实时预测 Dota2 胜率: 两个现代方法

龚林源
北京大学*
gonglinyuan@hotmail.com

谷晟
北京大学
gsxj2014@163.com

季卫明慧†
北京大学
jiweiminghui@pku.edu.cn

Abstract

我们抓取了 168451 场 Dota2 天梯比赛的记录, 建立了两个实时预测 Dota2 比赛胜率的模型. 我们利用对于 Dota2 的丰富知识, 对每个游戏状态提取了 977 个数值特征, 用 LightGBM 训练了第一个模型. 通过分析这个模型, 我们理解了各个特征的重要性; 我们还设计了基于注意力 (Attention) 机制的深度神经网络模型, 通过可视化注意力来理解模型的预测. 最后, 我们集成了两个模型, 在实时预测 Dota2 胜率任务上取得了远超 DotaPlus 的高准确率.

1 任务与数据集

Dota2 是一款多人在线战斗竞技场游戏 (multiplayer online battle arena, MOBA). 每一局比赛由两支队伍对抗, 每支队伍有五个玩家, 每个玩家控制一名英雄. 玩家控制的英雄在游戏地图上进行战斗, 在小兵的帮助下摧毁对方的防御塔和其它建筑. 当一方的特殊建筑 “遗迹” 被摧毁时, 游戏结束. 这次 UBS 量化大赛的题目就是: 给定一个实时的 Dota2 游戏状态, 预测这场比赛的胜负.

Kaggle 上提供的 Dota2 天梯比赛数据 (https://www.kaggle.com/devinanzelmo/dota-2-matches), 由于数据量小 (仅 50000 场), 年代久远, 版本不统一的缺点, 不能反映 Dota2 7.00 更新以来的全新游戏节奏. 因此我们通过 OpenDota API 重新抓取了 Dota2 7.19c 更新后的 168451 场天梯比赛. 7.19c 版本是 2018 年 9 月 14 日更新的版本, 是目前 (2018 年 10 月 10 日) Dota2 的最新版本. 这些比赛是按照以下标准筛选的:

- 游戏模式为天梯比赛中的 “全阵营选择”
- 比赛时长在 10 分钟以上, 且无人在 25 分钟前放弃比赛
- 玩家的平均等级达到 “十字军” 级别

这 168451 组数据分为三部分: 训练集 (143356 组), 验证集 (16718 组), 测试集 (8377 组), 比例约为 17 : 2 : 1. 所有模型都只在训练集上训练, 利用验证集进行提前终止 (early-stopping) 和超参数调优, 固定下来所有模型之后再在测试集上测试, 并报告结果.

2 特征工程

见表1, 对于一局 Dota2 比赛中的时间点, 可以提取出若干数值特征. 一类特征是与玩家个体相关的, 一类是与全局相关的.

- 对于阵容特征, 给每个队伍一个长度为 Dota2 英雄总数的向量, 如果这个队伍中有某个英雄, 就把这个英雄对应的位置置为 1, 其余位置置为 0, 每个队伍恰好有五个位置是 1
- 对于每一个个体特征, 每个队伍会有 5 个并列的数字; 我们把它们在队伍内从小到大排序. 这样虽然丢失了不同特征与同一个玩家的对应关系, 但是更容易让每个特征本身具有实际的意义

*在微软亚洲研究院实习期间参与此次比赛
†队长

Preprint. Work in progress.

个体特征	全局特征
阵容: 所选用的英雄 **金钱:** – 个人总财产 – 个人总财产占全场比例 – 最近 1 分钟, 最近 5 分钟的金钱增量 – 平均每分钟获得金钱 (GPM) **经验和等级:** – 总经验 – 等级 – 总经验占全场比例 – 最近 1 分钟, 5 分钟内的经验增量 – 平均每分钟获得经验 (XPM) **补刀**[3]**:** – 总正补数 – 正补数占全场的比例 – 总反补数 – 反补数占全场的比例 **插眼:** – 放置侦查守卫 (假眼) 数目 – 放置岗哨守卫 (真眼) 数目 **地图控制:** 从游戏开始到现在, 获得每一种神符的次数 **团战, Gank, 反 Gank 表现:** – 击杀次数 – 死亡次数 – 总 (造成) 英雄伤害 – 买活次数 **沟通与交流:** – 使用聊天轮盘次数 – 打字聊天的次数 – 使用聊天轮盘和打字聊天总次数 **物品与装备:** 购买每一种物品的次数 – 包括消耗品和装备 **士气:** 游戏开始时是否预测自己队伍会胜利	**时间:** 比赛已进行多少分钟 **比赛区域:** 如国服, 东南亚, 日本等 **建筑状态:** 当前各个建筑还剩下多少个 – 中路, 优势路, 劣势路的一, 二, 三塔 (9 种, 各 1 个) – 中路, 优势路, 劣势路的近战, 远战兵营 (6 种, 各 1 个) – 门牙塔 (1 种, 2 个) – 圣坛 (1 种, 2 个) **击杀 Roshan 次数** **金钱:** – 队伍总财产, 以及最近 1 分钟, 5 分钟的金钱增量 – 双方总财产之比, 以及最近 1 分钟, 5 分钟的金钱增量之比 – 队伍内五个玩家总财产的变异系数[4] **经验:** – 队伍总经验, 以及最近 1 分钟, 5 分钟的经验增量 – 双方总经验之比, 以及最近 1 分钟, 5 分钟的经验增量之比 – 队伍内五个玩家总经验的变异系数 **补刀:** – 队伍总正补数, 总反补数 – 双方总正补数, 总反补数之比 **插眼:** – 队伍放置侦查守卫 / 岗哨守卫的总数 – 双方放置侦查守卫 / 岗哨守卫次数之比 – 最近六分钟队伍放置侦查守卫 / 岗哨守卫的总数[5] **地图控制:** – 队伍获得神符总次数 – 两队获得神符次数之比 **团战/Gank/反 Gank 胜负:** – 队伍总击杀数 – 两队击杀次数之比 – 队伍买活总次数 – 双方买活总次数之差 **沟通与交流:** – 队伍使用聊天轮盘, 聊天轮盘的总次数, 以及两者之和 – 两队使用聊天轮盘次数之比 **士气:** – 队伍预测胜利的玩家人数 – 双方预测胜利的玩家人数之差

Table 1: 从 Dota2 比赛中一个时间点提取的特征.

  - 比如 “金钱” 特征, 在队伍内部从大到小排序后, 就是队伍内第一名, 第二名, 第三名, 第四名, 第五名的金钱
- 对于比赛区域, 我们把它标记为 “类别特征”, 即它的数字 (从 1 到 25) 不具有大小关系, 只是表示不同的类别

所有的特征都是在游戏进行到这个时间点时可以获得的, 不包含 “未来函数”. 所有特征最后共有 977 维.

3 LightGBM[5] 模型

LightGBM 是微软亚洲研究院 2017 年推出的一个开源的工业级机器学习库, 是梯度提升决策树 (Gradient Boosting Decision Tree, GBDT) 算法的改进及高效实现. GBDT 是机器学习中的一个传统算法; XGBoost[2] 是对它的巨大改进, 推出后一度统治了 Kaggle 上的众多数据挖掘竞赛; LightGBM 又是对 XGBoost 的改进, 速度提高了 7 至 20 倍. 由于我们的数据量大, 达到 40GB 之多, 所以使用 LightGBM 作为第一个模型.

3.1 算法概述

3.1.1 梯度提升决策树 (GBDT)

决策树是一个分段函数, 可以用分段空间 $\{R_j\}_1^J$ 和对应的函数值 $\{b_j\}_1^J$ 表示决策树的参数, 对于输入 x, 决策树的输出为:

$$f(x;\{R_j,b_j\}_1^J)=\sum_j b_j\mathbf{1}[x\in R_j]$$

传统机器学习中的有一系列基于决策树集成 (emsemble) 的算法, 如随机森林, GBDT 等. 它们的输出都是 K 个决策树的输出之和:

$$F=\sum_i f_i$$

决策树 ensemble 的损失函数定义为每个数据点上的损失值之和, 加上正则化 (Regularization) 项:

$$\mathcal{L}=\sum_{i=1}^N L(y_i,F(x_i))+\sum_{k=1}^K \Omega(f_k)$$

GBDT 是一个以决策树为基分类器的梯度提升 (Gradient Boosting) 算法, 梯度提升相当于在 n 维函数 $L(y_1,y_2,\dots,y_n)$ 的空间梯度下降. GBDT 每一轮都会计算损失函数关于对于每个数据点的预测值的梯度 (即残差), 训练一棵决策树拟合残差, 然后把新的决策树加到当前模型当中. 算法第 k 轮的步骤如下:

1. 计算残差: $\tilde{y}_i=-\dfrac{\partial\mathcal{L}(y_i,F_{k-1}(x_i))}{\partial F_{k-1}(x_i)},\ \forall 1\le i\le N$
2. 以与残差的**均方误差 (MSE)** 尽量小为目标训练决策树: $\{R_j,b_j\}_1^J=\text{train}(\{x_i,\tilde{y}_i\}_1^N)$
3. 令 $f_k=\rho f(\cdot,\{R_j,b_j\}_1^J),\ F_k=F_{k-1}+f_k$, 其中 ρ 是学习率

3.1.2 XGBoost

XGBoost 是对 GBDT 的改进, 把第 2 步和第 3 步合并, 再加上正则化项. XGBoost 每一轮都要最小化 $\sum_i^N L(y_i,F_{k-1}(x_i)+f_k(x_i))+\Omega(f_k)$. 把它在 $\{F_{k-1}(x_i)\}_1^N$ 点处 Taylor 展开到二阶, 得到:

$$
\begin{aligned}
\mathcal{L}_k &= \sum_i^N L(y_i, F_{k-1}(x_i) + f_k(x_i)) + \Omega(f_k) \\
&\approx \sum_i \left(L(y_i, F_{k-1}(x_i)) + \frac{\partial L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)} f_k(x_i) + \frac{1}{2} \frac{\partial^2 L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)^2} f_k(x_i)^2 \right) + \Omega(f_k) \\
&= \sum_i \left(L(y_i, F_{k-1}(x_i)) + g_i f_k(x_i) + \frac{1}{2} h_i f_k(x_i)^2 \right) + \Omega(f_k) \\
&= \sum_i \left(L(y_i, F_{k-1}(x_i)) + g_i \sum_j b_j \mathbf{1}[x_i \in R_j] + \frac{1}{2} h_i \sum_j b_j^2 \mathbf{1}[x_i \in R_j] \right) + \frac{\lambda}{2} \sum_j b_j^2 \\
&= \text{Const} + \sum_j \left(b_j \sum_{x_i \in R_j} g_i + \frac{1}{2} b_j^2 \sum_{x_i \in R_j} h_i + \frac{\lambda}{2} b_j^2 \right) \\
&= \text{Const} + \sum_j \left(b_j G_j + \frac{1}{2} b_j^2 (H_j + \lambda) \right)
\end{aligned}
$$

这里 $g_i \triangleq \dfrac{\partial L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)}$; $h_i \triangleq \dfrac{\partial^2 L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)^2}$; Const $\triangleq \sum_i L(y_i, F_{k-1}(x_i))$, 是与 $\{R_j, b_j\}_1^J$ 无关的常量; $G_j \triangleq \sum_{x_i \in R_j} g_i$; $H_j \triangleq \sum_{x_i \in R_j} h_i$.

接下来就是要找到使得 $\mathcal{L}_k$ 尽量小的 $\{R_j, b_j\}_1^J$. 如果给定 $\{R_j\}_1^J$, 只要对 b_j 求导, 让导数为 0, 即可得到最大值点为:

$$
b_j = -\frac{G_j}{H_j + \lambda}, \forall 1 \le j \le J
$$

代入得 $\mathcal{L}_k = \text{Const} - \dfrac{1}{2} \sum_j \dfrac{G_j^2}{H_j + \lambda}$, 是一个关于 $\{R_j\}_1^J$ 的函数.

精确求解最优的 $\{R_j\}_1^J$ 是一个 NP-Hard 的组合优化问题, 一般认为不可能在多项式时间内求解. XGBoost 的贪心策略是:

1. 枚举叶子节点, 枚举特征, 找出能使得损失函数减少得最多的分割点
2. 用最优的分裂方案把一个叶子节点分裂成两个

XGBoost 在损失函数的二阶 Taylor 展开上优化, 相比 GBDT 的一阶优化方法, 所需迭代次数更少, 准确度更高.

3.1.3 LightGBM

GBDT 在寻找分割点的时候找的是精确解, 需要枚举整个数据集; XGBoost 把数据点按照各个特征预排序 (pre-sort), 提高了速度, 但是在大数据集上训练仍然缓慢. LightGBM 把数据集的每个特征的分布建立了一个粒度较粗的直方图, 寻找分割点的时候在直方图上找近似解. 这样做的好处有:

- 基于集成 (Ensemble) 的算法并不要求每个弱学习器 (Weak Learner) 的预测结果都非常准确; 可以认为 LightGBM 的近似给每棵决策树的输出都加上了一个无偏的误差项, 只要增加决策树的数目就能让误差的影响忽略不计
- 直方图的段数一般比数据点个数小很多, 因此存储直方图所需的内存空间比存储预排序结果小得多; LightGBM 占用内存比 XGBoost 更小
- 同理, 在直方图上找分割点也比在整个数据集上找要快得多; LightGBM 达到和 XGBoost 相近的准确率, 所需训练时间更短

3.2 结果

我们取每一局 Dota2 比赛进行到 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% 时的 9 个状态, 作为 9 个独立的数据点. 我们在训练集上训练 LightGBM 模型, 迭代 300 轮, 每次迭代后在验证集上计算准确率, 保留准确率最高的模型; 我们尝试了多组超参数, 保留在验证集上准确率最高的一组:

1. 每棵决策树用 5% 的行 (数据点) 和 80% 的列 (特征) 训练
2. 每棵决策树至多有 60 个叶子节点; 每个叶子节点至少包括 600 个数据点
3. 学习率[6] 0.1
4. L2 正则化的 $\lambda = 0.3$

最后, 我们的 LightGBM 模型由 281 棵决策树组成, 在双路 Intel®Xeon®E5-2699 v3 CPU (18 核 36 线程) 上消耗 36.07 分钟即可完成训练, 其中 33.23 分钟用于预处理数据和提取特征, LightGBM 的训练仅用了 2 分 50 秒[7]. 模型在验证集上达到了 76.15% 的准确率, 在测试集上达到了 76.20% 的准确率.

3.3 模型解释

3.1.2节提到 LightGBM 每次都找出一个使得损失函数下降最多的方案来分裂叶子节点. 统计一个特征使得损失函数下降的总量, 就得到这个特征对于训练的贡献, 也就是这个特征的重要性. 图1列出了重要性排名前十五的特征:

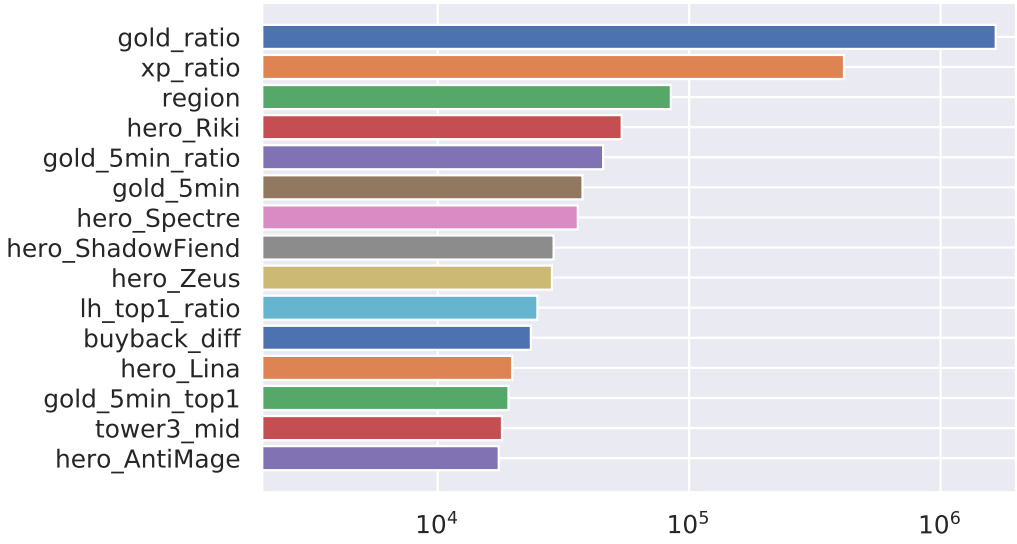

Figure 1: 重要性排名前十五的特征. 从高到低分别是: 双方总财产比例, 双方总经验比例, 比赛区域, 队伍中是否有力丸, 双方 5 分钟内金钱增量之比, 队伍 5 分钟内金钱增量, 队伍中是否有幽鬼, 队伍中是否有影魔, 队伍中是否有宙斯, 队伍中正补最多的玩家的正补数占全场的比例, 双方买活次数之差, 队伍中是否有莉娜, 队伍中 5 分钟内金钱增量最多的玩家的金钱增量, 中路三塔是否被摧毁, 队伍中是否有敌法师.

图1中的特征可以分为六类:

1. **金钱相关:** 双方总财产比例, 双方 5 分钟内金钱增量之比, 队伍 5 分钟内金钱增量, 队伍中正补最多的玩家的正补数占全场的比例, 队伍中 5 分钟内金钱增量最多的玩家的金钱增量

[6]也称作 shrinkage, 即每棵决策树乘以 0.1 再加到当前模型中
[7]我们也尝试过 sklearn 中的随机森林和 GBDT, 都比 LightGBM 慢至少 20 倍

- 前三个和团队财产相关, 后两个可近似认为是一号位打钱速度
- **原因:** Dota2 有很多强大而昂贵的属性型装备 (如蝴蝶, 代达罗斯之殇[8]) 和功能型装备 (如邪恶镰刀[9], 黑皇杖[10]). 一名装备领先的一号位英雄可以以一敌多
- **结论:** 金钱是决定 Dota2 胜负最重要的因素

2. **双方总经验比例**
   - **结论:** 很多 Dota2 玩家认为经验和金钱同等重要. 我们的模型认为, 经验虽然重要, 但它的重要性比金钱还是要差了很多, 即使是在推出了 "天赋" 系统的 "后 7.00" 版本
3. **地区:** 比赛区域
   - 即游戏所处的服务器. 比如中国大陆的玩家, 大多在完美世界代理的国服进行游戏
   - **原因:** 不同的国家和地区的文化不同; 大多数玩家都在一个相对固定的服务器游戏; 长期以来, 不同的游戏区域就形成了不同的游戏风格
4. **阵容:** 力丸[11], 幽鬼, 影魔, 宙斯, 莉娜[12], 敌法师
   - 力丸, 幽鬼, 宙斯刚好是 Dota2 该版本中胜率最高的英雄
   - 莉娜, 影魔是两个 Dota2 中单高爆发英雄, 也常常会遭到针对
   - 敌法师是 Dota2 最需要发育的英雄之一, 他的装备是否领先, 是队伍能否获胜的关键
   - 力丸, 幽鬼, 敌法师属于一号位英雄; 宙斯, 莉娜, 影魔属于二号位英雄
   - **结论:** 一号位和二号位的表现, 对胜率的影响更大
5. **双方买活次数之差**
   - "买活" 可以让玩家瞬间在泉水重生, 但是代价高昂: 大量金钱 (而且优先使用可靠金钱), 下一次复活时间延长 25 秒, 和 7 分钟的冷却时间
   - **原因:** 在双方看似旗鼓相当时, 如果其中一个队伍的核心成员都刚刚买活, 局势实际上对这个队伍非常不利: 一次团灭可能就是致命的
6. **中路三塔是否被摧毁**
   - **原因 1:** 三塔是高地塔, 地形易守难攻; 但一旦攻下, 就很难再阻挡对方的进攻
   - **原因 2:** 中路是三条路中最短的, 如果没有中路高地作为屏障, 一次团灭就可能被推平
   - **结论:** 中路高地塔是 Dota2 中除遗迹以外最重要的建筑

令人惊讶的是, 许多玩家重视的 "双方击杀数目比例" 特征, 即 "比分", 却榜上无名: 它对预测只有 5979.2 的贡献, 远不如正补数目重要. 因为 Dota2 的胜负并不像体育比赛一样取决于 "比分"; 如果仅仅杀得爽快, 追着对方的 "ATM" 提款, 却不压制对方核心英雄的经济, 而且己方核心英雄也不好好打钱, 那么所谓 "比分优势" 就是幻觉.

4 基于注意力 (Attention) 的深度神经网络

4.1 传统方法的弊端

大部分传统的机器学习算法中, 数据点是一个 $\mathbb{R}^d$ 中的向量. 但是 Dota2 比赛的状态是结构化的, 强行把状态表示成单个向量损害了模型的表达能力:

1. 第2节中提到, 我们把大部分 "个体特征" 都在队伍内排序. 这样做赋予了每个特征一个准确的含义, 但是打乱了个体特征之间的关系
   - 比如, 幽鬼最近 5 分钟有很多正补, 0 杀 0 死, 是好事; 光之守卫[13]最近 5 分钟有很多正补, 0 杀 0 死, 大概率是坏事
2. 游戏的 "事件" 不是一个固定维度的向量, 而是长度可变的离散时间序列

[8] 俗称 "大炮"
[9] 俗称 "羊刀"
[10] 简称 "BKB"
[11] 也被称作 "隐形刺客", 简称 "SA"
[12] 俗称 "火女"
[13] 俗称 "光法", 通常认为是辅助英雄

- 传统的做法是从时间序列中提取数值特征, 比如双方击杀 Roshan 次数, 拾取神符次数, 击杀英雄次数
- 这种提取损失了很多信息. 比如 “育母蜘蛛拾取了双倍伤害神符, 天辉的育母蜘蛛击杀了 Roshan, 夜魇的力丸拾取了不朽之守护, 力丸击杀了育母蜘蛛” 如果仅仅表示成 “拾取神符次数, 击杀 Roshan 次数, 击杀英雄次数”, 就显得过于苍白
- 要让模型直接从事件中 “理解” 游戏, 而不是从几个统计量中 “统计” 游戏

4.2 算法概述

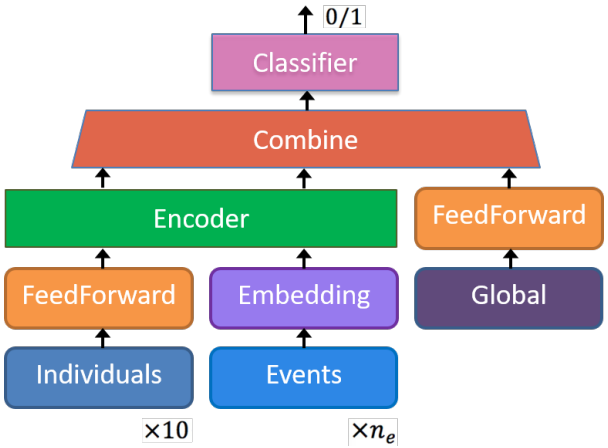

Figure 2: 我们的深度神经网络的总体框架. 个体特征和全局特征都通过多层前馈神经网络 (即全连接层) 提取成 $\mathbb{R}^d$ 中的向量. 分类器一般是逻辑回归 (Logistic Regression).

为了解决这两个问题, 我们设计了如图2所示的深度神经网络模型. 输入是 10 名玩家的个体特征, 全局特征, 以及发生过的游戏事件的离散编码. 神经网络的计算过程如下:

1. 用一个公共的 “个体特征提取器” 把每个玩家的个体特征变换为 $\mathbb{R}^d$ 中的向量 s_i, $i = 0, 1, \ldots, 9$, 称为**个体状态**
2. 用 “全局特征提取器” 把全局特征也变换为 $\mathbb{R}^d$ 中的向量 g, 称为**全局状态**
3. 把每种事件嵌入 (Embed) 到 $\mathbb{R}^d$ 中的一个向量 $e_i \in \mathbb{R}^d$, $0 \le i < n_e$. 嵌入层随机初始化, 再通过反向传播和其它参数一起优化.
4. 用 “编码器” 模块把个体特征和事件嵌入混合, 捕捉它们的相互关系
5. 把编码器的输出和全局特征通过某种方式合并, 得到一个 $\mathbb{R}^d$ 中的向量
6. 把上一步得到的向量输入分类器 C, 得到一个 0 到 1 之间的概率作为输出, 即队伍 1 的胜率

所有的模块都是可微的 (Differentiable), 通过反向传播可以得到各个参数的梯度, 然后使用各种改进的梯度下降法即可求出一个较好的局部最优解.

其中, “事件嵌入” 受到了自然语言处理领域的 “词嵌入 (即词向量)” 的启发, 因为事件序列和文本一样是变长的离散序列. 我们考虑的游戏事件有: 英雄击杀, 买活, 获取神符 (为了减少事件数目, 这里的神符不包括赏金符), 一血, 击杀 Roshan, 拾取不朽之守护, 摧毁建筑. 其中每个事件都再分为天辉和夜魇两种, 拾取每一种神符的事件编号也不同, 摧毁每一种不同的建筑的事件编号也不同. 因此总共有 58 种事件.

我们的目标是: 给定一个游戏状态的序列, 输出一个预测的胜率序列. 这是一个 “序列到序列 (seq2seq)” 的问题, 与机器翻译类似[14]. 因此我们参考了机器翻译领域目前效果最好的模型, Google 研究院于 2017 年发明的 Transformer[8]. 它彻底抛弃了低效的 RNN[15], LSTM 和过参数化

[14] 不过有一个重大区别: 预测一个游戏状态的胜率时看不到未来的状态; 我们在设计算法的过程中向神经网络隐藏了未来的状态, 防止出现 “未来函数”

[15] 循环神经网络

的 CNN[16]等结构, 完全基于 “多头自注意力 (Multihead Self-Attention)”, 具有参数少, 并行性高, 可解释的优势.

4.2.1 注意力 (Attention)

注意力是一种神经网络结构, 用于建模 “相关性查询”. 给定询问矩阵 $Q \in \mathbb{R}^{n_q \times d_k}$, 每一行都是一个询问, 由 d_k 维向量表示; 给定 key 矩阵 $K \in \mathbb{R}^{n_e \times d_k}$, 每一行是一个 key, 由 d_k 维向量表示; 给定 value 矩阵 $V \in \mathbb{R}^{n_e \times d_v}$, 每一行是一个 value, 由 d_v 维向量表示. 运算步骤如下:

1. 计算出 QK^T, 每个询问与每个 key 的内积, 即询问与 key 的相关程度
2. 乘以一个常数之后, 再对每一行分别做 softmax, 使得每一行都是一个 n_e 个元素上的概率分布; 第 i 行第 j 列的值, 就是第 i 个询问应该给予 V 中的第 j 个值多少 “注意力”
3. 把该矩阵乘以 V, 得到一个 $n_q \times d_v$ 的矩阵, 每一行都是 V 中所有行的凸组合

每个步骤都是可微的, 因此可以作为神经网络的一部分进行优化. 它的数学表述如下:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

自注意力 (Self-Attention) 指的是 Q, K, V 均为同一个矩阵的注意力模块, 可以用于捕捉一个集合 (或序列) 内各个元素之间的相互关系, 每个元素都从与之相关的其它元素那里获取一些信息.

传统的注意力机制弊端在于: softmax 生成的概率分布倾向于汇聚到单点, 而忽略其它的元素. 多头注意力 (Multihead Attention) 弥补了这一缺点. 它把每个询问, 键, 值都线性投影到 n_h 个不同的低维的子空间, 在子空间中分别做 Attention, 再把 n_h 个结果线性变换回原空间进行组合. 这就相当于有 n_h 个 “头” 在分别地进行询问, 容易看到更多不一样的元素:

$$\begin{aligned} h_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\ \text{MultiHead}(Q, K, V) &= \text{Concat}(h_1, h_2, \ldots, h_{n_h})W^O \end{aligned}$$

```
q = self.q_l(q).view(bs, n_q, self.n_h, self.d_k)
k = self.k_l(k).view(bs, n_e, self.n_h, self.d_k)
v = self.v_l(v).view(bs, n_e, self.n_h, self.d_k)
q = q.permute(0, 2, 1, 3).contiguous().view(bs * self.n_h, n_q, self.d_k)
k = k.permute(0, 2, 1, 3).contiguous().view(bs * self.n_h, n_e, self.d_k)
v = v.permute(0, 2, 1, 3).contiguous().view(bs * self.n_h, n_e, self.d_k)
if mask is not None:
    mask = mask.view(bs, 1, n_q, n_e)
    mask = mask.repeat(1, self.n_h, 1, 1)
    mask = mask.view(bs * self.n_h, n_q, n_e)
x, _ = attend(q, k, v, mask=mask)
x = x.view(bs, self.n_h, n_q, self.d_k)
x = x.permute(0, 2, 1, 3).contiguous().view(bs, n_q, self.d)
x = self.o_l(x)
```

Figure 3: 多头注意力机制模块. 节选自 model.py

4.2.2 编码器模型

Transformer 分为编码器和解码器两部分, 编码器把文本变换为一个隐状态序列, 解码器把隐状态序列变换为另一种语言的文本. 我们的算法主要参考了编码器部分.

编码器的输入是 $s_i \in \mathbb{R}^d$, $i = 0, 1, \ldots, 9$, 即每个玩家的个体状态; 还有 $\{e_i\}_0^{n_e-1}$, $e_i \in \mathbb{R}^d$, 即事件序列的嵌入.

如图4所示, 编码器由若干层组成. 第 i 层接受输入 $\{s^{(i-1)}\}^{10}$ 和 $\{e^{(i-1)}\}^{n_e}$, 其中 $s^{(0)} = s, e^{(0)} = e$. 每一层有四个多头注意力模块, 它们分别是:

[16]卷积神经网络

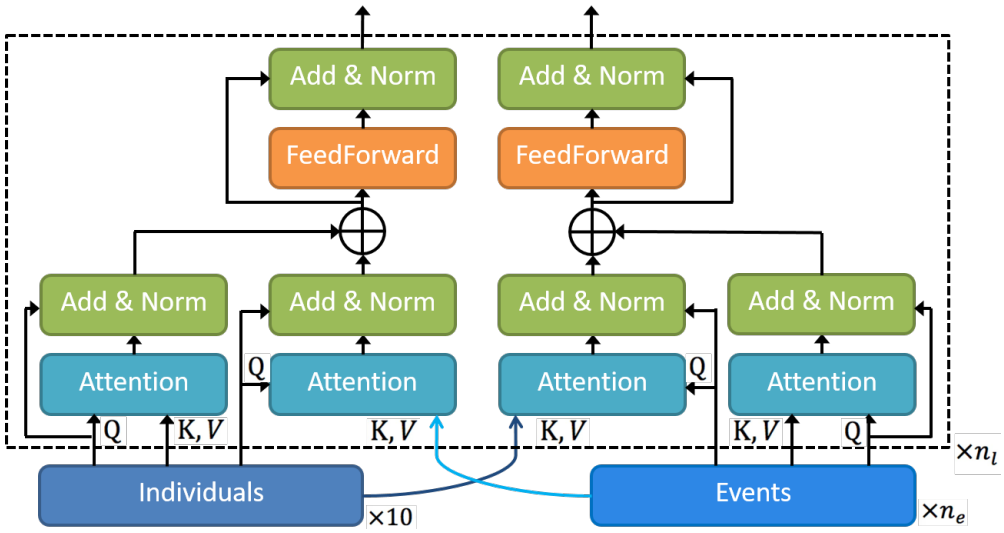

Figure 4: 编码器由若干个编码器层堆叠而成, 每一层的输出和输入有相同的形状. “Add & Norm” 层由一个残差连接和一次层规范化组成.

- $\{s^{(i-1)}\}^{10}$ 对 $\{s^{(i-1)}\}^{10}$ 的自注意力 (反映玩家之间的互相关系)
- $\{e^{(i-1)}\}^{n_e}$ 到 $\{e^{(i-1)}\}^{n_e}$ 的自注意力 (这里限制了每个事件只能看到它之前发生的所有事件, 反映事件之间的相互关系)
- $\{s^{(i-1)}\}^{10}$ 对 $\{e^{(i-1)}\}^{n_e}$ 的注意力 (这里限制了每个玩家个体状态只能看到与他相关的状态事件, 反映事件对玩家状态的影响)
- $\{e^{(i-1)}\}^{n_e}$ 对 $\{s^{(i-1)}\}^{10}$ 的注意力 (这里限制了每个事件只能看到事件发生时刻的与该事件相关的玩家的个体状态, 反映玩家状态对事件的影响)

注意力模块之后的运算如下:

1. 每个注意力模块后都有与输入的残差连接 (Residual Connection)[4], 使得梯度更容易传播, 加快训练
2. 残差连接之后增加了层规范化 (Layer Normalization)[1], 防止深层神经网络训练过程中 “梯度消失” 和 “梯度爆炸” 的现象, 能够让训练又快又稳定
3. 把结果按照 Attention 的询问两两合并
4. 每个 d 维向量分别通过通过前馈神经网络
5. 再次进行残差连接和层规范化, 就得到了一个编码器层的输出 $\{s^{(i)}\}^{10}$ 和 $\{e^{(i)}\}^{n_e}$

```
s = s.view(bs * max_t, 10, self.d)
s_s, _ = self.s_s_att(s, s, s)
s = s.view(bs, max_t * 10, self.d)
s_e, _ = self.s_e_att(s, e, e, mask=s_e_mask.view(bs, max_t * 10, max_e))
e_s, _ = self.e_s_att(e, s, s, mask=e_s_mask.view(bs, max_e, max_t * 10))
e_e, _ = self.e_e_att(e, e, e, mask=e_e_mask)
s = s_s.view(bs, max_t, 10, self.d) + s_e.view(bs, max_t, 10, self.d)
e = e_s + e_e
s = self.s_ff(s)
e = self.e_ff(e)
```

Figure 5: 编码器层. 节选自 model.py

把这个模块堆叠 n_l 层, 就得到了完整的编码器.

4.3 用注意力合并状态

编码器的输出 $\{s^{(n_l)}\}^{10}$ 和 $\{e^{(n_l)}\}^{n_e}$ 是个体特征和事件序列经过一系列充分的变换得到的, 把它们与全局状态合并, 变成单个 d 维向量, 就能放入分类器分类了. 这里我们再次利用注意力机制.

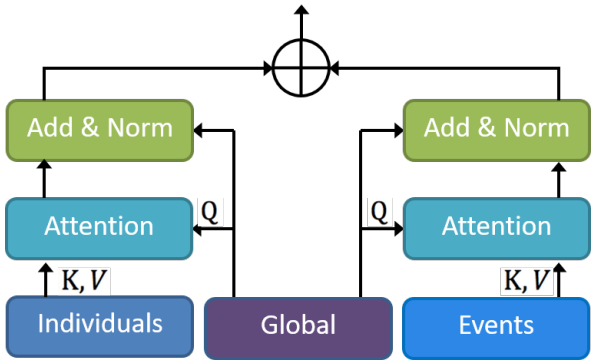

Figure 6: 三种 d 维向量的合并: 计算全局状态到编码器的两个输出的注意力, 然后相加.

如图6所示, 以全局状态 $g \in \mathbb{R}^d$ 为询问, 以 $\{s^{(i)}\}^{10}$ 为 key 和 value, 用一个多头注意力把它们合并成一个 d 维向量; 再以全局状态 $g \in \mathbb{R}^d$ 为询问, 以 $\{e^{(i)}\}^{n_e}$ 为 key 和 value, 用一个多头注意力把它们合并成一个 d 维向量; 把这两个向量相加, 得到了一个 d 维向量, 放入分类器 C 中分类即可得出预测的胜率.

4.3.1 时间编码

原始的注意力机制中, 元素的顺序对输出没有影响. 如果不用 CNN 来捕获局部性, 也不用 RNN 来捕获时间顺序, "序列编码器" 就退化成了 "集合编码器", 事件发生的时间顺序被完全忽视了. 自然语言处理领域的学者为了解决这个问题, 提出了 "位置嵌入 (Positional Embedding)"[3] 和 "正弦曲线时间编码 (Sinusoid Temporal Encoding)" 两种方法. 我们使用的就是后者:

$$PE_{(t,2i)} = \sin(\frac{t}{10000^{2i/d}})$$
$$PE_{(t,2i+1)} = \cos(\frac{t}{10000^{2i/d}})$$

在经过编码器之前, 我们就把它加到 $\{s^{(0)}\}^{10}$ 和 $\{e^{(0)}\}^{n_e}$ 上, 使编码器的输入带有和时间相关的信息.

4.4 训练与结果

对于神经网络的结构, 我们使用如下超参数:

- 每个前馈神经网络 (Feed Forward) 有 2 层, 使用 ReLU 激活函数 [7]
- 各个特征向量的维度 $d = 256$
- 带有激活函数的隐藏层的维度 $d_{hid} = 320$
- 多头注意力机制的 "头" 的数目 $n_h = 4$
- 编码器的层数 $n_l = 2$

对每一场比赛的每一分钟, 都进行一次预测, 并和比赛的真实结果对比, 以它和真实结果的交叉熵 (Cross Entropy, 即 Bernoulli 分布的对数似然函数的相反数) 作为这次预测的 loss. 一场比赛的总 loss, 就是每分钟的预测的 loss 的算术平均. 我们使用这个 loss 来训练, 即:

$$
\begin{aligned}
L(y, \{\hat{y}_t\}_1^T) &= \frac{1}{T}\sum_t \text{CrossEntropy}(y, \hat{y}_t) \\
&= -\frac{1}{T}\sum_t \left(y\log(\hat{y}_t) + (1-y)\log(1-\hat{y}_t)\right)
\end{aligned}
$$

神经网络的所有模块都是可微的, loss 函数关于任意一个参数的微分都可以通过反向传播求得, 所以可以用一阶方法来优化参数. 我们在验证集上经过试验, 确定使用 Adam[6] 算法作为优化器. Adam 是小批次随机梯度下降算法 (Mini-Batch Stochastic Gradient Descent) 的一个最著名的变体, 在非凸函数上收敛速度快, 而且鲁棒性较好, 对学习率设置不敏感, 广泛应用于深度学习中. 优化器的超参数设置如下:

- 批次大小为 32
- 学习率 $\alpha = 0.0003$
- momentum $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- 共训练 4 个 epoch
- 如果一个 epoch 内验证集上的 loss 不下降, 就把学习率降低到原来的 0.25 倍
- 所有参数都有 3×10^{-6} 的权值衰减 (weight decay, 等价于 L2 正则化)

在验证或测试时, 我们使用和 LightGBM 模型同样的 setting, 即取游戏进行到 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% 时的 9 个状态作为 9 个数据点, 统计总准确率.

每一 epoch 结束时, 都记录它在验证集上的准确率, 保留准确率最高的模型. 最后在测试集上测试并报告结果.

我们的模型在单 Nvidia Titan Xp GPU 上消耗 5 时 47 分即可完成训练, 最终模型有 3722625 个参数, 在验证集上达到了 76.66% 的准确率, 在测试集上达到了 76.61% 的准确率.

5 模型对比

	LightGBM	深度神经网络
准确率	76.20%	76.61%
硬件需求	普通 CPU	高端 GPU
训练时间	36 分 4 秒	5 时 47 分
参数数目	0.84M	3.72M
数据需求	小	大

Table 2: 基于决策树的 LightGBM 模型与基于注意力的深度神经网络模型的对比

表2对比了我们的两个模型. LightGBM 模型对数据的需求量更小, 只有 10000 组数据也能得到不错的结果, 而且训练耗时短, 成本低; 深度神经网络模型需要大量的数据来对抗过参数化带来的过拟合, 而且训练耗时耗力, 但是潜力极大. 这次量化大赛只有短短的一个月时间, 因此我们只收集了 16 万场天梯比赛的数据. 我们相信, 只要用更多的数据, 构建更深的神经网络, 这个模型的表现还能再提高. 我们将在比赛结束后继续进行这方面的研究.

6 模型集成与实验结果

我们的两个模型在设计上有非常大的差异, 因此把它们集成 (Ensemble) 在一起, 可进一步提升预测的准确率. 对于每一个游戏状态, 我们用两个模型分别预测, 各得到一个 0 到 1 之间的概率. 我们把两个模型的输出简单地做算术平均, 作为最终的输出. 模型集成后, 准确率达到了 76.78%, 超过了任何单个模型.

表3详细列出了我们的算法在游戏进行到 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% 时的预测准确率, 并与 Dota2 官方的收费数据服务 DotaPlus 进行了对比. 可以发现我们的预测准确率在大多数情况下要高于 DotaPlus, 尤其是在游戏前中期.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	Avg
LightGBM	0.643	0.675	0.701	0.720	0.751	0.786	0.819	0.860	0.905	0.762
Deep	0.647	0.684	0.705	**0.729**	0.749	0.788	0.820	0.862	0.910	0.766
Ensemble	**0.647**	**0.684**	**0.707**	0.728	**0.755**	**0.791**	**0.824**	**0.864**	0.910	**0.768**
DotaPlus	0.417	0.514	0.597	0.611	0.681	0.750	0.778	0.847	**0.944**	0.682

Table 3: 我们的两个模型, LightGBM, 深度神经网络, 以及这两个模型的集成 (Ensemble), 与 Dota2 的收费数据服务 DotaPlus 的比较. DotaPlus 的预测准确率数据来源于 [9] (这篇论文的数据集与我们的数据集不完全相同)

7 案例分析

我们挑选了两场比较有代表性的 Dota2 天梯比赛, 来评估我们的模型的预测结果是否合理.

7.1 比赛 4162609310

第一局是编号为 4162609310 的天梯比赛, 位于新加坡服务器 (即东南亚), 组别为 High, 大部分玩家的等级在 Legend 4 (一代传奇 4) 到 Ancient 5 (万古流芳 5) 之间. 天辉方的阵容是工程师[17], 复仇之魂, 帕吉[18], 恐怖利刃[19], 修补匠; 夜魇方的阵容是影魔, 沙王, 暗影萨满[20], 食人魔魔法师[21], 幽鬼. 比赛共持续 44 分 30 秒[22].

Figure 7: 比赛 4162609310. 我们的模型的预测结果与 Dota Plus 的对比. 图中黄线代表双方财产总和之差, 蓝线代表双方经验总和之差, 绿线是 Dota Plus 的预测结果, 红线是我们的双模型集成得到的预测结果

图7是我们的模型和 Dota Plus 的预测结果. 可以看出, Dota Plus 在游戏中期 (16:00 到 32:00) 错误地认为夜魇的胜算更大, 但是我们的模型除了 18:00 到 23:00 中出现了错判以外, 都比较坚定地认为天辉终将胜利. 在游戏后期, 两者差别不大.

为了理解预测结果出现差别的原因, 我们选取两者预测差距悬殊的 30:00. 此时, 我们的模型认为天辉有 80.39% 的胜算, 而 Dota Plus 却认为夜魇的有近 70% 的胜算. 我们利用深度神经网络模型

[17] 俗称 “炸弹人”
[18] 俗称 “屠夫”
[19] 简称 “TB”
[20] 俗称 “小 Y”
[21] 俗称 “蓝胖”
[22] 在第 42 分 29 秒结束, 多出来的两分钟是从选英雄到出兵之间的时间

的注意力机制, 把它在这一时刻对于不同的玩家和事件的注意力值可视化, 就能看到它做出判断的依据.

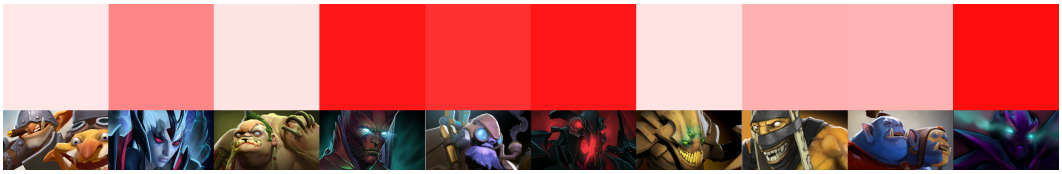

Figure 8: 比赛 4162609310. 30:00 时, 神经网络的全局状态对每个玩家的个体状态的注意力. 红色越深代表对这个玩家的注意力越大.

如图8所示, 我们的模型预测天辉胜利, 主要依据是天辉方恐怖利刃, 修补匠的表现和夜魇方影魔, 幽鬼的表现. 观看这局比赛的录像后, 我们发现这四位玩家在 30:00 前的表现分别是这样的:

1. 幽鬼 (夜魇): 总财产 13305, 全场第四, 不如同一队的辅助英雄暗影萨满. 前期一路顺风, 但出装以小件和肉装为主. 到 30:00, 身上装备是假腿, 先锋盾, 刃甲, 天鹰之戒, 魔抗斗篷. 还未做出核心的打钱/输出装: “辉耀”
2. 影魔 (夜魇): 总财产 15028, 全场第二. 中单对线优势, 补刀完压对方的修补匠, 前 22 分钟的正补在全场遥遥领先. 然而, 在大优势下, 他的出装思路并非主流: 假腿天鹰之后, 做出高分局并不常见的隐刀; 为了防止被对方控制针对, 于 26:55 做出能够每 13 秒抵挡一次指向性技能的 “林肯法球”, 而不是选择更加便宜, 更加主流的 “黑皇杖 (BKB)” 来反制对面的控制
3. 恐怖利刃 (天辉): 总财产 17124, 全场第一, 也以 250 个正补高居全场第一. 9 级就点满了镜像, 在此之后线野双收, 出装速度越来越快, 8:56 假腿, 14:54 魔龙枪, 19:45 夜叉, 27:43 冰眼, 在攒钱出鹰角弓 (合成顶级输出装 “蝴蝶” 的配件). 出装思路主流得有些 “功利”, 有肉有输出
4. 修补匠 (天辉): 前期对线被影魔压制, 13:05 才憋出远行鞋[23], 20:53 时终于集齐闪烁匕首, 灵魂之戒, 远行鞋三件套, 步入正轨. 26:25 做出 “以太之镜”, 可以增大指向性技能的施法距离. 虽然击杀数不多, 但是贡献了大量英雄伤害

从全局状态对于各个事件的注意力来看, 注意力较高的两个事件分别是 “26:56 恐怖利刃击杀了沙王” 和 “29:35 恐怖利刃击杀了暗影萨满”. 这两个事件表明恐怖利刃装备已经成型, 开始具有作战能力.

做为一名 Dota2 玩家, 我的预测和我们的模型一致, 就是天辉赢面更大. 夜魇方虽然财产总和有优势, 但是集中在非主流出装的影魔和辅助英雄暗影萨满身上, 一号位幽鬼本来就不肥, 而且还出肉装, 刷不起来. 反之, 天辉方的恐怖利刃韬光养晦, 拼命刷钱, 总财产早就默默超过了敌方的影魔, 出装也稳扎稳打; 中单修补匠虽然前期被压制, 无法起到带动全场节奏的作用, 但是很有团队精神, 没有出路人局常见的抢人头装备 “达贡之神力[24]”, 而是做出 “以太之镜” 之后开始憋 “邪恶镰刀[25]”, 以针对敌方的核心英雄.

后面游戏的走势果然符合我们模型的预测. 恐怖利刃成了天辉的主力, 游戏结束时以 20 杀 5 死 13 助攻拿下全场最佳战绩. 天辉的恐怖利刃和修补匠的出装仍然稳扎稳打: 在夜魇方的幽鬼带盾击杀天辉三人后, 恐怖利刃果断做出提供短时间魔法免疫的黑皇杖, 修补匠果断做出邪恶镰刀, 限制幽鬼的发挥, 一波把对方团灭后获得了游戏的胜利.

7.2 比赛 4161373900

图9是在另一局 Normal 级别天梯比赛中我们的模型与 Dota Plus 的对比. 我们的模型自始至终都是预测正确的, 即使是在 10:00 到 20:00 总财产和总经验双双落后的情况下仍未大幅度动摇. 而 Dota Plus 的预测则是跟着总财产和总经验曲线跑的, 出现了错误的预测.

[23] 俗称 “飞鞋”, 可以全图传送到小兵或建筑上
[24] 简称 “红杖”, 可以对单体瞬间造成大量伤害
[25] 俗称 “羊刀”, 造成 3.5 秒的沉默, 缴械, 禁用物品和减速, 是一件强力控制道具

Figure 9: 比赛 4161373900 中我们的模型的预测结果与 Dota Plus 的对比. 图中绿线是 Dota Plus 的预测结果, 红线是我们的双模型集成得到的预测结果

7.3 总结与对比

相比 Dota Plus, 我们的模型主要有以下几个优点:

- Dota Plus 的预测往往与总财产和总经验高度相关, 注重全局特征, 忽视了每个个体的打法和心态, 所以经常在中期错判比赛趋势. 我们的模型不仅考虑全局特征, 还通过注意力机制聚焦到了玩家个体: 有主次之分, 而不只是简单地把数值指标加总, 更符合人类分析 Dota2 比赛时的思路
- 我们的模型考虑更多的因素, 对特征的变换高度非线性, 预测能力更好; 因为 Dota2 游戏的数据非常容易获得, 我们使用大量数据来训练, 使得复杂模型也不会过拟合
- Dota Plus 是官方的收费服务, 一般玩家在使用时看不到哪些因素对预测结果有贡献, 是一个 "黑盒". 我们的 LightGBM 模型统计了每个特征贡献的损失函数减小量, 帮助我们理解各个特征的重要性; 深度神经网络模型通过可视化注意力, 帮用户更好地理解它的预测

我们的模型也存在一些问题:

- 比赛刚开始的时候, 模型的预测会出现大幅度的波动. 因为此时很多带有 "比例" 的特征, 即使已经经过了 Laplace 平滑, 仍然会出现比较极端的值. 我们在验证和测试的时候都从 10% 开始采样游戏状态, 所以这个问题一直到最后评估模型的时候才表现出来. 在接下来的研究中, 我们会通过多种平滑手段解决这一问题.
- 由于时间原因, 我们没有足够的时间做特征提取, 因此还有很多因素没有考虑进去. 比如:
  - 技能加点
  - 黑皇杖目前的持续时间
  - 永久性 Buff 的层数 (比如堆积腐肉, 智慧之刃, 决斗)

  增加了这些特征, 我们模型可能会有更好的表现

References

[1] J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[2] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[3] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[9] L. Yu, D. Zhang, X. Chen, and X. Xie. Moba-slice: A time slice based evaluation framework of relative advantage between teams in MOBA games. *CoRR*, abs/1807.08360, 2018.