PLE-Cluster
기술 참조서

Progressive Layered Extraction · CGC Gate · GroupTaskExpertBasket · 16 Tasks × 4 Groups × 20 Clusters

프로젝트

AIOps 추천 시스템

PLE-adaTT + Cluster Head

버전

v3.14

2026-03-05

이 문서는 AIOps 추천 시스템의 핵심 모델인 PLE-Cluster-adaTT의 전체 아키텍처, 각 모듈의 이론적 근거, 수학적 구조, 입출력 사양, 손실 함수 설계, 디버깅 가이드를 상세히 기술한다. 7개 Shared Expert(v3.15, RawScale 제거)의 결합, CGC 게이팅, HMM Triple-Mode 라우팅, GroupEncoder + ClusterEmbedding (4 그룹, 20 클러스터), Logit Transfer, 16개 태스크 타워를 포함하는 복합 멀티태스크 학습 모델의 진실의 원천(Single Source of Truth)이다.

목 차

PLE-Cluster-adaTT 전체 아키텍처

전체 데이터 흐름

아래 다이어그램은 `PLEClusterAdaTT.forward()` (라인 1185 1363)의 전체 추론 파이프라인을 도식화한 것이다.

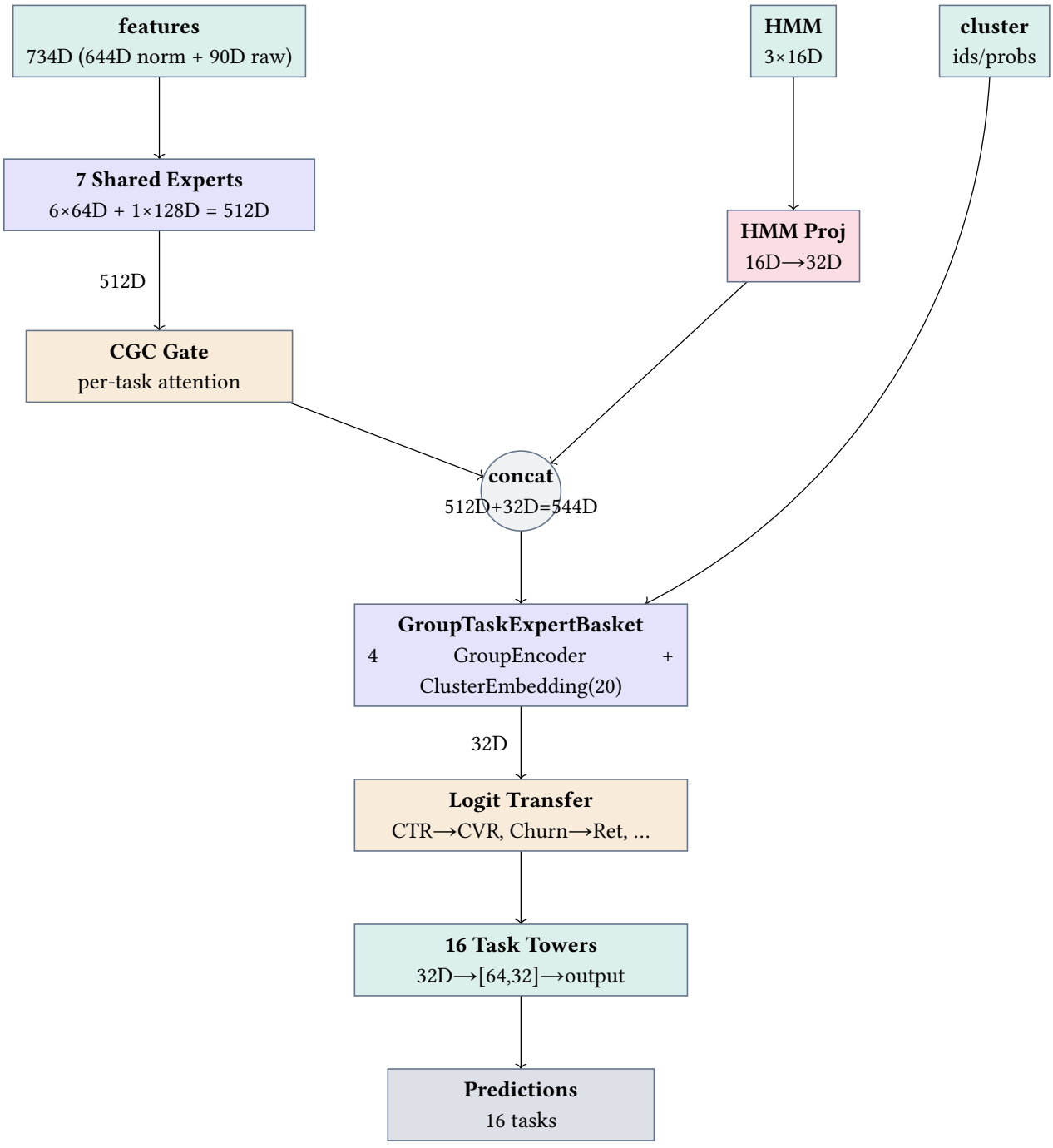

그림 1: PLE-Cluster-adaTT 전체 추론 파이프라인

핵심 설계 철학 – 태스크 간 지식 공유와 간섭 방지의 균형

PLE의 핵심은 Shared Expert와 Task-specific Expert를 명시적으로 분리하되, CGC 게이팅으로 태스크별 최적 결합 비율을 학습하는 것이다. 이를 통해 MMoE의 `Expert Collapse` 문제와 Shared-Bottom의 `Negative Transfer` 문제를 동시에 완화한다. 본 시스템은 이 아이디어 위에 GroupEncoder + ClusterEmbedding(4 그룹, 20 클러스터), HMM Triple-Mode 라우팅, Logit Transfer를 추가하여 확장하였다.

역사적 배경

멀티태스크 학습(MTL)의 학문적 계보는 다음과 같다:

- **1993**: Caruana가 MTL의 개념을 처음 체계화 (박사 논문, CMU)
- **1997**: Caruana가 Shared-Bottom 구조를 **Machine Learning** 저널에 공식 발표
- **1991/2017**: Jacobs et al.이 Mixture of Experts를 제안 / Shazeer et al.이 Sparsely-Gated MoE로 LLM에 적용
- **2018**: Ma et al.이 **MMoE**를 Google에서 KDD 2018에 발표 (YouTube 추천)
- **2020**: Tang et al.이 **PLE**를 Tencent에서 RecSys 2020에 발표 (동영상 추천)
- **2021-현재**: PLE 변형들이 산업계에서 광범위하게 채택

본 시스템은 PLE(2020)를 기반으로 Uncertainty Weighting(Kendall et al., CVPR 2018), Focal Loss(Lin et al., ICCV 2017), Evidential DL(Sensoy et al., NeurIPS 2018) 등 각 분야의 최신 기법을 통합한 복합 아키텍처이다.

최신 동향

2024-2025년 멀티태스크 추천 시스템의 핵심 트렌드:

- **PLE** 변형 연구: Tencent의 **AITM** (Adversarial-Improved PLE, CIKM 2021)과 Alibaba의 **ESMM2** (Entire Space Multi-Task Model v2, 2023)가 PLE의 게이팅을 전체 변환 퍼널(impression→click→conversion→purchase)로 확장
- **Foundation Model + MTL**: LLM을 backbone으로 사용하고 태스크별 adapter를 연결하는 **P5** (Geng et al., RecSys 2022), **InstructRec** (Zhang et al., 2024) 등 등장
- **Gradient** 기반 최적화: PCGrad (Yu et al., NeurIPS 2020), CAGrad (Liu et al., NeurIPS 2021), Nash-MTL (Navon et al., ICML 2022) 등 gradient 충돌을 직접 해결하는 방법 연구 활발
- 산업 적용 규모: Meta(Instagram, Facebook), Google(YouTube), TikTok(ByteDance), Spotify 등에서 PLE 계열 MTL이 수억 사용자 규모로 운영 중

주요 사양 요약

항목	값
Shared Expert 수	7개 (unified_hgcn, perslay, deepfm, temporal, lightgcn, causal, optimal_transport) — v3.15 RawScale 제거
Shared 결합 차원	512D (6×64D + 1×128D)
활성화 태스크 수	16개 (전체 18개 정의, uplift/category_uplift 비활성화)
GMM 클러스터 수	20개
HMM 모드 수	3개 (journey, lifecycle, behavior) × 16D ⟶ 32D 프로젝션
Task Expert 아키텍처	GroupTaskExpertBasket (v3.2, 88% 파라미터 감소)

Task Expert 출력	32D per task
Logit Transfer 쌍	5개 (CTR→CVR, Churn→Retention, CVR→LTV, NBA→Spending, Spending→Brand)
Loss Weighting	Uncertainty Weighting (Kendall et al.)
입력 피처 차원	734D (644D normalized + 90D raw power-law) + 별도 입력 (hyperbolic 20D, HMM 48D, TDA 70D)
CGC 게이팅	태스크별 nn.Linear(576→8) + Softmax
adaTT	4개 태스크 그룹, gradient 기반 적응적 전이

심층 기초 해설 – PLE의 근본 아이디어와 수학적 직관

이 장은 PLE(Progressive Layered Extraction)를 처음 접하거나, 수식은 읽었지만 "그래서 왜 이렇게 설계하는가"가 와닿지 않는 실무자를 위한 해설이다. 멀티태스크 학습의 근본 동기부터 시작하여, Shared-Bottom → MMoE → PLE로의 진화를 하나의 이야기로 관통하고, 본 문서에 등장하는 핵심 수식 각각이 실제로 무엇을 말하고 있는지 직관적으로 풀어낸다.

왜 멀티태스크 학습인가 – 단일 모델로는 부족한 이유

근본 동기: 하나를 알면 다른 것도 더 잘 안다

AIOps 추천 시스템은 CTR(클릭률), CVR(전환율), Churn(이탈), LTV(생애가치) 등 수십 가지 예측을 동시에 수행해야 한다. 직관적으로 생각하면 태스크마다 독립 모델을 만드는 것이 가장 자연스럽다. 하지만 실제로는 태스크들이 서로 관련되어 있다.

- CTR이 높은 고객은 CVR도 높을 가능성이 크다 (퍼널 관계)
- Churn 위험이 높은 고객은 Retention이 낮다 (역상관)
- 소비 패턴(Spending)은 LTV를 결정하는 핵심 신호다

이 관련성을 활용하면 데이터 효율이 극적으로 올라간다. 태스크 A를 학습하면서 발견한 패턴이 태스크 B에도 유용한 경우, 동일한 데이터로 더 풍부한 표현을 학습할 수 있다. 이것이 멀티태스크 학습(MTL)의 핵심 동기다.

> **비유: 외국어 학습**
>
> 스페인어를 배운 사람이 이탈리아어를 더 빨리 습득하는 것과 같다. 두 언어는 어근, 문법 구조, 발음 규칙을 상당 부분 공유한다. 하나를 학습하면서 획득한 "라틴어 계열 언어의 구조 이해"가 다른 언어 학습에 전이(transfer)된다. MTL에서 Shared Expert가 바로 이 "공통 구조 이해"를 담당한다.

통계적 관점: 귀납적 편향과 정규화

단일 태스크 모델은 해당 태스크의 데이터만으로 학습하므로 과적합(overfitting) 위험이 높다. MTL은 여러 태스크가 공유 표현을 통해 서로의 학습을 암묵적으로 정규화한다.

$$\mathcal{L}_{\text{MTL}} = \sum_{k=1}^{K} w_k \cdot \mathcal{L}_k\left(f_k\left(\boldsymbol{h}_{\text{shared}(\boldsymbol{x})}\right)\right)$$

이 총 손실을 최소화할 때, $\boldsymbol{h}_{\text{shared}}$는 어느 한 태스크에만 과적합하기 어렵다. 모든 태스크에 동시에 유용한 표현만이 $\boldsymbol{h}_{\text{shared}}$에 살아남는다. 이것은 L2 정규화나 Dropout과는 다른, 태스크 간 상호 정규화**(inter-task regularization)**다.

> **학부 수학**
>
> 가중합**(Weighted Sum)**이란? $\mathcal{L}_{\text{MTL}} = \sum_{k=1}^{K} w_k \cdot \mathcal{L}_k$에서 w_k는 태스크 k의 가중치(중요도)이고, $\mathcal{L}_k$는 해당 태스크의 손실이다. 가중합은 "각 항목에 중요도를 곱한 뒤 더하는 것"으로, 일상에서 학점 평균과 같은 원리다. 구체적 예시: 국어 90점(가중치 2), 수학 80점(가중치 3)이면 가중 평균 = $\frac{2\times90+3\times80}{2+3} = \frac{420}{5} = 84$점 – 수학에 더 비중을

둔 평균이다. MTL에서도 $w_{\text{CVR}} = 1.5$, $w_{\text{CTR}} = 1.0$이면 CVR 태스크의 손실이 총 손실에 1.5배 더 많이 반영되어, 모델이 CVR 예측 정확도를 더 중시하게 된다. $\boldsymbol{h}_{\text{shared}}$의 학습은 이 가중합의 gradient $\nabla \mathcal{L}_{\text{MTL}}$로 이루어지므로, w_k가 큰 태스크의 gradient가 공유 표현에 더 강하게 영향을 미친다.

Negative Transfer – 멀티태스크의 어두운 면

문제 정의

모든 태스크를 함께 학습하면 항상 좋을까? 아니다. **Negative Transfer**는 관련성이 낮은 태스크가 공유 표현을 오염시켜 오히려 단일 태스크 모델보다 성능이 하락하는 현상이다.

Seesaw 현상 – 실무에서의 경험

CTR 성능을 올리면 Churn 성능이 떨어지고, Churn을 올리면 CTR이 떨어지는 시소(seesaw) 현상은 MTL의 가장 흔한 실패 모드다. 이는 두 태스크의 gradient가 공유 파라미터 공간에서 서로 반대 방향을 가리키기 때문이다.

최적화 관점: Gradient 충돌

태스크 k의 손실을 $\mathcal{L}_k$라 하자. 공유 파라미터 $\boldsymbol{\theta}_{\text{shared}}$에 대한 각 태스크의 gradient는 다음과 같다.

$$\boldsymbol{g}_k = \nabla_{\boldsymbol{\theta}_{\text{shared}}} \mathcal{L}_k$$

두 태스크의 gradient가 같은 방향을 가리키면 협력(positive transfer)이고, 반대 방향을 가리키면 충돌(negative transfer)이다.

$$\cos(\boldsymbol{g}_i, \boldsymbol{g}_j) = \frac{\boldsymbol{g}_i \cdot \boldsymbol{g}_j}{\|\boldsymbol{g}_i\| \cdot \|\boldsymbol{g}_j\|}$$

- $\cos > 0$: 태스크 i와 j의 gradient가 협력 – 함께 학습하면 이득
- $\cos < 0$: gradient가 충돌 – 공유 파라미터가 양쪽 모두에 해로운 방향으로 업데이트
- $\cos \approx 0$: 무관 – 함께 학습해도 큰 효과 없음

학부 수학

내적(**Dot Product**)과 코사인 유사도: 두 벡터 $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$의 내적은 $\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i$이다. 기하학적으로 $\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\| \cdot \cos\theta$이므로, 코사인 유사도 $\cos\theta = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\|}$는 두 벡터의 방향적 유사성을 $[-1, 1]$ 범위로 측정한다. 구체적 예시: 2차원에서 $\boldsymbol{g}_{\text{CTR}} = (3, 1)$, $\boldsymbol{g}_{\text{CVR}} = (2, 2)$이면 $\cos = \frac{3\times2+1\times2}{\sqrt{10}\times\sqrt{8}} = \frac{8}{8.94} \approx 0.89$ (강한 협력). $\boldsymbol{g}_{\text{Churn}} = (-1, 2)$이면 $\cos = \frac{3\times(-1)+1\times2}{\sqrt{10}\times\sqrt{5}} = -\frac{1}{7.07} \approx -0.14$ (약한 충돌). 이처럼 gradient 벡터의 코사인 유사도가 음수면, 공유 파라미터를 업데이트할 때 한 태스크의 개선이 다른 태스크의 악화로 이어져 Negative Transfer가 발생한다.

직관적 해석: 줄다리기

공유 파라미터는 여러 태스크가 동시에 잡아당기는 줄과 같다. 모든 태스크가 같은 방향으로 당기면 빠르게 전진하지만, 반대 방향으로 당기면 제자리걸음이거나 오히려 후퇴한다. PLE의 핵심 아이디어는 "각 태스크에게 자기

만의 줄을 하나 더 주는 것"이다. Shared Expert는 공용 줄, Task-specific Expert(본 구현의 GroupTaskExpertBasket)는 태스크 전용 줄에 해당한다.

정보이론 관점: 태스크 간 상호정보량

태스크 A와 B의 레이블을 확률 변수 Y_A, Y_B라 하면, 두 태스크의 관계는 상호정보량(mutual information)으로 측정할 수 있다.

$$I(Y_A; Y_B) = \sum_{y_a, y_b} p(y_a, y_b) \cdot \log \frac{p(y_a, y_b)}{p(y_a) \cdot p(y_b)}$$

- $I(Y_A; Y_B)$가 높으면: 두 태스크가 공통 정보를 많이 공유 → 같은 Expert가 유용
- $I(Y_A; Y_B)$가 낮으면: 독립적 → 강제로 공유하면 noise만 주입

이상적인 MTL 아키텍처는 I가 높은 태스크끼리는 표현을 공유하고, I가 낮은 태스크끼리는 분리해야 한다. PLE의 Shared/Task-specific Expert 분리와 CGC 게이팅이 바로 이 원칙을 아키텍처 수준에서 구현한 것이다.

학부 수학

상호정보량**(Mutual Information)**과 **KL-Divergence**의 관계: 상호정보량 $I(X;Y)$는 실은 두 분포 간의 **KL-Divergence**로 표현된다: $I(X;Y) = D_{\mathrm{KL}(p(x,y) \,\|\, p(x)p(y))}$. KL-Divergence $D_{\mathrm{KL}(P \,\|\, Q)} = \sum P(x) \log \frac{P(x)}{Q(x)}$는 "분포 P를 분포 Q로 근사할 때 발생하는 정보 손실"이다. 따라서 $I(X;Y)$는 "결합 분포 $p(x,y)$와 독립 가정 $p(x)p(y)$ 사이의 정보 손실", 즉 "X와 Y가 독립이 아님으로 인해 생기는 추가 정보"를 측정한다. 구체적 예시: CTR과 CVR의 경우, 클릭한 고객이 전환할 확률이 높으므로 $p(\text{click}=1, \text{convert}=1) \gg p(\text{click}=1) \times p(\text{convert}=1)$이고, $I(\text{CTR}; \text{CVR})$이 크다. 반면 CTR과 Brand_prediction은 상대적으로 독립적이어서 I가 작고, 같은 Expert를 강제 공유하면 노이즈만 추가된다.

아키텍처 진화 – Shared-Bottom에서 PLE까지

Shared-Bottom: 가장 단순한 MTL

모든 태스크가 하나의 trunk(공유 네트워크)를 통과한 뒤, 태스크별 head(타워)로 분기하는 구조다.

$$\boldsymbol{h} = f_{\text{shared}(\boldsymbol{x})} \quad \rightarrow \quad \hat{y}_k = f_k^{\text{tower}(\boldsymbol{h})}$$

장점: 구현이 단순하고, 파라미터 효율적이다.
한계: 모든 태스크가 동일한 표현을 강제로 공유하므로, 태스크 간 관련성이 낮을 때 Negative Transfer가 심각하다. 비유하자면, 모든 학생에게 동일한 교과서 한 권만 제공하는 것과 같다.

MMoE: Expert를 여러 개 두고 선택하게 하자

Ma et al. (KDD 2018)의 MMoE는 N개의 동일 구조 Expert를 두고, 태스크별 gate가 Expert 출력의 가중합을 결정한다.

$$\boldsymbol{h}_k = \sum_{i=1}^{N} g_{k,i} \cdot f_i^{\text{expert}(\boldsymbol{x})}, \quad \boldsymbol{g}_k = \text{Softmax}\left(\boldsymbol{W}_k^{\text{gate}} \cdot \boldsymbol{x}\right)$$

장점: 태스크별로 다른 Expert 조합을 사용할 수 있다.
한계: **Expert Collapse** – 모든 태스크의 gate가 동일한 Expert를 선택하여 사실상 Shared-Bottom으로 퇴화하는 현상. 이는 gradient가 "인기 Expert"에 집중되면서, 나머지 Expert의 gradient가 소실되어 학습이 멈추기 때문이다.

> 역사적 배경
>
> **Shared-Bottom**은 MTL의 원조인 **Caruana (Machine Learning, 1997)**이 체계화한 구조다. Rich Caruana는 "관련 태스크를 함께 학습하면 귀납적 편향(inductive bias)이 개선된다"는 핵심 통찰을 제시하며, 의료 진단에서 폐렴 사망률 예측에 관련 태스크를 보조로 사용하여 성능이 향상됨을 보였다. 이후 20년간 MTL의 기본 프레임워크가 되었다.
>
> **MMoE**는 **Ma, Zhao, Yi, Chen, Hong & Chi (KDD 2018)**이 Google에서 제안하였다. YouTube의 참여도 예측과 만족도 예측이라는 두 태스크에서 Shared-Bottom의 Negative Transfer가 심각했고, Expert를 복수로 두되 gate로 선택하는 아이디어로 이를 완화했다. Mixture of Experts 자체는 **Jacobs, Jordan, Nowlan & Hinton (1991)**이 최초로 제안한 것으로, MMoE는 이를 멀티태스크 학습의 맥락에서 재해석한 것이다.

> 비유: 뷔페 레스토랑
>
> MMoE는 뷔페 레스토랑과 같다. 여러 음식(Expert)이 준비되어 있고, 각 손님(태스크)이 자기 접시에 원하는 만큼 담는다(gate). 문제는 모든 손님이 스테이크(인기 Expert)만 담고 나머지 음식은 아무도 먹지 않아 폐기되는 상황이다. PLE의 해결책은 "기본 식사(Shared Expert)"와 "개인 특선(Task Expert)"을 명시적으로 분리하여 제공하는 것이다.

PLE: 공유와 분리의 명시적 균형

Tang et al. (RecSys 2020)의 PLE는 Expert를 두 종류로 나눈다.

- **Shared Expert** ($\mathcal{E}^s$): 모든 태스크가 접근 가능한 공용 Expert
- **Task-specific Expert** ($\mathcal{E}^k$): 태스크 k만 접근 가능한 전용 Expert

각 태스크의 gate는 Shared Expert와 자기 전용 Expert를 모두 입력받아 최적 결합 비율을 학습한다.

$$\boldsymbol{h}_k = \sum_{i=1}^{|\mathcal{E}^s|} g_{k,i}^s \cdot \boldsymbol{e}_i^s + \sum_{j=1}^{|\mathcal{E}^k|} g_{k,j}^k \cdot \boldsymbol{e}_j^k$$

이 설계가 해결하는 문제들:

1. **Negative Transfer** 완화: 태스크 전용 Expert가 해당 태스크에만 특화된 패턴을 간섭 없이 학습할 수 있다.
2. **Expert Collapse** 방지: Shared Expert는 "반드시 모든 태스크에 유용한 정보"를 학습하고, Task Expert는 "특화 정보"를 학습하여 역할이 자연스럽게 분리된다.
3. **Progressive (점진적)**: 여러 Extraction Layer를 쌓아 저수준→고수준으로 점진적으로 태스크별 표현을 정제할 수 있다.

> 역사적 배경

PLE는 **Tang, Liu, Zhao & Gong (RecSys 2020)**이 Tencent의 동영상 추천 시스템에서 제안하였다. 당시 Tencent Video는 VCR(Video Completion Rate), VTR(Video Through Rate), Share Rate 등 다수의 참여도 지표를 동시에 최적화해야 했다. MMoE를 적용했지만 Expert Collapse와 Seesaw 현상이 심각하여, "Expert를 명시적으로 공유/전용으로 분리하면 어떨까?"라는 아이디어에서 PLE가 탄생했다. 논문에서 PLE는 MMoE 대비 모든 태스크에서 동시에 성능이 향상된 최초의 MTL 아키텍처로 보고되었다. 이후 Alibaba, JD.com, Kuaishou, ByteDance 등 중국 대형 플랫폼에서 광범위하게 채택되어 산업용 MTL의 사실상 표준(de facto standard)이 되었다.

Expert와 Gate의 역할 – 직관적 이해

Expert: 서로 다른 렌즈로 세상을 보는 전문가

Expert는 입력 데이터를 특정 관점으로 해석하는 전문가다. 본 시스템의 8개 Shared Expert는 각각 전혀 다른 도메인의 시각을 제공한다.

- **unified_hgcn**: 상품/카테고리의 계층 구조를 쌍곡 공간에서 해석
- **perslay**: 거래 데이터의 위상적 형태**(topological shape)**를 포착
- **deepfm**: 피처 간 교차 상호작용을 학습
- **temporal**: 시간적 패턴과 동역학을 포착
- **lightgcn**: 고객-상품 그래프 관계를 표현
- **causal**: 피처 간 인과 관계를 추출
- **optimal_transport**: 분포 간 거리를 측정
- **raw_scale**: 정규화 전 원시 피처의 멱법칙 분포 패턴을 보존 (v3.3)

이들은 동일한 고객 데이터를 8가지 서로 다른 "렌즈"로 바라본 결과다. 어떤 태스크에는 시간적 패턴이 중요하고(CTR), 어떤 태스크에는 계층적 관계가 중요하다(Brand Prediction).

8개 **Shared Expert** 비교**:** 입력**,** 학습 대상**,** 대체 불가능성

Expert	입력	학습 대상	다른 **Expert**로 대체 불가능한 이유	출력 차원
DeepFM	정규화 644D	피처 쌍의 대칭 상호작용	FM의 $O(nk)$ 2차 교차를 명시적으로 포착	64D
LightGCN	사전 계산된 64D	고객-가맹점 협업 신호	이분 그래프 기반 "비슷한 고객" 패턴	64D
Unified HGCN	사전 계산된 47D	가맹점 계층 구조 (가맹점 노드만)	쌍곡 공간에서 MCC 트리 + 공동방문 보정	128D
Temporal	시퀀스 $[B, 180, 16]$+ $[B, 90, 8]$	시간적 패턴 변화	Mamba+LNN+Transformer 앙상블	64D
PersLay	Persistence Diagram	위상적 구조	소비 패턴의 루프/클러스터/분기점	64D
Causal	정규화 644D	피처 간 방향성 인과 (DAG)	교란 변수 제거, 비대칭 인과 구조	64D

OT	정규화 644D	고객-프로토타입 분포 거리	Wasserstein 거리로 분포 기하학 인코딩	64D
RawScale	원시 90D	멱법칙 분포 패턴 (v3.3)	정규화 시 손실되는 원시 스케일/분포 정보 보존	64D

왜 8개 **Expert** 모두 필요한가

8개 Expert는 동일 고객의 다른 측면을 포착하며, CGC Gate가 태스크별로 최적 조합을 학습한다. 세 Expert(DeepFM, Causal, OT)가 동일 정규화 644D를 입력받지만 대칭/비대칭/거리라는 근본적으로 다른 수학 구조를 추출하므로 중복이 아니다. RawScaleExpert(v3.3)는 정규화 전 원시 90D 피처를 입력받아 멱법칙 분포 정보를 보존한다.

Gate: 어떤 전문가의 의견을 얼마나 들을 것인가

Gate는 태스크별로 "어떤 Expert의 의견을 얼마나 신뢰할 것인가"를 결정하는 주의**(attention)** 메커니즘이다.

$$\boldsymbol{w}_k = \text{Softmax}(\boldsymbol{W}_k \cdot \boldsymbol{h}_{\text{shared}} + \boldsymbol{b}_k) \in \mathbb{R}^8$$

이 수식이 말하는 것은 명확하다.

- $\boldsymbol{W}_k \cdot \boldsymbol{h}_{\text{shared}}$: 현재 입력을 보고 각 Expert의 관련성을 점수로 매긴다
- Softmax: 점수를 확률 분포로 변환하여, 총합이 1이 되게 한다
- $\boldsymbol{w}_k \in \mathbb{R}^8$: 태스크 k가 8개 Expert에게 부여하는 신뢰도 벡터

비유: 의료 진단 위원회

8명의 전문의(Expert)가 환자(입력 데이터)를 각자의 전문 분야에서 진단한다. 내과의, 외과의, 영상의학과 의사 등이 각각 소견서(Expert output)를 작성한다. Gate는 "이 환자의 상태를 판단할 때 어느 전문의의 소견을 얼마나 비중 있게 볼 것인가"를 결정하는 주치의의 역할이다. 심장 관련 증상이면 내과의 소견에 높은 가중치를, 외상이면 외과의에게 높은 가중치를 부여한다.

학부 수학

Softmax에서 왜 e^x**(**자연 지수 함수**)**를 사용하는가**?** Softmax는 임의의 실수 벡터를 확률 분포(양수, 합=1)로 변환한다: $\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$. e^x를 선택하는 이유는 세 가지다: (1) 양수 보장: $e^x > 0$ (모든 실수 x에 대해), 따라서 "음의 확률" 문제가 없다. (2) 단조 증가: $z_i > z_j$이면 $e^{z_i} > e^{z_j}$이므로 점수 순서가 보존된다. (3) 미분 편의: $\frac{d}{dx}e^x = e^x$이므로 gradient 계산이 간결하다. 구체적 계산: 점수 벡터 $\boldsymbol{z} = [2.0, 1.0, 0.1]$이면 $e^{\boldsymbol{z}} = [7.39, 2.72, 1.11]$, 합 = 11.22, Softmax $= [0.659, 0.242, 0.099]$ – 점수 차이가 확률 차이로 변환된다. 점수 차이가 클수록 확률 차이가 급격해지는 것이 e^x의 지수적 증가 특성 덕분이다. 다른 양수 함수(예: x^2)를 쓸 수도 있지만, 음수 입력에서 순서가 꼬이고($(-3)^2 > (-1)^2$) gradient가 0 근처에서 소실되는 문제가 생겨 e^x가 최적의 선택이다.

수학적 고찰 – 수식이 말하는 것들

Gating과 **Attention**의 연결

CGC gate의 Softmax 가중합은 Transformer의 Attention 메커니즘과 본질적으로 같은 구조다.

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

CGC에서는:

- **Query**: 태스크 k의 게이트 가중치 $\boldsymbol{W}_k$ (이 태스크가 원하는 정보)
- **Key**: 공유 표현 $\boldsymbol{h}_{\text{shared}}$ (현재 입력에 대한 각 Expert의 관련성)
- **Value**: 각 Expert의 출력 블록 $\boldsymbol{h}_i^{\text{expert}}$ (실제 정보)

차이점은 Transformer가 시퀀스의 각 토큰 간 attention을 계산하는 반면, CGC는 **Expert** 간 **attention**을 계산한다는 것이다. 동일한 수학적 원리 – "관련성에 비례하여 정보를 선택적으로 결합" – 가 다른 단위(토큰 vs Expert)에 적용된 것이다.

Expert 가중합의 함수 근사론적 의미

Expert 출력의 가중합이 왜 강력한가? 이를 함수 근사(function approximation)의 관점에서 이해할 수 있다.

$$\boldsymbol{h}_k = \sum_{i=1}^{N} g_{k,i} \cdot \boldsymbol{e}_{i(\boldsymbol{x})}$$

각 Expert $\boldsymbol{e}_i$는 입력 공간의 특정 영역에 특화된 기저 함수**(basis function)**로 볼 수 있다. Gate $g_{k,i}$는 이 기저 함수들의 혼합 계수다. 이는 Mixture of Experts의 이름에 담긴 의미 그대로, 전문가 혼합 모델**(Mixture of Experts model)**이다.

통계학의 혼합 밀도 모델(mixture density model)과 정확히 같은 구조다.

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{i=1}^{N} \pi_{i(\boldsymbol{x})} \cdot p_{i(\boldsymbol{y} \mid \boldsymbol{x})}$$

여기서 $\pi_{i(\boldsymbol{x})}$가 gate, p_i가 Expert에 대응한다. 각 Expert가 입력 공간의 서로 다른 영역을 담당하므로, 전체 모델은 단일 네트워크보다 더 복잡한 함수를 효율적으로 근사할 수 있다.

> **학부 수학**
>
> 가중합의 함수 근사론적 의미: 수학에서 임의의 함수를 기저 함수(basis function)의 선형 결합(가중합)으로 표현하는 것은 근사 이론의 핵심이다. 푸리에 급수 $f(x) = \sum a_n \cos(nx) + b_n \sin(nx)$가 대표적인 예시로, 삼각함수라는 기저의 가중합으로 모든 주기 함수를 근사할 수 있다. Expert의 가중합 $\boldsymbol{h} = \sum g_i \cdot \boldsymbol{e}_{i(\boldsymbol{x})}$도 같은 원리다. 각 Expert $\boldsymbol{e}_i$는 입력 공간의 특정 패턴에 특화된 "기저 함수"이고, gate g_i가 혼합 계수 역할을 한다. 차이점은 기저 자체도 학습된다는 것이다. 구체적 예시: 입력 $\boldsymbol{x}$가 시간적 패턴이 강한 고객이면 $g_{\text{temporal}} = 0.4$로 커지고, 그래프 관계가 중요하면 $g_{\text{lightgcn}} = 0.3$으로 커진다. 결과 표현 $\boldsymbol{h}$는 이 고객에게 가장 적합한 전문가 의견의 혼합이 된다.

> **최신 동향**
>
> Mixture of Experts(MoE)는 2024-2025년 LLM 분야에서 핵심 아키텍처로 부상했다. Mistral의 **Mixtral 8x7B** (2023), Google의 **Switch Transformer** (Fedus et al., 2022), DeepSeek의 **DeepSeek-MoE** (2024)가 대표적이다. 이들은 Sparse MoE(top-k 선택)를 사용하여 파라미터 수 대비 계산량을 크게 줄였다. 추천 시스템에서도 Alibaba의 **Star Topology Adaptive Recommender** (STAR, CIKM 2021), Kuaishou의 **PEPNet** (KDD 2023)이 입력 조건에 따라

> Expert를 선택하는 MoE 구조를 채택했다. 본 시스템의 CGC 게이팅은 Dense MoE(모든 Expert 활용)에 해당하며, Expert 수가 8개로 적어 sparse 선택 없이도 계산 효율이 유지된다.

Progressive 구조가 정보 흐름에 미치는 영향

원 논문의 PLE는 여러 Extraction Layer를 쌓는다. l번째 레이어의 태스크 k 출력은 다음과 같다.

$$\boldsymbol{h}_k^{(l)} = \text{Gate}_k^{(l)}\left(\boldsymbol{E}^{s,(l)}\left(\boldsymbol{h}^{(l-1)}\right), \boldsymbol{E}^{k,(l)}\left(\boldsymbol{h}_k^{(l-1)}\right)\right)$$

각 레이어를 지날 때마다:

1. **Shared** 표현은 모든 태스크에 공통으로 유용한 정보를 점진적으로 정제한다
2. **Task** 표현은 점점 더 태스크에 특화된다
3. Gate는 레이어마다 독립적으로 학습되어, 추상화 수준별로 다른 결합 전략을 사용할 수 있다

이것은 CNN에서 저수준(에지, 텍스처)→고수준(객체, 의미)으로 정보가 정제되는 것과 동일한 원리다.

본 구현에서는 단일 Extraction Layer를 사용하지만, **CGC → GroupTaskExpertBasket → Logit Transfer**라는 3단계 파이프라인이 사실상 Progressive한 정보 정제의 역할을 수행한다.

핵심 수식의 직관적 해석

이 절에서는 본 문서에 등장하는 주요 수식이 실제로 무엇을 의미하는지, "왜 이 수식이 여기에 필요한지"를 실무자 관점에서 해석한다.

PLE 게이팅 결합 (Section 2)

$$\boldsymbol{h}_k = \sum_{i=1}^{|\mathcal{E}^s|} g_{k,i}^s \cdot \boldsymbol{e}_i^s + \sum_{j=1}^{|\mathcal{E}^k|} g_{k,j}^k \cdot \boldsymbol{e}_j^k$$

해석: "태스크 k의 표현은, 공용 전문가들의 의견을 가중 합산하고, 자기 전용 전문가의 의견도 가중 합산한 뒤, 둘을 더한 것이다."

첫째 합 $\sum g_{k,i}^s \cdot \boldsymbol{e}_i^s$은 공유 지식에서 태스크 k에 유용한 부분만 골라내는 것이고, 둘째 합 $\sum g_{k,j}^k \cdot \boldsymbol{e}_j^k$는 전용 지식에서 특화 정보를 가져오는 것이다. gate 값 g가 클수록 해당 Expert의 발언권이 크다.

CGC Attention 가중치 (Section 5)

$$\boldsymbol{w}_k = \text{Softmax}(\boldsymbol{W}_k \cdot \boldsymbol{h}_{\text{shared}} + \boldsymbol{b}_k) \in \mathbb{R}^8$$

해석: "현재 입력($\boldsymbol{h}_{\text{shared}}$)을 보고 8개 Expert 각각의 관련성 점수를 매긴 뒤, Softmax로 비율화한다. Softmax를 거치면 합이 1이 되므로, 이것은 곧 '태스크 k가 현재 입력에 대해 Expert들에게 부여하는 주의(attention) 분포'다."

초기 bias($\boldsymbol{b}_k$)의 역할이 중요하다. `domain_experts`로 지정된 Expert에는 `bias_high=1.0`을, 나머지에는 `bias_low=-1.0`을 설정하여, 학습 초기에 이미 도메인 지식에 부합하는 Expert에 주의가 집중되도록 유도한다. 학습이 진행되면서 $\boldsymbol{W}_k$가 업데이트되어 데이터에 맞게 수정된다.

Entropy 정규화 (Section 5)

$$\mathcal{L}_{\text{entropy}} = \lambda_{\text{ent}} \cdot \left(-\frac{1}{|\mathcal{T}|}\right) \sum_{k \in \mathcal{T}} H(\boldsymbol{w}_k), \quad H(\boldsymbol{w}_k) = -\sum_{i=1}^{8} w_{k,i} \cdot \log\left(w_{k,i}\right)$$

해석: "gate 분포의 엔트로피가 낮으면(한 Expert에 집중) 페널티를 부여하여, 최소한 여러 Expert를 고르게 활용하도록 유도한다."

왜 필요한가? 엔트로피 정규화 없이 학습하면, gate가 가장 gradient가 큰 Expert 하나에 빠르게 수렴하여 나머지 Expert의 학습이 멈추는 **Expert Collapse**가 발생한다. 이 손실 항은 "모든 전문의의 소견을 최소한 참고는 하라"는 제약이다.

Focal Loss (Section 9)

$$\text{FL}(p_t) = -\alpha_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t)$$

해석: "이미 잘 맞추고 있는 쉬운 예제($p_t \approx 1$)의 손실은 거의 0으로 줄이고, 틀리고 있는 어려운 예제($p_t \approx 0$)에 학습 에너지를 집중한다."

$(1 - p_t)^{\gamma}$ 항이 핵심이다. $p_t = 0.9$(잘 맞추는 경우)이면 $(1 - 0.9)^2 = 0.01$로 가중치가 100분의 1이 된다. $p_t = 0.1$(틀리는 경우)이면 $(1 - 0.1)^2 = 0.81$로 가중치가 거의 유지된다. γ가 클수록 쉬운 예제의 감쇠가 강해진다.

α_t는 클래스 불균형을 보정한다. Churn 태스크의 $\alpha = 0.6$은 "이탈 고객을 놓치는 비용이 비이탈을 잘못 예측하는 비용보다 크므로, 양성(이탈) 예제에 더 높은 가중치를 부여한다"는 비즈니스 판단을 인코딩한 것이다.

> 학부 수학
>
> 거듭제곱 $(1 - p_t)^{\gamma}$의 감쇠 효과: 0과 1 사이의 수를 거듭제곱하면 지수가 클수록 값이 빠르게 0에 가까워진다. 이것이 Focal Loss의 핵심 메커니즘이다.
>
> $\gamma = 0$이면 $(1 - p_t)^0 = 1$으로 Focal Loss = 표준 Cross-Entropy (감쇠 없음). $\gamma = 1$이면 선형 감쇠: $p_t = 0.9 \to 0.1$, $p_t = 0.5 \to 0.5$. $\gamma = 2$이면 제곱 감쇠: $p_t = 0.9 \to 0.01$, $p_t = 0.5 \to 0.25$. $\gamma = 5$이면 급격 감쇠: $p_t = 0.9 \to 0.00001$, $p_t = 0.5 \to 0.03$.
>
> 즉 γ가 클수록 "쉬운 예제(높은 p_t)를 더 강하게 무시"하고 어려운 예제에 학습을 집중한다. 실무적으로 $\gamma = 2$가 가장 널리 사용되며, 이는 "잘 맞추는 예제의 기여를 대략 100분의 1로 줄이는" 적당한 강도이다.

Uncertainty Weighting (Section 9)

$$\mathcal{L}_k^{\text{uw}} = w_k \cdot (\exp(-s_k) \cdot \mathcal{L}_k + s_k)$$

해석: "본질적으로 예측이 어려운 태스크(높은 불확실성)의 가중치를 자동으로 낮추고, 쉬운 태스크의 가중치를 높인다. 이 균형을 모델이 스스로 학습한다."

$s_k = \log(\sigma_k^2)$는 태스크 k의 학습 가능한 불확실성이다.

- s_k가 크면(불확실성 높음): $\exp(-s_k)$가 작아져 손실 기여가 줄어든다. 동시에 $+s_k$ 항이 커져서 s_k를 무한히 키우는 것을 방지한다.
- s_k가 작으면(불확실성 낮음): $\exp(-s_k)$가 커져 손실을 적극 반영한다.

이 메커니즘 덕분에 수동으로 태스크 가중치를 튜닝할 필요가 줄어든다. 16개 태스크의 가중치를 일일이 조정하는 것은 조합 폭발이지만, Uncertainty Weighting은 이를 자동 균형으로 대체한다.

학부 수학

exp와 **log**는 왜 짝으로 나타나는가? $\exp(x) = e^x$와 $\log(x) = \ln(x)$는 역함수 관계이다: $\exp(\log(x)) = x$, $\log(\exp(x)) = x$. Uncertainty Weighting에서 $s_k = \log(\sigma_k^2)$로 정의하고 $\exp(-s_k)$를 사용하는 이유는: (1) σ_k^2(분산)은 반드시 양수여야 하는데, s_k는 제약 없는 실수이므로 최적화가 쉽다. (2) $\exp(-s_k) = \exp(-\log(\sigma_k^2)) = \frac{1}{\sigma_k^2}$이므로 precision(정밀도)이 된다. 구체적 계산: $s_k = 0$이면 $\exp(-s_k) = 1$ (표준 손실 그대로 반영). $s_k = 2$이면 $\exp(-2) \approx 0.135$ (손실의 13.5%만 반영 – 불확실한 태스크 감쇠). $s_k = -1$이면 $\exp(1) \approx 2.718$ (손실 2.7배 증폭 – 확실한 태스크 강조). 동시에 $+s_k$ 정규화 항은 $s_k = 2$일 때 $+2$를 더해 "불확실하다고 선언하는 데 비용이 든다"는 제약을 건다.

Evidential Uncertainty (Section 11)

$$u = \frac{K}{S}, \quad S = \sum_{k=1}^{K} \alpha_k$$

해석: "모델이 증거(evidence)를 많이 모았으면 확신하고(S 큼, u 작음), 증거가 부족하면 '모르겠다'고 솔직하게 말한다(S 작음, u 큼)."

기존 Softmax는 어떤 입력이든 항상 확률 분포를 출력한다. 학습 분포에서 벗어난(out-of-distribution) 데이터에도 자신 있게 예측하여 위험한 결정을 내릴 수 있다. Evidential 접근은 "확률의 확률" – 즉 Dirichlet 분포 –를 모델링하여, 예측 자체의 불확실성까지 정량화한다.

Soft Routing (Section 7)

$$\boldsymbol{e}_{\text{cluster}} = \sum_{c=0}^{19} p_c \cdot \boldsymbol{E}_c \in \mathbb{R}^{32}$$

해석: "GMM 클러스터 경계에 있는 고객은 하나의 클러스터에 강제 배정하지 않고, 여러 클러스터 임베딩 벡터를 소속 확률에 비례하여 혼합한다. 혼합된 임베딩이 GroupEncoder 출력과 결합되어 TaskHead를 통과한다."

이것은 hard clustering의 불연속성 문제를 해결한다. 클러스터 0과 1의 경계에 있는 고객이 id=0으로 배정되면, 클러스터 1의 지식을 전혀 활용하지 못한다. Soft routing은 $p_0 = 0.6, p_1 = 0.4$처럼 임베딩을 비례 혼합하여 경계 고객의 표현을 더 안정적으로 만든다.

전체 내러티브 – "왜 PLE인가"

이야기의 흐름

시작점: 16개 태스크를 동시에 예측해야 한다. 독립 모델 16개를 만들면 데이터 효율이 낮고, 공통 패턴을 활용하지 못한다.

첫 시도 **(Shared-Bottom)**: 하나의 공유 네트워크를 만들었다. 일부 태스크는 성능이 올라갔지만, CTR과 Churn처럼 본질적으로 다른 태스크가 서로의 학습을 방해하는 **Negative Transfer**가 발생했다.

두 번째 시도 **(MMoE)**: Expert를 여러 개 두고 gate로 선택하게 했다. 하지만 모든 gate가 같은 Expert를 선택하는 **Expert Collapse**가 발생하여, 사실상 Shared-Bottom과 다를 바 없어졌다.

해결 **(PLE)**: Expert를 공유(Shared)와 전용(Task-specific)으로 명시적으로 분리하고, CGC gate가 두 종류의 Expert를 최적 비율로 결합하게 했다. 공유 Expert는 모든 태스크에 유용한 기본 지식을, 전용 Expert는 각 태스크에만 필요한 특화 지식을 담당한다.

확장 **(본 프로젝트)**: PLE의 아이디어 위에 8개 이종 도메인 Expert(GCN, TDA, DeepFM, Temporal, Graph, Causal, OT, RawScale), GroupEncoder + ClusterEmbedding(4 그룹, 20 클러스터), HMM Triple-Mode 라우팅, Logit Transfer 체인, Evidential 불확실성, SAE 해석성을 추가하여 **AIOps** 추천 시스템에 특화된 **PLE-Cluster-adaTT**를 완성하였다.

설계 원칙 요약

원칙	구현
공유와 분리의 균형	8 Shared Expert + CGC gate + GroupTaskExpertBasket
태스크 간 간섭 최소화	Expert 분리 + Entropy 정규화 + domain_experts bias
태스크 간 지식 전달	Logit Transfer (명시적) + adaTT gradient 전이 (적응적)
클러스터별 특화	GroupEncoder + ClusterEmbedding + Soft Routing
시간 스케일 분리	HMM Triple-Mode (daily / monthly)
불확실성 인식	Evidential Deep Learning (Dirichlet)
자동 균형	Uncertainty Weighting (태스크 가중치 자동 조정)
해석 가능성	SAE (2304D sparse latent)

PLE 이론적 배경

논문 참조

RecSys 2020 Tang, H., Liu, J., Zhao, M., & Gong, X. *"Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations"*

Negative Transfer 문제

멀티태스크 학습(MTL)에서 가장 심각한 문제는 **Negative Transfer**이다. 관련성이 낮은 태스크가 표현 공간을 오염시켜, 단일 태스크 학습보다 오히려 성능이 하락하는 현상이다.

> **Negative Transfer의 실제 영향**
>
> AIOps 시스템에서 CTR(클릭률)과 Churn(이탈) 태스크는 본질적으로 다른 패턴을 학습해야 한다. CTR은 단기 참여도, Churn은 장기 이탈 신호에 집중한다. Shared-Bottom 구조에서 이 두 태스크가 동일한 표현을 공유하면, 한쪽 gradient가 다른 쪽의 학습을 방해하는 **seesaw** 현상이 발생한다.

Shared-Bottom, MMoE, PLE 비교

구분	Shared-Bottom	MMoE	PLE
Expert 구조	단일 Shared trunk	N개 Expert 전체 공유	Shared Expert + Task-specific Expert 명시 분리
게이팅	없음	태스크별 Softmax gate	CGC: Shared + Task Expert 결합 gate
Negative Transfer	높음 (모든 태스크가 간섭)	중간 (Expert Collapse 가능)	낮음 (명시적 분리로 간섭 최소화)
Expert Collapse	해당 없음	높음 (모든 태스크가 동일 Expert 선택)	낮음 (Shared/Task Expert 분리)
확장성	제한적	Expert 수 증가로 대응	Extraction Layer 추가로 대응

PLE 게이팅 공식

PLE에서 태스크 k의 출력은 Shared Expert 집합 $\mathcal{E}^s$와 Task-specific Expert 집합 $\mathcal{E}^k$의 게이팅 결합으로 결정된다.

$$\boldsymbol{h}_k = \sum_{i=1}^{|\mathcal{E}^s|} g^s_{k,i} \cdot \boldsymbol{e}^s_i + \sum_{j=1}^{|\mathcal{E}^k|} g^k_{k,j} \cdot \boldsymbol{e}^k_j$$

e_i^s: i번째 Shared Expert 출력, e_j^k: k 태스크의 j번째 Task Expert 출력
$g_{k,i}^s, g_{k,j}^k$: CGC gate 가중치 (Softmax 정규화)

본 프로젝트에서의 PLE 변형

원 논문의 PLE는 Task-specific Expert를 태스크당 독립 MLP로 구현하지만, 본 시스템에서는 **CGC Gate**가 **Shared Expert** 출력 블록에 스케일 가중치를 적용하고, 그 결과를 **GroupTaskExpertBasket(4 GroupEncoder + ClusterEmbedding)**으로 전달하는 2단 구조로 변형하였다. Task-specific Expert의 역할을 GroupTaskExpertBasket이 대신하며, 그룹 내 파라미터 공유 + 클러스터별 특화를 동시에 달성한다.

PLEClusterInput – 입력 데이터 구조

`PLEClusterInput` 데이터클래스 (라인 62 199)는 모델의 모든 입력을 캡슐화한다. 배치 단위로 디바이스 이동이 가능하며, HMM 모드 라우팅 로직을 내장한다.

전체 필드 사양

필드명	타입	차원	설명 / 출처
`features`	`Tensor`	`[B, 734]`	base 238 + multi_source 91 + domain 159 + multidisciplinary 24 + model_derived 27 + extended_source 84 + merchant 21 (= 644D normalized) + raw_power_law 90D
`cluster_ids`	`Tensor`	`[B]`	GMM 클러스터 ID (0 19)
`cluster_probs`	`Tensor?`	`[B, 20]`	Soft routing용 클러스터 확률 (경계 샘플 처리)
`hyperbolic_features`	`Tensor?`	`[B, 20]`	MCC(8D) + Product(8D) + Region(4D) Poincare 좌표
`tda_features`	`Tensor?`	`[B, 70]`	tda_short(24) + tda_long(36) + phase_transition(10)
`tda_short_diagrams`	`Tensor?`	`[B, 200, 3]`	Raw Persistence Diagram (birth, death, beta_idx)
`tda_short_mask`	`Tensor?`	`[B, 200]`	유효 쌍 마스크 (패딩 제외)
`tda_long_diagrams`	`Tensor?`	`[B, 150, 3]`	Long-term Persistence Diagram
`tda_long_mask`	`Tensor?`	`[B, 150]`	Long 유효 쌍 마스크
`tda_global_stats`	`Tensor?`	`[B, 30]`	short_global 12D + long_global 18D
`tda_phase_transition`	`Tensor?`	`[B, 10]`	상전이 피처
`hmm_journey`	`Tensor?`	`[B, 16]`	HMM Journey 모드 (10D base + 6D ODE dynamics)
`hmm_lifecycle`	`Tensor?`	`[B, 16]`	HMM Lifecycle 모드
`hmm_behavior`	`Tensor?`	`[B, 16]`	HMM Behavior 모드
`txn_seq`	`Tensor?`	`[B, 180, 16]`	거래 시퀀스: card(8) + deposit(8)
`session_seq`	`Tensor?`	`[B, 90, 8]`	세션 시퀀스
`collaborative_features`	`Tensor?`	`[B, 64]`	LightGCN 사전 계산 임베딩
`hierarchy_features`	`Tensor?`	`[B, 20]`	H-GCN 사전 계산 Poincare 좌표

`customer_segment`	`Tensor?`	`[B]`	0=anonymous, 1=cold_start, 2=warm_start
`coldstart_features`	`Tensor?`	`[B, 40]`	콜드스타트 static features
`anonymous_features`	`Tensor?`	`[B, 15]`	익명 static features
`targets`	`Dict?`	가변	태스크별 정답 레이블 (학습 시)

734D `features` 텐서 인덱스 매핑

`feature_schema.yaml`의 `continuous` 리스트 순서가 데이터로더에서 텐서 컬럼 순서를 결정한다. 아래 표는 `features` 텐서의 정확한 인덱스 범위를 정의한다.

피처 그룹	차원	인덱스 범위	소계	세부 구성
Base	238D	`[0, 237]`	238	RFM 34D + Category 64D + Transaction_Stats 80D + Temporal 60D
Multi-source	91D	`[238, 328]`	91	Deposit 20D + Membership 15D + Investment 18D + Credit 12D + Digital 14D + Product 12D
Domain	159D	`[329, 487]`	159	TDA_short 24D + TDA_long 36D + Phase_Transition 10D + GMM 22D + Mamba 50D + Economics 17D
Multidisciplinary	24D	`[488, 511]`	24	Chemical 6D + Epidemic 5D + Interference 8D + Crime 5D
Model-derived	27D	`[512, 538]`	27	Bandit 4D + HMM_summary 5D + LNN 18D
Extended source	84D	`[539, 622]`	84	Insurance 25D + Consultation 18D + Campaign 12D + Overseas 6D + OtherChannel 23D
Merchant hierarchy	21D	`[623, 643]`	21	MCC_L1 4D + MCC_L2 4D + Brand 8D + Stats 4D + Radius 1D

Domain 내부 세부 인덱스:

서브그룹	차원	인덱스 범위	설명
TDA-Short	24D	`[329, 352]`	앱 로그 기반 단기 위상 패턴 (H0+H1, 90일 윈도우)
TDA-Long	36D	`[353, 388]`	금융 거래 기반 장기 위상 패턴 (H0+H1+H2, 12개월 윈도우)
Phase Transition	10D	`[389, 398]`	W1 거리, 위상 변화량, 전이 확률/방향/크기 등
GMM Cluster	22D	`[399, 420]`	GMM 클러스터 소속 확률 + 거리 통계
Mamba Temporal	50D	`[421, 470]`	Mamba SSM 시계열 잠재 표현
Income Decomposition	8D	`[471, 478]`	소득 구조 분해 (경제학 피처)

서브그룹	차원	인덱스 범위	설명
Financial Behavior	9D	`[479, 487]`	금융 행동 지표 (경제학 피처)

Model-derived 내부 세부 인덱스:

서브그룹	차원	인덱스 범위	설명
Bandit (MAB)	4D	`[512, 515]`	Multi-Armed Bandit 탐색/활용 행동 통계
HMM Summary	5D	`[516, 520]`	지배적 상태, 지속기간, 안정성, 엔트로피, 변화율
LNN Model	18D	`[521, 538]`	분포 통계 4D + 주파수 4D + 변화점 3D + 자기상관 4D + 복잡성 3D

`_FEATURE_GROUP_DIMS_ORDER` 순서 수정 완료

`ple_cluster_adatt.py:407` 의 `_FEATURE_GROUP_DIMS_ORDER` 를 `feature_schema.yaml` 순서에 맞게 수정 완료: base → multi_source → **domain** → multidisciplinary → model_derived → extended_source → merchant. 검증 기준: `feature_schema.yaml` → `task_feature_mapper.py:FEATURE_GROUP_DIMS` → `feature_integrator.py:EXPECTED_GROUP_DIMS_CAN` .

학부 수학 – 734차원 입력 벡터의 의미

신경망의 입력 $\boldsymbol{x} \in \mathbb{R}^{734}$는 734개의 실수로 구성된 벡터다. 앞 644D는 정규화된 피처, 뒤 90D는 정규화 전 원시 power-law 피처이다. 선형대수에서 $\mathbb{R}^n$은 n차원 실수 벡터 공간으로, 각 축이 하나의 피처에 대응한다. 예를 들어 x_1이 "월평균 소비 금액", x_2가 "최근 로그인 빈도"라면, 한 고객은 734차원 공간의 한 점으로 표현된다. 인간은 3차원까지만 시각적으로 이해할 수 있지만, 수학적 연산(내적, 노름, 사영)은 차원 수에 관계없이 동일하게 적용된다. 734D 공간에서도 두 고객 벡터의 코사인 유사도를 계산하면 "행동 패턴이 얼마나 비슷한가"를 정량화할 수 있다. 차원의 저주**(Curse of Dimensionality)**: 고차원 공간에서는 데이터가 희소해져 거리 기반 방법이 비효율적이 된다. 이것이 Expert 네트워크가 차원을 64D나 128D로 압축하는 이유이며, 압축 과정에서 태스크에 유용한 정보만 남기는 것이 학습의 핵심이다.

HMM 모드 라우팅

`set_hmm_routing()` 클래스 메서드(라인 173 186)는 모델 초기화 시 1회 호출되어 config의 `hmm_triple_mode` 섹션에서 태스크별 HMM 모드 매핑을 구축한다.

```python
# ple_cluster_adatt.py:172-186 — config가 single source of truth
@classmethod
def set_hmm_routing(cls, hmm_config: dict) -> None:
    routing: Dict[str, str] = {}
    for mode in ["journey", "lifecycle", "behavior"]:
        for task in hmm_config.get(mode, {}).get("target_tasks", []):
            routing[task.lower().replace("-", "_")] = mode
    cls._default_hmm_routing = routing
```

`get_hmm_for_task()` (라인 188 198)는 태스크 이름으로 해당 HMM 텐서를 반환하며, 매핑에 없는 태스크는 `"behavior"` 모드를 기본값으로 사용한다.

Shared Expert 결합 (576D)

8개 Shared Expert 구성

`_build_shared_experts()` (라인 395 451)에서 `SharedExpertFactory.create_from_config()`를 호출하여 config의 `shared_experts` 섹션에서 활성화된 Expert를 동적으로 생성한다.

Expert 이름	입력	출력	역할
`unified_hgcn`	47D	128D	Hyperbolic GCN + Merchant 계층 구조 (hgcn+merchant_hgcn 통합)
`perslay`	70D	64D	Persistence Diagram 처리 (TDA 위상 피처)
`deepfm`	정규화 644D	64D	Feature Interaction (FM + Deep, 필드별 독립 임베딩 v3.11)
`temporal`	시퀀스	64D	Temporal Ensemble (Mamba + LNN + Transformer)
`lightgcn`	64D	64D	Graph-based CF (사전 계산 임베딩)
`causal`	정규화 644D	64D	SCM/NOTEARS 기반 인과 관계 추출
`optimal_transport`	정규화 644D	64D	Sinkhorn 기반 Wasserstein 거리 표현
`raw_scale`	원시 90D	64D	RawScaleExpert: 정규화 전 멱법칙 분포 보존 (LayerNorm+MLP, v3.3)

$$\boldsymbol{h}_{\text{shared}} = [\text{unified\_hgcn}_{128D} \,\|\, \text{perslay}_{64D} \,\|\, \text{deepfm}_{64D} \,\|\, \text{temporal}_{64D} \,\|\, \text{lightgcn}_{64D} \,\|\, \text{causal}_{64D} \,\|\, \text{OT}_{64D} \,\|\, \text{raw\_scale}_{64D}]$$

$$\dim(\text{shared\_concat}) = 7 \times 64 + 1 \times 128 = 576D$$

$\|$: 텐서 결합(concatenation). DeepFM/Causal/OT는 `inputs.features[:, :644]` (정규화 644D), RawScaleExpert는 `inputs.features[:, 644:]` (원시 90D) 입력

수식 직관

이 수식은 8명의 서로 다른 전문가가 각자의 분석 결과를 한 줄로 이어 붙인 것이다. 직관적으로, 그래프 구조 분석(128D)과 위상 분석(64D), FM 교차(64D) 등 이질적인 시각의 결과물을 하나의 576차원 벡터로 합치면, 후속 CGC 게이트가 "이 고객에게는 어떤 전문가의 의견이 중요한가"를 태스크별로 판단할 수 있는 재료가 된다. v3.3에서 추가된 RawScaleExpert(64D)는 정규화 시 손실되는 원시 스케일 정보를 보존한다.

최신 동향

이종(heterogeneous) Expert 결합은 2024-2025년 추천 시스템 연구의 핵심 트렌드다. 기존 MoE 연구(MMoE, PLE)는 동일 구조의 MLP Expert를 사용했지만, 최근에는 GNN + Transformer + CNN 등 서로 다른 아키텍처를 Expert로 결합하는 시도가 늘고 있다. Google의 **Multi-Aspect Expert Model** (MAEM, KDD 2024)은 행동/컨텍스트/프로필 각각에 특화된 Expert를 두어 YouTube 추천 성능을 개선했다. Meta의 **DHEN** (Deep Heterogeneous Expert Network, 2023)은 이종 Expert의 상호작용을 명시적으로 모델링하여 Instagram 피드 랭킹에 적용했다. 본 시스템의 8개 이종 Expert(GCN, PersLay, DeepFM, Temporal, LightGCN, Causal, OT, RawScale) 결합은 이러한 최신 흐름과 정확히 부합하며, 단일 도메인 Expert로는 포착할 수 없는 다면적(multi-aspect) 고객 표현을 구축한다.

Expert별 Forward 디스패치

`_forward_shared_experts()` (라인 1416 1567)에서 Expert 이름에 따라 서로 다른 입력을 전달한다.

```
# ple_cluster_adatt.py:1435-1565 — Expert별 디스패치 요약
for name, expert in self.shared_experts.items():
    if name in ("hgcn", "merchant_hgcn", "unified_hgcn"):
        # hierarchy_features(20D) + merchant slice(27D) = 47D
        out, hgcn_interpret, _ = expert(combined_input)
    elif name == "perslay":
        # Raw diagram mode 또는 pre-computed 70D fallback
        out, _ = expert(tda_short_diagrams / tda_features / zero)
    elif name == "deepfm":
        out, _ = expert(inputs.features[:, :644])   # 정규화 644D
    elif name == "temporal":
        out, _ = expert(txn_seq, session_seq, ...)  # 시퀀스
    elif name == "lightgcn":
        out, _ = expert(collaborative_features)  # 사전 계산 64D
    elif name in ("causal", "optimal_transport"):
        out, _ = expert(inputs.features[:, :644])   # 정규화 644D
    elif name == "raw_scale":
        out, _ = expert(inputs.features[:, 644:])   # 원시 90D (v3.3)
```

Zero Fallback 전략

모든 Expert는 입력 데이터가 `None`일 때 **zero** 텐서 **fallback**을 수행한다. 이는 배치 내에 해당 피처가 없는 샘플이 있을 때 안전하게 처리하기 위함이다. CGC 게이팅이 이후 단계에서 해당 Expert의 가중치를 자동으로 낮춘다.

Zero Fallback과 CGC의 상호작용

Zero 출력을 가진 Expert에 대해 CGC가 높은 가중치를 부여하면 전체 표현이 희석된다. CGC의 `domain_experts` 초기 bias가 이 문제를 완화하지만, 학습 초기에 특정 Expert의 입력이 대부분의 배치에서 누락되면 **dead expert** 현상이 발생할 수 있다. `_cgc_entropy_regularization`으로 분산을 유도하여 부분 완화한다.

CGC (Customized Gate Control) – 태스크별 Expert 가중치

이론적 배경

RecSys 2020 Ma, J., Zhao, Z., Yi, X., et al. *"Modeling Task Relationships in Multi-Task Learning with Multi-Gate Mixture-of-Experts"* (MMoE)

RecSys 2020 Tang et al. *"Progressive Layered Extraction (PLE)"*

CGC는 MMoE의 게이팅 메커니즘을 확장한 것으로, 태스크별 독립 게이트가 Shared Expert 출력에 서로 다른 가중치를 적용하여 태스크-Expert 친화도를 학습한다.

CGC 아키텍처

`_build_cgc()` (라인 566 677)에서 태스크별 독립적인 `nn.Sequential(Linear + Softmax)` 모듈을 `nn.ModuleDict`로 관리한다.

$$\boldsymbol{w}_k = \text{Softmax}(\boldsymbol{W}_k \cdot \boldsymbol{h}_{\text{shared}} + \boldsymbol{b}_k) \in \mathbb{R}^8$$

$$\tilde{\boldsymbol{h}}_{k,i} = w_{k,i} \cdot \boldsymbol{h}_i^{\text{expert}} \quad \text{for } i = 1, \ldots, 8$$

$$\boldsymbol{h}_k^{\text{cgc}} = \left[\tilde{\boldsymbol{h}}_{k,1} \parallel \tilde{\boldsymbol{h}}_{k,2} \parallel \ldots \parallel \tilde{\boldsymbol{h}}_{k,8}\right] \in \mathbb{R}^{576}$$

$\boldsymbol{W}_k \in \mathbb{R}^{8\times 576}$: 태스크 k의 gate 가중치
$\boldsymbol{h}_i^{\text{expert}}$: i번째 Expert의 출력 블록 (64D 또는 128D)
$w_{k,i}$: 태스크 k가 Expert i에 부여하는 attention 가중치

수식 직관

첫 번째 식은 576D 공유 표현을 보고 8개 Expert 각각의 "관련성 점수"를 산출한 뒤 Softmax로 확률화하는 과정이다. 두 번째 식은 이 확률(스칼라)을 각 Expert 출력 블록에 곱해 중요도를 조절한다. 세 번째 식은 가중 조절된 블록들을 다시 이어 붙여 원래와 동일한 576D를 복원한다. 결과적으로, 같은 576D 벡터라도 태스크마다 Expert별 기여 비중이 다르게 조합된다.

차원 유지 설계

CGC는 576D 입력을 576D 출력으로 변환한다. Expert별 블록에 스칼라 가중치를 곱하는 블록 스케일링 방식이므로 기존 파이프라인과 하위 호환된다. 가중치 합이 1 (Softmax)이므로 출력 스케일이 보존된다.

역사적 배경

CGC(Customized Gate Control)는 PLE 논문(Tang et al., RecSys 2020)에서 MMoE의 gate를 확장한 개념으로 처음 명명되었다. MMoE(Ma et al., KDD 2018)의 gate가 모든 Expert를 동등하게 취급하는 반면, CGC는 Shared Expert와 Task-specific Expert를 구분하여 게이팅한다. Attention 메커니즘과의 연결은 Transformer(Vaswani et al., NeurIPS 2017)의 Scaled Dot-Product Attention에서 영감을 받은 것으로, "관련성에 비례하여 정보를 선택적으로 결합"하는 동일 원리가 토큰 간(Transformer), Expert 간(CGC), 헤드 간(Multi-Head Attention) 등 다양한 단위에 적용된 것이다. 본 구현의 CGC는 원 논문과 달리 블록 스케일링 + 차원 정규화를 적용하여 이종 Expert 차원 불일치를 처리한다.

초기 Bias 설정 (domain_experts)

`_build_cgc()` (라인 621 649)에서 각 태스크의 config `domain_experts` 필드를 읽어 초기 bias를 설정한다.

```
# ple_cluster_adatt.py:626-638
bias_high = float(cgc_config.get("bias_high", 1.0))
bias_low = float(cgc_config.get("bias_low", -1.0))
linear_layer.weight.zero_()          # weight는 0 시작
for i, expert_name in enumerate(expert_names):
    if expert_name in domain_experts:
        linear_layer.bias[i] = bias_high   # 선호 Expert
    else:
        linear_layer.bias[i] = bias_low    # 비선호 Expert
```

태스크	**domain_experts (bias_high=1.0)**
CTR	`perslay` , `temporal` , `unified_hgcn`
CVR	`perslay` , `temporal` , `unified_hgcn`
Churn	`perslay` , `temporal`
Retention	`perslay` , `temporal`
NBA	`perslay` , `unified_hgcn` , `lightgcn`
Life-stage	`perslay` , `temporal`
Balance_util	`temporal`
Engagement	`temporal`
LTV	`temporal` , `deepfm`
Channel	`temporal`
Timing	`temporal`
Spending_category	`unified_hgcn` , `perslay`
Consumption_cycle	`temporal`
Spending_bucket	`deepfm`

Brand_prediction	`unified_hgcn`
Merchant_affinity	`unified_hgcn` , `temporal`

Entropy 정규화 (v2.3)

`_cgc_entropy_regularization()` (라인 748 768)은 CGC attention 분포의 엔트로피를 최대화하여 **Expert Collapse**를 방지한다.

$$\mathcal{L}_{\text{entropy}} = \lambda_{\text{ent}} \cdot \left(-\frac{1}{|\mathcal{T}|}\right) \sum_{k \in \mathcal{T}} H(\boldsymbol{w}_k)$$

$$H(\boldsymbol{w}_k) = -\sum_{i=1}^{8} w_{k,i} \cdot \log(w_{k,i})$$

$\mathcal{T}$: 활성화된 태스크 집합, $\lambda_{\text{ent}} = 0.01$ (config: `cgc.entropy_lambda`)
음의 엔트로피를 최소화하면 엔트로피가 증가하여 분산 유도

수식 직관

엔트로피 H는 "gate 분포가 얼마나 고르게 퍼져 있는가"의 척도다. 하나의 Expert에 가중치가 몰리면 H가 작고, 고르면 H가 크다. 이 손실 항은 $-H$를 최소화하므로 H를 키우는 방향, 즉 gate가 여러 Expert를 골고루 참조하도록 유도한다. λ_{ent}가 클수록 균등 분산 압력이 강해진다.

학부 수학

엔트로피**(Entropy)**의 정보이론적 유도: Shannon(1948)은 "불확실성의 측도"를 공리적으로 유도했다. 확률 분포 $\boldsymbol{w} = (w_1, \ldots, w_n)$에 대해 다음 세 공리를 만족하는 유일한 함수가 엔트로피다: (1) 연속성: w_i가 조금 변하면 H도 조금 변한다, (2) 최대성: 균등 분포일 때 H가 최대, (3) 결합: 독립 사건의 엔트로피는 덧셈적. 구체적 계산 예시: Expert 8개에 대해 균등 분포 $w_i = \frac{1}{8}$이면 $H = -8 \times \left(\frac{1}{8}\right) \times \log\left(\frac{1}{8}\right) = \log(8) \approx 2.079$비트 (최대 엔트로피). 한 Expert에 집중 $\boldsymbol{w} = (1, 0, \ldots, 0)$이면 $H = -1 \times \log(1) = 0$ (최소 엔트로피). 만약 $\boldsymbol{w} = (0.65, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05)$이면 $H = -(0.65 \log 0.65 + 7 \times 0.05 \log 0.05) \approx 1.33$ — 최대의 약 64%만 활용. 엔트로피 정규화는 이 값을 최대 쪽으로 밀어 Expert 활용을 분산시킨다.

Expert Collapse 위험

CGC entropy lambda가 0이면 정규화 비활성화. 이 경우 학습 중 특정 Expert(특히 unified_hgcn 128D)에 attention이 집중되어 나머지 Expert의 gradient가 소실될 수 있다. `entropy_lambda=0.01` 이 기본값이며, 실험적으로 0.005 0.02 범위가 안정적이다.

CGC Attention 적용 (forward)

`_apply_cgc_attention()` (라인 679 725)에서 Expert별 블록에 가중치를 곱한다.

```python
# ple_cluster_adatt.py:708-725
parts = []
offset = 0
for i, dim in enumerate(self._cgc_expert_dims):
    block = shared_concat[:, offset:offset + dim]
    # v3.3: 차원 정규화 — 128D Expert는 감쇠, 64D는 증폭
    if self._cgc_dim_normalize and dim != self._cgc_mean_dim:
        scale = math.sqrt(self._cgc_mean_dim / dim)
        block = block * scale
    part = block * attention_weights[:, i:i+1]  # broadcast
    parts.append(part)
    offset += dim
return torch.cat(parts, dim=-1)
```

차원 정규화 (v3.3)

`dim_normalize=true` 일 때 Expert 출력 차원 비대칭(128D vs 64D)에 의한 기여도 불균형을 스케일링으로 보정한다.

$$\text{scale}_i = \sqrt{\frac{\text{mean\_dim}}{\text{dim}_i}}$$

$$\text{mean\_dim} = \frac{128 + 64 \times 7}{8} = 72.0$$

unified_hgcn (128D): scale = $\sqrt{\frac{72.0}{128}} \approx 0.750$ (감쇠)
나머지 Expert (64D): scale = $\sqrt{\frac{72.0}{64}} \approx 1.061$ (증폭)
동일 attention = 동일 L2 기여

> **수식 직관**
>
> unified_hgcn(128D)은 다른 Expert(64D)보다 출력 차원이 2배 크므로, 동일한 attention 가중치를 받더라도 L2 노름 기준 기여가 과대하다. 이 스케일링은 "차원이 큰 Expert는 줄이고 작은 Expert는 키워서" attention $w_{k,i} = 0.125$ (균등)일 때 모든 Expert의 실질 기여가 동등하도록 보정한다.

CGC Freeze 동기화

`on_epoch_end()` (라인 1921 1942)에서 adaTT `freeze_epoch` 에 도달하면 CGC attention 파라미터도 함께 고정한다.

```python
# ple_cluster_adatt.py:1934-1942
if (freeze_epoch is not None
        and epoch >= freeze_epoch
        and not self._cgc_frozen.item()):
    for param in self.task_expert_attention.parameters():
        param.requires_grad = False
    self._cgc_frozen.fill_(True)
```

CGC-adaTT 동기화 이유

adaTT가 전이 가중치를 고정한 뒤에도 CGC가 계속 학습하면, 두 메커니즘이 상충하는 방향으로 진화할 수 있다. 동시 고정으로 학습 후반부의 안정성을 보장한다.

HMM Triple-Mode 라우팅

3개 HMM 모드

HMM Triple-Mode (v2.0)는 고객의 행동을 3가지 시간 스케일로 분리하여 태스크별로 가장 적합한 행동 모드를 주입한다.

모드	입력	시간 스케일	대상 태스크
Journey	16D	daily	CTR, CVR, Engagement, Uplift
Lifecycle	16D	monthly	Churn, Retention, Life-stage, LTV
Behavior	16D	monthly	NBA, Balance_util, Channel, Timing, Spending_category, Consumption_cycle, Spending_bucket, Merchant_affinity, Brand_prediction

각 모드는 10D base 상태 확률 + 6D ODE dynamics bridge로 구성된다.

> 역사적 배경 – **Hidden Markov Model**의 기원과 발전
>
> HMM(Hidden Markov Model)은 **Baum & Petrie (1966)**이 통계적 언어 모델링을 위해 정식화하였다. 핵심 아이디어는 "관측 가능한 사건(observation) 뒤에 관측 불가능한 은닉 상태(hidden state)가 존재하고, 상태 간 전이가 Markov 성질을 따른다"는 것이다. 1970년대 **Rabiner & Juang**이 음성 인식에 체계적으로 적용하여 HMM이 대중화되었고, 이후 생물정보학(유전자 서열 분석), 금융(시장 상태 추정), NLP(품사 태깅) 등에 확산되었다. 본 시스템에서는 고객의 관측 가능한 행동(거래, 로그인) 뒤에 숨겨진 "여정 상태(journey)", "생애주기 상태(lifecycle)", "행동 패턴 상태(behavior)"를 HMM으로 추정하고, 각 태스크에 가장 적합한 시간 스케일의 상태 정보를 주입한다. ODE dynamics bridge는 **Neural ODE (Chen et al., NeurIPS 2018)**에서 영감을 받아 이산 HMM 상태를 연속 시간으로 보간(interpolation)하는 확장이다.

HMM 프로젝터 구조

`_build_hmm_projectors()` (라인 452 496)에서 모드별 프로젝터를 생성한다.

$$\boldsymbol{h}^{m}_{\text{hmm}} = \text{SiLU}(\text{LayerNorm}(\text{Linear}_{16\to 32}(\boldsymbol{x}^{m}_{\text{hmm}})))$$

$m \in \{\text{journey}, \text{lifecycle}, \text{behavior}\}$, 각 프로젝터 독립 학습

> 수식 직관
>
> 이 수식은 HMM이 출력한 16차원 상태 벡터(10D 상태 확률 + 6D ODE 동역학)를 모델 내부에서 사용하기 좋은 32차원으로 확장하는 과정이다. Linear로 차원을 키운 뒤, LayerNorm으로 스케일을 안정화하고, SiLU 활성화로

비선형성을 부여한다. 세 모드(journey/lifecycle/behavior) 각각이 독립 프로젝터를 가지므로, "일별 여정 패턴"과 "월별 생애주기 패턴"이 서로 다른 변환을 학습한다.

학부 수학 – **SiLU** 활성화 함수와 비선형성의 필요성

SiLU(Sigmoid Linear Unit)는 $\mathrm{SiLU}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1+e^{-x}}$로 정의된다. $\sigma(x)$는 시그모이드 함수로, 입력을 $[0, 1]$ 범위로 압축하는 "부드러운 스위치"다. SiLU는 x에 이 스위치를 곱하여 "양수 입력은 거의 그대로, 음수 입력은 부드럽게 억제"한다. 왜 비선형 활성화가 필요한가? Linear 변환만 쌓으면 $\boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x}) = (\boldsymbol{W}_2\boldsymbol{W}_1)\boldsymbol{x}$로 하나의 Linear와 동치이다. 비선형 함수를 사이에 넣어야 레이어를 쌓는 의미가 생긴다. 활성화 함수 비교: $\mathrm{ReLU}(\max(0, x))$는 $x < 0$에서 기울기가 0이 되어 "뉴런 사망" 문제가 있고, $\mathrm{GELU}(x \cdot \Phi(x)$, Gaussian CDF)는 SiLU와 유사하나 계산이 더 비싸다. SiLU는 ReLU의 경량성과 GELU의 부드러움을 절충한 것으로, $\mathrm{Mish}(x \cdot \tanh(\mathrm{Softplus}(x)))$와 함께 2020년 이후 표준 활성화 함수로 자리잡았다.

```python
# ple_cluster_adatt.py:482-486
self.hmm_projectors[mode] = nn.Sequential(
    nn.Linear(hmm_dim, proj_dim),     # 16 → 32
    nn.LayerNorm(proj_dim),
    nn.SiLU(),
)
```

학습 가능한 Default Embedding

HMM 피처가 없는 샘플(all-zero row)에 대해 zero 대신 학습 가능한 **default embedding**을 사용한다 (라인 488 493).

```python
# ple_cluster_adatt.py:488-493
self.hmm_default_embeddings = nn.ParameterDict({
    mode: nn.Parameter(torch.zeros(proj_dim))
    for mode in ["journey", "lifecycle", "behavior"]
})
```

`_forward_hmm_projectors()` (라인 1365 1414)에서 샘플별 마스킹으로 유효 샘플만 프로젝션하고, 무효 샘플은 default embedding으로 대체한다.

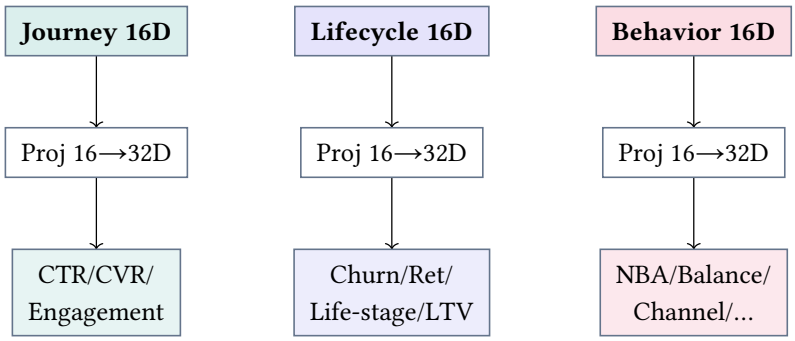

그림 2: HMM Triple-Mode 프로젝션 및 태스크 라우팅

GroupTaskExpertBasket – GroupEncoder + ClusterEmbedding (v3.2)

GroupTaskExpertBasket vs ClusterTaskExpertBasket

v3.2에서 `use_group_encoder=true` (기본값) 설정 시 `GroupTaskExpertBasket`을 사용하며, 기존 `ClusterTaskExpertBasket` 대비 **88%** 파라미터 감소를 달성한다.

항목	ClusterTaskExpertBasket (레거시)	GroupTaskExpertBasket (v3.2)
아키텍처	태스크×클러스터 독립 MLP	GroupEncoder 공유 + ClusterEmbedding
파라미터	3.0M	362K
클러스터 특화	독립 서브헤드 가중치	클러스터 임베딩 주입
일반화	낮음 (클러스터별 과적합)	높음 (공유 인코더)

GroupEncoder 아키텍처

`_build_task_experts()` (라인 498 560)에서 `GroupTaskExpertBasket`을 생성한다.

```
# ple_cluster_adatt.py:533-543
self.task_experts = GroupTaskExpertBasket(
    input_dim=task_expert_input_dim,     # 576 + 32 = 608
    group_hidden_dim=128,
    group_output_dim=64,
    cluster_embed_dim=32,
    subhead_output_dim=32,
    n_clusters=20,
    task_names=self.task_names,
    task_groups=task_groups,             # adaTT config에서 추출
)
```

GroupEncoder의 단일 태스크 forward는 다음과 같다:

$$\boldsymbol{e}_{\text{cluster}} = \text{Embedding}(\text{cluster\_id}) \in \mathbb{R}^{32}$$

$$\boldsymbol{x}_{\text{input}} = \left[\text{CGC\_output}_{576D} \parallel \text{HMM\_proj}_{32D} \parallel \boldsymbol{e}_{\text{cluster}_{32D}}\right] \in \mathbb{R}^{640}$$

$$\boldsymbol{h}_{\text{expert}} = \text{MLP}_{640\to128\to64\to32}(\boldsymbol{x}_{\text{input}})$$

실제 input_dim = 608D (shared + HMM) + 32D (cluster_embed) = 640D
그룹 내 태스크는 GroupEncoder를 공유, 그룹 간은 독립

수식 직관

이 수식은 "이 고객이 어떤 클러스터에 속하는가"라는 정보를 모델 내부에 주입하는 과정이다. 먼저 클러스터 ID를 32D 임베딩으로 변환하고, CGC 출력(576D) + HMM 프로젝션(32D)과 이어 붙여 총 640D 입력을 만든다. 이후 3단 MLP(640→128→64→32)가 이를 압축하여 태스크별 최종 표현(32D)을 생성한다. 같은 태스크 그룹(예: Engagement 그룹의 CTR/CVR)은 동일한 GroupEncoder를 공유하여 파라미터를 절약하면서도 클러스터 임베딩으로 차별화한다.

학부 수학 – 임베딩(Embedding)이란 무엇인가

Embedding(cluster_id) $\in \mathbb{R}^{32}$는 정수 인덱스를 연속 벡터로 변환하는 룩업 테이블이다. 내부적으로 $\boldsymbol{E} \in \mathbb{R}^{20\times 32}$ 행렬을 저장하고, cluster_id = c이면 $\boldsymbol{E}$의 c번째 행 $\boldsymbol{e}_c \in \mathbb{R}^{32}$를 꺼내는 것이다. 이것은 **one-hot** 인코딩 **+ Linear** 변환과 수학적으로 동치다: one-hot $\boldsymbol{v}_c \in \mathbb{R}^{20}$ (c번째만 1)을 만들면 $\boldsymbol{v}_c^T \boldsymbol{E} = \boldsymbol{e}_c$. 그러나 one-hot은 희소 벡터 연산이 필요하여 비효율적이므로, 직접 인덱싱이 더 빠르다. 핵심은 이 행렬 $\boldsymbol{E}$가 학습 가능한 파라미터라는 점이다. 학습이 진행되면서 유사한 클러스터의 임베딩 벡터는 가까워지고, 다른 클러스터는 멀어져서, 이산적 ID가 의미 있는 연속 표현으로 변환된다. Word2Vec(Mikolov et al., 2013)에서 단어를 벡터로 표현한 것과 동일한 원리다.

Soft Routing (cluster_probs)

클러스터 경계에 위치한 샘플(cluster_probs가 분산된 경우)에 대해 **soft routing**으로 여러 클러스터 임베딩의 가중 평균을 사용한다.

$$\boldsymbol{e}_{\text{cluster}} = \sum_{c=0}^{19} p_c \cdot \boldsymbol{E}_c \in \mathbb{R}^{32}$$

$$\boldsymbol{h}_{\text{expert}} = \text{TaskHead}([\text{GroupEncoder}(\boldsymbol{x}) \parallel \boldsymbol{e}_{\text{cluster}}])$$

p_c: GMM 클러스터 c의 사후 확률, $\boldsymbol{E}_c$: 클러스터 c의 학습 가능 임베딩 벡터 (32D)

수식 직관

이 수식은 경계 고객을 부드럽게 처리하는 핵심이다. 예를 들어 어떤 고객이 클러스터 3에 60%, 클러스터 7에 30%, 나머지 10% 확률로 소속되면, 각 클러스터의 임베딩 벡터를 이 비율대로 혼합한다. 구현에서는 `cluster_probs @ embedding.weight` ($[B, 20] \times [20, 32] = [B, 32]$)로 한 번의 행렬곱으로 완료된다. 혼합된 임베딩이 GroupEncoder 출력과 결합되어 TaskHead를 통과하므로, 클러스터 조건 신호가 부드럽게 주입된다. 하나의 클러스터에 강제 배정하는 hard routing과 달리, 경계 고객의 예측이 클러스터 할당 변동에 민감하지 않게 된다.

`forward_single_task()` 호출 시 `cluster_probs` 인자가 전달되면 hard assignment 대신 soft routing을 수행한다 (라인 1247 1250).

학부 수학 – GMM과 베이즈 정리: Soft Routing의 수학적 기반

클러스터 확률 p_c는 가우시안 혼합 모델**(GMM)**의 사후 확률이다. GMM은 데이터가 K개의 가우시안 분포의 혼합에서 생성되었다고 가정한다: $p(\boldsymbol{x}) = \sum_{c=1}^{K} \pi_c \cdot \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ 여기서 π_c는 혼합 가중치(사전 확률), $\mathcal{N}$은 가우시안 분포다. 베이즈 정리를 적용하면 관측 $\boldsymbol{x}$가 클러스터 c에 속할 사후 확률은: $p(c \mid \boldsymbol{x}) = \frac{\pi_c \cdot \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$ 이것이 바로 soft routing에 사용되는 p_c이다. 클러스터 중심($\boldsymbol{\mu}_c$)에서 먼 고객일수록 여러 클러스터의 사후 확률이 비슷해져 soft routing의 효과가 커지고, 중심에 가까운 고객은 하나의 클러스터에 집중되어 hard routing과 유사해진다. EM(Expectation-Maximization) 알고리즘으로 $\pi_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c$를 추정하며, 이는 오프라인에서 사전 계산된다.

최신 동향 – 클러스터 기반 추천의 산업 적용

클러스터 기반 조건 주입(conditional embedding) 전략은 2023-2025년 대규모 추천 시스템에서 핵심 기법이 되었다. Kuaishou의 **POSO (Personalized Cold-Start, KDD 2022)**는 사용자 세그먼트별 독립 gate를 두어 콜드스타트 문제를 완화하였고, Alibaba의 **CL4CTR (2023)**은 클러스터별 대조 학습으로 사용자 표현을 정제하였다. ByteDance의 **SAMD (KDD 2024)**는 클러스터 임베딩과 MoE를 결합하여 TikTok의 단기 동영상 추천에서 CTR을 2.3% 개선했다. 본 시스템의 20-클러스터 임베딩 + soft routing 설계는 이러한 산업 트렌드와 부합하며, 특히 금융 도메인에서 고객 세그먼트(VIP, 일반, 청년, 시니어 등)에 따른 행동 패턴 차이를 명시적으로 모델링하는 접근이다.

Logit Transfer – 태스크 간 명시적 정보 전달

전이 쌍 정의

`_build_logit_transfer()` (라인 984 1055)에서 `task_relationships` config로부터 전이 쌍을 등록한다.

Source	Target	유형	강도	비즈니스 의미
CTR	CVR	Sequential	0.5	클릭한 고객만 전환 (AARRR 퍼널)
Churn	Retention	Inverse	0.5	이탈 확률의 역 = 유지 기반
CVR	LTV	Feature	0.5	전환 확률이 생애가치에 영향
NBA	Spending_category	Feature	0.5	추천 행동이 소비 카테고리 결정
Spending_category	Brand_prediction	Feature	0.5	소비 카테고리가 브랜드 선택에 영향

전이 메커니즘

```python
# ple_cluster_adatt.py:1266-1277 — forward()에서 logit transfer 적용
for task_name in execution_order:
    tower_input = task_expert_outputs[task_name]
    if task_name in self.logit_transfer_sources:
        source_task = self.logit_transfer_sources[task_name]
        if source_task in predictions:
            src_out = predictions[source_task]
            if src_out.dim() == 1:
                src_out = src_out.unsqueeze(-1)
            proj = self.logit_transfer_proj[task_name](src_out)
            tower_input = tower_input + strength * proj
```

$$\boldsymbol{h}^t_{\text{tower}} = \boldsymbol{h}^t_{\text{expert}} + \alpha \cdot \text{SiLU}(\text{LayerNorm}(\text{Linear}(\text{pred}^s)))$$

$\alpha = 0.5$ (`transfer_strength`), pred^s: source 태스크 예측값

`Linear` : source output_dim ⟶ task_expert_output_dim (32D)

Projection 모듈: `nn.Sequential(Linear, LayerNorm, SiLU)`

수식 직관

이 수식은 "선행 태스크의 예측 결과를 후행 태스크의 입력에 더해주는" 명시적 전이다. 예를 들어 CTR⟶CVR 전이에서, CTR 모델이 "이 고객은 클릭 확률이 높다"고 예측하면 그 정보가 프로젝션을 거쳐 CVR 타워의 입력에 추

가된다. $\alpha = 0.5$는 원래 Expert 출력 대비 전이 신호의 상대적 강도를 조절한다. 직관적으로, 전이가 잔차(residual) 형태로 더해지므로 source 정보가 유용하지 않으면 프로젝션 가중치가 0에 수렴하여 자연스럽게 무시된다.

실행 순서 (Topological Sort)

`_derive_task_order_from_config()` (라인 1093 1155)에서 `task_relationships`의 의존 관계를 Kahn's algorithm으로 위상 정렬하여 실행 순서를 자동 도출한다.

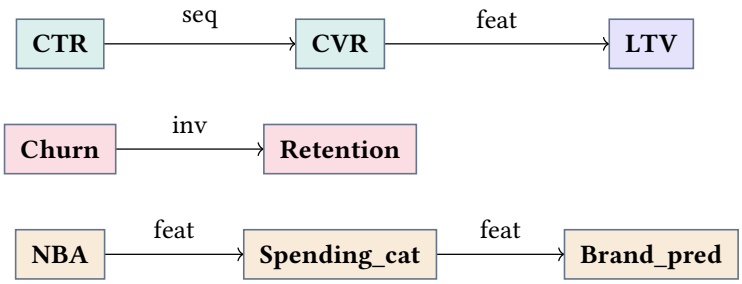

그림 3: Logit Transfer 의존 관계 그래프

실행 순서 위반 시

위상 정렬 실패(사이클 감지) 시 `_get_task_execution_order_fallback()` 이 하드코딩된 순서로 폴백한다 (라인 1069 1091). 새 전이 관계 추가 시 `task_relationships` config에 등록하면 자동으로 순서가 반영된다.

학부 수학 – 위상 정렬(Topological Sort)과 DAG

위상 정렬은 방향 비순환 그래프(DAG: Directed Acyclic Graph)의 노드를 간선 방향을 위반하지 않는 순서로 나열하는 알고리즘이다. 간선 $A \to B$가 있으면 A가 B보다 앞에 와야 한다. **Kahn's algorithm (1962)**은 다음과 같이 동작한다: (1) 진입 차수(in-degree)가 0인 노드를 큐에 삽입한다. (2) 큐에서 꺼낸 노드를 결과에 추가하고, 해당 노드의 출력 간선을 제거한다. (3) 새로 진입 차수가 0이 된 노드를 큐에 삽입한다. (4) 큐가 빌 때까지 반복한다. 모든 노드를 방문하지 못하면 사이클이 존재한다. 시간 복잡도: $O(V + E)$로, 노드 수 V와 간선 수 E에 비례한다. 본 시스템에서 CTR $\to$ CVR $\to$ LTV 체인은 "CTR을 먼저 예측해야 CVR에 전이할 수 있고, CVR을 먼저 예측해야 LTV에 전이할 수 있다"는 인과적 선후 관계를 그래프로 표현한 것이다.

역사적 배경 – 잔차 연결(Residual Connection)과 Logit Transfer

Logit Transfer의 `tower_input = tower_input + alpha * proj` 형태는 **He, Zhang, Ren & Sun (CVPR 2016)**이 ResNet에서 제안한 잔차 연결(skip connection)과 동일한 구조다. He et al.은 152층 깊은 네트워크에서 기울기 소실 없이 학습하려면 $\boldsymbol{y} = \boldsymbol{x} + \mathcal{F}(\boldsymbol{x})$처럼 "입력을 그대로 더해주는 지름길"이 필요함을 보였다. 이 아이디어는 **Highway Networks (Srivastava et al., 2015)**에서 먼저 탐구되었으나, ResNet의 극단적 단순화($\boldsymbol{y} = \boldsymbol{x} + \mathcal{F}(\boldsymbol{x})$, gate 없음)가 더 효과적이었다. Logit Transfer에서도 source 태스크의 예측이 잔차 형태로 더해지므로, 전이 정보가 유용하지 않으면 프로젝션 가중치가 0으로 수렴하여 원래 Expert 출력만 남게 되는 안전한 기본값(**safe default**)을 보장한다. $\alpha = 0.5$는 이 잔차의 상대적 크기를 제한하는 스케일링 계수다.

Logit Transfer vs adaTT – 두 전이 메커니즘의 차이와 보완 관계

본 시스템은 태스크 간 지식 전달을 두 가지 서로 다른 수준에서 동시에 수행한다.

특성	**Logit Transfer**	**adaTT**
작동 계층	Feature/Logit 수준 (forward pass 중)	Loss 수준 (backward pass 전)
전달 내용	선행 태스크의 예측값/은닉 표현	태스크 간 gradient 친화도
방향성	단방향 DAG (CTR→CVR→LTV)	전방향 행렬 (모든 태스크 쌍)
학습 가능성	고정 구조 (수동 설계)	적응적 (EMA로 친화도 학습)
목적	순차적 의존성 명시적 전달	Negative Transfer 자동 완화

Logit Transfer와 adaTT는 '태스크 간 지식 전달'이라는 같은 목표를 다른 수준에서 수행한다. Logit Transfer는 CTR→CVR처럼 비즈니스 로직상 순차적인 태스크에 예측값을 직접 전달하고, adaTT는 gradient 수준에서 모든 태스크 쌍의 상호 영향을 적응적으로 조절한다. 둘은 상호 보완적이며 동시에 작동한다.

상세 adaTT 메커니즘은 *adaTT* 기술 참조서 를 참조한다.

Task Tower – 최종 예측

Tower 아키텍처

`TaskTower` 클래스 (라인 244 293)는 모든 태스크에 공통된 MLP 구조를 사용한다.

$$\boldsymbol{y} = \text{Linear}_{32\to\text{ out}} \circ \text{Dropout} \circ \text{SiLU} \circ \text{LayerNorm} \circ \text{Linear}_{64\to32} \circ \text{Dropout} \circ \text{SiLU} \circ \text{LayerNorm} \circ \text{Linear}_{32\to64}(\boldsymbol{h}_{\text{expert}})$$

입력: 32D (Task Expert 출력), hidden_dims: [64, 32], dropout: 0.2
Regression 태스크는 activation=None, Binary는 sigmoid, Multiclass는 softmax

수식 직관

Task Tower는 32D Expert 출력을 받아 최종 예측값으로 변환하는 "마지막 한 걸음"이다. 32→64로 확장하여 표현력을 키운 뒤, 64→32로 압축하고, 마지막에 출력 차원으로 사영한다. 각 층 사이에 LayerNorm(스케일 안정화) + SiLU(비선형성) + Dropout(과적합 방지)을 끼워 얕은 MLP이면서도 안정적 학습이 가능하다. 태스크 유형에 따라 출력 활성화가 달라지며, Binary는 sigmoid로 0 1 확률을, Multiclass는 softmax로 클래스 분포를, Regression은 활성화 없이 실수값을 출력한다.

```python
# ple_cluster_adatt.py:268-280
layers = []
prev_dim = input_dim  # 32
for hidden_dim in hidden_dims:  # [64, 32]
    layers.extend([
        nn.Linear(prev_dim, hidden_dim),
        nn.LayerNorm(hidden_dim),
        nn.SiLU(),
        nn.Dropout(dropout),
    ])
    prev_dim = hidden_dim
layers.append(nn.Linear(prev_dim, output_dim))
```

학부 수학 – LayerNorm의 수학적 정의와 역할

Layer Normalization은 각 샘플의 은닉 벡터를 독립적으로 정규화한다: $\text{LayerNorm}(\boldsymbol{x}) = \gamma \cdot \frac{\boldsymbol{x}-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta$ 여기서 $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$ (평균), $\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i-\mu)^2$ (분산), $\gamma, \beta \in \mathbb{R}^d$는 학습 가능한 스케일/시프트 파라미터, $\varepsilon \approx 10^{-5}$는 분모 0 방지다. 왜 정규화가 필요한가? 신경망의 각 레이어는 이전 레이어의 출력을 입력으로 받는데, 학습 중 이전 레이어의 파라미터가 변하면 입력 분포가 계속 바뀌는 내부 공변량 이동**(Internal Covariate Shift)**이 발생한다. 이는 학습률 설정을 어렵게 만든다. LayerNorm은 각 레이어 입력을 평균 0, 분산 1로 정규화하여 분포를 안정화한다. **BatchNorm**과의 차이: BatchNorm은 배치 내 같은 뉴런을 정규화하고(배치 크기 의존), LayerNorm은 한 샘플 내 모든 뉴런을 정규화한다(배치 크기 무관). Task Tower처럼 배치 크기가 가변적인 추론 환경에서는 LayerNorm이 더 안정적이다.

태스크별 Loss 유형

태스크	유형	Loss	출력 dim	Activation	Weight
CTR	Binary	Focal (γ=2.0, α=0.25)	1	sigmoid	1.0
CVR	Binary	Focal (γ=2.0, α=0.20)	1	sigmoid	1.5
Churn	Binary	Focal (γ=2.0, α=0.60)	1	sigmoid	1.2
Retention	Binary	Focal (γ=2.0, α=0.20)	1	sigmoid	1.0
NBA	Multiclass	NLL (softmax후)	12	softmax	2.0
Life-stage	Multiclass	NLL	6	softmax	0.8
Balance_util	Regression	Huber (δ=1.0)	1	none	1.0
Engagement	Regression	MSE	1	none	0.8
LTV	Regression	Huber (δ=1.0)	1	none	1.5
Channel	Multiclass	NLL	3	softmax	0.8
Timing	Multiclass	NLL	28	softmax	0.8
Spending_category	Multiclass	NLL	12	softmax	1.2
Consumption_cycle	Multiclass	NLL	7	softmax	0.8
Spending_bucket	Regression	Huber (δ=1.0)	1	none	0.8
Brand_prediction	Contrastive	InfoNCE (τ=0.07)	128	none	2.0
Merchant_affinity	Regression	Huber (δ=1.0)	1	none	1.0

학부 수학 – Huber Loss: MSE와 MAE의 절충

Huber Loss는 **Peter J. Huber (1964)**가 제안한 로버스트(robust) 손실 함수다: $\mathcal{L}_{\text{Huber}(y,\hat{y})} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \le \delta \\ \delta \cdot \left(|y-\hat{y}| - \frac{\delta}{2}\right) & \text{otherwise} \end{cases}$ 오차가 작을 때($|y-\hat{y}| \le \delta$)는 MSE처럼 L_2(제곱), 클 때는 MAE처럼 L_1(절대값)로 동작한다. 왜 **MSE**만으로 부족한가? MSE = $(y-\hat{y})^2$는 이상치(outlier)에서 제곱으로 인해 손실이 폭발하여 gradient가 매우 커지고, 모델이 이상치 하나에 과도하게 끌려간다. MAE = $|y-\hat{y}|$는 이상치에 강하지만, 0에서 미분 불가능하고 gradient가 상수(± 1)라서 0 근처에서 수렴이 느리다. Huber Loss는 0 근처에서는 MSE의 부드러운 gradient를, 멀리에서는 MAE의 이상치 내성을 결합한다. $\delta = 1.0$은 "오차 1 이내는 정밀 추적, 1 이상은 이상치 방어"라는 경계를 설정한다. LTV(생애가치) 예측처럼 극단적 고액 소비자가 존재하는 태스크에서 Huber Loss가 MSE보다 안정적이다.

최신 동향 – InfoNCE와 대조 학습: Brand Prediction의 손실 함수

Brand Prediction 태스크에 사용된 **InfoNCE (Noise-Contrastive Estimation)**은 **Oord, Li & Vinyals (2018)**이 CPC(Contrastive Predictive Coding)에서 제안한 손실이다: $\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp(\boldsymbol{q}\cdot\boldsymbol{k}_+/\tau)}{\sum_{i=0}^{N}\exp(\boldsymbol{q}\cdot\boldsymbol{k}_i/\tau)}$ 여기서 $\boldsymbol{q}$는 쿼리, $\boldsymbol{k}_+$는 양성 키, $\boldsymbol{k}_i$는 음성 키, $\tau = 0.07$은 온도 파라미터다. **SimCLR (Chen et al., ICML 2020)**과 **MoCo (He et al., CVPR 2020)**가 이를 시각 표현 학습에 적용하여 대조 학습의 붐을 일으켰고, 2023-2025년에는 추천 시스템의 **SASRec-CL**, **CL4Rec** 등 시퀀셜 추천에 광범위하게 채택되었다. 본 시스템에서 Brand Prediction을 대조 학습으로 학습하

는 이유는 수천 개의 브랜드를 직접 분류하는 대신, 브랜드 임베딩 공간에서 유사 브랜드를 가깝게, 비유사 브랜드를 멀리 배치하는 것이 확장성과 일반화에 유리하기 때문이다.

Focal Loss 구현

`_compute_task_losses()` (라인 1765 1780)에서 확률값 기반 Focal Loss를 계산한다. TaskTower가 이미 sigmoid를 적용하므로 이중 **sigmoid** 방지를 위해 logits 기반이 아닌 확률값 기반으로 구현한다.

$$\text{FL}(p_t) = -\alpha_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t)$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

$$\alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{if } y = 0 \end{cases}$$

$\gamma = 2.0$: focusing parameter (쉬운 예제 감쇠 강도)
α: 양성 클래스 가중치 (태스크별 차별화, 상세 설계는 config 참조)

수식 직관

Focal Loss는 표준 Cross-Entropy에 $(1 - p_t)^{\gamma}$ 가중치를 곱한 것이다. p_t는 "모델이 정답에 부여한 확률"이므로, 잘 맞추는 예제(p_t가 큼)에는 가중치가 급격히 줄어들고, 틀리는 예제(p_t가 작음)에는 가중치가 유지된다. 직관적으로, "쉬운 문제를 계속 풀어봐야 실력이 안 오르니, 어려운 문제에 집중하라"는 학습 전략을 손실 함수로 구현한 것이다. α_t는 클래스 불균형 보정으로, 희소한 양성 클래스(예: 이탈 고객)를 놓치지 않도록 가중치를 높인다.

학부 수학

Cross-Entropy는 왜 $-\log(p)$인가? 정보이론에서 어떤 사건의 정보량**(information content)**은 $I(x) = -\log_2(p(x))$로 정의된다. 확률이 낮은 사건(놀라운 사건)일수록 정보량이 크다. 예: 동전 앞면($p = 0.5$)의 정보량 = $-\log_2(0.5) = 1$비트, 주사위 1($p = \frac{1}{6}$)의 정보량 = $-\log_2\left(\frac{1}{6}\right) \approx 2.58$비트. **Cross-Entropy** $H(p, q) = -\sum p(x) \log q(x)$는 "실제 분포 p를 따르는 데이터를 모델 분포 q로 인코딩할 때 필요한 평균 비트 수"이다. 이진 분류에서 정답이 $y = 1$이고 모델 예측이 p이면, $-\log(p)$는 "정답에 모델이 부여한 확률이 낮을수록 큰 벌점"이 된다. **Focal Loss**와의 관계: $\text{FL} = -(1 - p_t)^{\gamma} \log(p_t)$에서 $(1 - p_t)^{\gamma}$는 이 벌점에 "예측 틀린 정도"에 비례하는 가중치를 곱한 것이다. 구체적 계산: $p_t = 0.9$이면 $\text{CE} = -\log(0.9) = 0.105$, $\text{FL} = 0.1^2 \times 0.105 = 0.00105$ (100배 감소). $p_t = 0.1$이면 $\text{CE} = -\log(0.1) = 2.303$, $\text{FL} = 0.9^2 \times 2.303 = 1.865$ (거의 유지).

역사적 배경

Focal Loss는 **Lin, Goyal, Girshick, He & Dollár (ICCV 2017)**이 물체 탐지(Object Detection)에서 제안하였다. 당시 one-stage 탐지기(예: YOLO, SSD)가 two-stage 탐지기(Faster R-CNN)보다 정확도가 낮았는데, 원인은 배경(easy negative)이 전경(hard positive)보다 압도적으로 많아 쉬운 예제의 gradient가 학습을 지배하기 때문이었다. Focal Loss는 $(1 - p_t)^{\gamma}$ 항으로 쉬운 예제의 기여를 동적으로 줄여, one-stage 탐지기(RetinaNet)가 처음으로 two-stage를

능가하는 결과를 이끌어냈다. 이후 추천 시스템, 의료 영상, 자연어 처리 등 클래스 불균형이 심한 거의 모든 분야에서 표준 손실 함수로 채택되었다. CTR 예측(양성 비율 3 8%)은 Focal Loss의 전형적인 적용 사례다.

```
# ple_cluster_adatt.py:1774-1780 — fp16 AMP 안전 focal loss
p_f = pred.squeeze().float().clamp(1e-7, 1 - 1e-7)
t_f = target.float()
bce = -(t_f * torch.log(p_f) + (1 - t_f) * torch.log(1 - p_f))
p_t = p_f * t_f + (1 - p_f) * (1 - t_f)
alpha_t = alpha * t_f + (1 - alpha) * (1 - t_f)
focal_weight = alpha_t * (1 - p_t) ** gamma
loss = (focal_weight * bce).mean()
```

Focal Alpha 설계 기준

`focal_alpha` 는 양성 비율과 비즈니스 **FN** 비용의 두 요인으로 결정된다.

- CTR (양성 3 8%, FN비용 중간): $\alpha = 0.25$ (표준)
- CVR (양성 0.5 3%, FN비용 높음): $\alpha = 0.20$ (음성 경계 학습 강화)
- Churn (양성 5 15%, FN비용 매우 높음): $\alpha = 0.60$ (이탈 놓침 방지, recall 극대화)
- Retention (양성 85 95%, FN비용 중간): $\alpha = 0.20$ (소수 이탈전조 탐지)

Uncertainty Weighting (Kendall et al.)

`loss_weighting.strategy: "uncertainty"` 설정 시 (라인 1818 1827) 태스크별 학습 가능한 log variance로 **homoscedastic uncertainty**를 모델링한다.

$$\mathcal{L}_k^{\text{uw}} = w_k \cdot (\exp(-s_k) \cdot \mathcal{L}_k + s_k)$$

$s_k = \log(\sigma_k^2)$: 학습 가능한 log variance (`task_log_vars[k]`)
$\exp(-s_k)$: precision (불확실성 높으면 가중치 낮춤)
s_k: 불확실성을 무한히 키우는 것을 방지하는 정규화 항
s_k는 $[-4.0, 4.0]$으로 clamp, precision은 $[10^{-3}, 100]$으로 clamp

수식 직관

이 수식은 "어떤 태스크가 본질적으로 예측하기 어려우면, 그 태스크의 손실이 전체 학습을 지배하지 않도록 자동으로 가중치를 낮추는" 메커니즘이다. $\exp(-s_k)$는 정밀도(precision)로서, 불확실성이 높으면(s_k 큼) 작아져 손실 기여를 줄인다. 동시에 $+s_k$ 항이 정규화 역할을 하여, 모델이 "모든 태스크를 불확실하다고 선언하여 손실을 0으로 만드는" 편법을 방지한다. 16개 태스크의 가중치를 수동으로 튜닝하는 대신, 모델이 자동으로 균형을 찾는다.

NeurIPS 2018 Kendall, A., Gal, Y., & Cipolla, R. *"Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics"*

역사적 배경

Uncertainty Weighting은 **Kendall, Gal & Cipolla (CVPR 2018)**이 제안하였다. 원래 자율주행의 장면 이해(scene understanding) 문제에서 깊이 추정, 표면 법선 추정, 의미 분할이라는 세 태스크의 가중치를 자동으로 조정하기 위해 고안되었다. 이론적 기반은 **homoscedastic uncertainty**(데이터 포인트가 아닌 태스크 자체의 불확실성)의 최대 우도 추정(MLE)이다. 각 태스크의 likelihood를 Gaussian으로 가정하면 log-likelihood를 정리했을 때 자연스럽게 $\exp(-s_k) \cdot \mathcal{L}_k + s_k$ 형태가 유도된다. 이전에는 수동 grid search나 GradNorm (Chen et al., ICML 2018)처럼 gradient 크기를 균등화하는 방법이 사용되었으나, Uncertainty Weighting은 추가 하이퍼파라미터 없이 학습 가능한 파라미터로 대체하여 널리 채택되었다.

최신 동향

2024-2025년 MTL loss balancing 분야는 Uncertainty Weighting을 넘어서는 방법들이 연구되고 있다. **Nash-MTL** (Navon et al., ICML 2022)은 태스크 간 gradient를 Nash 협상(bargaining) 게임으로 모델링하여 Pareto 최적 해를 찾고, **Aligned-MTL** (Senushkin et al., CVPR 2023)은 gradient 방향을 정렬하여 충돌을 최소화한다. **Auto-Lambda** (Liu et al., 2024)는 메타 학습으로 가중치를 적응적으로 조정한다. 그러나 실무에서는 Uncertainty Weighting이 구현 단순성과 안정적 성능으로 여전히 가장 널리 사용되며, 특히 태스크 수가 10개 이상인 대규모 MTL에서 검증된 선택지로 남아 있다.

총 손실 집계

`forward()` (라인 1284 1344)에서 다음 손실들이 합산된다:

1. 태스크 손실: adaTT 적용 후 enhanced losses 합계 (또는 단순 합계)
2. **CGC Entropy** 정규화: $\lambda_{\text{ent}} \times \mathcal{L}_{\text{entropy}}$ (학습 시, CGC 미고정 시)
3. **Causal Expert DAG** 정규화: acyclicity + sparsity
4. **SAE** 손실: reconstruction + L1 sparsity (weight=0.01, detached)

SAE (Sparse Autoencoder) – Expert 해석성

목적

Shared Expert 결합 표현(576D)에서 해석 가능한 sparse feature를 추출한다. Anthropic의 Sparse Autoencoder 접근법에서 영감을 받아, 신경망 내부 표현을 인간이 해석 가능한 단위로 분해하는 것이 목적이다.

아키텍처

`_build_sae()` (라인 877 896)에서 `SparseAutoencoder`를 생성한다.

$$\boldsymbol{z} = \mathrm{ReLU}(\boldsymbol{W}_{\text{enc}} \cdot \boldsymbol{h}_{\text{shared}} + \boldsymbol{b}_{\text{enc}}) \in \mathbb{R}^{2304}$$

$$\hat{\boldsymbol{h}} = \boldsymbol{W}_{\text{dec}} \cdot \boldsymbol{z} + \boldsymbol{b}_{\text{dec}} \in \mathbb{R}^{576}$$

$$\mathcal{L}_{\text{SAE}} = \|\boldsymbol{h}_{\text{shared}} - \hat{\boldsymbol{h}}\|_2^2 + \lambda_1 \, \|\boldsymbol{z}\|_1$$

`expansion_factor=4` : latent_dim = 576 × 4 = 2304
`l1_lambda=0.001` : sparsity 유도
`tied_weights=true` : $\boldsymbol{W}_{\text{dec}} = \boldsymbol{W}_{\text{enc}}^T$ (파라미터 절약)
`loss_weight=0.01` : 총 손실에 기여하는 비율

수식 직관

첫째 식은 인코딩: 576D 공유 표현을 4배 확장한 2304D 희소 벡터 $\boldsymbol{z}$로 변환한다. ReLU 덕분에 대부분의 원소가 0이 되어, 활성화된 소수의 원소만이 "이 고객의 표현에서 어떤 개념이 켜져 있는가"를 나타낸다. 둘째 식은 디코딩: 희소 벡터에서 원래 576D를 복원하여 정보 손실을 최소화한다. 셋째 식의 손실은 복원 오차(L_2)와 희소성 제약(L_1)의 합이다. 직관적으로, "가능한 적은 수의 개념으로 전문가 표현을 설명하되, 원래 정보를 잃지 말라"는 두 목표의 균형이다.

학부 수학

L1 노름과 **L2** 노름의 차이: $\|\boldsymbol{z}\|_1 = \sum_i |z_i|$는 각 원소의 절댓값 합이고, $\|\boldsymbol{h} - \hat{\boldsymbol{h}}\|_2^2 = \sum_i \left(h_i - \hat{h}_i\right)^2$는 차이의 제곱합이다. L1 노름을 최소화하면 많은 원소가 정확히 **0**이 되는 희소(sparse) 해를 유도한다. 이는 L1의 기하학적 성질 때문인데, L1 공의 꼭짓점이 축 위에 있어서 제약 하 최적화 시 해가 축 위(= 나머지 좌표 0)에 놓이기 쉽다. 반면 L2 노름은 원(구)의 형태로 해가 고르게 분산되어 0이 잘 나오지 않는다. 구체적 예시: $\boldsymbol{z} = [3, 0, 0, 2, 0]$이면 $\|\boldsymbol{z}\|_1 = 5$, 0이 아닌 원소가 2개뿐이다. 이렇게 희소한 $\boldsymbol{z}$는 "5개 개념 중 2개만 활성"이라는 해석이 가능하다.

역사적 배경 – 오토인코더의 역사: 차원 축소에서 해석성까지

오토인코더(Autoencoder)의 개념은 **Rumelhart, Hinton & Williams (1986)**의 역전파 논문에서 "자기 자신을 복원하도록 학습하면 중간 은닉층에 유용한 표현이 형성된다"는 관찰로 시작되었다. 이후 **Vincent et al. (ICML 2008)**의 Denoising Autoencoder, **Kingma & Welling (ICLR 2014)**의 Variational Autoencoder(VAE)로 발전하였다. **Sparse Autoencoder**는 은닉 표현에 L1 페널티를 부여하여 소수의 뉴런만 활성화되도록 강제하는 변형으로, **Andrew Ng**이 2011년 Stanford 강의에서 체계화하였다. 핵심 아이디어는 PCA(주성분 분석)와 유사하게 차원을 축소하되, 비선형 변환을 허용하고, 과완전**(overcomplete)** 표현($\dim(z) > \dim(x)$)에서도 L1 희소성으로 의미 있는 특징을 추출할 수 있다는 점이다. 본 시스템의 SAE는 576D → 2304D 과완전 인코딩을 사용하여, 576차원에 뒤섞여 있는 Expert 정보를 2304개의 해석 가능한 단위로 "풀어헤치는" 역할을 한다.

최신 동향

Sparse Autoencoder를 신경망 해석에 적용하는 기계적 해석성**(Mechanistic Interpretability)**은 2023년 Anthropic의 연구("Towards Monosemanticity", Bricken et al., 2023)에서 대규모 언어 모델의 잔차 스트림에 SAE를 적용하여 해석 가능한 특징(feature)을 추출한 것이 기폭제가 되었다. 2024-2025년에는 OpenAI, DeepMind, EleutherAI 등에서도 SAE 기반 해석을 활발히 연구 중이며, Templeton et al. (Anthropic, 2024)은 Claude 3 Sonnet에서 수백만 개의 해석 가능한 특징을 추출했다. 추천 시스템 분야에서도 모델의 내부 표현을 SAE로 분해하여 "왜 이 상품을 추천했는가"를 설명하는 연구가 증가하고 있으며, EU AI Act(2024 발효)의 설명 가능성 요건이 이러한 추세를 가속화하고 있다.

Main Path Gradient 차단

`forward()` (라인 1216)에서 `shared_concat.detach()`로 SAE 입력을 분리한다. SAE 손실은 SAE 자체 가중치만 업데이트하며, Shared Expert의 학습에 영향을 주지 않는다.

```
# ple_cluster_adatt.py:1216
_, sae_latent, sae_loss = self.sae(shared_concat.detach())
```

SAE latent 활용

`PLEClusterOutput.sae_latent` (2304D sparse vector)는 추론 후 **Expert Neuron Dashboard**에서 활성화 패턴 분석에 사용된다. 예를 들어 "자주 활성화되는 latent 147은 '카드론 이용 패턴'에 대응"과 같은 해석이 가능하다.

Evidential Deep Learning – 불확실성 정량화

목적

태스크 예측의 **epistemic uncertainty**(모델이 "얼마나 모르는지")를 정량화하여 추천 신뢰도를 평가한다. 높은 불확실성을 가진 예측은 서빙 시 **fallback** 로직으로 전환된다.

원리

NeurIPS 2018 Sensoy, M., Kaplan, L., & Kandemir, M. *"Evidential Deep Learning to Quantify Classification Uncertainty"*

분류 태스크에서 Softmax 출력 대신 **Dirichlet** 분포의 파라미터를 예측한다.

$$\boldsymbol{\alpha} = \text{evidence} + 1 \quad (\boldsymbol{\alpha} \in \mathbb{R}_+^K)$$

$$S = \sum_{k=1}^{K} \alpha_k \quad \text{(Dirichlet strength)}$$

$$\hat{p}_k = \frac{\alpha_k}{S} \quad \text{(expected probability)}$$

$$u = \frac{K}{S} \quad \text{(epistemic uncertainty)}$$

K: 클래스 수, S 클수록 확신, u 클수록 불확실
evidence가 0이면 $\boldsymbol{\alpha} = \mathbf{1} \rightarrow$ 균등 분포 $\rightarrow$ 최대 불확실

수식 직관

기존 Softmax 분류기는 "어떤 입력이든 항상 하나의 확률 분포를 출력"하여, 학습 데이터에 없던 패턴에도 자신 있게 예측하는 위험이 있다. Evidential 접근은 "확률 분포의 분포"(Dirichlet)를 모델링한다. $\boldsymbol{\alpha}$는 Dirichlet 분포의 농도 파라미터로, evidence(증거)가 많을수록 $S = \sum \alpha_k$가 커져 분포가 뾰족해지고(확신), 불확실성 $u = K/S$가 줄어든다. 직관적으로, "증거가 충분히 쌓이면 확신하고, 증거가 없으면 솔직하게 모르겠다고 말한다"는 인식론적 불확실성의 정량화다.

학부 수학 – Dirichlet 분포: "확률의 확률"을 모델링하는 분포

Dirichlet 분포 $\text{Dir}(\boldsymbol{p} \mid \boldsymbol{\alpha})$는 확률 심플렉스(simplex) 위의 분포다. K-차원 확률 벡터 $\boldsymbol{p} = (p_1, \ldots, p_K)$ ($\sum p_k = 1$, $p_k \geq 0$)를 생성하며, 확률 밀도 함수는 $f(\boldsymbol{p} \mid \boldsymbol{\alpha}) = \frac{\Gamma(\sum \alpha_k)}{\prod \Gamma(\alpha_k)} \prod_{k=1}^{K} p_k^{\alpha_k - 1}$이다. $\Gamma(n)$은 감마 함수로, 자연수에서는 $\Gamma(n) = (n-1)!$(팩토리얼의 연속 확장)이다. 직관적 이해: α_k가 모두 1이면 균등 분포(어떤 $\boldsymbol{p}$든 동등), α_k가 모두 크면 중심 $(1/K, \ldots, 1/K)$ 근처에 집중(확신), 특정 α_k만 크면 해당 클래스 쪽으로 치우침. 구체적 예시 ($K = 3$): $\boldsymbol{\alpha} = (1, 1, 1)$이면 삼각형 위에 균일하게 분포한다. $\boldsymbol{\alpha} = (10, 10, 10)$이면 $(1/3, 1/3, 1/3)$ 근처에 집중 – "세 클래스

확률이 비슷하다고 확신". $\boldsymbol{\alpha} = (100, 1, 1)$이면 $(1, 0, 0)$ 근처에 집중 – "클래스 1이 거의 확실하다고 확신". 이처럼 Dirichlet 분포의 $\boldsymbol{\alpha}$를 신경망이 예측하면, "예측 확률 자체의 분산"까지 정량화하여 불확실성을 표현할 수 있다.

역사적 배경

Evidential Deep Learning은 **Sensoy, Kaplan & Kandemir (NeurIPS 2018)**이 제안하였다. 이들은 Dempster-Shafer 증거 이론(1968, 1976)과 주관적 논리(Subjective Logic, Jøsang 2016)를 신경망에 접목하여, Softmax 출력의 과신(overconfidence) 문제를 해결하고자 했다. 핵심 아이디어는 "확률의 확률"을 모델링하는 것으로, 베이지안 통계에서 사후 분포(posterior)에 Dirichlet prior를 놓는 전통(Ferguson, 1973)에서 영감을 받았다. 이후 Amini et al. (NeurIPS 2020)이 회귀 문제로 확장한 **Evidential Regression**을 제안하여 Normal-Inverse-Gamma(NIG) 분포로 연속값 예측의 불확실성을 정량화하였다.

최신 동향

2024-2025년 Evidential DL 분야는 교정**(calibration)** 개선과 **OOD(Out-of-Distribution)** 탐지 성능 향상에 집중되고 있다. Pandey & Yu (AAAI 2023)의 Posterior Network과 Charpentier et al.의 Natural Posterior Network이 Normalizing Flow를 결합하여 증거 추정의 정확도를 높였다. 산업적으로는 자율주행(Waymo, 2024), 의료 진단(Google Health), 금융 리스크 평가에서 모델 불확실성 정량화가 규제 요건으로 부상하면서 실무 채택이 가속화되고 있다. 특히 LLM의 hallucination 감지에 evidential 접근을 적용하는 연구(Ren et al., 2024)도 주목받고 있다.

구현

`_build_evidential_layers()` (라인 898 931)에서 태스크별 `EvidentialLayer`를 생성한다.

```python
# ple_cluster_adatt.py:921-927
self.evidential_layers[task_name] = EvidentialLayer(
    input_dim=self.task_expert_output_dim,  # 32D
    task_type=task_type,
    output_dim=output_dim,
    kl_lambda=0.01,
    annealing_epochs=10,
)
```

Forward (라인 1253 1260)에서 Task Expert 출력(32D)에 병렬로 적용되며, `compute_evidential_loss()` (라인 1838 1841)로 보조 KL 손실을 가산한다.

$$\mathcal{L}_{\text{evi}} = \mathcal{L}_{\text{task}} + \lambda_{\text{KL}} \cdot \min\left(1, \frac{\text{epoch}}{\text{anneal}}\right) \cdot \text{KL}(\text{Dir}(\boldsymbol{\alpha}) \parallel \text{Dir}(\mathbf{1}))$$

`kl_lambda=0.01`, `annealing_epochs=10` : 초기에는 KL 기여 작게 시작
학습 초반 KL이 너무 강하면 모든 예측이 균등 분포로 수렴하는 문제 방지

수식 직관

이 손실은 두 부분으로 구성된다. 첫째, 원래 태스크 손실($\mathcal{L}_{\text{task}}$)은 예측 정확도를 높인다. 둘째, KL 항은 "증거가 없는 클래스의 α를 1(무정보 상태)로 되돌리라"는 압력이다. annealing 계수 $\min\left(1, \frac{\text{epoch}}{\text{anneal}}\right)$은 학습 초반에 KL 기여를 약하게 시작하여 모델이 먼저 기본적인 분류 능력을 갖춘 후에 불확실성 교정에 집중하도록 한다. 직관적으로, "처음에는 정답 맞추기에 집중하고, 어느 정도 실력이 쌓이면 자신의 확신도를 정직하게 표현하는 법을 배워라"이다.

18 태스크 전체 사양

아래는 시스템에 정의된 전체 18개 태스크의 완전한 사양이다. 현재 16개가 활성화되어 있으며, uplift과 category_uplift이 비활성화 상태이다.

태스크	그룹	Loss	dim	HMM 모드	Weight	활성화
CTR	Engagement	Focal	1	journey	1.0	O
CVR	Engagement	Focal	1	journey	1.5	O
Engagement	Engagement	MSE	1	journey	0.8	O
Uplift	Engagement	MSE	1	journey	1.0	X
Churn	Lifecycle	Focal	1	lifecycle	1.2	O
Retention	Lifecycle	Focal	1	lifecycle	1.0	O
Life-stage	Lifecycle	NLL	6	lifecycle	0.8	O
LTV	Lifecycle	Huber	1	lifecycle	1.5	O
Balance_util	Value	Huber	1	behavior	1.0	O
Channel	Value	NLL	3	behavior	0.8	O
Timing	Value	NLL	28	behavior	0.8	O
NBA	Consumption	NLL	12	behavior	2.0	O
Spending_category	Consumption	NLL	12	behavior	1.2	O
Consumption_cycle	Consumption	NLL	7	behavior	0.8	O
Spending_bucket	Consumption	Huber	1	behavior	0.8	O
Category_uplift	Consumption	MSE	12	behavior	1.5	X
Brand_prediction	Consumption	InfoNCE	128	behavior	2.0	O
Merchant_affinity	Consumption	Huber	1	behavior	1.0	O

태스크 그룹 (adaTT config)

그룹	멤버	intra 강도	inter 강도
Engagement	CTR, CVR, Engagement, (Uplift)	0.8	0.3
Lifecycle	Churn, Retention, Life-stage, LTV	0.7	0.3
Value	Balance_util, Channel, Timing	0.6	0.3
Consumption	NBA, Spending_category, Consumption_cycle, Spending_bucket, Merchant_affinity, Brand_prediction	0.7	0.3

학부 수학 – **intra/inter** 전이 강도의 의미

adaTT의 **intra** 강도(= 0.6 ∼ 0.8)는 같은 그룹 내 태스크 간 gradient 전이 비율이고, **inter** 강도(= 0.3)는 다른 그룹 태스크 간 전이 비율이다. 수학적으로 태스크 i의 gradient가 태스크 j에 전이되는 양은: $\boldsymbol{g}_j^{\text{transferred}} = \boldsymbol{g}_j + \alpha_{ij} \cdot \text{proj}(\boldsymbol{g}_i, \boldsymbol{g}_j)$ 여기서 $\text{proj}(\boldsymbol{g}_i, \boldsymbol{g}_j) = \frac{\boldsymbol{g}_i \cdot \boldsymbol{g}_j}{\|\boldsymbol{g}_j\|^2} \cdot \boldsymbol{g}_j$는 $\boldsymbol{g}_i$를 $\boldsymbol{g}_j$ 방향으로 사영(projection)한 것이다. 사영**(projection)**은 "벡터 $\boldsymbol{a}$에서 벡터 $\boldsymbol{b}$ 방향 성분만 추출"하는 연산으로, $\text{proj}_{\boldsymbol{b}}(\boldsymbol{a}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{b}\|^2}\boldsymbol{b}$로 정의된다. 직관적으로, 같은 그룹(예: CTR과 CVR)은 gradient 방향이 유사하여 큰 전이($\alpha = 0.8$)가 유익하고, 다른 그룹(예: CTR과 Churn)은 gradient가 충돌할 수 있어 작은 전이($\alpha = 0.3$)가 안전하다.

최신 동향 – **Gradient** 기반 멀티태스크 최적화의 최전선

adaTT의 gradient 기반 전이는 **PCGrad (Yu et al., NeurIPS 2020)**에서 시작된 연구 흐름의 연장이다. PCGrad는 충돌하는 gradient를 서로의 법선(normal) 방향으로 사영하여 충돌을 제거하고, **CAGrad (Liu et al., NeurIPS 2021)**는 모든 태스크의 최소 개선을 보장하는 방향을 찾는다. **Nash-MTL (Navon et al., ICML 2022)**은 이를 Nash 협상 게임으로 정식화하여 Pareto 최적 해를 유도하였다. 2024-2025년에는 **Aligned-MTL (Senushkin et al., CVPR 2023)**이 gradient 행렬의 SVD를 이용해 정렬된 업데이트 방향을 찾고, **FairGrad (Mahapatra & Rajan, 2024)**가 태스크 간 공정성까지 고려하는 방법을 제안하였다. 본 시스템의 adaTT는 이 중 그룹 구조를 명시적으로 활용하는 점이 차별적이며, intra/inter 강도를 별도로 설정하여 도메인 지식(예: CTR-CVR 관계)을 반영한다.

논문 vs 구현 비교

PLE (Tang et al., 2020) 비교

항목	원 논문	본 구현
Expert 구조	Shared + Task-specific MLP	8개 도메인 Shared Expert (GCN, PersLay, DeepFM, ..., RawScale)
Extraction Layer	다중 PLE Layer 스택	단일 레이어 (CGC → GroupTaskExpertBasket)
Task Expert	태스크별 독립 MLP	GroupEncoder + ClusterEmbedding (20 clusters)
Gate	Shared+Task Expert → gate	Shared Expert 블록 스케일링 (576D 유지)
Knowledge Transfer	암묵적 (Expert 공유)	명시적 Logit Transfer + adaTT gradient 기반
Cluster 특화	없음	GMM 20-cluster 임베딩 + GroupEncoder
HMM 라우팅	없음	Triple-Mode (journey/lifecycle/behavior)
Loss Weighting	고정	Uncertainty Weighting (Kendall et al.)
불확실성	없음	Evidential Deep Learning (Dirichlet)

MMoE (Ma et al., 2018) 비교

KDD 2018 Ma, J., et al. *"Modeling Task Relationships in Multi-Task Learning with Multi-Gate Mixture-of-Experts"*

항목	**MMoE**	본 구현
Expert 수	동일 구조 N개	이종 8개 (GCN, PersLay, DeepFM, ..., RawScale)
Expert 구조	동일 MLP	각각 도메인 특화 아키텍처
Gate	Linear(input → N) + Softmax	Linear(576 → 8) + Softmax (CGC)
Expert Collapse	심각 (모든 태스크가 동일 Expert)	완화 (Entropy 정규화 + domain_experts bias)
초기 편향	없음 (무작위)	domain_experts 기반 warm start
태스크 특화	gate만으로 분리	CGC + HMM routing + GroupTaskExpertBasket

주요 아키텍처 혁신

본 프로젝트만의 고유한 설계 요소:

1. 이종 **Expert** 결합: 단일 구조 Expert 대신 GCN, PersLay, DeepFM, RawScale 등 8개 이종 도메인 Expert를 결합
2. **CGC** 차원 정규화: Expert 출력 차원 비대칭(128D vs 64D) 보정
3. **HMM Triple-Mode** 라우팅: 태스크별 시간 스케일에 맞는 HMM 모드 선택적 주입
4. **GroupTaskExpertBasket**: GroupEncoder + ClusterEmbedding으로 88% 파라미터 감소 (v3.2)
5. **Logit Transfer** 체인: 위상 정렬 기반 자동 실행 순서 도출
6. **Evidential + SAE**: 예측 불확실성 정량화 + Expert 표현 해석성

디버깅 가이드

Expert 출력 진단

증상	원인	확인 방법
특정 Expert 출력 all-zero	입력 데이터 None → zero fallback	`shared_expert_outputs` dict에서 해당 Expert 텐서 확인
unified_hgcn 출력 NaN	Poincare 좌표 overflow	`hierarchy_features` 값 범위 확인, curvature 조정
temporal 출력 all-zero	`txn_seq` None	DataLoader의 시퀀스 로딩 활성화 확인
perslay 출력 불안정	Raw diagram 패딩 오류	`tda_short_mask` 유효 비율 확인

CGC Attention 분포 분석

증상	원인	해결
단일 Expert에 0.9+ 집중	Expert Collapse	`entropy_lambda` 증가 (0.01→0.02)
모든 Expert 균등 (0.125)	CGC 미학습 또는 과도한 entropy	`entropy_lambda` 감소, 학습률 확인
domain_experts 외 Expert에 높은 가중치	CGC가 domain 편향을 극복함	정상일 수 있음 — cross-domain transfer 패턴
학습 후반 attention 급변	CGC freeze 미적용	`freeze_epoch` 설정 확인

Loss 관련 문제

증상	원인	해결
Loss NaN/Inf 발생	fp16 underflow + focal loss	`.float().clamp(1e-7, 1-1e-7)` 확인 (M-2 FIX)
특정 태스크 loss 0	target 전체 −1 (결측)	`ignore_index=-1` 정상 동작, 데이터 확인
Uncertainty weight 발산	`task_log_vars` clamp 미적용	clamp(-4.0, 4.0) 및 precision clamp 확인
총 loss 급증	Evidential KL annealing 완료	annealing_epochs 이후 KL 기여 확인
adaTT 이후 loss 급변	Negative transfer 감지	`negative_transfer_threshold` 조정

Gradient Flow 진단

증상	원인	해결
Shared Expert gradient 0	Phase 2에서 freeze_shared 활성	`freeze_shared_in_phase2` 확인
CGC gradient 0 (학습 중)	CGC frozen (freeze_epoch 도달)	의도적 고정 — 정상
`_extract_task_gradients` OOM	retain_graph=True 누적	`adatt_grad_interval` 증가 (10→50)
Brand prediction gradient 약함	InfoNCE temperature 너무 높음	`temperature` 감소 (0.07→0.05)

HMM 관련 문제

증상	원인	해결
HMM 프로젝션 출력 동일	모든 샘플에 default embedding	HMM 파이프라인 데이터 생성 확인
특정 모드만 학습	다른 모드 대상 태스크가 적음	`target_tasks` 균형 확인
Default embedding이 학습 안 됨	대부분 샘플이 유효 HMM 보유	정상 (default는 소수에만 적용)

Logit Transfer 문제

증상	원인	해결
CVR이 CTR에 과의존	`transfer_strength` 너무 높음	0.5→0.3 감소
Transfer 미적용	Source 태스크 비활성화	Source가 `self.task_names`에 포함 확인
실행 순서 오류	위상 정렬 실패 → 폴백	로그에서 “하드코딩 폴백 사용” 경고 확인

부록

코드 파일 매핑

파일	역할
`models/ple_cluster_adatt.py`	PLEClusterAdaTT 메인 모델 (2125 라인)
`models/experts/registry.py`	ExpertRegistry, SharedExpertFactory
`models/experts/cluster_task_expert.py`	ClusterTaskExpertBasket, GroupTaskExpertBasket
`models/adatt.py`	AdaptiveTaskTransfer (gradient 기반 전이)
`models/layers/sae_layer.py`	SparseAutoencoder
`models/layers/evidential_layer.py`	EvidentialLayer (Dirichlet/Beta/NIG)
`models/tasks/task_registry.py`	TaskRegistry, TaskManager, TASK_GROUPS
`models/tasks/base_task.py`	BaseTask, TaskConfig, TaskOutput, TaskType
`models/tasks/classification_tasks.py`	CTR, CVR, Churn, Retention, NBA, ...
`models/tasks/regression_tasks.py`	Engagement, BalanceUtil, LTV, Uplift
`models/tasks/merchant_tasks.py`	BrandPrediction, MerchantAffinity, ContrastiveLoss
`configs/model_config.yaml`	전체 모델 설정 (진실의 원천)

파라미터 카운트 추정

모듈	파라미터	비고
Unified H-GCN	200K	128D output, merchant 계층 구조
PersLay	50K	Raw diagram + global stats
DeepFM	169K	v3.11: 필드별 독립 임베딩
Temporal Ensemble	500K	Mamba + LNN + Transformer
LightGCN	20K	사전 계산 임베딩 → 경량
Causal	100K	NOTEARS DAG + 인과 인코더
Optimal Transport	100K	Sinkhorn + 기준 분포
RawScaleExpert	12K	90D→64D LayerNorm+MLP (v3.3)
CGC (16 태스크)	75K	16 × Linear(576→8)

HMM 프로젝터	5K	3 × Linear(16⟶32)
GroupTaskExpertBasket	362K	4 GroupEncoder × 20 clusters
Task Towers (16)	80K	16 × MLP(32⟶64⟶32⟶out)
adaTT	10K	전이 행렬 + affinity
SAE	2.7M	576D × 4 expansion (분석 전용)
Evidential	30K	16 × Linear(32⟶out)
Auxiliary 프로젝터	40K	coldstart + anonymous + gate
총계 **(SAE 제외)**	**1.7M**	학습 대상 파라미터
총계 **(SAE 포함)**	**4.4M**	SAE는 detach (분석 전용)

파라미터 카운트 확인 방법

`model.summary()` 메서드 (라인 1967 2073)가 모듈별 파라미터 수를 출력한다. 위 수치는 추정값이며, 실제 값은 config 설정에 따라 달라질 수 있다. 정확한 수치는 모델 초기화 후 `summary()` 출력으로 확인한다.

학습 설정 요약

항목	값
Optimizer	AdamW (lr=0.0005, weight_decay=0.01)
Scheduler	CosineAnnealingWarmRestarts (T0=10, Tmult=2)
Batch Size	16384
Max Epochs	100
Early Stopping	patience=7
Gradient Clipping	5.0
Mixed Precision	fp16 (AMP)
Phase 1	Shared Expert 학습 (15 epochs)
Phase 2	Cluster Subhead Fine-tuning (8 epochs, shared frozen)
adaTT Warmup	0 epoch (프로덕션: 10)
adaTT Freeze	1 epoch (프로덕션: 28)
CGC Freeze	adaTT freeze_epoch과 동기화

학부 수학 – **AdamW** 옵티마이저의 수학적 구조

AdamW는 **Loshchilov & Hutter (ICLR 2019)**이 제안한 옵티마이저로, Adam에 분리된 가중치 감쇠**(decoupled weight decay)**를 적용한 것이다. 기본 Adam의 파라미터 업데이트 규칙은: $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ (1차 모멘트 = gradient 이동평균) $\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$ (2차 모멘트 = gradient 제곱 이동평균) $\widehat{\boldsymbol{m}}_t = \boldsymbol{m}_t/(1-\beta_1^t)$, $\hat{\boldsymbol{v}}_t = \boldsymbol{v}_t/(1-\beta_2^t)$ (편향 보정) $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \widehat{\boldsymbol{m}}_t/\left(\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon\right)$ 여기서 η는 학습률, $\beta_1 = 0.9$, $\beta_2 = 0.999$가 일반적이다. 직관: $\widehat{\boldsymbol{m}}_t$는 "어느 방향으로 가야 하는가"(1차 모멘트 = 관성), $\sqrt{\hat{\boldsymbol{v}}_t}$는 "이 방향의 gradient가 얼마나 변동하는가"(2차 모멘트 = 적응적 학습률). 변동이 큰 파라미터는 학습률을 자동으로 줄여 안정화한다. AdamW의 핵심 차이는 가중치 감쇠를 gradient가 아닌 파라미터 자체에 직접 적용하여 $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}(1-\eta\lambda) - \eta \cdot \widehat{\boldsymbol{m}}_t/\left(\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon\right)$ ($\lambda = 0.01$ = `weight_decay`)로 L2 정규화를 올바르게 구현하는 것이다.

역사적 배경 – **Cosine Annealing** 학습률 스케줄러

Cosine Annealing은 **Loshchilov & Hutter (ICLR 2017)**이 SGDR(Warm Restarts) 논문에서 제안하였다. 학습률을 코사인 함수로 감소시킨다: $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\pi t/T_0))$ 학습률이 주기적으로 최댓값으로 복원(warm restart)되어, 국소 최솟값(local minimum)에서 탈출할 기회를 반복적으로 제공한다. $T_0 = 10$은 첫 주기 길이, $T_{\text{mult}} = 2$는 주기가 매번 2배씩 늘어남을 의미한다 (10→20→40 에포크). 이 방식은 StepLR(계단식 감소)보다 부드러운 전이를 제공하고, Exponential Decay보다 warm restart 덕분에 더 넓은 손실 경관을 탐색한다. 2020년 이후 대부분의 대규모 모델 학습에서 cosine 스케줄러가 표준으로 자리잡았으며, GPT-3, PaLM 등 LLM 학습에서도 warm-up + cosine decay 조합이 사용된다.

Config 핵심 경로

모델의 진실의 원천(Single Source of Truth)은 `configs/model_config.yaml` 이다. 주요 섹션과 모델 메서드의 매핑:

Config 섹션	설명	읽는 메서드
`global`	클러스터 수, dropout, input_dim	`__init__`
`shared_experts`	8개 Expert 설정	`_build_shared_experts`
`cgc`	CGC 활성화, bias, entropy	`_build_cgc`
`hmm_triple_mode`	3모드 라우팅, 대상 태스크	`_build_hmm_projectors`
`task_experts.common`	GroupEncoder 설정	`_build_task_experts`
`task_experts.tasks`	16+2 태스크 개별 설정	`_build_task_experts`, `_compute_task_losses`
`adatt`	전이 강도, 태스크 그룹	`_build_adatt`
`task_relationships`	Logit Transfer 쌍	`_build_logit_transfer`
`task_towers`	Tower 구조, activation	`_build_task_towers`
`sae`	SAE 활성화, expansion	`_build_sae`
`evidential`	Evidential 활성화, KL	`_build_evidential_layers`
`training`	lr, batch, epochs, loss weighting	`__init__` (task_log_vars)