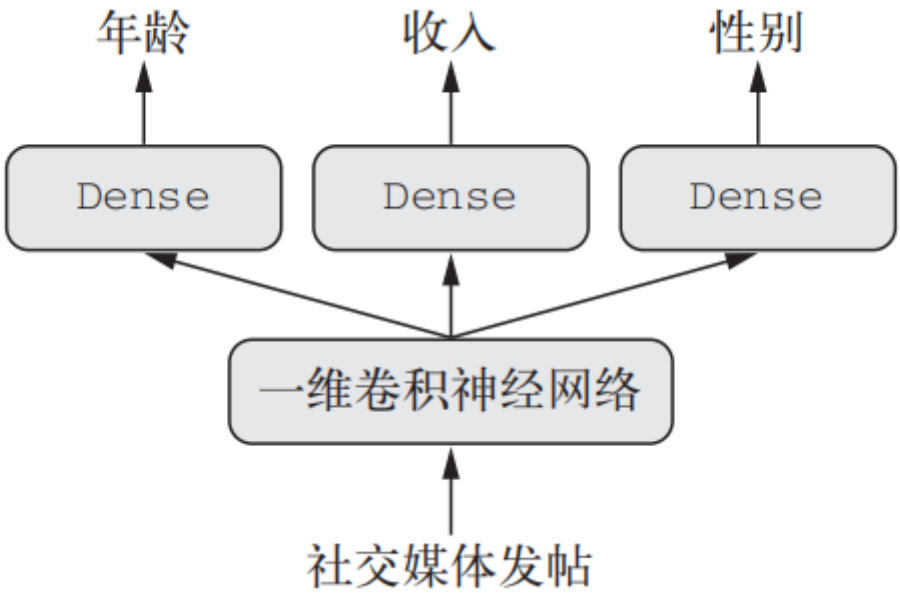
多输出

一个网络试图同时预测数据的不同性质

eg.一个网络，输入某个人的一系列社交媒体发帖，然后尝试预测那个人的属性，比如年龄、性别和收入水平



In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [3]:

```
from tensorflow.keras import Input, layers, Model
```

In [4]:

```
vocabulary_size = 50000
num_income_groups = 10
```

In [5]:

```
posts_input = Input(shape=(None, ), dtype='int32', name='posts') # 评论长度是可变的
```

In [6]:

```
embedded_posts = layers.Embedding(vocabulary_size, 256) (posts_input)
```

In [7]:

```
embedded_posts.shape
```

Out[7]:

```
TensorShape([None, None, 256])
```

In [8]:

```
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
```

In [9]:

```
x.shape
```

Out[9]:

```
TensorShape([None, None, 128])
```

In [10]:

```
x = layers.MaxPooling1D(5)(x)
```

In [11]:

```
x.shape
```

Out[11]:

```
TensorShape([None, None, 128])
```

In [12]:

```
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)
```

In [13]:

```
age_pred = layers.Dense(1, name='age')(x)
income_pred = layers.Dense(num_income_groups, activation='softmax', name='income')(x)
gender_pred = layers.Dense(1, activation='sigmoid', name='gender')(x)
```

In [14]:

```
model = Model(posts_input, [age_pred, income_pred, gender_pred])
```

In [15]:

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected t
o
==========================================================================================
posts (InputLayer)              [(None, None)]       0
__________________________________________________________________________________________
embedding (Embedding)           (None, None, 256)    12800000    posts[0][0]
__________________________________________________________________________________________
conv1d (Conv1D)                 (None, None, 128)    163968      embedding
[0][0]
__________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, None, 128)    0           conv1d[0]
[0]
__________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, None, 256)    164096      max_pooling
1d[0][0]
__________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, None, 256)    327936      conv1d_1[0]
[0]
__________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, None, 256)    0           conv1d_2[0]
[0]
__________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, None, 256)    327936      max_pooling
1d_1[0][0]
__________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, None, 256)    327936      conv1d_3[0]
[0]
__________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 256)          0           conv1d_4[0]
[0]
__________________________________________________________________________________________
dense (Dense)                   (None, 128)          32896       global_max_
pooling1d[0][0]
__________________________________________________________________________________________
age (Dense)                     (None, 1)            129         dense[0][0]
__________________________________________________________________________________________
income (Dense)                  (None, 10)           1290        dense[0][0]
__________________________________________________________________________________________
gender (Dense)                  (None, 1)            129         dense[0][0]
```

```
==================================================================================
====================
Total params: 14,146,316
Trainable params: 14,146,316
Non-trainable params: 0
__________________________________________________________________________________
____________________
```

三个输出需要指定不同的损失函数

年龄是回归，收入是多分类，性别是二分类，需要不同的训练过程

但是，梯度下降要求将一个标量最小化，所以为了能够训练模型，我们必须将这些损失合并为单个标量(全局损失)。合并不同损失最简单的方法就是对所有损失求和

In [16]:

```
model.compile(optimizer='rmsprop', loss=['mse','categorical_crossentropy','binary_crossentr
```

In [17]:

```
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'
               })  # 仅限输出层有名字时
```

严重不平衡的损失贡献会导致模型表示针对单个损失值最大的任务优先进行优化，而不考虑其他任务的优化。为了解决这个问题，我们可以为每个损失值对最终损失的贡献分配不同大小的重要性。如果不同的损失值具有不同的取值范围，那么这一方法尤其有用

用于年龄回归任务的均方误差（MSE）损失值通常在 3~5 左右，而用于性别分类任务的交叉熵损失值可能低至 0.1。在这种情况下，为了平衡不同损失的贡献，我们可以让二元交叉熵损失的权重取 10，而 MSE 损失的权重取 0.25，多分类交叉熵损失权重取 1

In [18]:

```
model.compile(optimizer='rmsprop',
              loss=['mse','categorical_crossentropy','binary_crossentropy'],
              loss_weights = [0.25, 1., 10.]
             )
```

In [19]:

```
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'},
              loss_weights={'age': 0.25,
                            'income': 1.,
                            'gender': 10.})
```

In [20]:

```
import numpy as np
```

In [21]:

```
num_samples = 100
```

In [22]:

```
age_targets = np.random.randint(12, 55, (num_samples))
```

In [23]:

```
age_targets.shape
```

Out[23]:

```
(100,)
```

In [24]:

```
income_targets = np.random.randint(1, 10, (num_samples))
```

In [25]:

```
income_targets.shape
```

Out[25]:

```
(100,)
```

In [26]:

```
from tensorflow.keras import utils
```

In [27]:

```
income_targets = utils.to_categorical(income_targets, num_classes=10)
```

In [28]:

```
income_targets.shape
```

Out[28]:

```
(100, 10)
```

In [29]:

```
gender_targets = np.random.randint(0, 1, (num_samples))
```

In [30]:

```
gender_targets = utils.to_categorical(gender_targets, num_classes=2)
```

In [31]:

```
gender_targets.shape
```

Out[31]:

```
(100, 2)
```

In [ ]:

```
model.fit(posts_input,  [age_targets, income_targets, gender_targets], epochs=10, batch_siz
```

In [ ]:

```
model.fit(posts, {'age': age_targets,
                  'income': income_targets,
                  'gender': gender_targets},
          epochs=10, batch_size=64)
```

In [ ]:

```

```

In [34]:

```
np.random.randint(12, 55, (34))
```

Out[34]:

```
array([35, 14, 26, 28, 43, 46, 39, 27, 25, 36, 41, 13, 35, 43, 24, 18, 27,
       14, 46, 44, 46, 28, 42, 29, 25, 31, 35, 29, 26, 18, 39, 26, 28, 46])
```