多任务学习

一、MMOE[2018]

1. 近年来，深度神经网络模型已成功应用于很多现实世界的大规模应用程序`application`中，例如推荐系统。这些推荐系统通常需要同时优化多个目标。例如，当给用户推荐电影时，我们希望用户不仅“购买”和“观看”电影，还希望用户能够喜欢这些电影，从而吸引用户回头观看更多的电影。即，我们可以创建模型来同时预测用户的购买情况和他们的评分情况。

   事实上，很多大规模推荐系统已经采用了基于深度神经网络模型的多任务学习。研究人员表示：多任务学习模型可以利用正则化和迁移学习来改善所有任务的模型预测。

   但是在实践中，多任务学习模型并不总是在所有任务上都超越相应的单任务模型。实际上很多基于`DNN`的多任务学习模型对诸如数据分布差异和任务之间关系之类的因素很敏感。来自任务差异的固有冲突`inherent conflict`实际上会损害至少一部分任务的预测，尤其是当模型参数在所有任务之间广泛共享时。因此，重要的是研究特定任务目标`task-specific objective`和任务间关系`inter-task relationship`之间建模的权衡`tradeoff`。

   早期的工作研究了多任务学习中的任务差异`task difference`，方法是为每个任务假设特定的数据生成过程，并根据假设来度量`measuring`任务差异，然后根据任务的差异来提出建议。然而，由于实际应用程序通常具有更复杂的数据模式，因此很难度量任务差异并利用这些工作中建议的方法。

   最近的一些工作提出了新的建模技术来处理多任务中的任务差异，而不依赖于显式的任务差异度量。但是，这些技术通常涉及为每个任务添加更多的模型参数，从而适应任务差异。由于大规模推荐系统可能包含数百万或数十亿个参数，这些额外的参数可能是欠约束的`under-constrained`（即无法得到充分学习），这可能会损害模型质量。另外，由于`serving`资源有限，这些参数的额外计算成本在实际生产环境中通常也是不可实现的。

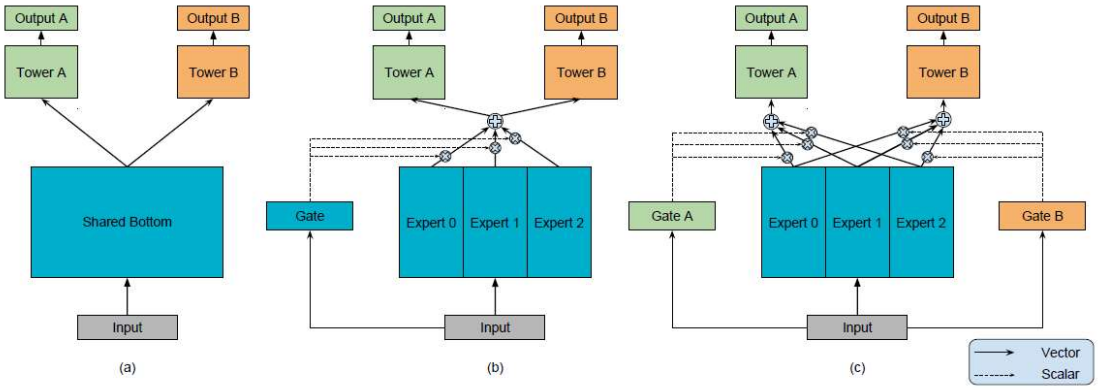
   在论文`《Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts》`中，我们提出了一种新颖的、基于`Multi-gate Mixture-of-Expert: MMoE`结构的多任务学习方法，该方法受到`Mixture-of-Expert: MoE`模型和最近的`MoE layer`的启发。

   `MMoE`显式地对任务关系`task relationship`建模，并学习`task-specific`函数以利用共享的`representation`。它允许自动分配参数来捕获共享的任务信息`shared task information`或特定于任务的信息`task-specific information`，而无需为每个任务添加许多新参数。

   `MMoE`的骨干基于最常用的共享底部`Shared-Bottom`多任务`DNN`架构。`Shared-Bottom`模型架构如图`(a)`所示，其中输入层之后的几个底层`bottom layer`在所有任务之间共享，然后每个任务在`bottom representation`的顶部都有一个独立的`tower`网络。

   - `MMoE`模型不是所有任务共享一个底部网络`bottom network`，而是有一组`bottom network`，如图`(c)`所示。每个`bottom network`称作一个专家子模型`expert submodel`。论文中每个专家都是前馈神经网络。
   - 然后我们为每个任务引入一个门控网络`gating network`。门控网络馈入输入特征`input feature`，并输出`softmax`的门控权重，并以这些权重来集成`assembling`所有的专家。
   - 集成专家的结果被传递到任务特定`task-specific`的`tower`网络中。通过这种方式，不同任务的门控网络可以学习专家集成的不同混合模式，从而捕获任务关系。

   以这种方式，我们允许不同的任务以不同的方式利用专家。



为了理解 MMoE 是如何为不同级别 level 任务相关性来学习它的专家和任务门控网络 task gating network 的，我们进行了一项人工合成实验 synthetic experiment 。在该实验中，我们可以通过任务之间的皮尔逊相关系数 Pearson correlation 来度量和控制任务相关性。我们使用两个人工合成回归任务，并使用正弦函数作为数据生成机制来引入非线性。

- 实验表明：MMoE 优于 baseline 方法，尤其是当任务相关性较低的情况下。
- 在这组实验的多次运行中，我们还发现 MMoE 更容易训练（即可训练性 trainability ），并且收敛到一个更低的 loss 。这与最近的发现有关，即调制 modulation 和门控机制可以提高训练非凸深度神经网络的可训练性 trainability 。

我们进一步评估了 MMoE 在 benchmark 数据集（UCI Census-income 数据集）上的性能，该数据集具有多任务的配置。我们比较了几种 state-of-the-art 的多任务模型，这些模型通过软参数共享 soft parameter sharing 来建模任务关系，并观察到 MMOE 方法的提升。

最后，我们在一个真实的大规模内容推荐系统上测试 MMoE 。在这个系统中，当向用户推荐 item 时，我们同时学习两个分类任务。我们用数千亿个训练样本来训练 MMoE 模型，并将其与 Shared-Bottom 的生产模型 production model 进行比较。我们观察到离线指标（如 AUC ）有显著提升。此外，我们的 MMoE 模型在在线实验中不断提升在线指标。

通过在 benchmark 数据集和真实的大型推荐系统上进行的实验，我们证明了 MMoE 方法在几种 state-of-the-art 多任务学习模型上的成功。

除了上述效果上的优点之外，实际机器学习生产系统的另一个主要设计因素是计算效率，这也是 Shared-Bottom 多任务模型被广泛使用的最重要原因之一。该模型的共享部分在 serving 时节省了大量的计算。 MMoE 模型在很大程度上保留了计算优势，因为门控网络通常都是轻量级的，而专家网络在所有任务中共享。此外，通过将门控网络设计为稀疏的 top-k 门， MMoE 模型有可能实现更好的计算效率。

论文贡献：

- 首先，我们提出了一种新颖的 Multi-gate Mixture-of-Expert: MMOE 模型，该模型显式地对任务关系进行建模。通过调制 modulation 和门控 gating 网络，我们的模型在建模共享信息 shared information 和建模任务特定信息 task-specific information 之间自动地调整。
- 其次，我们对人工合成数据进行控制实验。我们报告了任务相关性 task relatedness 如何影响多任务学习中的训练动态 training dynamics ，以及 MMoE 如何提高模型表达能力和可训练性 trainability 。
- 最后，我们对真实的 benchmark 数据集、以及具有数亿用户和 item 的大型生产推荐系统进行了实验。我们的实验验证了我们提出的方法在现实环境中的效率 efficiency 和效果 effectiveness 。

2. 相关工作：

- DNN 中的多任务学习：多任务模型可以学习不同任务的共性 commonalities 和差异 differences 。这样做可以提高每个任务的效率和模型质量。

- 《Multitask learning》和《Multitask learning: A knowledge-based source of inductive bias》提出了一种广泛使用的多任务学习模型，它具有共享底部 shared-bottom 模型结构，其中底部隐层在任务之间共享。这种结构大大降低了过拟合的风险，但是可能会因为任务差异导致优化冲突 optimization conflicts，因为所有任务都需要在共享底层上使用相同的 parameters。
- 为了了解任务相关性如何影响模型质量，早期的工作生成不同的任务相关性的人工合成数据，从而评估多任务模型的有效性。
- 最近的一些方法不是在任务之间共享隐层和相同的模型参数，而是对特定任务的参数 task-specific parameters 添加了不同类型的约束。

  例如，对于两个任务，《Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser》在两组参数之间添加了 L2 约束。

  《Cross-stitch networks for multi-task learning》为每个任务学习 task-specific 隐层 embedding 的 unique combination。

  《Deep multi-task representation learning: A tensor factorisation approach》使用张量分解模型为每个任务生成隐层参数。

  与 shared-bottom 模型相比，这些方法具有更多的 task-specific 参数，并且在任务差异导致共享参数更新的冲突时，可以获得更好的性能。但是，大量 task-specific 参数需要更多的训练数据来拟合，并且在大模型中可能效率不高。

- 子网集成 subnetwork ensemble & 专家混合 expert mixture：在本文中，我们应用了深度学习中的一些最新发现，例如参数调制 parameter modulation 和集成方法 ensemble method 来为多任务学习的任务关系建模。在 DNN 中，集成模型 ensemble model 和子网络集成 subnetwork ensemble 已被证明能够提高模型性能。
  - 《Learning factored representations in a deep mixture of experts》和《Outrageously large neural networks: The sparsely-gated mixture-of-experts layer》将 mixture-of-experts: MOE 模型转换为基础构建块（MoE layer），并将它们堆叠在 DNN 中。MoE layer 在训练训练期间和推断期间，根据 layer input 来选择子网（即 expert）。因此，该模型不仅在建模方面更强大，而且通过将稀疏性引入门控网络来降低计算成本。
  - 类似地，PathNet 是为通用人工智能处理不同任务而设计的，是一个巨大的神经网络，具有多层、以及每层内多个子模块。在为每个任务训练时，多条路径随机选择并由不同的 workers 并行训练。最佳路径的 parameters 是固定的，并选择新的路径来训练新任务。

  我们从这些工作中获取灵感，通过使用子网（即 expert）的集成来实现迁移学习，同时节省计算成本。

- 多任务 Application：由于分布式机器学习系统的发展，许多大规模的现实世界 application 都采用了 DNN-based 多任务学习算法，并观察到了实质性的质量提升。
  - 在多语言机器翻译任务上，通过参数共享，训练数据有限的翻译任务（例如小语种翻译任务）可以通过与拥有大量训练数据的其它翻译任务联合学习来改进。
  - 在推荐任务上，多任务学习有助于提升上下文感知推荐。
  - 在《Ask the gru: Multitask learning for deep text recommendations》中，通过共享 feature representations 和 lower level hidden layers 可以改进文本推荐任务。
  - 在《Deep neural networks for youtube recommendations》中，shared-bottom 模型用于学习视频推荐的 ranking 算法。

与这些早先的工作类似，我们在现实世界的大规模推荐系统上评估了我们的建模方法。我们证明了我们的方法具有可扩展性，并且与其它 `state-of-the-art` 建模方法相比具有良好的性能。

1.1 人工合成数据生成

1. 先前的工作表明：多任务学习模型的性能高度依赖于数据中固有的任务相关性。然而，实际应用中很难直接研究任务相关性如何影响多任务模型，因为在实际应用中我们无法轻易改变任务之间的相关性并观察其效果。为了对这种影响建立实证研究，我们首先使用人工合成数据，因为这样我们可以轻松地度量和控制任务相关性。

   受到 `Kang` 等人的启发，我们生成了两个回归任务，并使用这两个任务 `label` 的 `Pearson correlation` 作为任务相关性的定量指标。由于我们关注于 `DNN` 模型（而不是线性模型），所以我们将回归模型设置为正弦函数的组合。

   具体而言，我们通过以下步骤来生成人工合成数据：

   - 给定输入特征向量的维度 d ，我们随机生成两个正交的单位向量 $\vec{\mathbf{u}}_1, \vec{\mathbf{u}}_2 \in \mathbb{R}^d$ ，即：

     $$\vec{\mathbf{u}}_1^\top \vec{\mathbf{u}}_2 = 0, \quad \|\vec{\mathbf{u}}_1\|_2 = \|\vec{\mathbf{u}}_2\|_2 = 1$$

   - 给定一个缩放常量 c 、以及一个相关系数分 `correlation score` $-1 \le p \le 1$，生成两个权重向量 $\vec{\mathbf{w}}_1, \vec{\mathbf{w}}_2$，使得：

     $$\vec{\mathbf{w}}_1 = c\vec{\mathbf{u}}_1, \quad \vec{\mathbf{w}}_2 = c\left(p \times \vec{\mathbf{u}}_1 + \sqrt{(1-p^2)} \times \vec{\mathbf{u}}_2\right)$$

   - 随机采样一个输入样本 $\vec{\mathbf{x}} \in \mathbb{R}^d$ ，其中每个元素来自于标准正态分布 $\mathcal{N}(0,1)$ 。

   - 为两个回归任务生成两个标签 y_1, y_2：

     $$y_1 = \vec{\mathbf{w}}_1^\top \vec{\mathbf{x}} + \sum_{i=1}^{m} \sin\left(\alpha_i \vec{\mathbf{w}}_1^\top \vec{\mathbf{x}} + \beta_i\right) + \epsilon_1$$

     $$y_2 = \vec{\mathbf{w}}_2^\top \vec{\mathbf{x}} + \sum_{i=1}^{m} \sin\left(\alpha_i \vec{\mathbf{w}}_2^\top \vec{\mathbf{x}} + \beta_i\right) + \epsilon_2$$

     其中：

     - $\alpha_1, \beta_i, i = 1, \cdots, m$ 是给定的参数，它们控制了正弦函数的形状。
     - ϵ_1, ϵ_2 都是独立同分布的随机变量，它们来自于正态分布 $\mathcal{N}(0, 0.01)$ 。

   - 重复采样 $\vec{\mathbf{x}}$ 和 y_1, y_2 ，直到生成足量的数据。

2. 由于采用非线性数据生成过程，因此要生成具有给定皮尔逊相关系数 `label` 的任务并非易事。相反，我们控制权重向量的余弦相似度，即 $\cos(\vec{\mathbf{w}}_1, \vec{\mathbf{w}}_2) = p$ ，然后度量结果 `label` 的皮尔逊相关系数。

   注意到在线性情况下：

   $$y_1 = \vec{\mathbf{w}}_1^\top \vec{\mathbf{x}} + \epsilon_1, \quad y_2 = \vec{\mathbf{w}}_2^\top \vec{\mathbf{x}} + \epsilon_2$$

   y_1, y_2 的标签皮尔逊相关系数 `label Pearson correlation` 刚好是 p 。

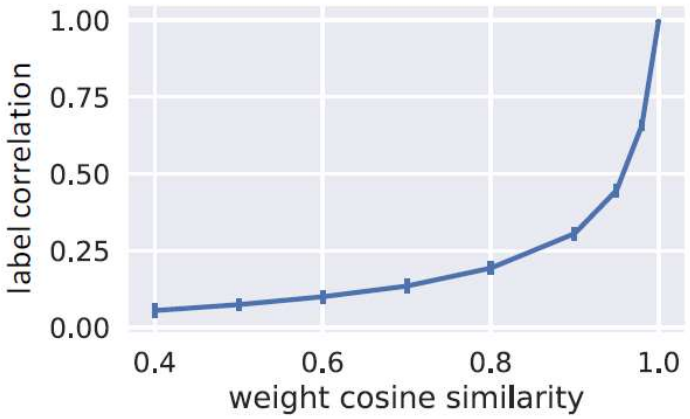
   在非线性情况下：

   $$y_1 = \vec{\mathbf{w}}_1^\top \vec{\mathbf{x}} + \sum_{i=1}^{m} \sin\left(\alpha_i \vec{\mathbf{w}}_1^\top \vec{\mathbf{x}} + \beta_i\right) + \epsilon_1$$

   $$y_2 = \vec{\mathbf{w}}_2^\top \vec{\mathbf{x}} + \sum_{i=1}^{m} \sin\left(\alpha_i \vec{\mathbf{w}}_2^\top \vec{\mathbf{x}} + \beta_i\right) + \epsilon_2$$

   此时 y_1, y_2 也是正相关的，如下图所示。下图给出了标签皮尔逊相关系数（纵轴）和权重余弦相似度 `weight cosine similarity` （横轴，即 p 值）之间的关系。对于每个 p 值，我们生成 `10k` 个带有两个标签的数据点，并计算这两个标签之间的皮尔逊相关系数。我们我们重复这个过程，并绘制均值，其中 `error bar` 表示 `100` 次实验中标准差的 `2` 倍。

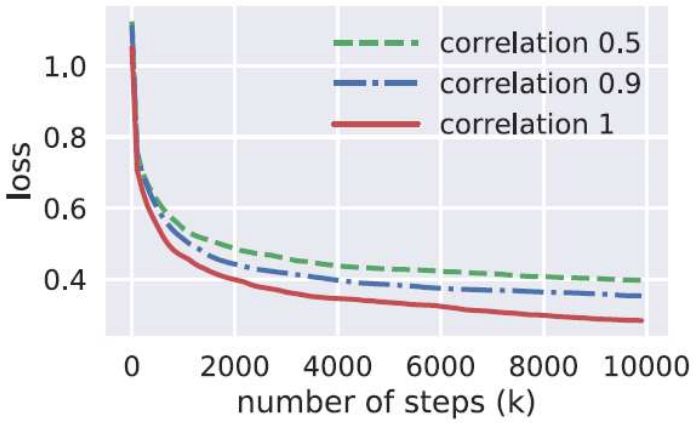
在本文的剩余部分，为简单起见，我们将权重向量的余弦相似度 p 称作任务相关系数 `task correlation` 。



1.2 任务相关性影响

1. 为了验证在 `baseline` 多任务模型设置下，低任务相关性会损害模型质量，我们对人工合成数据进行了如下控制实验。
    - 给定任务相关系数列表，为每个相关系数生成一个人工合成数据集。
    - 在控制所有模型和训练超参数保持不变的同时，分别在每个数据集上训练一个 `Shared-Bottom` 多任务模型。
    - 对独立生成的数据集重复上述两步数百次，但是控制任务相关系数列表和超参数相同。
    - 对每个任务相关系数计算模型的平均性能。

    下图给出了不同任务相关系数的 `loss` 曲线。正如预期所示：随着任务相关性的降低，模型的性能呈下降趋势。对于许多不同的超参数设置，这种趋势是普遍存在的。这一现象验证了我们的假设：即传统的多任务模型对任务关系很敏感。
    - 任务相关系数为 `1.0` 表示两个任务具有完全相同的权重向量，但是具有各自独立的噪声。横轴表示训练 `step` 的数量，纵轴表示 `200` 次独立运行的平均 `loss` 。注意：这两个回归任务是对称的，因此只需报告一个任务的结果即可。
    - `Shared-Bottom` 网络是宽度为 `16` 的单层网络、每个 `tower` 是宽度为 `8` 的单层网络。模型使用 `TensorFlow` 实现，并且使用带默认配置的 `Adam` 优化器进行训练。



1.3 模型

1. 共享底部多任务模型 `Shared-Bottom Multi-task Model` ：如下图所示为 `Shared-Bottom` 多任务模型，我们将它视为多任务建模的典型 `baseline` 方法。

    给定 K 个任务，该模型由以下部分组成：
    - 一个共享底部网络 `shared-bottom network` ，以函数 $f(\cdot)$ 来表示。

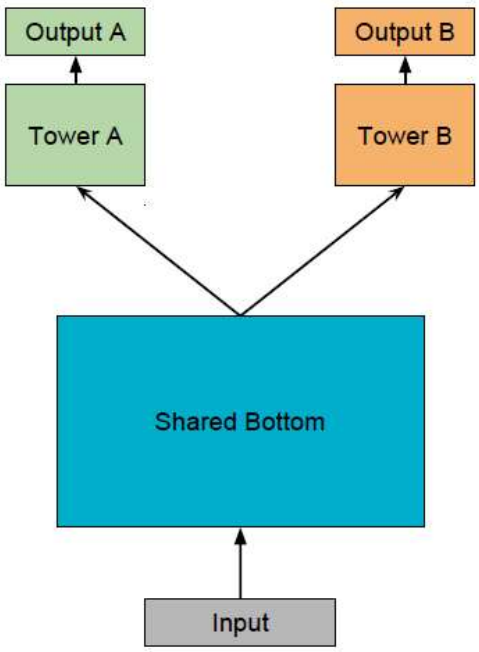
- K 个 `tower network`，以函数 $h^{(k)}(\cdot), k=1,2,\cdots,K$ 来表示，每个任务对应一个 `tower network`。

输入层 `input layer` 之后是 `shared-bottom network`，而之后是 `tower network`。然后每个任务的 `tower` 产生对应任务的输出 y_k。对于任务 k，其输出可以公式化为：

$$y_k = h^{(k)}\left(f\left(\vec{\mathbf{x}}\right)\right)$$

其中 $\vec{\mathbf{x}}$ 为输入的样本特征。

> 这里还有一个核心问题是：多个 `output` 的目标函数如何融合？我们知道 `DNN` 的优化过程只能优化单个目标函数，但是这里有两个 `output`，意味着有两个目标函数。如何融合多任务的目标函数也是一个难点，最简单的方法是各个目标函数直接相加。



2. `MoE`：原始的 `Mixture-of-Expert:MoE` 模型可以形式化为：

$$y = \sum_{i=1}^{n} g_i \times f_i\left(\vec{\mathbf{x}}\right)$$

其中：

- $f_i\left(\vec{\mathbf{x}}\right)$ 是第 i 个专家 `expert` 的输出，共有 n 个专家。
- $\vec{\mathbf{g}} = g\left(\vec{\mathbf{x}}\right) \in \mathbb{R}^n$ 为门控网络的输出，第 i 个分量 $g_i \in \mathbb{R}$ 给出第 i 个专家的权重，表示专家 f_i 的概率。且满足 $\sum_{i=1}^{n} g_i = 1,\ g_i \geq 0$。

门控网络基于输入得到 n 个专家的权重分布，最终的输出是所有专家输出的加权和。

> `MoE` 是单任务的、多个 `expert` 的集成方法。

3. `MoE Layer`：`MoE` 最初是作为多个单体模型 `individual model` 的集成方法 `ensemble method` 而开发的，但是有些工作将其转变为基本构建块 `basic building block`，并将其堆叠在 `DNN` 中。

- `MoE Layer` 具有和 `MoE` 模型相同的结构，但是采用前一层的输出作为后一层的输入。然后以端到端的方式训练整个模型。
- `MoE Layer` 的主要目标是实现条件计算，其中每个样本中只有网络的一部分是活跃的。对于每个输入样本，模型都可以通过以输入为条件的门控网络来选择全体专家的一个子集。

4. `MMoE`：我们提出了一种新的 `MoE` 模型，该模型旨在捕获任务差异，而不需要比 `Shared-Bottom` 多任务模型多得多的模型参数。新模型被称作 `Multi-gate Mixture-of-Expert: MMoE` 模型，其关键思想是用 `MoE Layer` 替代 `shared-bottom network` f。更重要的是，我们为每个任务 k 添加了一个单独的门控网络 $g^{(k)}(\cdot)$。

具体而言，任务 k 的输出为：

$$y_k = h^{(k)}\left(f^{(k)}\left(\vec{\mathbf{x}}\right)\right)$$

$$f^{(k)}\left(\vec{\mathbf{x}}\right) = \sum_{i=1}^{n} g_i^{(k)} f_i\left(\vec{\mathbf{x}}\right)$$

其中：

- $\vec{\mathbf{x}} \in \mathbb{R}^d$ 为输入 `embedding` 。
- $\vec{\mathbf{g}}^{(k)} = g^{(k)}\left(\vec{\mathbf{x}}\right) \in \mathbb{R}^n$ 为任务 k 的门控网络，而 $g_i^{(k)}$ 为 $\vec{\mathbf{g}}^{(k)}$ 的第 i 项，对应于任务 k 中第 i 个专家的权重。$k = 1, 2, \cdots, K$ 。
- $f_i\left(\vec{\mathbf{x}}\right)$ 为第 i 个专家。$i = 1, 2, \cdots, n$ 。
- $f^{(k)}\left(\vec{\mathbf{x}}\right)$ 为所有 n 个专家在任务 k 上的加权和，它是任务 k `tower network` 的输入。
- $h^{(k)}(\cdot)$ 为任务 k 的 `tower network`。

我们 `MMoE` 的实现由采用 `ReLU` 激活函数的多层感知机组成。门控网络只是对输入线性变换然后通过一个 `softmax` 层：

$$\vec{\mathbf{g}}^{(k)} = g^{(k)}\left(\vec{\mathbf{x}}\right) = \text{softmax}\left(\mathbf{W}_{g^{(k)}}\vec{\mathbf{x}}\right) \in \mathbb{R}^n$$

其中 $\mathbf{W}_{g^{(k)}} \in \mathbb{R}^{n \times d}$ 是线性变换的参数，d 是输入特征的维度。

5. 每个门控网络 $g^{(k)}\left(\vec{\mathbf{x}}\right)$ 都可以学习在给定输入 $\vec{\mathbf{x}}$ 的条件下 `select` 所有专家的一个子集。对于多任务学习场景下的灵活参数共享，这是理想的。

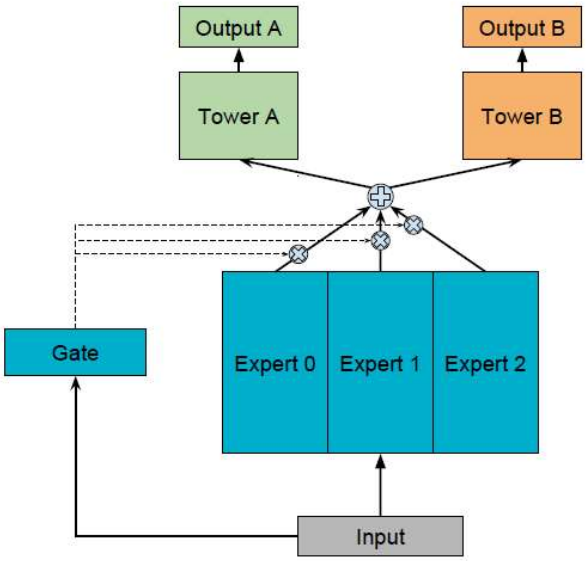
   作为一种特殊情况，如果仅选择一个具有最高 `gate score` 的专家，那么每个门控网络实际上将输入空间线性地划分为 n 个区域，每个区域对应一个专家。`MMoE` 通过决定不同 `gate` 产生的间隔如何相互重叠，从而能够以一个复杂的方式来建模任务关系。如果任务的相关性较低，则共享专家将受到惩罚，这些任务的门控网络将学会使用不同的专家。

   > 如果任务之间的相关性较低，则不同任务之间倾向于选择不同的专家；如果任务之间的相关性较高，则不同任务之间倾向于选择相同的专家。

   和 `Shared-Bottom` 模型相比，`MMoE` 只有几个额外的门控网络，而门控网络中的模型参数数量可以忽略不计。因此，在多任务学习中，整个模型仍然尽可能地享受知识迁移的好处。

   为理解为每个任务引入单独的门控网络如何帮助模型学习 `task-specific` 的信息，我们比较了所有任务共享一个门控的模型结构。我们称之为 `One-gate Mixture-of-Expert: OMoE` 模型，如下图所示。这是 `MoE Layer` 对 `Shared-Bottom` 多任务模型的直接适配。

   > `OMoE` 可以视为多个子模型的 `ensemble`，因此它的效果要强于 `shared-bottom` 模型。



1.4 实验

1.4.1 人工合成数据集

1. 为了解`MMoE`模型是否可以更好地处理任务相关性较低的情况，我们改变了人工合成数据的任务相关性，并观察了不同模型的行为如何变化。

   我们还进行了可训练性`trainability`分析，并表明和`Shared-Bottom`模型相比，`MMoE`模型更易于训练。

2. 配置：

   - 输入维度为`100`。
   - 所有`MoE based`模型都有`8`个专家，每个专家都实现为一个隐层维度为`16`的单层网络。`tower`网络实现为一个隐层维度为`8`的单层网络。

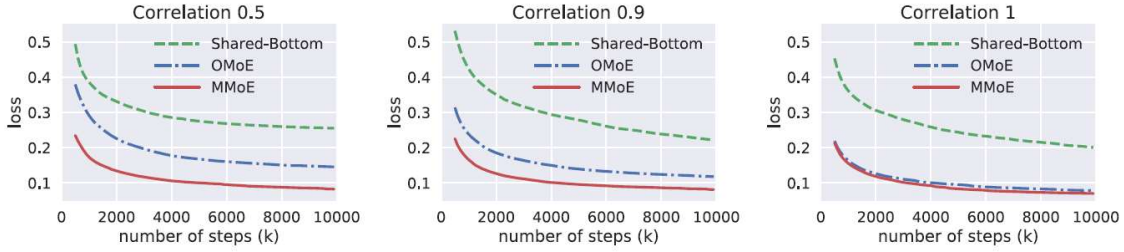
     我们注意到，共享的专家和`tower`中所有参数的总量是$100 \times 16 \times 8 + 16 \times 8 \times 2 = 13056$。
   - 对于`baseline`的`Shared-Bottom`模型，我们仍然将`tower`网络实现为一个隐层维度为`8`的单层网络。我们设置单层的`shared-bottom network`，其隐层维度为$13056/(100 + 8 \times 2) \simeq 113$。
   - 所有模型都使用`Adam`优化器训练，学习率在`[0.0001, 0.001, 0.01]`中进行网格搜索。

   注意：由于两个回归任务是对称的，因此只需报告一个任务的`loss`值即可。

3. 对于每种模型配置，我们重复`200`次独立运行，每次使用随机生成的数据和随机的模型初始化。平均结果如下图所示。可以看到：

   - 对于所有模型，具有较高相关性的数据，其性能要优于具有较低相关性的数据。其中性能以损失函数`loss`来衡量。
   - 在具有不同相关性的数据上，`MMoE`模型的性能差距要远小于`OMoE`模型和`Shared-Bottom`模型。当我们将`MMoE`模型和`OMoE`模型进行比较时，这种趋势尤为明显：
     - 在两个任务相同的极端情况下，`MMoE`模型和`OMoE`模型的性能几乎没有区别。
     - 当两个任务之间的相关性降低时，`OMoE`模型的性能就会明显降低，而`MMoE`模型的影响很小。

     因此，在低关联性的情况下，具有`task-specific gate`对任务差异建模至关重要。
   - 就平均性能而言，两种`MoE`模型在所有情况下都优于`Shared-Bottom`模型。这表明`MoE`结构本身带来了额外的好处。根据这一观察，我们在后续展示了`MoE`模型比`Shared-Bottom`模型具有更好的可训练性`trainability`。



(a) Performance with correlation 0.5 (b) Performance with correlation 0.9 (c) Performance with two identical tasks

4. 对于大型神经网络模型，我们非常关心它们的可训练性`trainability`，即模型在一系列超参数设置和模型初始化的鲁棒性。

   最近，`Collins`等人发现，某些门控`RNN`模型（如`LSTM`和`GRU`），这些模型我们认为比普通`RNN`性能更好，更容易训练，而不是具有更好的模型容量。

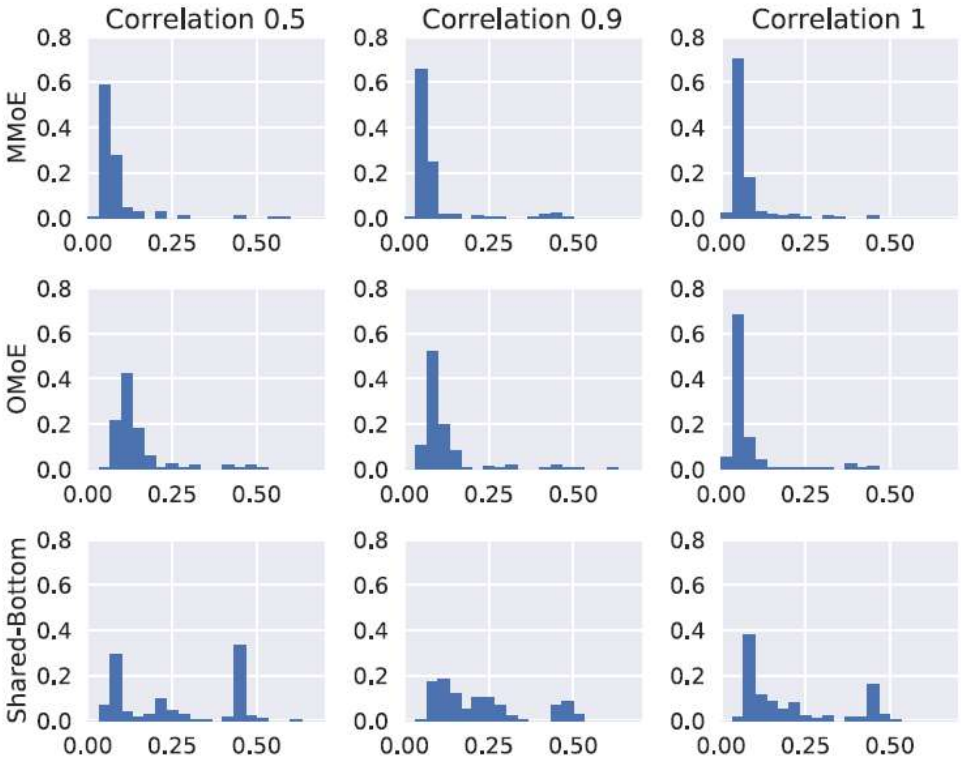
虽然我们已经证明 `MMoE` 可以更好地处理任务相关性较低的情况，但是我们也希望更深入地了解它在可训练性方面的表现。利用我们的人工合成数据，我们可以自然地研究我们的模型对于数据和模型初始化中的随机性的鲁棒性。

我们在每种配置下重复实验多次。每次数据都是在相同的分布、但是不同的随机种子生成的，模型的初始化也不同。我们在下图中绘制了重复运行的最终 `loss` 值的直方图。在直方图中有三个有趣的观察结果：

- 首先，在所有配置中，`Shared-Bottom` 模型的性能差异要比 `MoE-based` 模型大得多。这意味着 `Shared-Bottom` 模型通常比 `MoE-based` 模型具有更差质量的局部极小值。
- 其次，当任务相关度为 `1` 时，`OMoE` 模型的性能方差与 `MMoE` 模型的性能方差具有相似的鲁棒性。而当任务相关性降低到 `0.5` 时，`OMoE` 模型的鲁棒性有显著的下降。

  注意到 `MMoE` 和 `OMoE` 唯一的区别在于是否采用多门结构。这验证了多门结构在解决由任务差异导致的冲突 `conflict` ，从而带来的不良局部极小值方面的有效性。
- 最后，值得观察的是：所有三种模型的最低 `loss` 是可比的。这并不奇怪，因为神经网络在理论上是通用的函数逼近器 `approximator` 。只要有足够的模型容量，就应该存在一个 “正确的” `Shared-Bottom` 模型来很好地学习这两个任务。

  但是请注意：这是 `200` 个独立实验的分布。而且我们怀疑，对于更大、更复杂的模型（例如，当 `shared-bottom network` 是 `RNN` 时），获得任务关系的 “正确” 模型的机会可能更低。因此，显式建模任务关系仍然是可取的。



1.4.2 真实数据集

1. `baseline` 方法：除了 `Shared-Bottom` 多任务模型之外，我们还比较了几种 `state-of-the-art` 多任务深度神经网络模型。

   - `L2-Constrained`：该方法是为一个拥有两个任务的跨语言问题 `cross-lingual problem` 设计的。在该方法中，不同任务的参数由 `L2` 约束软性共享 `shared softly` 。

     假设任务 $k, k=1,2$ 的 `ground truth label` 为 y_k ，任务 k 的预估结果为 $\hat{y}_k = f\left(\vec{\mathbf{x}};\theta_k\right)$ ，其中 θ_k 为模型参数。该方法的目标函数为：

$$\mathcal{J} = \mathbb{E}\left[L\left(y_1, f\left(\vec{\mathbf{x}};\theta_1\right)\right)\right] + \mathbb{E}\left[L\left(y_2, f\left(\vec{\mathbf{x}};\theta_2\right)\right)\right] + \alpha\|\theta_1 - \theta_2\|_2^2$$

其中：

- y_1 为任务 1 的 ground truth label ， y_2 为任务 2 的 ground truth label 。
- $L(\cdot,\cdot)$ 为损失函数。
- α 为超参数。该方法使用 α 的大小来建模任务相关性。

- Cross-Stitch：该方法通过引入一个 Cross-Stitch 单元，从而在两个任务之间共享知识。Cross-Stitch 单元从任务 1 和任务 2 中获取隔离 separated 的隐层 $\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2$ 作为输入，并通过以下等式分别输出 $\tilde{\vec{\mathbf{x}}}_1^{(i)},\tilde{\vec{\mathbf{x}}}_2^{(i)}$ ：

$$\begin{bmatrix}\tilde{\vec{\mathbf{x}}}_1^{(i)}\\ \tilde{\vec{\mathbf{x}}}_2^{(i)}\end{bmatrix}=\begin{bmatrix}\alpha_{1,1} & \alpha_{1,2}\\ \alpha_{2,1} & \alpha_{2,2}\end{bmatrix}\begin{bmatrix}\vec{\mathbf{x}}_1^{(i)}\\ \vec{\mathbf{x}}_2^{(i)}\end{bmatrix}$$

其中：

- $\alpha_{j,k},j,k=1,2$ 为可训练的参数，代表从任务 k 到任务 j 的交叉迁移 cross transfer 。
- $\tilde{\vec{\mathbf{x}}}_1^{(i)},\tilde{\vec{\mathbf{x}}}_2^{(i)}$ 表示在任务 1 和任务 2 中被发送到更高的层的 representation 。i 表示第 i 层。

- Tensor-Factorization：在该方法中，将多个任务的权重张量进行张量分解从而用于跨任务的参数共享。为进行比较，我们实现了 Tucker 分解以学习多任务模型，据报道该模型可以提供最可靠的结果。

  例如，给定输入隐层维度为 m 、输出隐层维度为 n 、任务数量 K 的情况下，权重 $\mathbf{W}$ 是一个 $m\times n\times K$ 的张量，它由以下等式得到：

$$\mathbf{W}=\sum_{i_1}^{r_1}\sum_{i_2}^{r_2}\sum_{i_3}^{r_3}S(i_1,i_2,i_3)\times\mathbf{U}_1(:,i_1)\circ\mathbf{U}_2(:,i_2)\circ\mathbf{U}_3(:,i_3)$$

其中：

- $\mathbf{S}\in\mathbb{R}^{r_1\times r_2\times r_3}$ 为一个张量，它为可训练的参数。
- $\mathbf{U}_1\in\mathbb{R}^{m\times r_1},\mathbf{U}_2\in\mathbb{R}^{n\times r_2},\mathbf{U}_3\in\mathbb{R}^{K\times r_3}$ 为矩阵。它们都是可训练的参数。
- $\circ$ 为向量的外积。
- r_1,r_2,r_3 为超参数。

2. 配置：

- 我们采用了一种超参数调优器 hyper-parameter tuner 为所有模型搜索最佳超参数，调优算法是一个高斯过程模型 Gaussian Process model 。

  除了调优所有方法的学习率和训练 step 的数量之外，我们还调优了一些 model-specific 的超参数：

  - MMoE：专家的数量、每个专家的隐层维度。
  - L2-Constrained：隐层维度、α 系数。
  - Cross-Stitch：隐层维度、Cross-Stitch layer 的维度。
  - Tensor-Factorization：系数 r_1,r_2,r_3、隐层维度。
- 为使得公平地比较，我们通过为每层隐单元数量设置相同的上限来限制所有方法的最大模型大小，即 2048 。对于 MMoE ，它就是 “专家数量” 乘以 “每个专家的隐层维度” 。
- 我们的方法和所有 baseline 方法都是使用 Tensorflow 来实现的。

a. Census-income 数据集

1. UCI census-income 数据集：从 1994 年人口普查数据库中抽取的，包含 299285 个美国成年人的人口信息，每个样本有 40 个特征。

   通过将某些特征设置为预测目标，我们从该数据集中构造了两个多任务学习问题，并计算了 10000 个随机样本中任务标签的皮尔逊相关系数的绝对值：

- 任务`1`：预测收入是否超过`5`万美元；任务`2`：预测他/她是否从未结婚。绝对皮尔逊相关系数为`0.1768`。
- 任务`1`：预测教育程度是否至少为大学；任务`2`：预测他/她是否从未结婚。绝对皮尔逊相关系数为`0.2373`。

数据集中有`199523`个训练样本和`99762`个测试样本。我们进一步按`1:1`的比例将测试样本随机划分为验证集和测试集。注意：我们从输入特征中删除教育和婚姻状况，因为这些特征被视为标签。

2. 配置：
   - 由于这两组任务都是二元分类问题，因此我们将`AUC`分数作为评估指标。在这两组中，我们都将婚姻状况任务作为辅助任务，而将第一组中的收入任务、第二组中的教育任务视为主要任务。我们关注主要任务的`AUC`。
   - 对于超参数调优，我们使用验证集上主要任务的`AUC`作为目标。
   - 对于每种方法，我们使用超参数调优器进行数千次实验，以找到最佳的超参数配置。

     在找到最佳超参数之后，我们使用随机参数初始化在训练集上对每种方法进行`400`次训练，并在测试集上报告结果（主要任务的平均`AUC`）。
3. 下表给出了两组任务的实验结果，可以看到：
   - 由于这两组中的任务相关性（大致由`label`的皮尔逊相关系数来衡量）都不是很强，因此`Shared-Bottom`模型在多任务模型中几乎总是最差的（`Tensor-Factorization`除外）。
   - `L2-Constrained`和`Cross-Stitch`都为每个任务提供了单独的模型参数，并增加了如何学习这些参数的约束，因此它们的性能要比`Shared-Bottom`模型更好。

     但是，对模型参数学习的约束很大程度上依赖于任务关系的假设，这不如`MMoE`使用的参数调制机制`parameter modulation mechanism`灵活。因此`MMoE`在第二组任务中的所有方面都超越其他多任务模型。
   - `Tensor-Factorization`方法在两组中效果都是最差的。这是因为它倾向于对所有任务的隐层权重进行低秩张量和低秩矩阵的泛化。该方法对于任务相关性可能非常敏感，因为当任务之间的相关性较低时，它倾向于过度泛化`over-generalize`，并且需要更多的数据和更长的训练时间。
   - 多任务模型未针对验证集上的婚姻状况辅助任务进行调优，而单任务模型则针对辅助任务进行了调优。因此，单任务模型在辅助任务上获得最佳性能是合理的。

Group 1	AUC/Income		AUC/Marital Stat	
	best	mean	w/ best income	mean
Single-Task	0.9398	0.9337	**0.9933**	0.9922
Shared-Bottom	0.9361	0.9295	0.9915	0.9921
L2-Constrained	0.9389	0.9359	0.9922	0.9918
Cross-Stitch	0.9406	**0.9361**	0.9917	0.9922
Tensor-Factorization	0.7460	0.6765	0.8175	0.8412
OMoE	0.9387	0.9319	0.9928	0.9923
MMoE	**0.9410**	0.9359	0.9926	**0.9927**

Group 2	AUC/Education		AUC/Marital Stat	
	best	mean	w/ best education	mean
Single-Task	0.8843	0.8792	**0.9933**	0.9922
Shared-Bottom	0.8836	0.8813	0.9927	0.9917
L2-Constrained	0.8855	0.8823	0.9923	0.9918
Cross-Stitch	0.8855	0.8819	0.9919	0.9921
Tensor-Factorization	0.7367	0.7256	0.7453	0.7497
OMoE	0.8852	0.8813	0.9915	0.9912
MMoE	**0.8860**	**0.8826**	0.9932	**0.9924**

b. 大规模内容推荐数据集

1. 我们在谷歌的大型内容推荐系统上进行实验，其中为数十亿用户推荐数十亿个`item`。具体而言，给定用户当前消费`item`的行为，推荐系统旨在向用户推荐接下来要消费的`item`列表。

   我们的推荐系统采用一些现有内容推荐框架，包括一个候选生成器`candidate generator`、以及一个深度排序模型。在我们的设置中，深度排序模型可以针对两种类型的排序目标进行优化：

   - 针对互动`engagement`相关的目标进行优化，如点击率`CTR`、互动时长。
   - 针对满意度`satisfaction`相关的目标进行优化，如喜欢率`like rate`。

   我们的训练数据包括数以千亿记的用户隐式反馈，如点击`click`和喜欢`like`。如果单独训练，则每个任务的模型需要学习数十亿个参数。因此，和分别独立学习多个目标相比，`Shared-Bottom`架构具有更小规模的优势。实际上，`Shared-Bottom model`已经在生产环境中使用。

2. 配置：

   - 我们通过为深度排序模型创建两个二元分类任务来评估多任务模型：预测用户互动相关的行为、预测用户满意度相关的行为。我们将这两个任务命名为`engagement`子任务、`satisfaction`子任务。
   - 推荐系统使用稀疏特征的`embedding`，并将所有稠密特征归一化为`[0.0,1.0]`之间。
   - 对于`Shared-Bottom`模型，我们将`shared-bottom network`实现为一个前馈神经网络，它具有几个全连接层并使用`ReLU`激活函数。每个任务在`shared-bottom network`之上构建全连接层作为`tower`网络。
   - 对于`MMoE`，我们只需要将`shared-bottom network`的顶层修改为`MMoE layer`，并保持`MMoE layer`的输出维度不变。因此，我们不会在模型训练和`serving`中增加额外的计算成本。
   - 我们还实现了其他`baseline`方法，例如`L2-Constrained`和`Cross-Stitch`。由于它们的模型架构，与`Shared-Bottom`模型相比，它们的参数数量大约翻了一倍。
   - 我们不和`Tensor-Factorization`进行比较，因为如果没有高效率的工程实现，`Tucker`乘积的计算不能扩展到十亿级。
   - 所有模型均采用`batch size = 1024`的`mini-batch`随机梯度下降法进行优化。

3. 离线效果评估：对于离线评估，我们在固定的`300`亿条用户隐式反馈集合上训练模型，并在`100`万条数据的`hold-out`数据集上进行评估。鉴于`satisfaction`子任务的标签要比`engagement`子任务的标签稀疏的多，离线结果噪音很大。因此我们只在下表中给出`engagement`子任务的`AUC`得分和平方误差`R-Squared`得分。

   我们分别展示了训练`200`万步（千亿级样本、`batch size = 1024`）、`400`万步、`600`万步之后的结果。可以看到：

   - 在所有指标上，`MMoE`均优于其它模型。

     > 这里`OMoE`效果并没有超越`Shared-Bottom`，这不符合预期（`OMoE`是集成模型）。

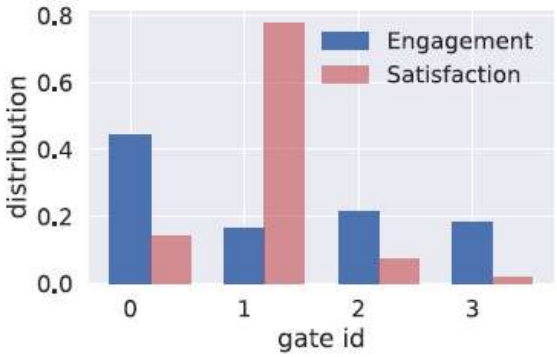
- `L2Constrained` 和 `Cross-Stitch` 比 `Shared-Bottom` 模型更差。这很可能是由于这两个模型是建立在两个独立的单任务模型上，并且有太多的模型参数使得难以很好地约束 `constrained` 。

> 注：这里用户隐式反馈数据 `300` 亿条，但是训练样本有千亿级，是因为还有些负采样的样本。

Metric	AUC@2M	AUC@4M	AUC@6M	R2@2M	R2@4M	R2@6M
Shared-Bottom	0.6879	0.6888	0.6900	0.08812	0.09159	0.09287
L2-Constrained	0.6866	0.6881	0.6895	0.08668	0.09030	0.09213
Cross-Stitch	0.6880	0.6885	0.6899	0.08949	0.09112	0.09332
OMoE	0.6876	0.6891	0.6893	0.08749	0.09085	0.09230
MMoE	**0.6894**	**0.6897**	**0.6908**	**0.08978**	**0.09263**	**0.09362**

为更好地理解门控 `gate` 是如何工作的，我们在下图中显示了每个任务的 `softmax gate` 的分布。可以看到：

- `MMOE` 学到了这两个任务之间的差异，并自动平衡 `balance` 了共享参数和非共享参数。
- 由于 `satisfaction` 子任务的标签比 `engagement` 子任务的标签更加稀疏，因此 `satisfaction` 子任务的 `gate` 更多地聚焦于单个专家上。



4. 在线效果评估：最后，我们在内容推荐系统上对我们的 `MMoE` 模型进行了在线实验。我们不进行 `L2-Constrained` 和 `CrossStitch` 方法的在线实验，因为这两种模型通过引入更多的参数使得 `serving` 时间加倍。

   我们进行了两组实验：

   - 第一组实验是比较 `Shared-Bottom` 模型和单任务 `Single-Task` 模型。 `Shared-Bottom` 模型在 `engagement` 子任务和 `satisfaction` 子任务上都进行了训练。单任务模型仅在 `engagement` 子任务上进行训练。

     注意：尽管没有在 `satisfaction` 子任务上进行训练，但是单任务模型在测试时用作排序模型，因此我们也可以计算它推荐结果的 `satisfaction` 指标。

   - 第二组实验是将我们的 `MMoE` 模型和第一个实验中的 `Shared-Bottom` 模型进行比较。

   这两组实验都是使用相同数量的在线流量完成的。下表给出了在线实验的结果，可以看到：

   - 首先，和单任务模型相比，通过使用 `Shared-Bottom` 模型，我们看到在线 `satisfaction` 指标大幅提升了 `19.72%` 、在线 `engagement` 指标略有下降（ `-0.22%` ）。
   - 其次，和 `Shared-Bottom` 模型相比，通过使用 `MMoE` 模型，我们同时提升了这两个指标。

   注意：在该推荐系统中， `engagement` 指标的原始值比 `satisfaction` 指标的原始值大得多。并且在提高 `satisfaction` 指标的同时，希望没有 `engagement` 指标的下降、甚至是能够有 `engagement` 指标的提升。

Live experiment	Engagement Metric	Satisfaction Metric
Shared-Bottom Improvement over Single-Task	-0.22% *	19.72% **
MMoE Improvement over Shared-Bottom	0.25% **	2.65% **

* indicates confidence interval level 90%

** indicates confidence interval level 95%