6 神经网络与全连接层

6.1 数据加载

- ✓ keras.datasets
- ✓ tf.data.Dataset.from_tensor_slices
  - ✓ shuffle
  - ✓ map
  - ✓ batch
  - ✓ repeat

keras.datasets

- ✓ bosto housing ←→ Boston housing price regression dataset
- ✓ mnist/fashion mnist ←→MNIST/Fashion-MNIST dataset
- ✓ cifar10/100 ←→ small images classification dataset
- ✓ imdb ←→ sentiment classification dataset

MNIST

```
In [1]: import os

In [2]: os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

In [3]: import tensorflow as tf

In [4]: from tensorflow import keras

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

In [6]: x.shape
Out[6]: (60000, 28, 28)

In [7]: y.shape
Out[7]: (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]
Out[12]:
<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)>
```

训练集

测试集

都是numpy格式的

这三个函数都是numpy的api函数

CIFAR10/100

```
In [4]: from tensorflow import keras

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

In [6]: x.shape,y.shape,x_test.shape,y_test.shape
Out[6]: ((50000, 32, 32, 3), (50000, 1), (10000, 32, 32, 3), (10000, 1))

In [7]: x.min(), x.max()
Out[7]: (0, 255)

In [8]: y[:4]
Out[8]:
array([[6],
       [9],
       [9],
       [4]], dtype=uint8)
```

tf.data.Dataset

```
In [9]: (x,y),(x_test,y_test) = keras.datasets.cifar10.load_data()

In [10]: db = tf.data.Dataset.from_tensor_slices(x_test)

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

In [12]: # 下面不能使用 [x_test,y_test]

In [13]: db = tf.data.Dataset.from_tensor_slices((x_test,y_test))

In [14]: next(iter(db))[0].shape
Out[14]: TensorShape([32, 32, 3])

In [15]: next(iter(db))[1].shape
Out[15]: TensorShape([1])
```

numpy格式

转化为Dataset格式

创建迭代器，每次迭代一个样本

Dataset.shuffle

```
In [16]: db = tf.data.Dataset.from_tensor_slices((x_test,y_test))

In [17]: db = db.shuffle(10000)
```

该参数可以稍微给大一点，打散范围：0~10000

Dataset.map

```
In [18]: 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 [19]: db2 = db.map(preprocess)

In [20]: res = next(iter(db2))

In [21]: res[0].shape, res[1].shape
Out[21]: (TensorShape([32, 32, 3]), TensorShape([1, 10]))

In [22]: res[1][:2]
Out[22]: <tf.Tensor: id=60, shape=(1, 10), dtype=float32, numpy=array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)>
```

对一张图片进行预处理，所以x的shape为[32,32,3]

应用map函数

Dataset.batch

```
In [23]: db3 = db2.batch(32)

In [24]: res = next(iter(db3))

In [25]: res[0].shape, res[1].shape
Out[25]: (TensorShape([32, 32, 32, 3]), TensorShape([32, 1, 10]))
```

标签y的1需要squeeze掉

Dataset.repeat

```
In [26]: db4 = db3.repeat()
```

对数据迭代无限次

```
In [27]: db4 = db3.repeat(2)
```

对数据迭代两次

例子

```
In [43]: 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
    ...:
    ...:

In [44]: def mnist_dataset():
    ...:     (x,y),(x_val,y_val) = keras.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
    ...:
    ...:
```

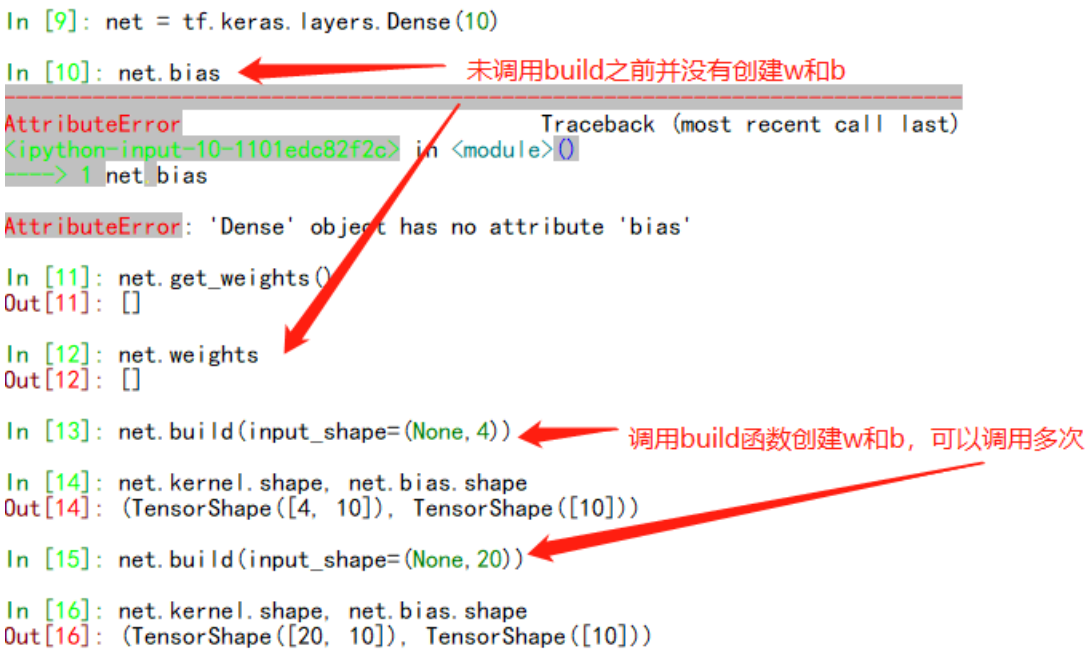
6.2 测试张量(实战)

```
# 实现前向传播
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets

# x : [60k, 28, 28],  x_test : [10k, 28, 28]
# y : [60k], y_test : [10k]
(x, y),(x_test, y_test) = datasets.mnist.load_data()

# x : [0~255] => [0~1]
x = tf.convert_to_tensor(x, dtype=tf.float32) / 255.
y = tf.convert_to_tensor(y, dtype=tf.int32)
```

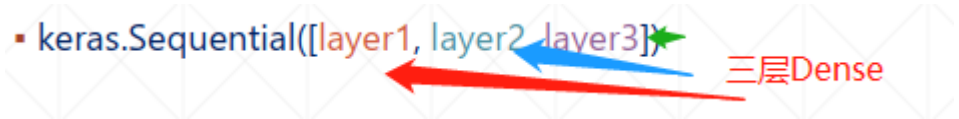
```
x_test = tf.convert_to_tensor(x_test, dtype=tf.float32) / 255.
y_test = tf.convert_to_tensor(y_test, dtype=tf.int32)

train_db = tf.data.Dataset.from_tensor_slices((x, y)).batch(128)
test_db = tf.data.Dataset.from_tensor_slices((x_test,y_test)).batch(1
28)
# [b,784] => [b,256] => [b,128] => [b,10]
# w = [dim_in, dim_out] ; b = [dim_out]
w1 = tf.Variable(tf.random.truncated_normal([784,256],stddev=0.1))
b1 = tf.Variable(tf.zeros([256]))
w2 = tf.Variable(tf.random.truncated_normal([256,128],stddev=0.1))
b2 = tf.Variable(tf.zeros([128]))
w3 = tf.Variable(tf.random.truncated_normal([128,10],stddev=0.1))
b3 = tf.Variable(tf.zeros([10]))

lr = 1e-3

for epoch in range(100):     # iterate 数据集 for 10 次
    for step,(x,y) in enumerate(train_db):      # for every batch
        x = tf.reshape(x, [-1, 28*28])
        with tf.GradientTape() as tape:
            h1 = tf.matmul(x,w1) + b1
            h1 = tf.nn.relu(h1)
            h2 = tf.matmul(h1,w2) + b2
            h2 = tf.nn.relu(h2)
            out = tf.matmul(h2,w3) + b3

            # 计算误差
            # out: [b,10] ;  y: [10]
            y_onehot = tf.one_hot(y, depth=10)

            loss = tf.square(y_onehot - out)        # loss: [b,10]
            # mean ==> scalar
            loss = tf.reduce_mean(loss)

        # compute gradients
        grads = tape.gradient(loss,[w1,b1,w2,b2,w3,b3])

        w1.assign_sub(lr * grads[0])
        b1.assign_sub(lr * grads[1])
        w2.assign_sub(lr * grads[2])
        b2.assign_sub(lr * grads[3])
        w3.assign_sub(lr * grads[4])
```

```
        b3.assign_sub(lr * grads[5])

        if step % 100 == 0:
            print(epoch, step, 'loss:', float(loss))

    # test / evaluation
    total_correct, total_num = 0, 0
    for step,(x,y) in enumerate(test_db):
        # [b, 28, 28] ==> [b, 28*28]
        x = tf.reshape(x, [-1, 28*28])
        h1 = tf.nn.relu(x@w1 + b1)
        h2 = tf.nn.relu(h1@w2 + b2)
        out = h2@w3 + b3

        # out : [b, 10]
        prob = tf.nn.softmax(out, axis=1)

        # [b, 10] ==> [b]
        pred = tf.argmax(prob, axis=1)
        pred = tf.cast(pred, dtype=tf.int32)

        # y : [b]
        correct = tf.cast(tf.equal(pred, y), dtype=tf.int32)
        correct = tf.reduce_sum(correct)
        total_correct += int(correct)
        total_num += x.shape[0]

    acc = total_correct / total_num
    print("test acc ", acc)
```

6.3 全连接层



keras.Sequential




```
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)
```

输出信息：

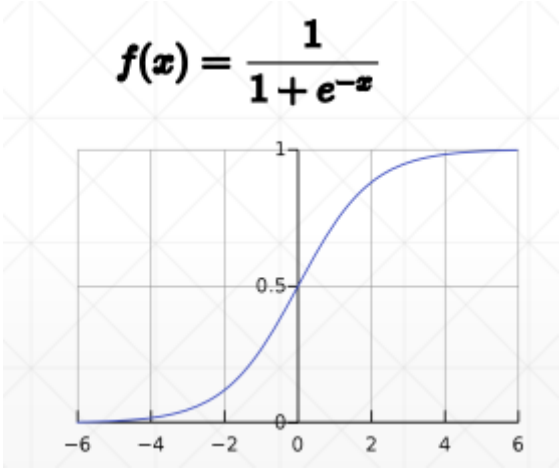
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  8
_________________________________________________________________
dense_1 (Dense)              multiple                  6
_________________________________________________________________
dense_2 (Dense)              multiple                  6
=================================================================
```

```
=======
Total params: 20
Trainable params: 20
Non-trainable params: 0

_________________________________________________________________
dense/kernel:0 (3, 2)
dense/bias:0 (2,)
dense_1/kernel:0 (2, 2)
dense_1/bias:0 (2,)
dense_2/kernel:0 (2, 2)
dense_2/bias:0 (2,)
```

6.4 输出方式

tf.sigmoid

应用 tf.sigmoid，可以使得：$y_i \in [0,1]$

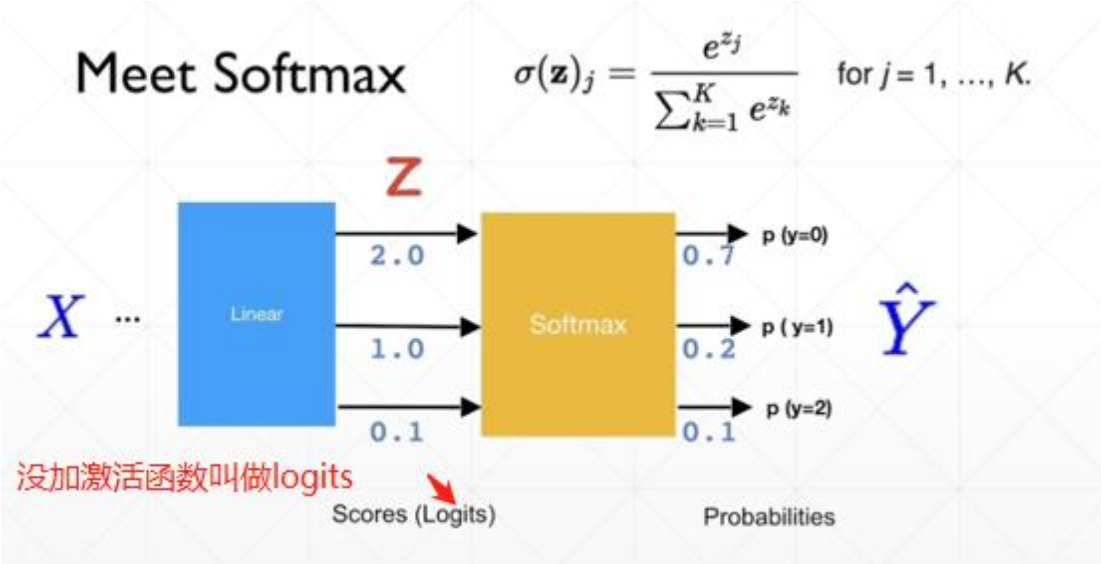



```
In [21]: a = tf.linspace(-6., 6, 10)

In [22]: tf.sigmoid(a)
Out[22]:                                                    压缩到0~1
<tf.Tensor: id=114, 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 ],
      dtype=float32)>

In [23]: x = tf.random.normal([1,28,28])*5

In [24]: tf.reduce_min(x), tf.reduce_max(x)
Out[24]:
(<tf.Tensor: id=124, shape=(), dtype=float32, numpy=-15.806613>,
 <tf.Tensor: id=126, shape=(), dtype=float32, numpy=14.872971>)

In [25]: x = tf.sigmoid(x)

In [26]: tf.reduce_min(x), tf.reduce_max(x)
Out[26]:
(<tf.Tensor: id=129, shape=(), dtype=float32, numpy=1.1920929e-07>,
 <tf.Tensor: id=131, shape=(), dtype=float32, numpy=0.99999964>)
```

sigmoid 能保证单个输出值在 0~1 之间，但是并不能保证所有值之和为 1.

tf.softmax

应用 tf.softmax，可以使得：$y_i \in [0,1]$，$\Sigma y_i = 1$




```
In [27]: a = tf.linspace(-2.,2,5)

In [28]: tf.sigmoid(a)
Out[28]:
<tf.Tensor: id=136, shape=(5,), dtype=float32, numpy=
array([0.11920291, 0.26894143, 0.5       , 0.7310586 , 0.880797  ],
      dtype=float32)>

In [29]: tf.nn.softmax(a)
Out[29]:
<tf.Tensor: id=137, shape=(5,), dtype=float32, numpy=
array([0.01165623, 0.03168492, 0.08612854, 0.23412167, 0.6364086 ],
      dtype=float32)>

In [30]: logits = tf.random.uniform([1,10],minval=-2,maxval=2)

In [31]: logits
Out[31]:
<tf.Tensor: id=144, shape=(1, 10), dtype=float32, numpy=
array([[ 0.10090733, -1.0773139 , -1.2953243 , -1.1285634 , -1.8821301 ,
         0.21339846,  0.64386034,  1.451633  , -1.511538  , -0.7488241 ]],
      dtype=float32)>

In [32]: prob = tf.nn.softmax(logits, axis=1)

In [33]: prob
Out[33]:
<tf.Tensor: id=145, shape=(1, 10), dtype=float32, numpy=
array([[0.10737965, 0.03305423, 0.02657947, 0.03140289, 0.01478086,
        0.12016454, 0.18480918, 0.4145097 , 0.02141144, 0.045908  ]],
      dtype=float32)>

In [34]: tf.reduce_sum(prob, axis=1)
Out[34]: <tf.Tensor: id=147, shape=(1,), dtype=float32, numpy=array([0.99999994], dtype=float32)>
```

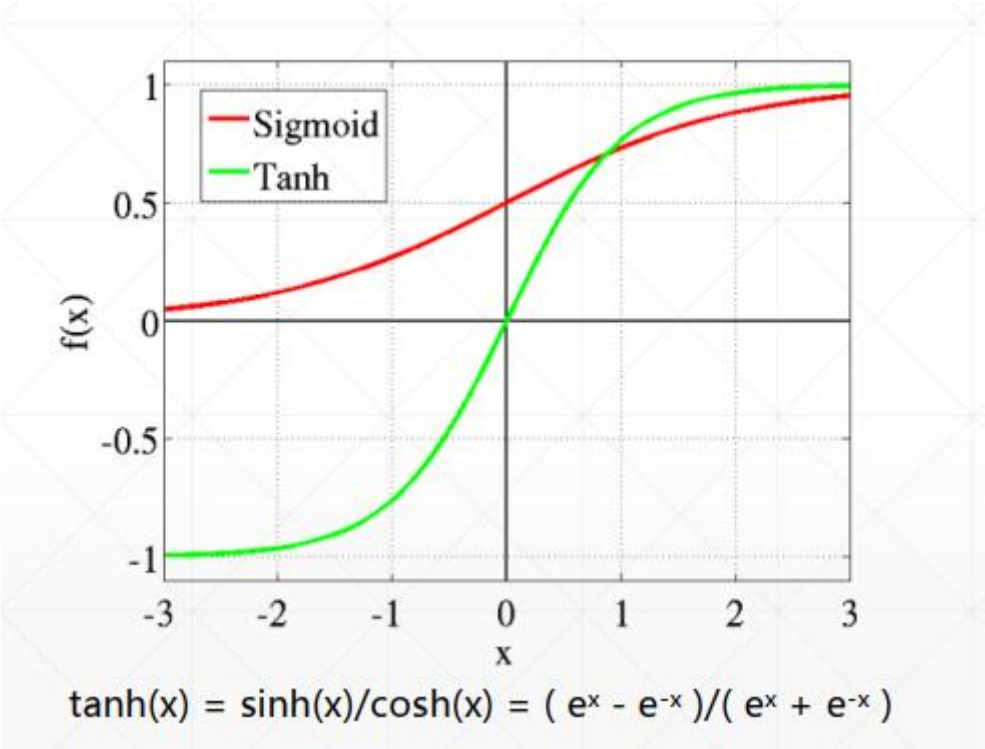
所有和不为1

所有和为1

经过softmax，单个值被压缩到0~1之间，并且和为1

tf.tanh

应用 tf.tanh，可以使得：$y_i \in [-1,1]$



6.5 误差计算

- ✓ MSE
- ✓ Cross Entropy Loss
- ✓ Hinge Loss

MSE

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

```
In [35]: y = tf.constant([1,2,3,0,2])
In [36]: y = tf.one_hot(y, depth=4)
In [37]: y = tf.cast(y, dtype=tf.float32)
In [38]: out = tf.random.normal([5,4])
In [39]: loss1 = tf.reduce_mean(tf.square(y-out))
In [40]: loss2 = tf.square(tf.norm(y-out))/(5*4)
In [41]: loss3 = tf.reduce_mean(tf.losses.MSE(y,out))
In [42]: loss1, loss2,loss3
Out[42]:
(<tf.Tensor: id=162, shape=(), dtype=float32, numpy=1.1129806>,
 <tf.Tensor: id=171, shape=(), dtype=float32, numpy=1.1129806>,
 <tf.Tensor: id=176, shape=(), dtype=float32, numpy=1.1129806>)
```

Σ / (b * depth)

返回的不是一个标量，是一个向量

三者等价，值是一样的

Entropy

熵用来衡量不确定度，熵越低，越不稳定(确定)，那么包含的信息也就越多。

$$Entropy = -\sum_i P(i)log_2 P(i)$$

```
In [43]: a = tf.fill([4], 0.25)

In [44]: a
Out[44]: <tf.Tensor: id=179, shape=(4,), dtype=float32, numpy=array([0.25, 0.25, 0.25, 0.25], dtype=float32)>

In [45]: a*tf.math.log(a) / tf.math.log(2.)
Out[45]: <tf.Tensor: id=184, shape=(4,), dtype=float32, numpy=array([-0.5, -0.5, -0.5, -0.5], dtype=float32)>

In [46]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))
Out[46]: <tf.Tensor: id=192, shape=(), dtype=float32, numpy=2.0>

In [47]: a = tf.constant([0.1, 0.1, 0.1, 0.7])

In [48]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))
Out[48]: <tf.Tensor: id=201, shape=(), dtype=float32, numpy=1.3567796>

In [49]: a = tf.constant([0.01, 0.001, 0.001, 0.97])

In [50]: -tf.reduce_sum(a*tf.math.log(a)/tf.math.log(2.))
Out[50]: <tf.Tensor: id=210, shape=(), dtype=float32, numpy=0.12899514>
```

tensorflow中的log是以e为底的，所以要除以个以2为底的

概率

熵 → 越低，惊喜度越高

Cross Entropy

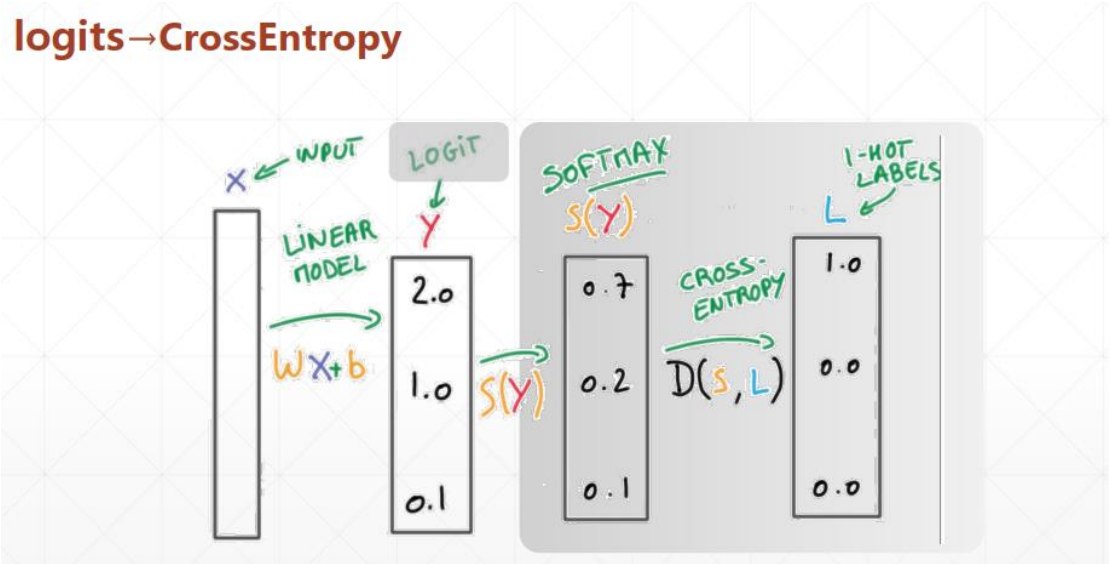
$$H(p,q) = -\sum p(x)\log q(x)$$

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

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

```
In [51]: tf.losses.categorical_crossentropy([0,1,0,0], [0.25,0.25,0.25,0.25])
Out[51]: <tf.Tensor: id=227, shape=(), dtype=float32, numpy=1.3862944>

In [52]: tf.losses.categorical_crossentropy([0,1,0,0], [0.1,0.1,0.8,0.1])
Out[52]: <tf.Tensor: id=244, shape=(), dtype=float32, numpy=2.3978953>

In [53]: tf.losses.categorical_crossentropy([0,1,0,0], [0.1,0.7,0.1,0.1])
Out[53]: <tf.Tensor: id=261, shape=(), dtype=float32, numpy=0.35667497>

In [54]: tf.losses.categorical_crossentropy([0,1,0,0], [0.01,0.97,0.01,0.01])
Out[54]: <tf.Tensor: id=278, shape=(), dtype=float32, numpy=0.030459179>

In [55]: tf.losses.BinaryCrossentropy()([1], [0.1])
Out[55]: <tf.Tensor: id=312, shape=(), dtype=float32, numpy=2.3025842>

In [56]: tf.losses.binary_crossentropy([1], [0.1])
Out[56]: <tf.Tensor: id=337, shape=(), dtype=float32, numpy=2.3025842>
```

从 softmax 到 cross_entropy 可能会出现除 0 操作，从而使得 loss 为 nan。



Numerical Stability

```
In [61]: x = tf.random.normal([1,784])

In [62]: w = tf.random.normal([784,2])

In [63]: b = tf.zeros([2])

In [64]: logits = x@w+b

In [65]: logits
Out[65]: <tf.Tensor: id=369, shape=(1, 2), dtype=float32, numpy=array([[-18.068073, -17.47423 ]], dtype=float32)>

In [66]: prob = tf.math.softmax(logits, axis=1)  经过one_hot

In [67]: tf.losses.categorical_crossentropy([0,1], logits, from_logits=True)   传入logits   推荐方式，解决数值不稳定问题
Out[67]: <tf.Tensor: id=405, shape=(1,), dtype=float32, numpy=array([0.43967378], dtype=float32)>

In [68]: tf.losses.categorical_crossentropy([0,1], prob)   不推荐方式，logits经过softmax之后得到的prob
Out[68]: <tf.Tensor: id=421, shape=(1,), dtype=float32, numpy=array([0.43967384], dtype=float32)>
```