多任务学习之MMOE模型

王多鱼 python科技园 6月10日

1

前言

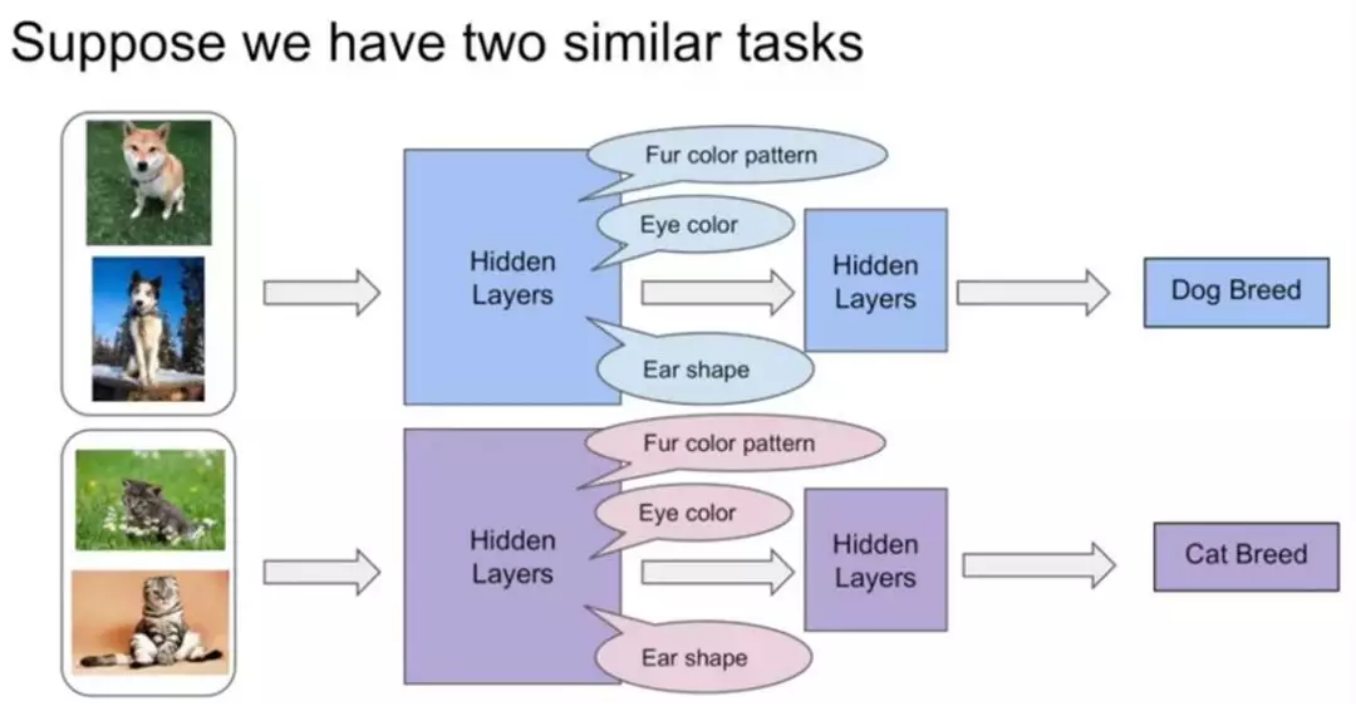
MMOE模型，全称为：Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts 。

论文地址为：https://dl.acm.org/doi/10.1145/3219819.3220007

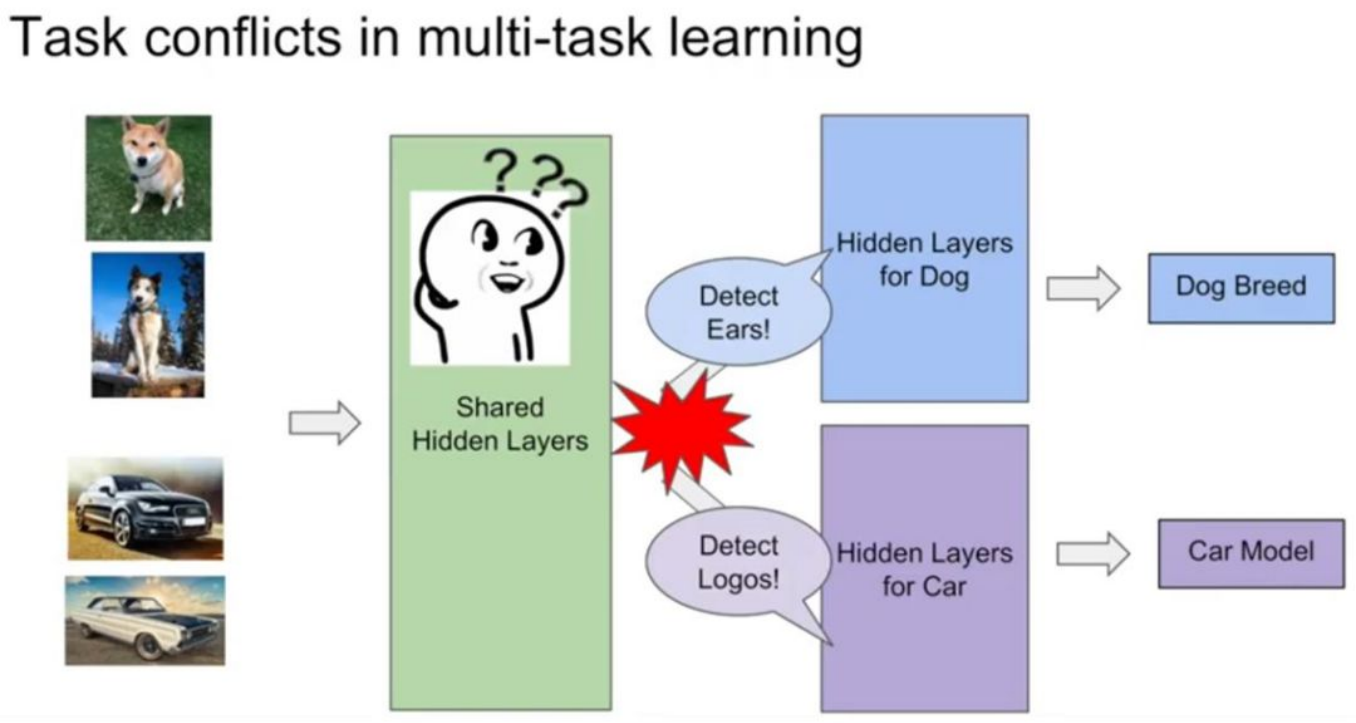
代码地址为：https://github.com/drawbridge/keras-mmoe

多任务学习在工业界中已经逐渐开始普及，例如在推荐系统中，不仅要考虑用户感兴趣的物品（产生点击），还要尽可能地促成转化（产生购买），因此要同时对Ctr和Cvr两种指标建模。阿里提出的ESSM模型就是同时对Ctr和Cvr进行建模，该模型属于典型的Shared-Bottom结构，参见：ESMM多任务学习算法在推荐系统中的应用。

由于多任务学习本质上是共享表示层，子任务之间互相影响。如果子任务的差异较小，它们就可以很好地进行底层特征共享，如图（a）狗和猫的识别；但是对于不相似的子任务来说，共享表示层的效果可能不明显，进行参数共享时会互相冲突或噪声太强，对多任务学习而言非常不友好，如图（b）狗和汽车的识别。



（a）狗和猫的识别



（b）狗和汽车的识别

因此，论文中提出了一个 Multi-gate Mixture-of-Experts(MMoE) 的多任务学习模型。MMoE模型刻画了子任务之间的相关性，基于共享表示来学习特定子任务的函数，避免了明显增加参数的缺点，同时提高了子任务的效果。

2

MMOE原理介绍

关于共享隐层方面，MMoE和一般多任务学习模型的区别：

- **一般多任务学习模型**：接近输入层的隐层作为一个整体被共享；
- **MMoE模型**：将共享的底层表示层分为多个Expert，同时设置了Gate，使得不同的任务可以多样化的使用共享层；

如图1，a）是最原始的多任务学习模型，也就是 Shared-Bottom 模型；b）是加入单门的 One-gate MoE 多任务学习模型；c）是文章提出的 Multi-gate MoE 模型。

可以看出，c）本质上是将Shared-Bottom换成了MoE Layer，并对每个任务都增加Gate。

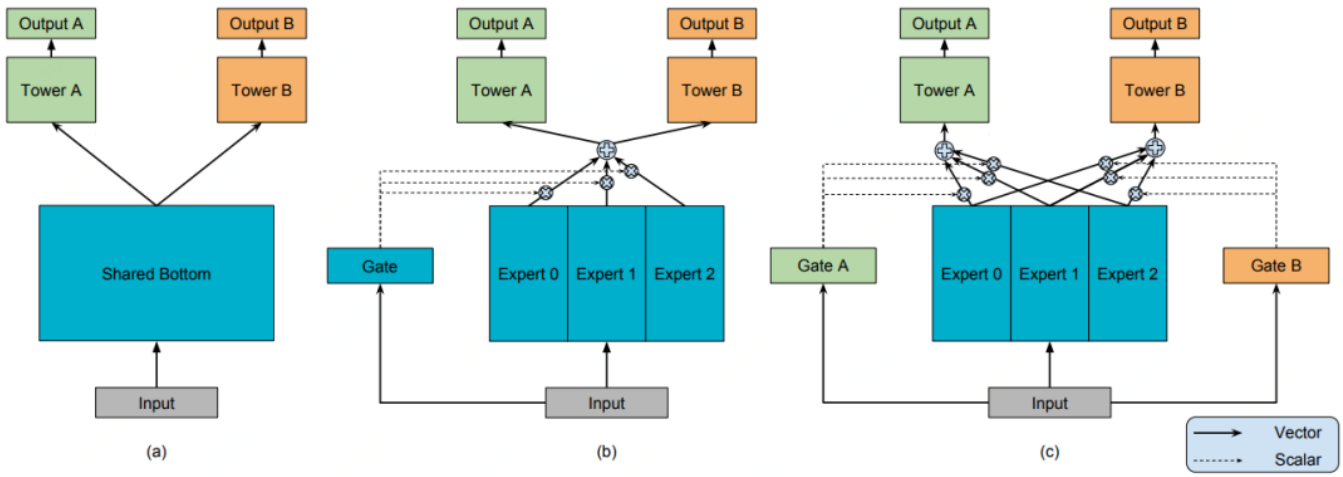


Figure 1: (a) Shared-Bottom model. (b) One-gate MoE model. (c) Multi-gate MoE model.

1. Shared-Bottom model

图a）结构的多任务模型最为常见，多个子任务共享模型的底层结果，即 Shared-Bottom model结果（用函数$f(x)$表示）。K 个子任务分别对应一个Tower Network（用 h^k 表示），每个子任务的输出结果为 $y_k = h^k(f(x))$ 。

2. One-gate MoE mdoel

MoE模型可以用函数：$y = \sum_{i=1}^{n} g(x)_i f_i(x)$ 表示。**其中：**

- $\sum_{i=1}^{n} g(x)_i = 1$ ， $g(x)_i$ 是输入的第 i 个logits；

- $f_i, i = 1, \ldots, n$ 表示 n 个Expert Networks，每个Expert Network可以认为是一个隐层神经网络；

g 是Gate kernel，对应 n 个 Expert Networks上的概率分布，最终作用于 n 个 Expert Networks。MoE模型的共享底层可以看做是基于多个独立模型的集成方法。

3. Multi-gate MoE model

MMoE模型是在One-gate MoE模型的基础上，对于每一个子任务，专门分配一个Gate kernel，表达式为：

$$
\begin{aligned}
y_k &= h^k\left(f^k(x)\right) \\
&= h^k\left(\sum_{i=1}^{n} g^k(x)_i f_i(x)\right)
\end{aligned}
$$

其中：

- $g^k(x) = softmax(W_{gk}x), W_{gk} \in \mathbb{R}^{d \times n}$ ；
- $f(x) \in \mathbb{R}^{n \times d}$， n 是Expert Networks个数，d 是特征的维度。

优势：
（1）MMoE模型增加了多个Gate kernels，可以有效的捕捉到子任务的相关性和区别；
（2）相对于MoE模型，MMoE模型并没有增加太多的参数，从论文的效果来看有明显的提升；

3

小试MMOE模型中Export和Gate模块

1. 模拟数据

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input
from keras import backend as K

# 1. expert_kernels (input_features * hidden_units * num_experts = 4 * 2 * 3)
```

```
# input_features = 4
# hidden_units = 2
# num_experts = 3

expert_kernels = tf.constant([
    [[1., 1., 1.], [2., 2., 1.]], \
    [[0.1, 0.5, 1.], [0.4, 0.1, 1.]], \
    [[1., 1., 1.], [2., 2., 1.]], \
    [[0., 1., 6.], [0., 2., 0.]]
    ], dtype=tf.float64)

print("expert_kernels: \n", expert_kernels)


# 2. gate_kernels (input_features * num_experts * num_tasks = 4 * 3 * 2)

# input_features = 4
# num_experts = 3
# num_tasks = 2

gate_kernels = [tf.constant([[0.1, 0.5, 1.], [0.4, 0.1, 1.], [1., 1., 1.], [2., 2., 1.]], dtype=tf.float64
    tf.constant([[1., 2., 1.], [4., 0.2, 1.5], [2., 1., 0.], [5., 2., 1.]], dtype=tf.float64)]

print("\n"*3)
print("gate_kernels: \n", gate_kernels)
print("\n"*3)


# 3. input samples (samples * input_features = 2 * 4)

# samples = 2
# input_features = 4
inputs = tf.constant([[1., 2., 1., 0.], [4., 0.2, 1., 1.]], dtype=tf.float64)

print("inputs: \n", inputs)
```

数据结果：

```
expert_kernels:
tf.Tensor(
[[[1. 1. 1. ]
```

```
 [2. 2. 1. ]]

 [[0.1 0.5 1. ]
 [0.4 0.1 1. ]]

 [[1. 1. 1. ]
 [2. 2. 1. ]]

 [[0. 1. 6. ]
 [0. 2. 0. ]]], shape=(4, 2, 3), dtype=float64)


gate_kernels:
[<tf.Tensor: shape=(4, 3), dtype=float64, numpy=
array([[0.1, 0.5, 1. ],
   [0.4, 0.1, 1. ],
   [1. , 1. , 1. ],
   [2. , 2. , 1. ]])>, <tf.Tensor: shape=(4, 3), dtype=float64, numpy=
array([[1. , 2. , 1. ],
   [4. , 0.2, 1.5],
   [2. , 1. , 0. ],
   [5. , 2. , 1. ]])>]

inputs:
tf.Tensor(
[[1. 2. 1. 0. ]
[4. 0.2 1. 1. ]], shape=(2, 4), dtype=float64)
```

2. Expert 结构

```
# result 1: expert_outputs = input * expert_kernels (samples * hidden_units * num_experts
# f_{i}(x) = activation(W_{i} * x + b)

# samples = 2
# hidden_units = 2
# num_experts = 3

expert_outputs = tf.tensordot(a=inputs, b=expert_kernels, axes=1)
print("expert_outputs: \n", expert_outputs)
print("\n"*3)
```

数据结果：

```
expert_outputs:
tf.Tensor(
[[[ 2.2   3.    4.  ]
  [ 4.8   4.2   4.  ]]

 [[ 5.02  6.1  11.2 ]
  [10.08 12.02  5.2 ]]], shape=(2, 2, 3), dtype=float64)
```

3. Gate结构

```
# result 2: gate_outputs = input * gate_kernels (num_tasks * samples * num_experts = 2 * 2
# g^{k}(x) = activation(W_{g,k} * x + b)

# num_tasks = 2

# samples: 2
# num_experts = 3

gate_outputs = []

for index, gate_kernel in enumerate(gate_kernels):
    gate_output = K.dot(x=inputs, y=gate_kernel)
    gate_outputs.append(gate_output)

gate_outputs = tf.nn.softmax(gate_outputs)
print("gate_outputs: \n", gate_outputs)
```

数据结果：

```
gate_outputs:
 tf.Tensor(
[[[1.00151222e-01 8.19968851e-02 8.17851893e-01]
  [4.79733364e-02 2.23775958e-01 7.28250705e-01]]

 [[9.98589658e-01 4.99745626e-04 9.10595901e-04]
  [6.80656487e-01 3.18320187e-01 1.02332564e-03]]], shape=(2, 2, 3), dtype=float64)
```

4. Gate结构 * Expert 结构

```
# result 3: final_outputs = gate_outputs * expert_outputs (num_tasks * samples * hidden_u
# 每个 task 的权重值 (gate_output) 分别作用于 expert_outputs，根据 hidden_units 维度进行加
```

```
    # f^{k}(x) = sum_{i=1}^{n} (g^{k}_{i}(x) * f_{i}(x))

    final_outputs = []
    hidden_units = 2

    for gate_output in gate_outputs:
        expanded_gate_output = K.expand_dims(gate_output, axis=1)
        #print("expanded_gate_output", expanded_gate_output)
        #print("\n"*2)

        weighted_expert_output = expert_outputs * K.repeat_elements(expanded_gate_output, h
        #print("weighted_expert_output: ", weighted_expert_output)
        #print("\n"*3)

        final_outputs.append(K.sum(weighted_expert_output, axis=2))

    print("final_outputs: \n", final_outputs)
```

数据结果：

```
final_outputs:
[<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[3.73773092, 4.09652035],
       [9.76226739, 6.96026192]])>, <tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[ 2.20203887, 4.79897168],
       [ 5.37010995, 10.69254733]])>]
```

欢迎关注 **“python科技园”** 及 **添加小编** 进群交流。