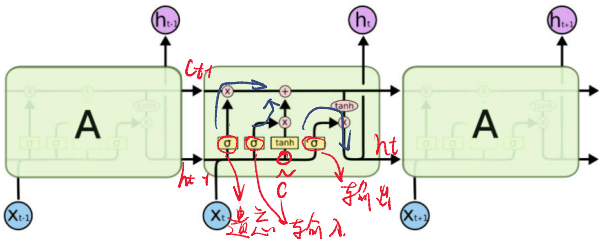
LSTM =

日期: /

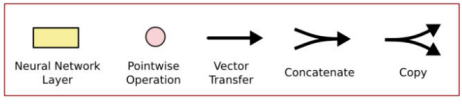
有目的的过滤 "门" : σ (sigmoid)

(遗忘)

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

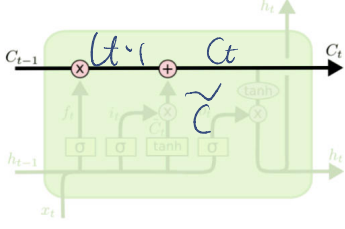


σ (w [h, x])

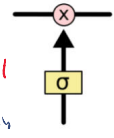
0 ~ 1

h → h'

The Core Idea Behind LSTMs : Cell State



Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



or Ct-1

σ: 0 ~ 1

门 控

An LSTM has three of these gates, to protect and control the cell state.

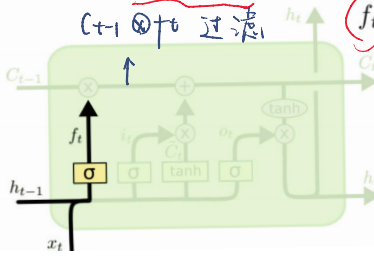
日期: /

$f_t=1$ 全部记住 $f_t=0$ 忘记

remember

forget

LSTM : Forget gate



C_{t-1} ⊗ f_t 过滤

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

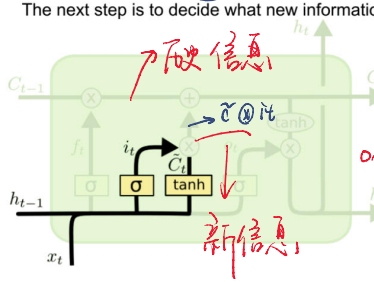
A 1 represents "completely keep this" while a 0 represents "completely get rid of this".

W_C, W_i, W_f 自动更新生成

b_C, b_i, b_f

LSTM : Input gate and Cell State

The next step is to decide what new information we're going to store in the cell state.



确定信息

$\tilde{C}_t$ ⊗ i_t

新信息

a sigmoid layer called the "**input gate layer**" decides which values we'll update.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

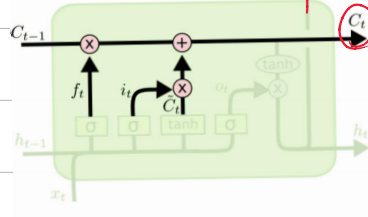
only

a tanh layer creates a vector of new candidate values, that could be added to the state.

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM : Input gate and Cell State

It's now time to update the old cell state into the new cell state



新输出

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

旧 新信息

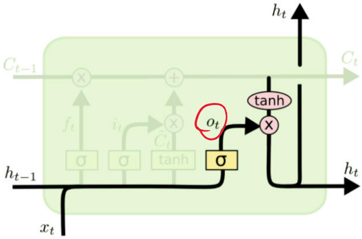
We multiply the old state by ft forgetting the things we decided to forget earlier.

Then, we add the new candidate values, scaled by how much we decided to update each state value.

日期: /

LSTM : Output

Finally, we need to decide what we're going to output.



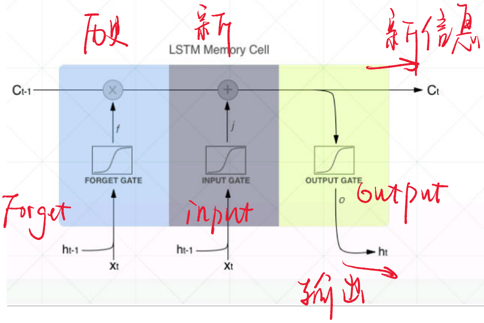
First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

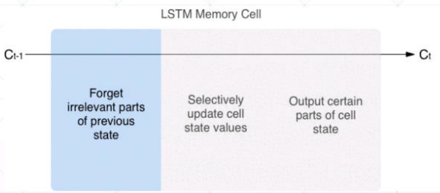
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

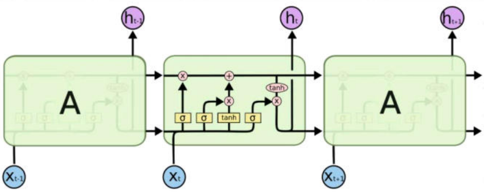
$$h_t = o_t * \tanh\left(C_t\right)$$ → 新输出

Ct → ht (另一个输出) 处理后的输入.





输入



☆

$$\begin{pmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{o}^{(t)} \\ \tilde{C} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \mathbf{W} \begin{pmatrix} \mathbf{x}^{(t)} \\ \mathbf{h}^{(t-1)} \end{pmatrix} \quad (6)$$

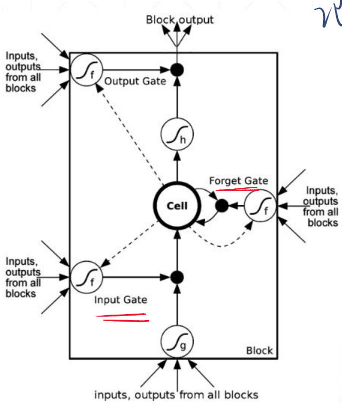
过去+现在.

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tilde{C} \quad (7)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh(\mathbf{c}^{(t)}). \quad (8)$$

out ⊙ tanh(ct)

记忆 + 输入·



真值表:

input gate	forget gate	behavior
0	1	remember the previous value
1	1	add to the previous value
0	0	erase the value
1	0	overwrite the value

过去 · 新信息 · 添加到记忆 · 忽略当前信息 · 读写 · 删除之前信息

How to solve Gradient Vanishing?

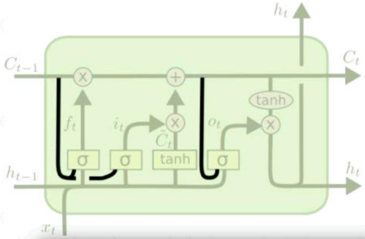
$$\frac{\partial h^k}{\partial h_i} = N \cdot W_{hh}^k$$

将 $h \to C$

$$\frac{\partial C_t}{\partial C_{t-1}} = \frac{\partial C_t}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial i_t}\frac{\partial i_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial \tilde{C}_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial C_{t-1}}$$

累加!!!

$$\frac{\partial C_t}{\partial C_{t-1}} = C_{t-1}\sigma'(\cdot)W_f * o_{t-1}tanh'(C_{t-1}) + \tilde{C}_t\sigma'(\cdot)W_i * o_{t-1}tanh'(C_{t-1}) + i_t \tanh'(\cdot)W_C * o_{t-1}tanh'(C_{t-1}) + f_t$$



$$\frac{\partial h_{k+1}}{\partial h_k} = diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

$$\frac{\partial h_k}{\partial h_1} = \prod_i^k diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

不可能出现过小过大。

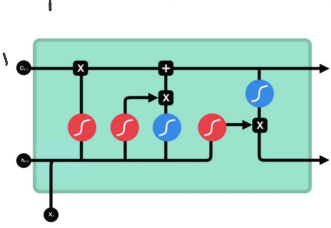
https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html

日期: /

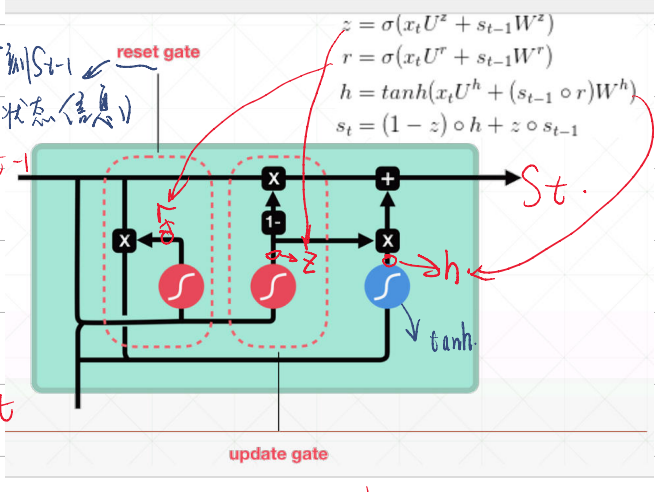
LSTM 实战

GRU: simpler, lower computation cost

LSTM:



GRU:



控制上一时刻S_{t-1}的阈值(状态信息)

S_{t-1}

S_t

h_t

dropout标志

RNN → GRU

unroll = False.

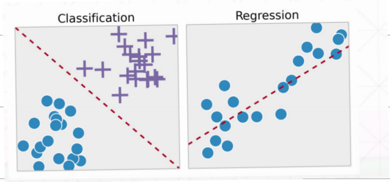
self.rnn_cell0 = layer.GRUCell(units, dropout=0.5)

状态useful一个 self.state0=[tf.zeros([batchsz, units])]

layers.LSTM layers.GRU.

日期: 无监督学习.

Auto-Encoders.



"Label" → 监督 通过人的先验知识 进行labeling.
评级
标定

强化学习.

Massive Unlabel Data.

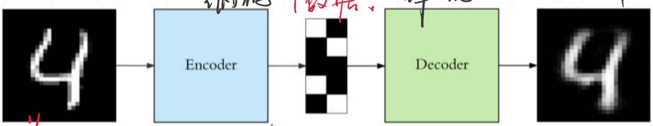
Unsupervised Learning.

- Dimension reduction 降维. → 64x64 → 20x10
- Preprocessing: Huge dimension, say 224x224, is hard to process
- 可视化. Visualization: https://projector.tensorflow.org/ 特征提取
- Taking advantages of unsupervised data
- 压缩 去噪 超分辨率.
  Compression, denoising, super-resolution ...

* 特殊的连接层
① 输入输出维度一致
② 有一个code.

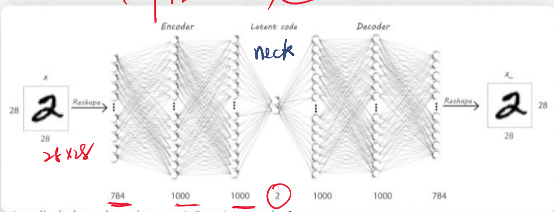
Auto-Encoder

编码 (数据. 解码.



"reconstruct" Input (神经网络) Code Output

降维, 意义相当.



neck
28x28
→ 2维.

日期:

- Loss function for binary inputs

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\hat{x}_k) + (1-x_k)\log(1-\hat{x}_k)\right)$$

  - Cross-entropy error function (reconstruction loss) $f(\mathbf{x}) \equiv \hat{\mathbf{x}}$

- Loss function for real-valued inputs

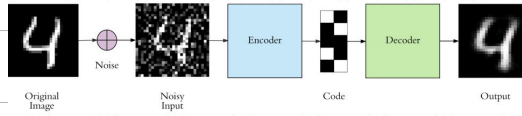
$$l(f(\mathbf{x})) = \frac{1}{2}\sum_k (\hat{x}_k - x_k)^2$$

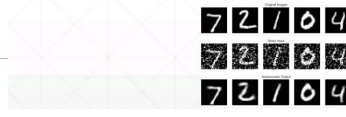
  - sum of squared differences (reconstruction loss)
  - we use a linear activation function at the output

PCA vs. Auto-Encoder

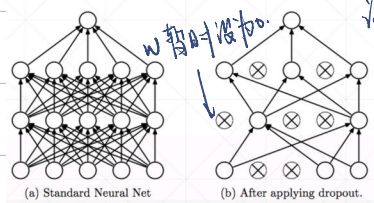
- PCA, which finds the directions of maximal variance in high-dimensional data, select only those axes that have the largest variance.
- The linearity of PCA, however, places significant limitations on the kinds of feature dimensions that can be extracted.

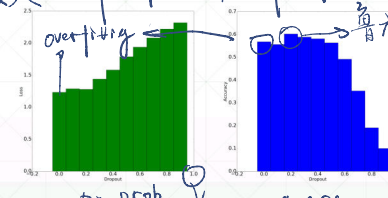
Denoising Auto Encoders.





Dropout Auto Encoders.



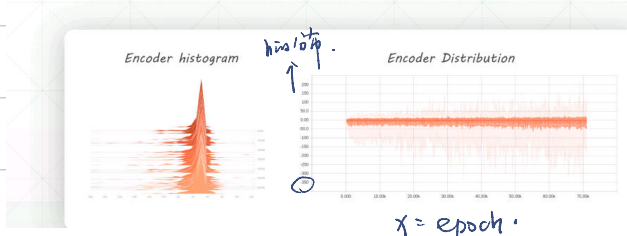


日期: /

Adversarial Auto Encoders.

- Distribution of hidden code

$x \rightarrow$ hidden vector $\rightarrow z$

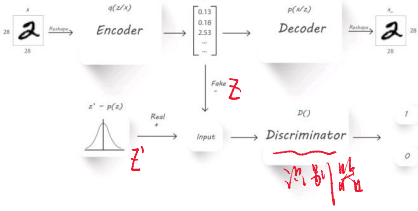


hidden 分布.

x = epoch.

- Give more details after GAN

希望 z 属于预设的分布.



目的一致

仅 P,Q 分布相近

数据分布.

过程: $x \rightarrow z \rightarrow \bar{x}$

Another Approach: $q(z) \rightarrow p(z)$

- Explicitly enforce

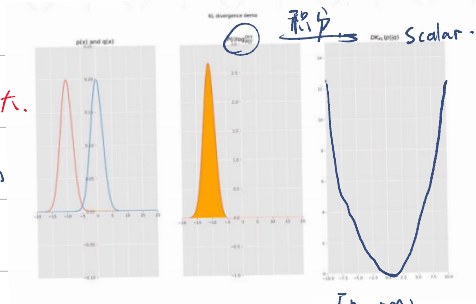
$\log P_\theta(x_i|z)$ 最大.

(AE) 重建误差 + 分布约束. 真实, 逼真.

$$l_i(\theta,\phi) = -E_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z))$$

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x)\log\frac{p(x)}{q(x)}\,dx$$

使通用图与原图数相似.



KL 散度.

两个分布. P, Q.

当 P,Q 分布相近时

KL(P||Q) 变小

[0, +∞)

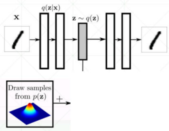
完全重叠时 KL(P||Q) = 0

Minimize KL Divergence

- Evidence Lower BOund

$$KL(q_\theta(z|x_i)||p(z))$$

重建 + 逼近.



高斯分布.

假设.

$p(z_i) \sim N(\mu_1, \sigma_1^2)$ 分布.

$q(z_i) \sim N(\mu_2, \sigma_2^2)$

实际

KL(p||q) 和 KL(q||p) 是不对称的.

$$KL(p,q) = -\int p(x)\log q(x)dx + \int p(x)\log p(x)dx$$

$$= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log 2\pi\sigma_1^2)$$

$$= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

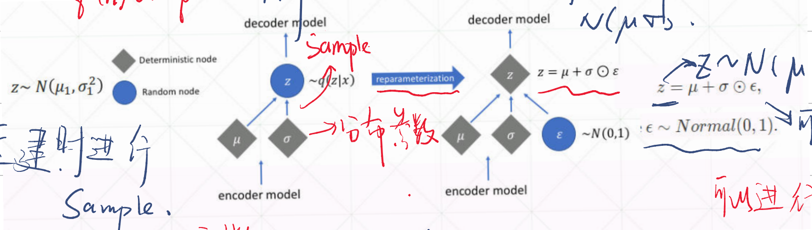
方差之间　　方差和均值

(μ, σ^2) distribution

$x \rightarrow h \rightarrow \hat{x}$

q(h) simple → 采样

假设 h 属于 q 分布. $N(\mu, \sigma^2)$.



sample

分布参数

$z \sim N(\mu, \sigma^2)$.

$z = \mu + \sigma \odot \epsilon$,

$\epsilon \sim Normal(0, 1)$. 可导!

重建时进行 Sample.

可以进行

不可微.

backprop

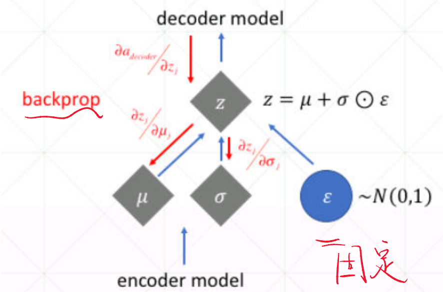
再参数化

Reparameterization trick

将 $z \sim N(\mu, \sigma^2)$

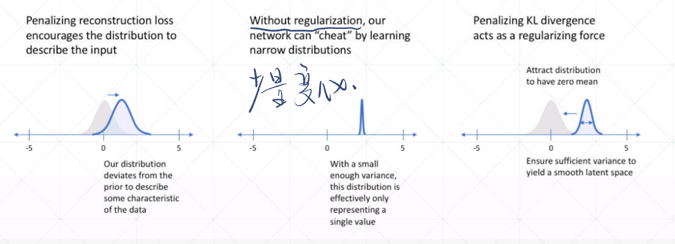
表达为 $z = \mu + \sigma \cdot \varepsilon$.

$\varepsilon \sim N(0,1)$



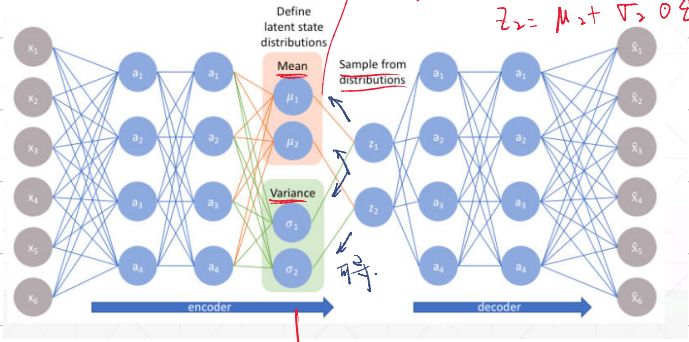
固定

RL + β × KL(q||p)



方差变小.

日期: /

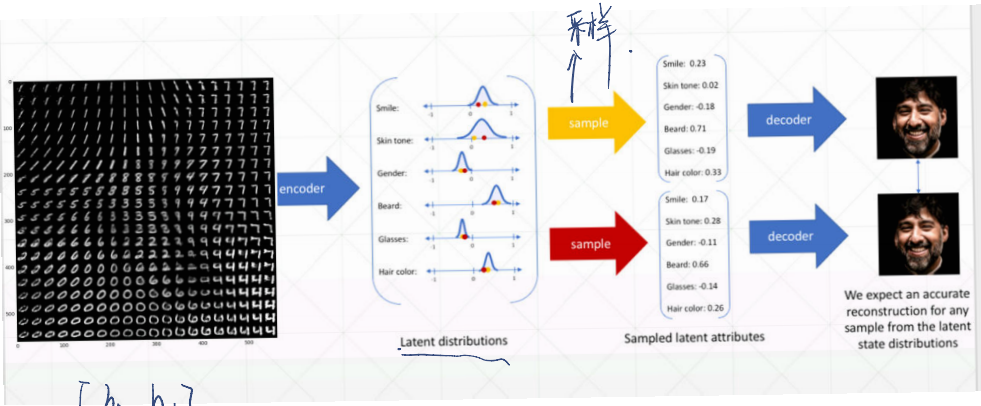


Sample. $z_1 = \mu_1 + \sigma_1 \odot \varepsilon$. $z_2 = \mu_2 + \sigma_2 \odot \varepsilon$.

采样.

VAE的问题: Sample. 直接方式的问

variational-autoencoders — VAE



采样.

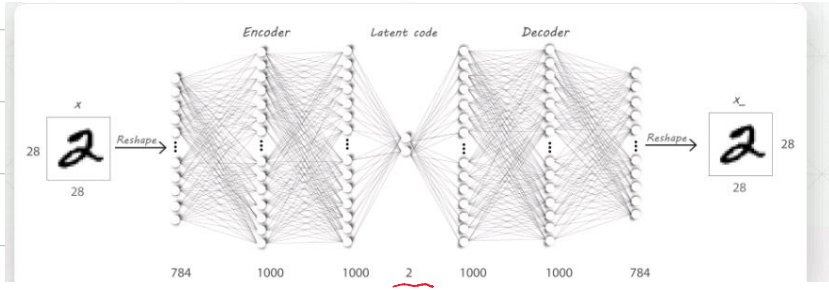
$[h_0, h_1]$ → $N(\mu_1, \sigma_1^2)$.

↓

$N(\mu_0, \sigma_0^2)$

日期: /

VAE/AE 实战



- 定义参数: h_dim (784→n) batchsz. lr.
- 划分数据集/测试集 对图 y (label)
- 新建类 (模型) class AE(keras.Model) 继承.

初始化 =

#Encoder: self.encoder = Sequential([

* 784→h_dim

784→256 layers.Dense(256, acti = tf.nn.relu)

256→128 layers.Dense(128, acti = tf.nn.relu)

128→h_dim layers.Dense(h_dim)])

* h_dim→784

#Decoder: self.decoder = Sequential([

h_dim → 128 layers.Dense(128, acti = tf.nn.relu)

128 → 256 layers.Dense(256, acti = tf.nn.relu)

256 → 784 layers.Dense(784)])

日期: /

前向传播. def call (self, inputs, traing=None):

(hidden)

[b, 784] → [b, 10] h = self.encoder(inputs)

[b, 10] → [b, 784] x_hat = self.encoder(h)

return. x_hat

参数:

model. AE( ).

model. build (input_shap=(None, 784))

model. summary.

optimizers. Adam(lr=lr)

for epoch in range ( ):

for step, x in enumerate (train_db).

x = tf. reshape

with ... as tape:

x_rec_logits = model (x)

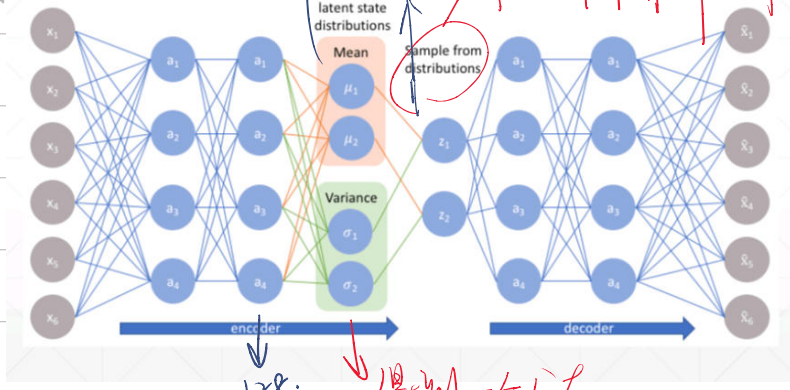
rec_loss = tf. loss. binary_cross ...

#求均值. rec_loss = tf. reduce_mean (rec_loss)

#计算梯度 ads = tape. gradient (loss, variable)

#更新 optimizer. apply_gradients ( )

VAE实战



z_dim

在分布中采样，得到特征。

128.

得到一个分布

z_dim = 10　　中间层维度

class VAE (keras.Model):

super(VAE, self).__init__()　　# Encoder:

128
fc1 → fc2 → z　　self.fc1 = layers.Dense(128)

μ: self.fc2 = layers.Dense(z_dim)

σ: self.fc3 = layers.Dense(z_dim)

Decoder: self.fc4 = layers.Dense(128)

[b, z] → [b, 784]　　self.fc5 = layers.Dense(784)

用 sequential ?

日期: /

x → fc1 ⟨ fc2 → mu ; fc3 → var

```
def Encoder (self, x):
mean:   h = tf.nn.relu(self.fc1(x))
        mu = self.fc2(h)
variance  log_var = self.fc3(h)
        return mu, logvar

def Decoder (self, x):
        out = tf.nn.relu(fc4(z))
        out = self.fc(out)
        return out

def reparameterize (self, mu, log_var)
        eps = tf.random.normal(inputs)
        std = tf.exp(log_var) ** 0.5
        z = mu + std * eps.
        return z.
```

ε 正态分布

z = μ + σ ⊙ ε.

日期:　　/

```
def call (self, input, traing = None)
  ① mu, log_var = self.encoder (inputs)
    # reparameterize trick
  ② z = self.reparameterize (mu, log_var)
  ③ x_hat = self.decoder (z)
    return x_hat, mu, log_var
```

↓ 画图.

```
x_hat, mu, log_var = model (x)
```

#AE的 loss

```
rec_loss = tf.loss.binary_cross ...
rec_loss = tf.reduce_mean (rec_loss)
```

简化的 KL 操作: 符合正态分布

```
kl_div = -0.5 * (log_var + 1 - mu**2 - tf.exp(log_var))
kl_div = tf.reduce_sum (kl_dim) / x.shape[0].
loss = rec_loss + 1. * kl_div
```

两部分 重建误差 + kl　(x ** 2 = x^2)

测试: { sample 画图.
　　　　x_hat (reconstruct 画图) → 效果好!

日期: /

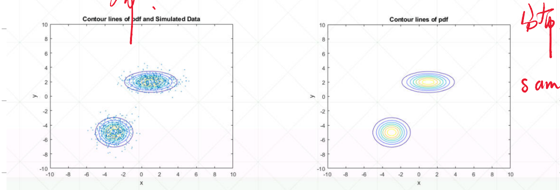
What I cannot create, I do not understand
—Richard Feynman.

对抗生成网络.

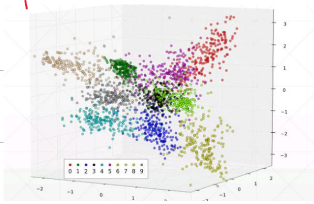
Our Goal: $p(x)$ 复现数据.

分布.

```
13 mu = [1 2;-3 -5]
14 sigma = cat(3,[2 0;0 .5],[1 0;0 1])
15 p = ones(1,2)/2
16 gm = gmdistribution(mu,sigma,p)
```



分布. sample → p(x). look like?



Mnist. 降维可视化.

画家的成长 → 学习优秀作品分布的过程. / 被鉴别/鉴别.

同时成长 → 标准提高 → 优化 → 无法鉴别.
↘ 效果提高 ↗

Nash Equilibrium.
50% 蒙混过关. 50% 被鉴别

收获: 如何画一幅名作 / 成为一名画家.

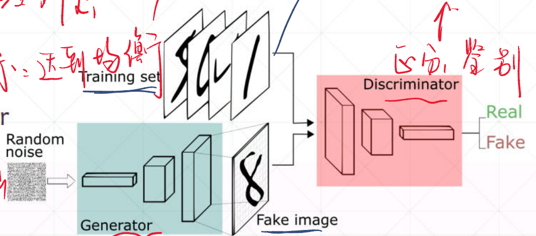
- Painter or Generator:
- Discriminator

画家 ↖

→ 不被鉴别出
目标: 达到均衡 ($P_r = P_g$). r = real

鉴别师 ↗ → 鉴别出

诸对抗



$P_r(x)$ 原作. 鉴别技巧 ↑ 区分, 鉴别. 1 0

创造. ↘ $P_g(x)$ → 模仿技巧.

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

$$V(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x)]$$

对于 $V(D,G)$ 鉴别师 试图使其变大

画家试图减小它. → 鉴别为真.

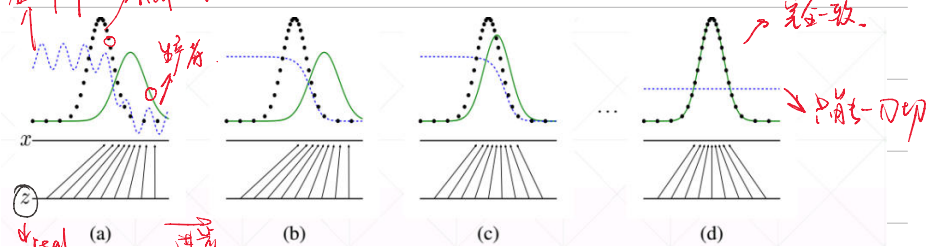
$x \rightarrow D \rightarrow D(x)$ $\quad z \rightarrow G \rightarrow x_g$

- https://reiinakano.github.io/gan-playground/
- https://affinelayer.com/pixsrv/
- https://www.youtube.com/watch?v=9reHvktowLY&feature=youtu.be
- https://github.com/ajbrock/Neural-Photo-Editor
- https://github.com/nashory/gans-awesome-applications

- Q1. Where will D converge, given fixed G
- Q2. Where will G converge, after optimal D

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$
$$= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x)]$$

$P_g(x) = P_r(x)$



日期: /

① Where will D go (fixed G). Distributor · 固定 Generator

* 得到 $D^* = \frac{P_r}{P_r + P_g}$

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \quad (2)$$

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$E D \log D(x)$ $E[\log(1-D(g(z)))]$

$$V(G, D) = \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{\boldsymbol{z}} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz$$

$$= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx \quad (3)$$

$z \to x_g$

$P(z) = P(x_g)$.

$p_{data}(x)$ 真实值. constant = A B

导数

$$f(\tilde{x}) = A\log\tilde{x} + B\log(1 - \tilde{x})$$

$$\frac{df(\tilde{x})}{d\tilde{x}} = A\frac{1}{ln10}\frac{1}{\tilde{x}} - B\frac{1}{ln10}\frac{1}{1-\tilde{x}}$$

$$= \frac{1}{ln10}\left(\frac{A}{\tilde{x}} - \frac{B}{1-\tilde{x}}\right)$$

$$= \frac{1}{ln10}\frac{A - (A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})} = 0$$

$A = (A+B)D(x)$,

$D(x) = \frac{A}{A+B}$.

Thus, set $\frac{df(\tilde{x})}{d\tilde{x}} = 0$, we get the best value of the discriminator: $D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0, 1]$

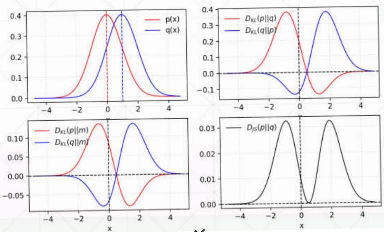
最优时. G 固定 → D 成为 $\frac{P_r}{P_r+P_g}$. = $\frac{原作}{原作 + 模仿}$

实现对称 KL divergence → 差异 散度

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

JS = KL(一半 + 一半)

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q\|\frac{p+q}{2})$$



→ 定义 (KL)

② Where will G go (After D^*).

↓ 固定训练好的 D^*

$$D_{JS}(p_r\|p_g) = \frac{1}{2}D_{KL}(p_r\|\frac{p_r+p_g}{2}) + \frac{1}{2}D_{KL}(p_g\|\frac{p_r+p_g}{2})$$

展开

$$= \frac{1}{2}\left(\log 2 + \int_x p_r(x) \log \frac{p_r(x)}{p_r + p_g(x)} dx\right) +$$

$$\frac{1}{2}\left(\log 2 + \int_x p_g(x) \log \frac{p_g(x)}{p_r + p_g(x)} dx\right)$$

$$= \frac{1}{2}\left(\log 4 + L(G, D^*)\right)$$

展开，

$D^* = \frac{P_g}{P_r + P_g}$.

$D_{JS}(p_r \| p_g) \geq 0$

$p_r = p_g$

min max $V(G, D^*)$
↓ D ↓ G = 固定

$$L(G, D^*) = 2D_{JS}(p_r\|p_g) - 2\log 2$$

= log4

∴ 固定 D^*. min $V(G, D^*)$ = L = 2 log 2.

前提: $P_r = P_g$

G, D —固G→ D^* —固D^*→ L (min) = 2 log 2.

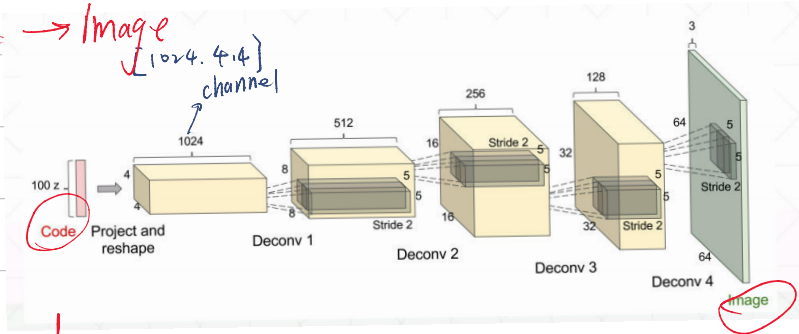
↓ 最优

日期: /

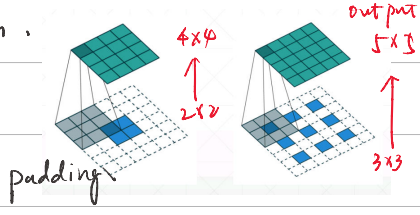
DC GAN (Deconvolution GAN).

code → Image [1024. 4.4]

channel



Transposed Deconvolution.



升维.

维度变大.

* Training Stability

小稳定.

$x \sim P_r$ 和 $x \sim P_g$ 有高维.

GAN开始时, Preal 和 Pgenerate 也可能有 overlap 的.

- In most cases, P_G and P_{data} are not overlapped.
- 1. The nature of data

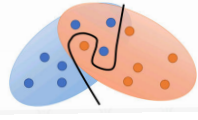
  Both P_{data} and P_G are low-dim manifold in high-dim space.

  The overlap can be ignored.

- 2. Sampling

  Even though P_{data} and P_G have overlap.

  If you do not have enough sampling .....



当 Pr 和 Pg 小重合时.

比/J 无法正常训练.

Toy example

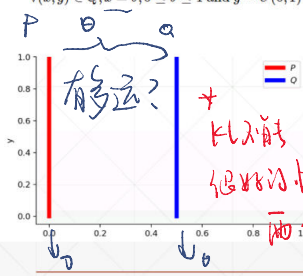
$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2})$$

极端情况：不相交

$\forall (x,y) \in P, x = 0 \text{ and } y \sim U(0,1)$

$\forall (x,y) \in Q, x = \theta, 0 \le \theta \le 1 \text{ and } y \sim U(0,1)$



When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0,y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta,y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P,Q) = \frac{1}{2}\left(\sum_{x=0,y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0,y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

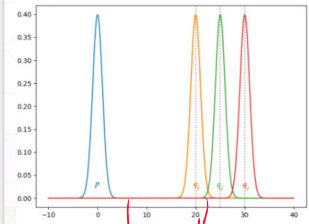
$$W(P,Q) = |\theta|$$

But when $\theta = 0$, two distributions are fully overlapped:

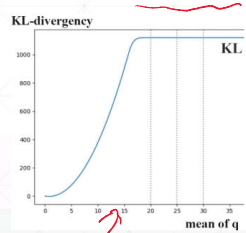
$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P,Q) = 0$$

$$W(P,Q) = 0 = |\theta|$$

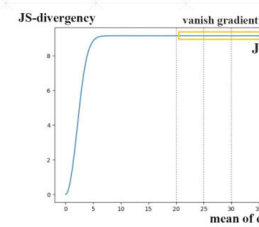
KL有一定的局限性.



MLE is kind of minimize KLD



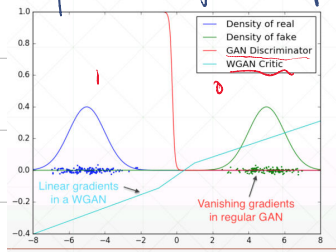
GAN



重合面积都很小，比差不多，都很大.

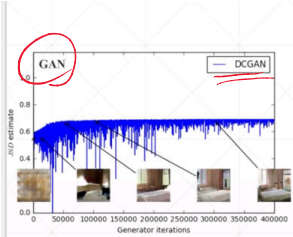
梯度没了!!! gradient vanish!

所以梯时. P_r from P_g 距离远. 找不到训练方向.

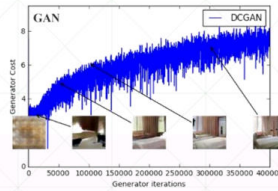


在不相关区域 GAN 梯度为 0

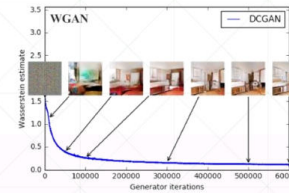
WGAN 还有.



$\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)$

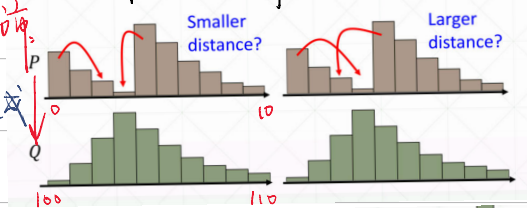


$\frac{1}{m}\sum_{i=1}^{m}-\log\left(D\left(G\left(z^{(i)}\right)\right)\right)$



$\frac{1}{m}\sum_{i=1}^{m}f_w\left(x^{(i)}\right)-\frac{1}{m}\sum_{i=1}^{m}f_w\left(g_\theta\left(z^{(i)}\right)\right)$

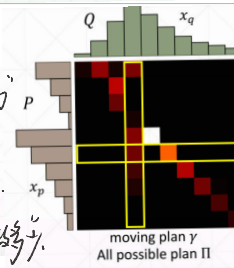
JS / KL 都不能引导



缺点：走多少步一样的直接报

优点

代价



A "moving plan" is a matrix

The value of the element is the amount of earth from one position to another.

Average distance of a plan γ:

$$B(\gamma)=\sum_{x_p,x_q}\gamma(x_p,x_q)\|x_p-x_q\|$$

Earth Mover's Distance:

$$W(P,Q)=\min_{\gamma\in\Pi}B(\gamma)$$

The best plan

怎么算最简单的方案

Wasserstein Distance.

解决了 overlap.

$$W(\mathbb{P}_r,\mathbb{P}_g)=\inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)}\mathbb{E}_{(x,y)\sim\gamma}\left[\|x-y\|\right],$$

$$w\longrightarrow\left[f\left(x^{(i)}\right)-f\left(G\left(z^{(i)}\right)\right)\right]$$

real

新映射 f ← Discriminator/Critic

		Generator
GAN	JS	$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}-\log\left(D\left(G\left(z^{(i)}\right)\right)\right)$
WGAN	WD	$\nabla_\theta\frac{1}{m}\sum_{i=1}^{m}-f\left(G\left(z^{(i)}\right)\right)$

$\nabla_w\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right)+\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)\right]$

$\nabla_w\frac{1}{m}\sum_{i=1}^{m}\left[f\left(x^{(i)}\right)-f\left(G\left(z^{(i)}\right)\right)\right]$

用神经网络逼近 f 由 Wasserstein Distance 学习得到

约束：在上任取两点 x_1, x_2，梯度小于1.

$$|f(x_1)-f(x_2)|\le|x_1-x_2|.$$

1-Lipschitz function

threshold.

Weight Clipping [Martin Arjovsky, et al., arXiv, 2017]

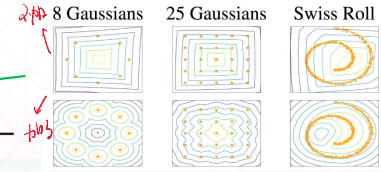
Force the parameters w between c and -c
After parameter update, if w > c, w = c;
if w < -c, w = -c

How to fulfill this constraint?

非常数 ≤1 1−Lipschitz?

1−Lipschitz?

Sort of 约束 Regularization



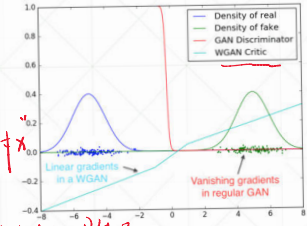
WGAN-Gradient Penalty

1-Lf

$\hat{x} \in [x_r, x_g]$

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

对D对x的导数.



在1这附近.

$P_r \rightarrow P_g$

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \lambda \underbrace{\mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

随便sample个点.

where $\hat{\boldsymbol{x}}$ sampled from $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$ with t uniformly sampled between 0 and 1

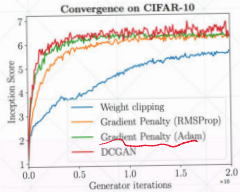
$$\hat{\boldsymbol{x}} = t\tilde{\boldsymbol{x}} + (1-t)\boldsymbol{x} \text{ with } 0 \leq t \leq 1$$

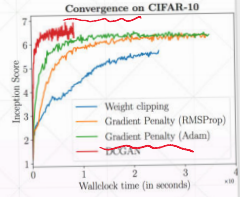
两点间差值.

$\geq \frac{dD}{dx}$, avg

DCGAN: 给个常数.

WGAD 增加复杂度. $\frac{dP}{dx}$.

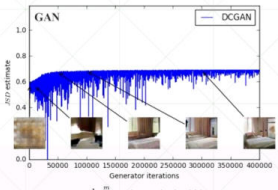




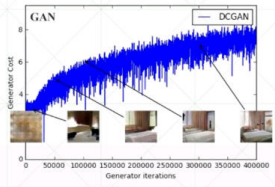
日期:

Unsupervised		Supervised	
Method	Score	Method	Score
ALI [8] (in [27])	$5.34 \pm .05$	SteinGAN [26]	6.35
BEGAN [4]	5.62	DCGAN (with labels, in [26])	6.58
DCGAN [22] (in [11])	$6.16 \pm .07$	Improved GAN [23]	$8.09 \pm .07$
Improved GAN (-L+HA) [23]	$6.86 \pm .06$	AC-GAN [20]	$8.25 \pm .07$
EGAN-Ent-VI [7]	$7.07 \pm .10$	SGAN-no-joint [11]	$8.37 \pm .08$
DFM [27]	$7.72 \pm .13$	WGAN-GP ResNet (ours)	$8.42 \pm .10$
WGAN-GP ResNet (ours)	$7.86 \pm .07$	**SGAN** [11]	$8.59 \pm .12$

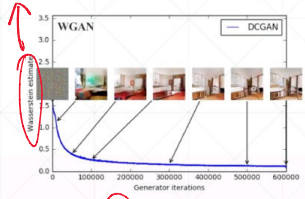
可以反映出距离.



$$\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)$$



$$\frac{1}{m}\sum_{i=1}^{m}-\log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$



$$\frac{1}{m}\sum_{i=1}^{m} f_w\left(x^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{m} f_w\left(g_\theta\left(z^{(i)}\right)\right)$$

日期: /

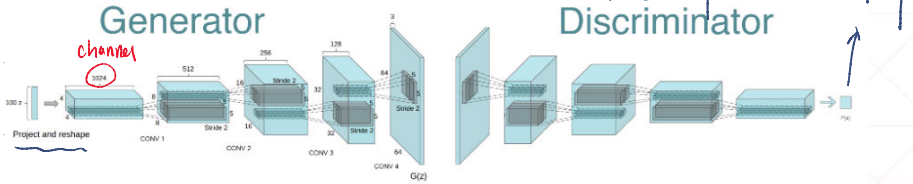
GAN 实战

数据集: Anime [b 96×96×3]

上采样
升维.

降维
classifier.

判断



采用卷积 convolution 操作.

定义 Generator: class Generator

预处理 def __init__

channel 降, size 升

z = [b, 100] → [b, 3×3×512] → [b, 3, 3, 512] → [b, 64, 64, 3]

反向卷积, 升维.

strides

self.fc = layers.Dense(3×3×512)

self.conv1 = layers.Conv2DTranspose(256, 3, 3)

self.bn1

channel

self.conv2 = (128, 5, 2) conv3 = (3, 4, 2)

日期: / 参数 channel : 512 → 256 → 128 → 3 最后一层.

size : 调节 strides ! 最终为 64×64×3.

```
def call:
    x = self.fc(input)  [2, 100] → [2, 4*4*512]
    x = tf.reshape(x, [-1, 4, 4, 512])  4*4*512 → [4, 4, 512]
    x = tf.nn.relu(bn1(conv1(x)))
    x = tf.nn.relu(bn2(conv2(x)))
    x = self.conv3(x)
    x = tf.tanh(x)        [-1, 1] 激活函数.
    return x.             size [b, 64, 64, 3]
```

鉴别器

Discriminator. class Discriminator.

判断为真 def __init__:

[b, 64, 64, 3] → [b, 1] → 0/1

3个(conv + bn) + 1 fc.

channel 升, size 降.

conv1 = (64, 5, 3) conv2 = (128, 5, 3)

conv3 = (256, 5, 3).

[b, h, w, 3] → [b, -1] 打平. layer.Flatten()

self.fc = layers.Dense(1)

日期: /

前向传播. def call.

x = tf.nn.leaky_relu(self.conv1(inputs))

x = tf.nn.leaky_relu(self.bn2(self.conv2(x)))

x = tf.nn.leaky_relu(self.bn3(self.conv3(x)))

x = self.flatten.

logits = self.fc(x) [b, -1] → [b, 1]

return logits. size: [b, 1]

测试网络.

def main:

d = Discriminator() x = tf.random.normal([2, 64, 64, 3])

g = Generator() z = tf.random.normal([2, 100])

prob = g(x) x_hat = g(z)

日期: /

采样生成两个batch

训练过程: Dataset —sample→ batch_z 真

Generator —Sample→ batch_x 假

真采样送入↘

batch_z —Generator→ Fake Image. 得到一张假图

batch_x (Real) → Discriminator (区别鉴别真假.) → real_logits

Fake Imge → Discriminator → fake_logits

使假logits→0

fake_logits —计算loss (=0)→ d_loss_Fake

使真logits=1

real_logits —计算loss (=1)→ d_loss_Real

Discriminator 的 Loss = d_loss_fake + d_loss_real

G 只 需要 Fake 且希望 Fake 过 D 之后得1. (欺骗) 骗过

fake_logits —计算loss (=1)→ g_loss_fake

Generator 的 loss = g_loss_fake

d_loss 分别计算梯度 grads —optimizer→ 更新

g_loss

日期: /

WGAN-GP

超参数

假 → D 作微分趋于 1

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

where $\hat{\boldsymbol{x}}$ sampled from $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$ with t uniformly sampled between 0 and 1

$\hat{\boldsymbol{x}} = t\tilde{\boldsymbol{x}} + (1-t)\boldsymbol{x}$ with $0 \le t \le 1$

新图片 = 假 + 真. 随机 sample.

惩罚

只用改一个地方. Loss fn 一样. gradient penalty

网络模型不用改

```
def gradient_penalty.
    batchsz = batch_x.shape.
    t = tf.random.uniform([batchsz, 1, 1, 1])
    [b, 1, 1, 1] → [b, h, w, c]
    t = tf.broadcast_to(t, batch_x.shape)
    interplate. = t * batch_x + (1-t) * fake_imge.
    tape.watch(d_interplate_logits. = discriminator(interplate))
    grads = tape.gradient(d_interplate_logits, interplate)
```

日期:

[b.h.w.c] → [b.-1] batchsz.

grads = reshape(grads, [grads.shape[0], -1])

gp = tf.norm(grads, axis=1) [b] = 范数

gp = tf.reduce_mean((gp-1)**2) MSE

return gp.

def d_loss_fn

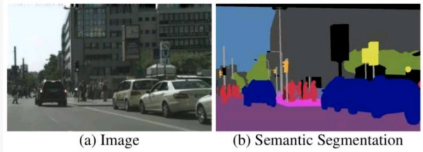
gp = gradient_penalty(discriminator, batch_x, fake_image)

loss = d_loss_fake + d_loss_real + 1 * gp

↓ 可调

日期: /

图像语义分割.

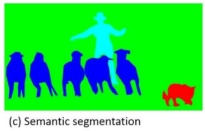


(a) Image (b) Semantic Segmentation

像素级. 对每一个像素点进行分类

这里需要和实例分割区分开来。它没有分离同一类的实例；我们关心的只是每个像素的类别，如果输入对象中有两个相同类别的对象，则分割本身不将他们区分为单独的对象。

总结点 2.物体 有九个



(c) Semantic segmentation

(d) This work segment individual object instances

语义分割的目标

[h.w.3] / [h.w.1]

一般是将一张RGB图像（height*width*3）或是灰度图（height*width*1）作为输入，

大小一样，没有像素值

输出的是分割图，其中每一个像素包含了其类别的标签（height*width*1）

每个像素都有label

[h.w.1]

Input Semantic Labels

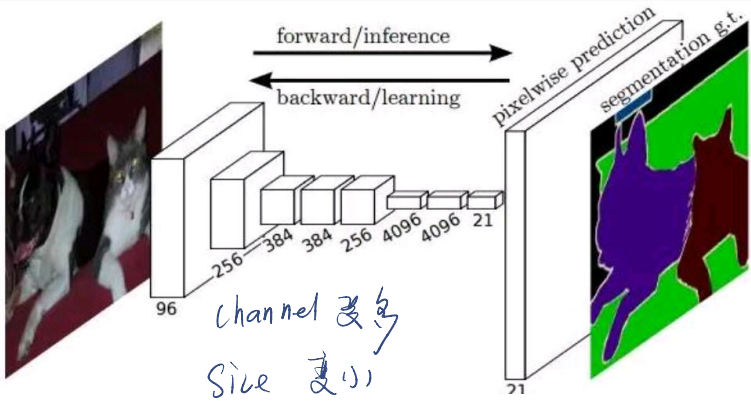
0: Background/Unknown
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

FCN:

目前在图像分割领域比较成功的算法，有很大一部分都来自于同一个先驱：Long等人提出的Fully Convolutional Network（FCN），或者叫全卷积网络。

FCN将分类网络转换成用于分割任务的网络结构，并证明了在分割问题上，可以实现端到端的网络训练。

FCN成为了深度学习解决分割问题的奠基石。

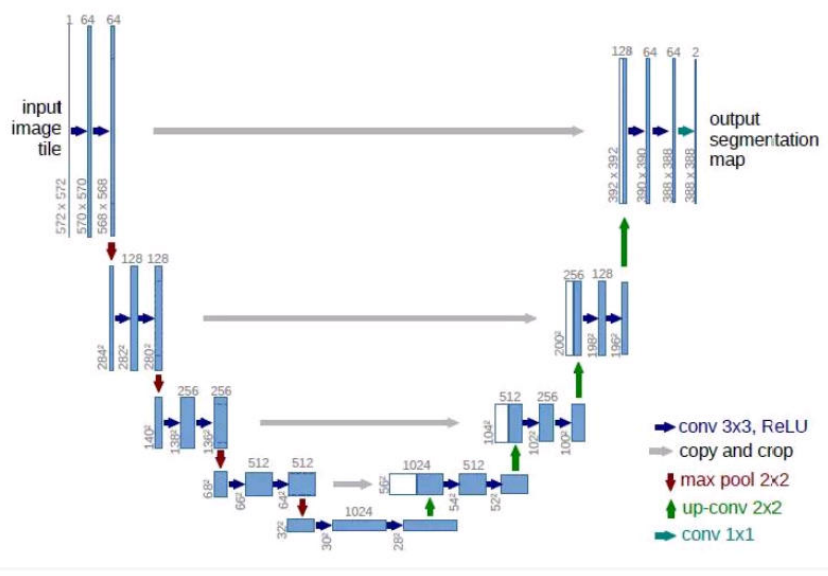


UNET.

数据集小

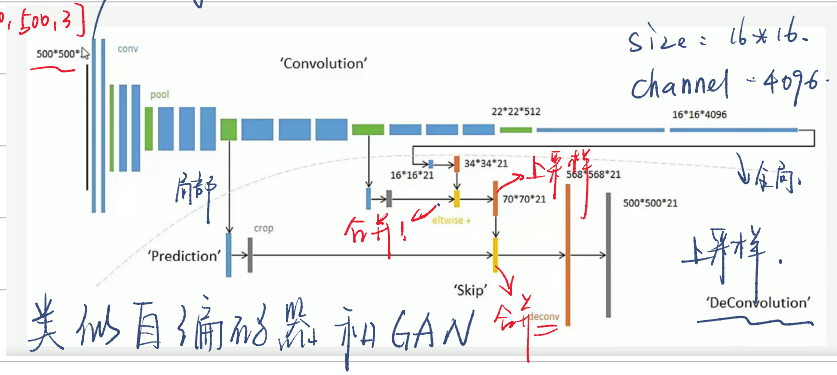
Unet是2015年诞生的模型，它几乎是当前segmentation项目中应用最广的模型。

Unet能从更少的训练图像中进行学习。当它在少于 40 张图的生物医学数据集上训练时，IOU 值仍能达到 92%。



日期: /

图像语义分割 的网络结构



1. 分类网络 (conv) → 上采样
2. 结合 之前 的信息
3. 输入 任意尺寸

Input_Size = Output_Size

Output_Channel = N +1

↓目标的类数 背景

全部使用 conv, 无 FC.

先 strides 再卷积

上采样: Upsampling.

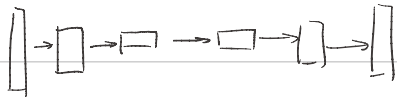
通过训练达大图像.

差值法

反池化

反卷积

日期: /



类自编码器结构

Skip. 跳跃结构

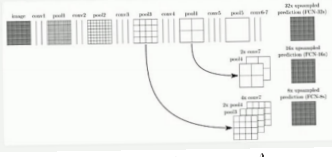
细节 → 抽象. 局部 → 全局.

1/32

将底层（stride 32）的预测（FCN-32s）进行2倍的上采样得到的图像，并与从pool4层（stride 16）进行的预测融合起来（相加），这一部分的网络被称为FCN-16s。随后将这一部分的预测再进行一次2倍的上采样并与从pool3层得到的预测融合起来，这一部分的网络被称为FCN-8s。

通过上采样使其 size 相同，可以运算.

增加Skips结构将最后一层的预测（有更富的全局信息）和更浅层（有更多的局部细节）的预测结合起来，这样可以在遵守全局预测的同时进行局部预测。

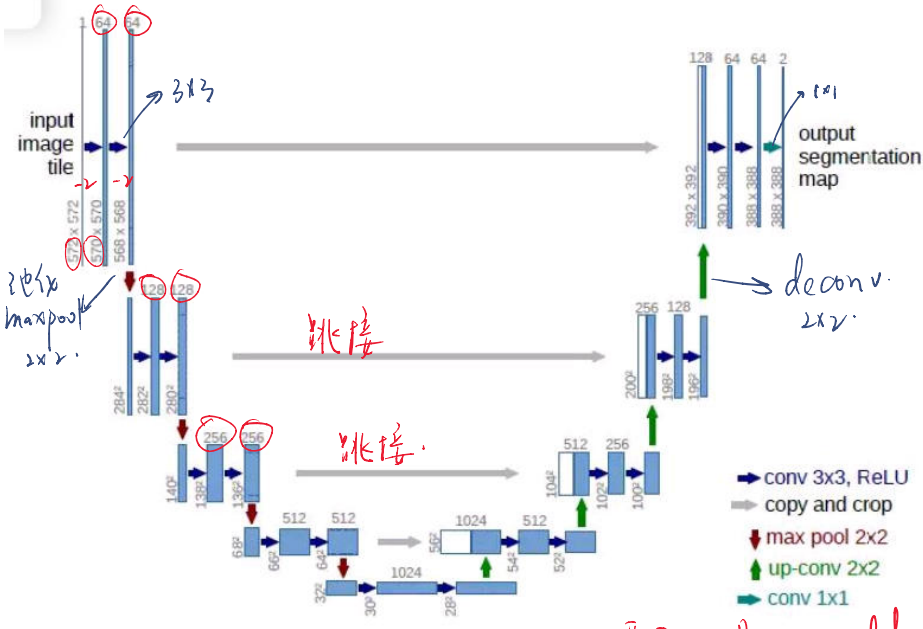


缺点：对细节不够敏感，效果不好.

日期: /

UNet 网络结构

前半部分 特征提取，后半部分是上采样.



FCN采用的是 什么add

采用的特征融合方式：拼接 (tf.concat).

对边缘采用更高的权重.

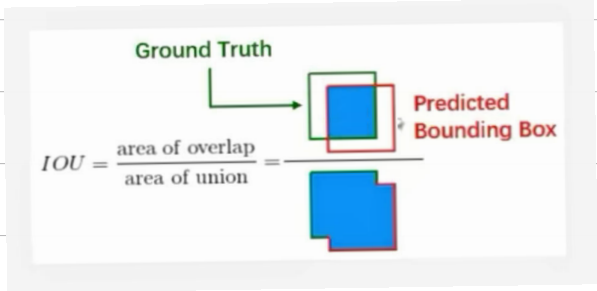
输入：任意彩图

输出：n(目标类别数) +1 (背景)

日期：交并比.

IOU 评价指标.

表示 产生的候选框 (candidate bound) 与原标记框 (ground truth bound)



日期: /

tf.data.

输入 + 处理文件.

tf.data.Dataset 表示一系列元素.

包含多个Tensor对象. 单个训练样本

-对表示数据/标签的张量.

可以通过两种不同的方式来创建tf.data.Dataset

1. 直接从 Tensor 创建 Dataset

例如 Dataset.from_tensor_slices()；

当然 Numpy 也是可以的，TensorFlow 会自动将其转换为 Tensor。

2. 通过对一个或多个 tf.data.Dataset 对象来使用变换（例如 Dataset.zip）来创建 Dataset。

一个 Dataset 对象包含多个元素，每个元素的结构都相同。每个元素包含一个或多个 tf.Tensor 对象，这些对象被称为组件。

张量, 元组, 张量元组,

可以直接迭代.

日期: /