多目标学习(MMOE/ESMM/PLE)在推荐系统的实战经验分享

原创 绝密伏击 深度传送门 昨天

收录于话题
#召回 3 #推荐系统 3 #深度学习 3

作者 | 绝密伏击
知乎 | https://zhuanlan.zhihu.com/p/291406172
整理 | 深度传送门

一、前言

最近搞了一个月的视频多目标优化，同时优化点击率和衍生率(ysl，点击后进入第二个页面后续的点击次数)，线上AB实验取得了不错的效果，总结一下优化的过程，更多的偏向实践。

Models	AUC CTR	MSE YSL	logloss CTR	训练方式	点击提升	YSL提升
Single-Task	0.7206	3.405	0.5186			
Loss加权	0.7178		0.5285	w=log(1+ysl)	-1.5%	+5%
PLE	0.7322	3.298	0.4675	Joint Training+UWL	+5.5%	+5%

表1：线上实验结果

二、业界方案

2.1 样本Loss加权

保证一个主目标的同时，将其它目标转化为样本权重，改变数据分布，达到优化其它目标的效果。

$$Loss = -\frac{1}{n}\sum_{i=1}^{n} w_i \times y_i \times log\left(p_i\right) + w_i \times \left(1-y_i\right) \times log\left(\left(1-p_i\right)\right) \quad (1)$$

如果 $y_i = 0$,则 $w_i = 1$.

优点：

- 模型简单，仅在训练时通过梯度乘以样本权重实现对其它目标的加权
- 模型上线简单，和base完全相同，不需要额外开销
- 在主目标没有明显下降时，其它目标提升较大(线上AB测试，主目标点击降低了1.5%，而衍生率提升了5%以上)

缺点：

- 本质上并不是多目标建模，而是将不同的目标转化为同一个目标。样本的加权权重需要根据AB测试才能确定。

2.2 多任务学习-Shared-Bottom Multi-task Model

模型结构如图1所示：

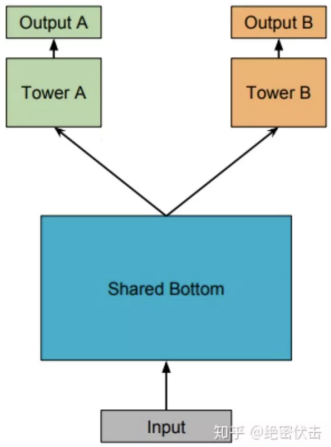

图1：Shared-Bottom Multi-task Model

Shared-Bottom 网络通常位于底部，表示为函数 f ，多个任务共用这一层。往上， K 个子任务分别对应一个 tower network，表示为 h^k ，每个子任务的输出为：

$$y_k = h^k(f(x))$$

优点：

- 浅层参数共享，互相补充学习，任务相关性越高，模型的loss可以降低到更低

缺点：

- 任务没有好的相关性时，这种Hard parameter sharing会损害效果

2.3 多任务学习-MOE

模型结构如图2所示：

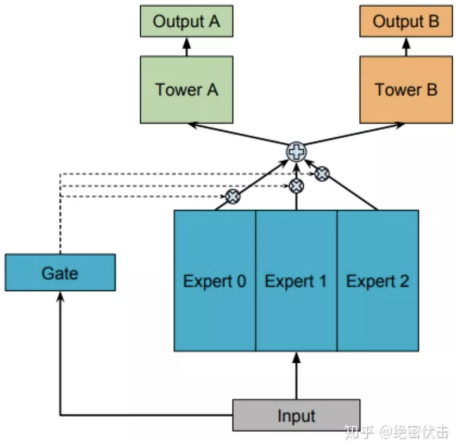

图2：MOE(One-gate Mixture-of-Experts)

前面的Shared-Bottom是一种Hard parameter sharing，会导致不相关任务联合学习效果不佳，为了解决这个问题，Google提出了Soft parameter sharing，MOE是其中的一种实现。

MOE由一组专家系统(Experts)组成的神经网络结构替换原来的Shared-Bottom部分，每一个Expert都是一个前馈神经网络，再加上一个门控网络(Gate)。MOE可以表示为：

$$y_k = h^k\left(f^k\left(x\right)\right) \tag{2}$$

$$f^k\left(x\right) = \sum_{i=1}^{n} g(x)_i f_i\left(x\right) \tag{3}$$

y_k 是第 k 个任务的输出， $f_i\left(i=1,..,n\right)$ 是 n 个expert network(expert network可认为是一个神经网络)， g 是门控网络，可以表示为：

$$g\left(x\right) = Softmax\left(W_g x\right) \tag{4}$$

可以看出 g 产生 n 个experts上的概率分布，最终的输出是所有experts的加权和。MOE可以看成多个独立模型的集成方法。

2.4 多任务学习-MMOE

MMOE(Multi-gate Mixture-of-Experts)是在MOE的基础上，使用了多个门控网络， k 个任就对应 k 个门控网络，模型结构如图3所示：

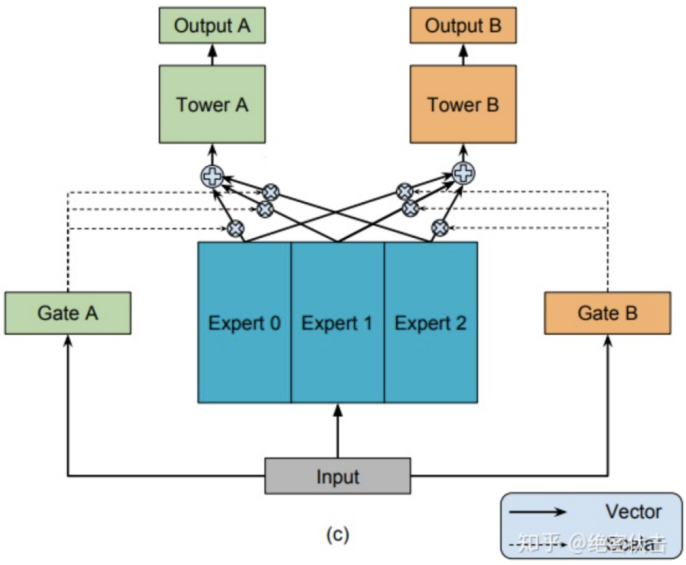

图3：MMOE(Multi-gate Mixture-of-Experts)

MMOE可以表示为：

$$f^k\left(x\right) = \sum_{i=1}^{n} g^k(x)_i f_i\left(x\right) \tag{5}$$

$$g^k\left(x\right) = Softmax\left(W_{gk} x\right) \tag{6}$$

其中， g^k 是第 k 个子任务中组合 experts 结果的门控网络，每一个任务都有一个独立的门控网络。

优点：

- MMOE是MOE的改进，相对于 MOE的结构中所有任务共享一个门控网络，MMOE的结构优化为每个任务都单独使用一个门控网络。这样的改进可以针对不同任务得到不同的 Experts 权重，从而实现对 Experts 的选择性利用，不同任务对应的门控网络可以学习到不同的Experts 组合模式，因此模型更容易捕捉到子任务间的相关性和差异性。

2.5 多任务学习-ESMM

ESMM(Entire Space Multi-Task Model)是针对任务依赖而提出，比如电商推荐中的多目标预估经常是ctr和cvr，其中转换这个行为只有在点击发生后才会发生。

cvr任务在训练时只能利用点击后的样本，而预测时，是在整个样本空间，这样导致训练和预测样本分布不一致，即样本选择性偏差。同时点击样本只占整个样本空间的很小比例，比如在新闻推荐中，点击率通常只有不到10%，即样本稀疏性问题。

为了解决这个问题，ESMM提出了转化公式：

$$p(z=1|y=1,x)=\frac{p(y=1,z=1|x)}{p(y=1|x)} \tag{8}$$

那么，我们可以通过分别估计pctcvr(即 $p(y=1,z=1|x)$)和pctr(即 $p(y=1|x)$)，然后通过两者相除来解决。而pctcvr和pctr都可以在全样本空间进行训练和预估。但是这种除法在实际使用中，会引入新的问题。因为pctr其实是一个很小的值，预估时会出现pctcvr>pctr的情况，导致pcvr预估值大于1。ESSM巧妙的通过将除法改成乘法来解决上面的问题。它引入了pctr和pctcvr两个辅助任务，训练时，loss为两者相加。

模型的Loss为：

$$L(\theta_{cvr},\theta_{ctr})=\sum_{i=1}^{N}l(y_i,f(x_i;\theta_{ctr}))+\sum_{i=1}^{N}l(y_i\&z_i,f(x_i;\theta_{ctr})\times f(x_i;\theta_{cvr})) \tag{9}$$

其中 θ_{ctr} 和 θ_{cvr} 分别是ctr和cvr任务的网络参数。这样模型可以同时得到pctr,pcvr,pctcvr三个任务的预估结果。模型结构如图4所示：

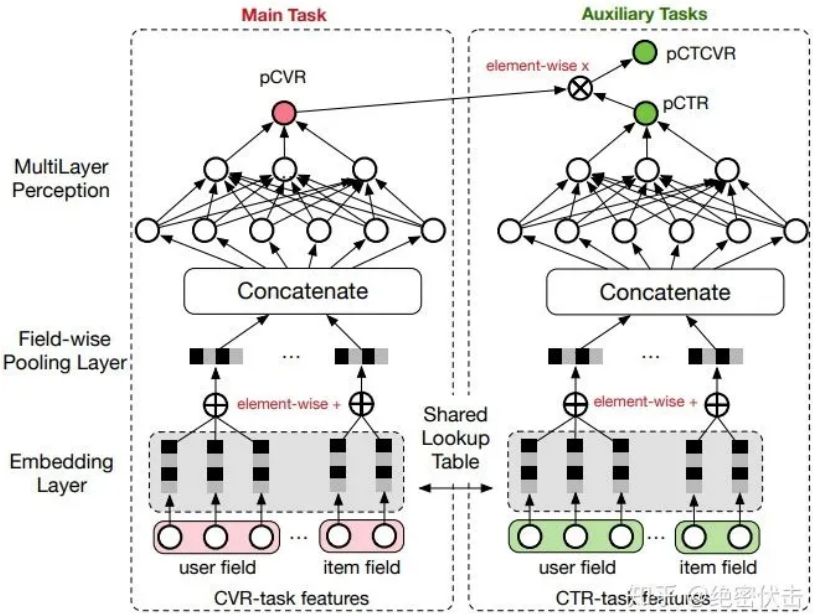

图4：ESMM模型结构

三、实践方案

具体的实践中，我们主要参考了腾讯的PLE(Progressive Layered Extraction)模型，PLE相对于前面的MMOE和ESMM，主要解决以下问题：

> 多任务学习中往往存在跷跷板现象，也就是说，多任务学习相对于多个单任务学习的模型，往往能够提升一部分任务的效果，同时牺牲另外部分任务的效果。即使通过MMoE这种方式减轻负迁移现象，跷跷板现象仍然是广泛存在的。

前面的MMOE模型存在以下两方面的缺点

- MMOE中所有的Expert是被所有任务所共享的，这可能无法捕捉到任务之间更复杂的关系，从而给部分任务带来一定的噪声
- 不同的Expert之间没有交互，联合优化的效果有所折扣

PLE针对上面第一个问题，每个任务有独立的Expert，同时保留了共享的Expert，模型结构如图5所示：

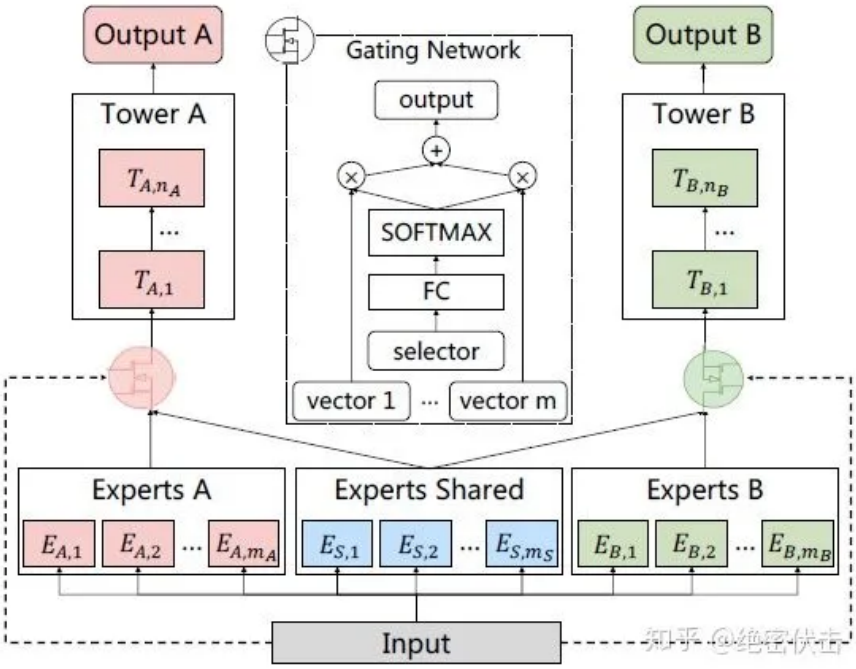

图5：Customized Gate Control (CGC) Model

图中ExpertsA和ExpertsB是任务A和B各自的专家系统，中间的Experts Shared是共享的专家系统。图中的selector表示选择的专家系统。对于任务A，使用Experts A和Experts Shared里面的多个Expert的输出。

任务 k 的输出可以表示为：

$$y^k(x) = t^k\left(g^k(x)\right) \tag{10}$$

其中， t^k 表示任务 k 的tower network， g^k 是门控网络，可以表示为：

$$g^k(x) = w^k(x)\,S^k(x) \tag{11}$$

其中 w^k 是选择专家系统 S^k 中所有Expert的权重，可以表示为：

$$w^k(x) = Softmax\left(W_g^k x\right) \tag{12}$$

其中 $W_g^k \in R^{(m_k+m_s)\times d}$ ， m_s 和 m_k 分别是共享Experts个数以及任务 k 独有Experts个数， d 是输入维度。

S^k 由共享Experts和任务 k 的Experts组成，可以表示为：

$$S^k(x) = \left[E_{(k,1)}^T, E_{(k,2)}^T, .., E_{(k,m_k)}^T, E_{(s,1)}^T, E_{(s,2)}^T, .., E_{(s,m_s)}^T\right]^T \tag{13}$$

PLE针对前面的第二个问题，考虑了不同Expert之前的交互，模型结构如图6所示：

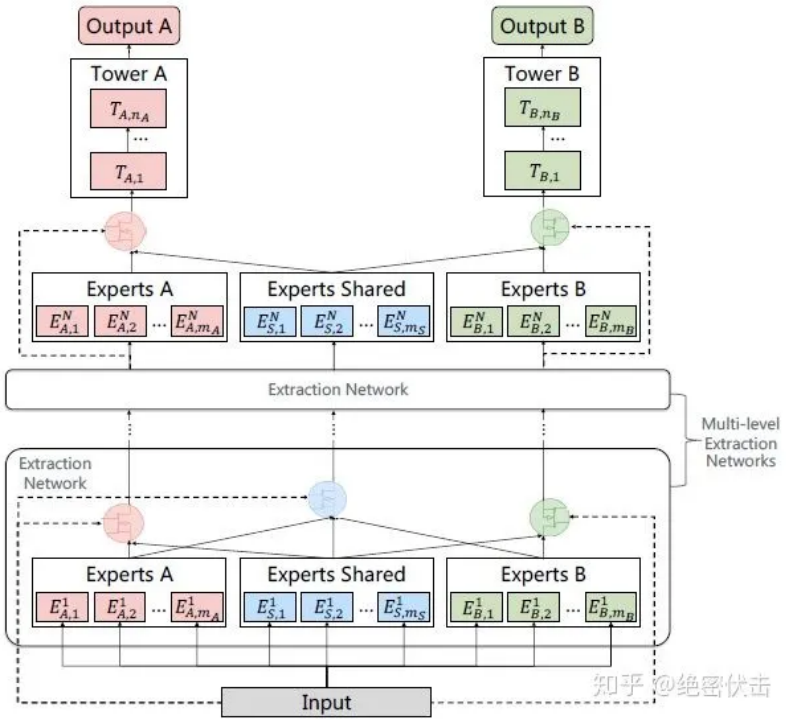

图6：Progressive Layered Extraction (PLE) Model

PLE中第 j 层的输出表示为：

$$g^{k,j}\left(x\right)=w^{k,j}\left(g^{k,j-1}\left(x\right)\right)S^{k,j} \tag{14}$$

这里面， $S^{k,j}$ 包含两部分，可以表示为：

$$S^{k,j}=\left[E^{T}_{(k,1)}\left(g^{k,j-1}\right),..,E^{T}_{(k,m_k)}\left(g^{k,j-1}\right),E^{T}_{(s,1)}\left(g^{j-1}_{shared}\right),..,E^{T}_{(s,m_s)}\left(g^{j-1}_{shared}\right)\right]^{T}$$

其中 $E^{T}_{(k,1)}\left(g^{k,j-1}\right)$ 表示第 j 层Experts k 的输入为 $g^{k,j-1}$ ，而 $E^{T}_{(s,1)}\left(g^{k,j-1}_{shared}\right)$ 表示Experts Shared的输入是 g^{j-1}_{shared} ， g^{j-1}_{shared} 表示共享部分的gating network，这部分gating network的输入(selector)包含了所有的Experts(即包含Experts A,Experts B和Experts Shared)，可以表示为： $g^{j}_{shared}\left(x\right)=w^{j}_{shared}\left(g^{j-1}_{shared}\left(x\right)\right)S^{j}_{all}$ ，这里面 S_{all} 就是所有的Experts。

最终每个任务的输出表示为：

$$y^{k}\left(x\right)=t^{k}\left(g^{k,N}\left(x\right)\right) \tag{15}$$

下面是PLE用tensorflow的一个简单实现，只考虑两个任务。

```
def multi_level_extraction_network(
        hidden_layer,
        num_level,
        experts_units,
        experts_num):
    """
```

```
    :param hidden_layer:
    :param num_level:
    :param experts_units:
    :param experts_num:
    :return:
    """
    gate_output_task1_final = hidden_layer
    gate_output_task2_final = hidden_layer
    gate_output_shared_final = hidden_layer
    selector_num = 2
    for i in range(num_level):
        # experts shared
        experts_weight = tf.get_variable(
            name='experts_weight_%d' % (i),
            dtype=tf.float32,
            shape=(gate_output_shared_final.get_shape()[1], experts_units, e
            initializer=init_ops.glorot_uniform_initializer()
        )

        experts_bias = tf.get_variable(
            name='expert_bias_%d' % (i),
            dtype=tf.float32,
            shape=(experts_units, experts_num),
            initializer=init_ops.glorot_uniform_initializer()
        )

        # experts Task 1
        experts_weight_task1 = tf.get_variable(
            name='experts_weight_task1_%d' % (i),
            dtype=tf.float32,
            shape=(gate_output_task1_final.get_shape()[1], experts_units, ex
            initializer=init_ops.glorot_uniform_initializer()
        )

        experts_bias_task1 = tf.get_variable(
            name='expert_bias_task1_%d' % (i),
            dtype=tf.float32,
            shape=(experts_units, experts_num),
            initializer=init_ops.glorot_uniform_initializer()
        )
```

```
47
48          # experts Task 2
49          experts_weight_task2 = tf.get_variable(
50              name='experts_weight_task2_%d' % (i),
51              dtype=tf.float32,
52              shape=(gate_output_task2_final.get_shape()[1], experts_units, ex
53              initializer=init_ops.glorot_uniform_initializer()
54          )
55
56          experts_bias_task2 = tf.get_variable(
57              name='expert_bias_task2_%d' % (i),
58              dtype=tf.float32,
59              shape=(experts_units, experts_num),
60              initializer=init_ops.glorot_uniform_initializer()
61          )
62
63          # gates shared
64          gate_shared_weigth = tf.get_variable(
65              name='gate_shared_%d' % (i),
66              dtype=tf.float32,
67              shape=(gate_output_shared_final.get_shape()[1], experts_num * 3)
68              initializer=init_ops.glorot_uniform_initializer()
69          )
70          gate_shared_bias = tf.get_variable(
71              name='gate_shared_bias_%d' % (i),
72              dtype=tf.float32,
73              shape=(experts_num * 3,),
74              initializer=init_ops.glorot_uniform_initializer()
75          )
76
77          # gates Task 1
78          gate_weight_task1 = tf.get_variable(
79              name='gate_weight_task1_%d' % (i),
80              dtype=tf.float32,
81              shape=(gate_output_task1_final.get_shape()[1], experts_num * se
82              initializer=init_ops.glorot_uniform_initializer()
83          )
84          gate_bias_task1 = tf.get_variable(
85              name='gate_bias_task1_%d' % (i),
86              dtype=tf.float32,
```

```
            shape=(experts_num * selector_num,),
            initializer=init_ops.glorot_uniform_initializer()
        )

        # gates Task 2
        gate_weight_task2 = tf.get_variable(
            name='gate_weight_task2_%d' % (i),
            dtype=tf.float32,
            shape=(gate_output_task2_final.get_shape()[1], experts_num * sel
            initializer=init_ops.glorot_uniform_initializer()
        )
        gate_bias_task2 = tf.get_variable(
            name='gate_bias_task2_%d' % (i),
            dtype=tf.float32,
            shape=(experts_num * selector_num,),
            initializer=init_ops.glorot_uniform_initializer()
        )

        # experts shared outputs
        experts_output = tf.tensordot(gate_output_shared_final, experts_weig
        experts_output = tf.add(experts_output, experts_bias)
        experts_output = tf.nn.relu(experts_output)

        # experts Task1 outputs
        experts_output_task1 = tf.tensordot(gate_output_task1_final, experts
        experts_output_task1 = tf.add(experts_output_task1, experts_bias_tas
        experts_output_task1 = tf.nn.relu(experts_output_task1)

        # experts Task2 outputs
        experts_output_task2 = tf.tensordot(gate_output_task2_final, experts
        experts_output_task2 = tf.add(experts_output_task2, experts_bias_tas
        experts_output_task2 = tf.nn.relu(experts_output_task2)

        # gates Task1 outputs
        gate_output_task1 = tf.matmul(gate_output_task1_final, gate_weight_t
        gate_output_task1 = tf.add(gate_output_task1, gate_bias_task1)
        gate_output_task1 = tf.nn.softmax(gate_output_task1)
        gate_output_task1 = tf.multiply(
            concat_fun([experts_output_task1, experts_output], axis=2),
            tf.expand_dims(gate_output_task1, axis=1)
```

```
        )
        gate_output_task1 = tf.reduce_sum(gate_output_task1, axis=2)
        gate_output_task1 = tf.reshape(gate_output_task1, [-1, experts_units
        gate_output_task1_final = gate_output_task1

        # gates Task2 outputs
        gate_output_task2 = tf.matmul(gate_output_task2_final, gate_weight_t
        gate_output_task2 = tf.add(gate_output_task2, gate_bias_task2)
        gate_output_task2 = tf.nn.softmax(gate_output_task2)
        gate_output_task2 = tf.multiply(
            concat_fun([experts_output_task2, experts_output], axis=2),
            tf.expand_dims(gate_output_task2, axis=1)
        )
        gate_output_task2 = tf.reduce_sum(gate_output_task2, axis=2)
        gate_output_task2 = tf.reshape(gate_output_task2, [-1, experts_units
        gate_output_task2_final = gate_output_task2

        # gates shared outputs
        gate_output_shared = tf.matmul(gate_output_shared_final, gate_shared
        gate_output_shared = tf.add(gate_output_shared, gate_shared_bias)
        gate_output_shared = tf.nn.softmax(gate_output_shared)
        gate_output_shared = tf.multiply(
            concat_fun([experts_output_task1, experts_output, experts_output
            tf.expand_dims(gate_output_shared, axis=1)
        )
        gate_output_shared = tf.reduce_sum(gate_output_shared, axis=2)
        gate_output_shared = tf.reshape(gate_output_shared, [-1, experts_uni
        gate_output_shared_final = gate_output_shared


    return gate_output_task1_final, gate_output_task2_final
```

四、PLE训练优化

4.1 联合训练(Joint Training)

联合训练方式如下图所示：

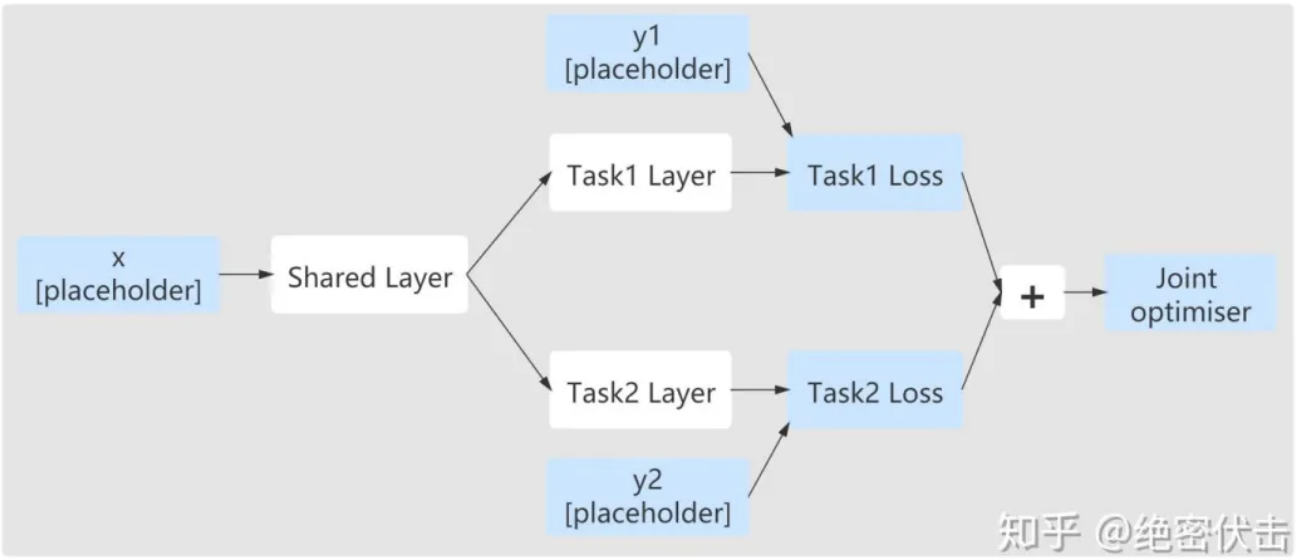

图7：联合训练(Joint Training)

可以看出最终将每个任务的Loss加权和合并成一个Loss，使用一个优化器训练，tensorflow里面可以表示为：

```
final_loss = tf.reduce_mean(loss1 + loss2)
train_op = tf.train.AdamOptimizer().minimize(final_loss)
```

联合训练比较适合在同一数据集进行训练，使用同一feature，但是不同任务输出不同结果，比如前面的pctr和pctcvr任务。

最开始上线的版本中，使用联合训练的方式，并且ctr和ysl两个任务的Loss系数都是1，后来考虑到ysl的均值是4左右，而ctr的均值不到0.2，模型会偏向于ysl。但是手动调节权值非常耗时，考虑使用UWL(Uncertainty to Weigh Losses)，优化不同任务的权重系数。

对于ysl回归任务，定义其取值的概率服从以 $f^W(x)$ 为均值的高斯分布，即：

$$p\left(y|f^W(x)\right) = N\left(f^W(x), \sigma^2\right) \tag{16}$$

对于ctr分类任务，其取值概率为：

$$p\left(y|f^W(x)\right) = Softmax\left(f^W(x)\right) \tag{17}$$

其中 $f^W(x)$ 为PLE的输出。

多任务模型，似然函数为：

$$p\left(y_1, y_2, .., y_k|f^W(x)\right) = p\left(y_1|f^W(x)\right) p\left(y_2|f^W(x)\right) .. p\left(y_k|f^W(x)\right) \tag{18}$$

对于回归任务，对数似然为：

$$log\, p\left(y|f^W(x)\right) \propto -\frac{1}{2\sigma^2}\left\|y - f^W(x)\right\|^2 - log\sigma \tag{19}$$

对于分类任务，添加一个缩放系数 σ^2 ：

$$p\left(y|f^W\left(x\right),\sigma^2\right)=Softmax\left(\frac{1}{\sigma^2}f^W\left(x\right)\right) \tag{20}$$

对数似然表示为：

$$log\,p\left(y=c|f^W\left(x\right),\sigma^2\right)=\frac{1}{\sigma^2}f_c^W\left(x\right)-log\left(\sum_{c'}exp\left(\frac{1}{\sigma^2}f_{c'}^W\left(x\right)\right)\right) \tag{21}$$

分类和回归任务的联合Loss表示为：

$$\begin{aligned}Loss&=-\log p\left(y_1,y_2=c|f^W\left(x\right)\right)\\&=-\log N\left(y_1;f^W\left(x\right),\sigma_1^2\right).Softmax\left(y_2=c;f^W\left(x\right),\sigma_2\right)\\&=\frac{1}{2\sigma_1^2}\left\|y-f^W\left(x\right)\right\|^2+\log\sigma_1-\log p\left(y_2=c|f^W\left(x\right),\sigma_2\right)\\&=\frac{1}{2\sigma_1^2}L_1\left(w\right)+\frac{1}{\sigma_2^2}L_2\left(w\right)+\log\sigma_1+\log\frac{\sum_{c'}exp\left(\frac{1}{\sigma_2^2}f_{c'}^W\left(x\right)\right)}{\left(\sum_{c'}exp\left(f_{c'}^W\left(x\right)\right)\right)^{\frac{1}{\sigma_2^2}}}\\&\approx\frac{1}{2\sigma_1^2}L_1\left(w\right)+\frac{1}{\sigma_2^2}L_2\left(w\right)+\log\sigma_1+\log\sigma_2\end{aligned} \tag{22}$$

在具体实现时，设 $s_1=\log\sigma_1^2,s2=\log\sigma_2^2$ ，则Loss可以表示为：

$$Loss=exp\left(-s1\right)\times L_1\left(w\right)+2\times exp\left(-s2\right)\times L_2\left(w\right)+s_1+s_2 \tag{23}$$

其中 L_1 是回归任务， L_2 是分类任务。tensorflow可以表示为：

```
## combine loss
ctr_log_var = tf.get_variable(
    name='ctr_log_var',
    dtype=tf.float32,
    shape=(1,),
    initializer=tf.zeros_initializer()
)
ysl_log_var = tf.get_variable(
    name='ysl_log_var',
    dtype=tf.float32,
    shape=(1,),
    initializer=tf.zeros_initializer()
)
loss_final = 2 * loss_ctr * tf.exp(-ctr_log_var) + loss_ysl * tf.exp(-ysl_log
```

模型迭代过程中，权重的变化曲线如下图所示：

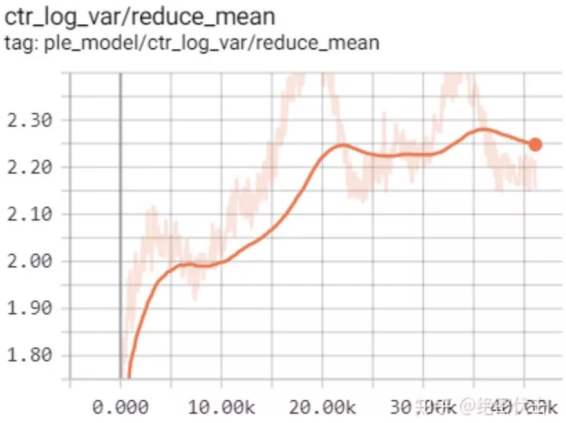

图8：ctr任务loss权重

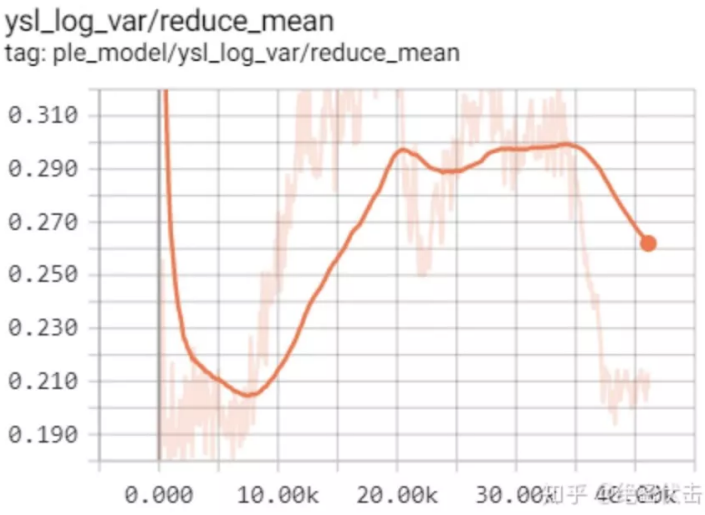

图9：ysl回归任务loss权重

对于不同的量纲，模型学习出的ctr权重系数会高于ysl，最后收敛到一个合理值范围。

4.2 交替训练(Alternative Training)

训练方式如下图所示：

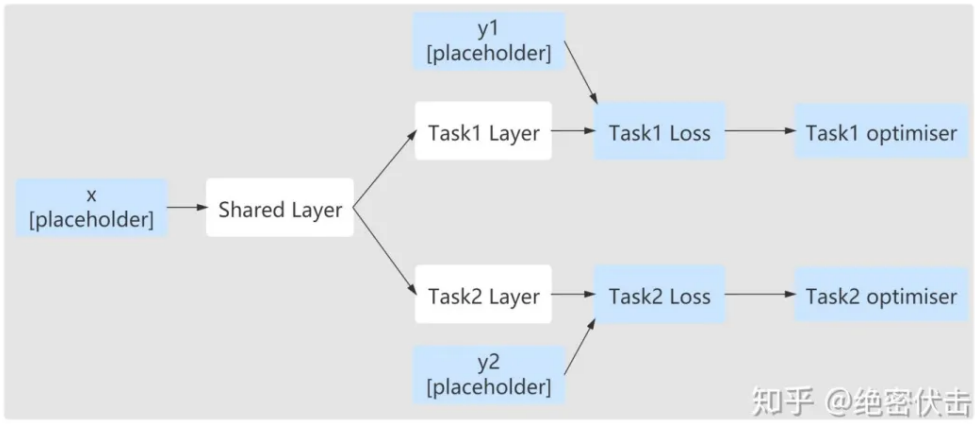

图10：交替训练(Alternative Training)

可以看出两个任务有各自的优化器，tensorflow可以表示为：

```
train_op1 = tf.train.AdamOptimizer().minimize(loss1)
train_op2 = tf.train.AdamOptimizer().minimize(loss2)
final_train_op = tf.group(train_op1 train_op2)
```

Alternative Training在训练任务A时，不会影响任务B的Tower，同样训练任务B不会影响任务A的Tower，这样就避免了如果任务A的Loss降低到很小，训练任务B时影响任务A的Tower，以及学习率的影响。

Alternative Training比较适合在不同的数据集上输出多个目标，多个任务不使用相同的feature，比如WDL模型，Wide侧和Deep侧用的特征不一样，使用的就是Alternative Training，Wide侧用的是FTRL优化器，Deep侧用的是Adagrad或者Adam。

tensorflow中WDL的Alternative Training实现如下：

```
def _train_op_fn(loss):
    """Returns the op to optimize the loss."""
    train_ops = []
    global_step = training_util.get_global_step()
    if dnn_logits is not None:
      train_ops.append(
          dnn_optimizer.minimize(
              loss,
              var_list=ops.get_collection(
                  ops.GraphKeys.TRAINABLE_VARIABLES,
                  scope=dnn_absolute_scope)))
    if linear_logits is not None:
      train_ops.append(
          linear_optimizer.minimize(
              loss,
              var_list=ops.get_collection(
                  ops.GraphKeys.TRAINABLE_VARIABLES,
                  scope=linear_absolute_scope)))
```

可以看出分别使用了dnn_optimizer和linear_optimizer两个优化器。

使用Alternative Training，两个任务拥有各自的学习率等信息。如果存在有的loss的返回值远小于其他loss的情况，这种训练方式比较有优势。后面会在实际应用中对比下两种训练方式的效果。

五、离线实验对比

分两组实验，一组实验对ysl做了平滑，取 $ysl = \log(1 + ysl)$ ，一组不做平滑。

Alternative Training训练时，CTR任务优化器更新共享参数的所有部分，包括特征的embedding，共享专家系统等。ysl任务优化器只更新ysl专家系统、门控网络以及Tower ysl，如图11所示：

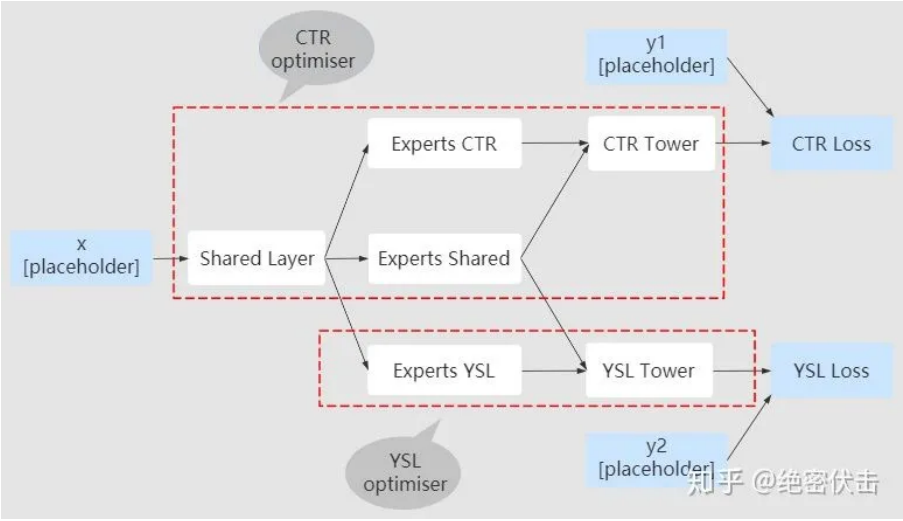

图11：Alternative Training优化器

如图11所示，Alternative Training中，CTR任务和Single-Task CTR是一个效果，YSL任务不更新共享参数。

第一组实验结果如表2所示。

实验	ysl样本空间	训练方式	Loss加权系数	AUC CTR	logloss CTR
1	全语料	Alternative Train		0.7250	0.4492
2	全语料	Joint Train	CTR: 1, YSL: 1	0.7336	0.4462
3	全语料	Joint Train	UWL自动加权	0.7328	0.4473
4	点击语料	Alternative Train		0.7250	0.4492
5	点击语料	Joint Train	CTR: 1, YSL: 1	0.7290	0.4484
6	点击语料	Joint Train	UWL自动加权	0.7316	0.4479

表2：不同训练方式实验对比(ysl平滑：ysl=log(1+ysl))

从表2可以得出以下两个结论：

- ysl平滑后，UWL和Loss直接相加效果相当，主要也是因为两个Loss的均值很接近
- 两个任务样本空间不一致时，Joint Training主任务效果会有下降

ysl不做平滑，实验结果如图3所示：

实验	ysl样本空间	训练方式	Loss加权系数	AUC CTR	logloss CTR
1	全语料	Alternative Train		0.7250	0.4492
2	全语料	Joint Train	CTR: 1, YSL: 1	0.7214	0.4526
3	全语料	Joint Train	UWL自动加权	0.7327	0.4469
4	点击语料	Alternative Train		0.7250	0.4492
5	点击语料	Joint Train	CTR: 1, YSL: 1	0.7075	0.4581
6	点击语料	Joint Train	UWL自动加权	0.7292	0.4481

表3：不同训练方式实验对比

从表3可以得出以下结论：

- 不同任务Loss相差很大时，UWL会比直接Loss加和效果好

汇总表2和表3，得出以下结论：

- 两个任务样本空间不一致时，Joint Training主任务效果会有下降
- 不同任务Loss相差很大时，UWL会比直接Loss加和效果好
- 使用Joint Training，对Loss大的任务做平滑，效果会更好
- Alternative Training在训练主任务时，效果和Single-Task一样，和其它任务训练完全独立

六、参考文献

1. Entire Space Multi-Task Model: An Effective Approach for Estimating Post-Click Conversion Rate
2. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts
3. Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations
4. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics
5. Multi-Task Learning in Tensorflow (Part 1)
6. 腾讯 at RecSys2020最佳长论文 - 多任务学习模型PLE
7. yymWater：详解谷歌之多任务学习模型MMoE(KDD 2018)
8. 多目标学习在推荐系统中的应用
9. 鱼罐头啊：从谷歌到阿里，谈谈工业界推荐系统多目标预估的两种范式

关于深度传送门