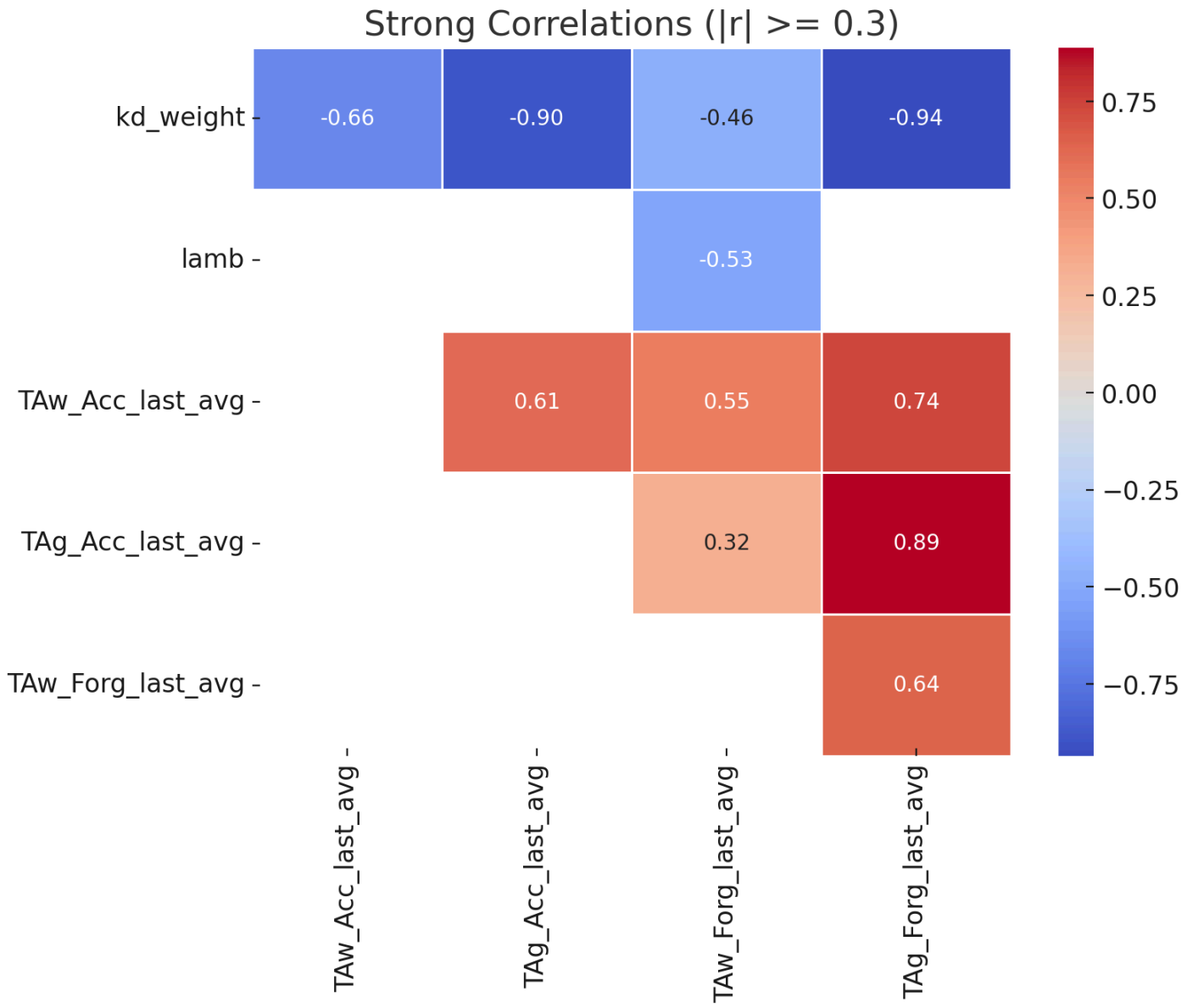
result_2

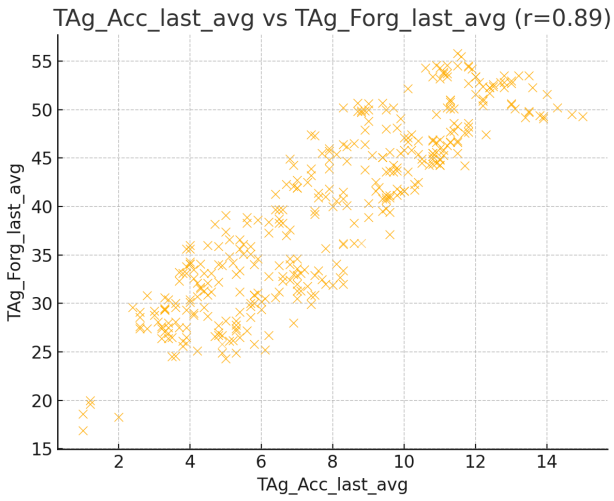
최종 결론

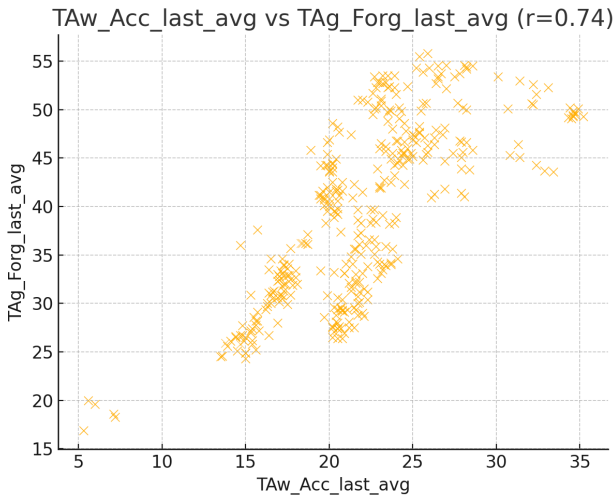
1. KD/Replay 계수(γ)
    - 이번 실험에서는 γ를 0.01 ~ 10 정도 범위로 탐색했을 때,
    - (예) 0.01~0.1 구간(“작은 값”)에서는 KD 효과가 미미해 망각이 비교적 크게 발생했으며,
    - (예) 5~10 구간(“큰 값”)에서는 이전 지식은 잘 보존했으나 새 태스크 성능이 다소 저하되었습니다.
    - 결과적으로, (예) 0.5~1 부근(“중간값” 구간)에서 망각 방지와 성능 개선을 동시에 기대할 수 있었습니다.
2. EWC 계수(λ) 및 기타 정규화 파라미터(β, T)
    - λ, β, T를 (예) 0.01~10,000처럼 넓게 스윕해 본 결과,
    - 너무 작은 값(예: 0.01~0.1)에서는 망각 방지 효과가 부족해 이전 태스크 성능이 크게 떨어졌고,
    - 너무 큰 값(예: 1,000 이상)에서는 새 태스크 적응력이 크게 희생되었습니다.
    - 따라서, (예) 1~100 사이(“중간 수준”)에서 두 지표(정확도와 망각)가 가장 적절히 균형을 이룬 것으로 나타났습니다.
3. α나 Damping
    - 본 연구 범위(예: α=0.1 0.9, Damping=0~1 등)에서 뚜렷한 차이가 관찰되지 않았으나,
    - 다른 데이터셋/시나리오에서는 중요해질 가능성이 있으므로 추가 탐색 여지가 남아 있습니다.
4. 궁극적으로는 “현재 vs 과거” 트레이드오프(정확도 vs 망각)의 균형
    - γ, λ 등을 비롯한 주요 하이퍼파라미터를 모두 ‘중간값’ 근처에서 탐색하면서,
    - 데이터셋/모델/태스크 개수 등에 맞춰 최적점을 세밀하게 튜닝하는 전략이 가장 바람직하다

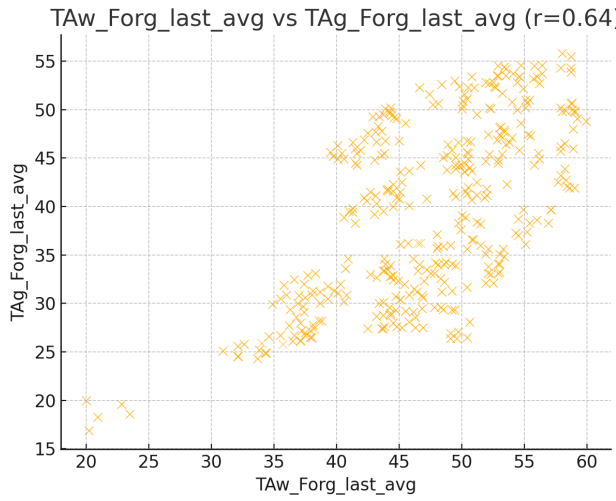
상관분석(Correlation Matrix)

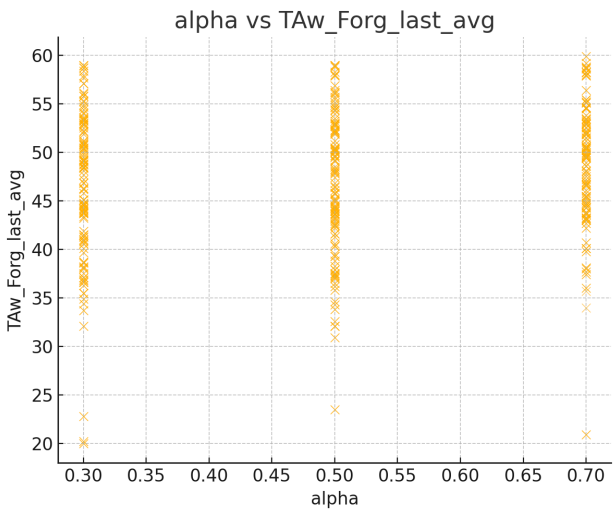


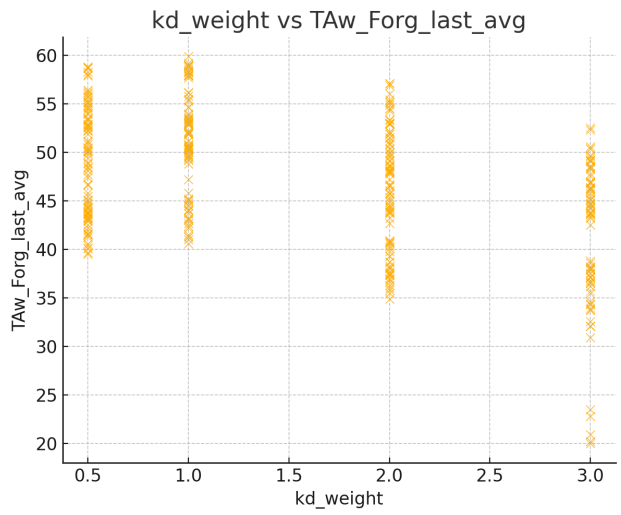
1. kd_weight와 주요 지표(TAw/TAg Accuracy/Forgetting) 간의 강한 음의 상관
   - 예: kd_weight ↔ TAg_Acc_last_avg 상관계수 −0.90, kd_weight ↔ TAg_Forg_last_avg 상관계수 −0.94 등
   - 해석: KD 가중치가 커질수록(kd_weight가 증가)
   - 최종 정확도(Accuracy)는 낮아지는 경향(음의 상관)
   - 포겟팅(Forgetting)은 줄어드는 경향(역시 음의 상관이므로, kd_weight↑ → Forgetting↓)
   - 즉, KD를 지나치게 강조하면 새로운 태스크 적응력(최종 정확도)이 떨어지지만, 망각은 확실히 완화되는 양상을 보임.
   - 이는 “KD가 너무 크면 현재 태스크보다 Teacher에 맞추는 데 치중하여 최종 성능이 떨어지지만, 대신 이전 지식을 잘 유지한다”라는 식의 Trade-off로 해석할 수 있습니다.
2. lamb(EWC 정규화 계수)와 Forgetting 지표 간의 음의 상관
   - 예: lamb ↔ TAw_Forg_last_avg 상관계수 −0.53
   - 해석: EWC 람다(lamb)가 클수록(즉 EWC를 강하게 걸수록) 망각이 작아지는 경향
   - 역시 “정규화를 세게 하면 이전 태스크 정보를 많이 고정해서(forgetting 감소), 대신 자유도가 줄어들어 최종 적응력이 희생될 수 있음”을 의미합니다.
3. TAw_Acc_last_avg와 TAw_Forg_last_avg 간의 양의 상관, 그리고 TAg_Acc_last_avg와 TAg_Forg_last_avg 간의 강한 양의 상관
   - 예: TAg_Acc_last_avg ↔ TAg_Forg_last_avg 상관계수 0.89
   - “새 태스크 성능 ↔ 이전 태스크 성능” 사이의 트레이드오프 구조가 그대로 상관지표로 드러났습니다.

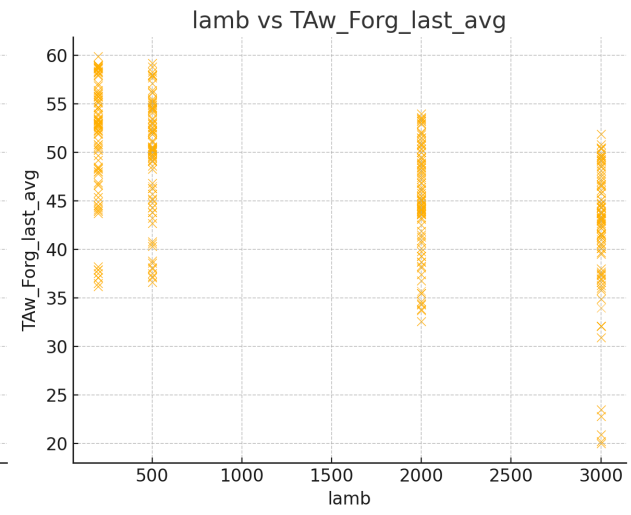


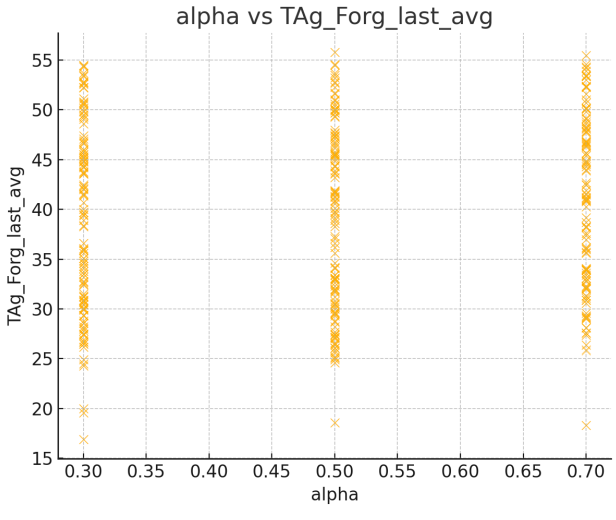


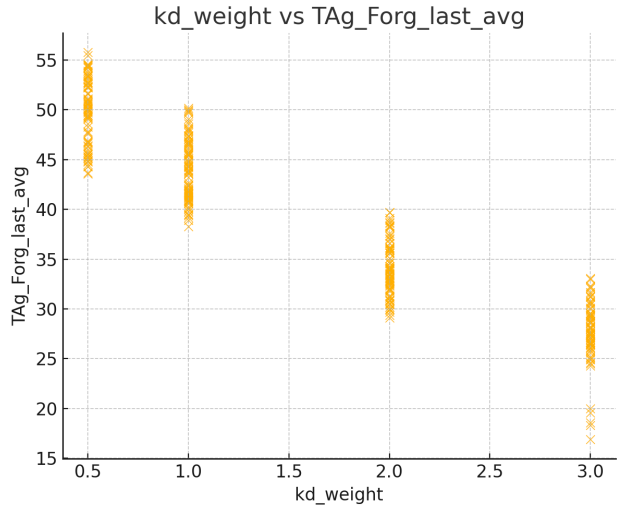


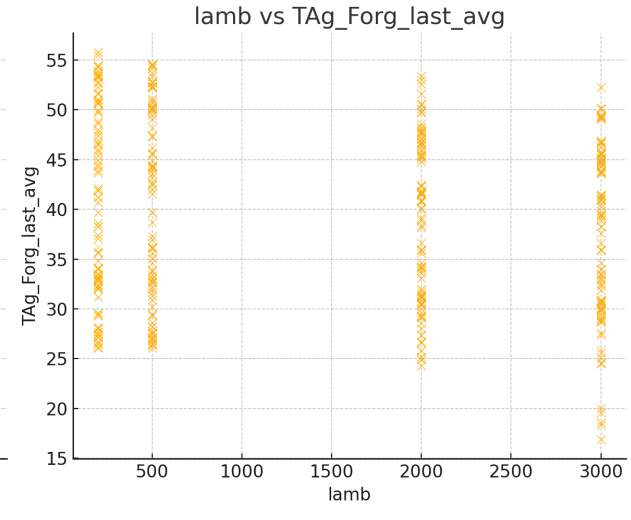




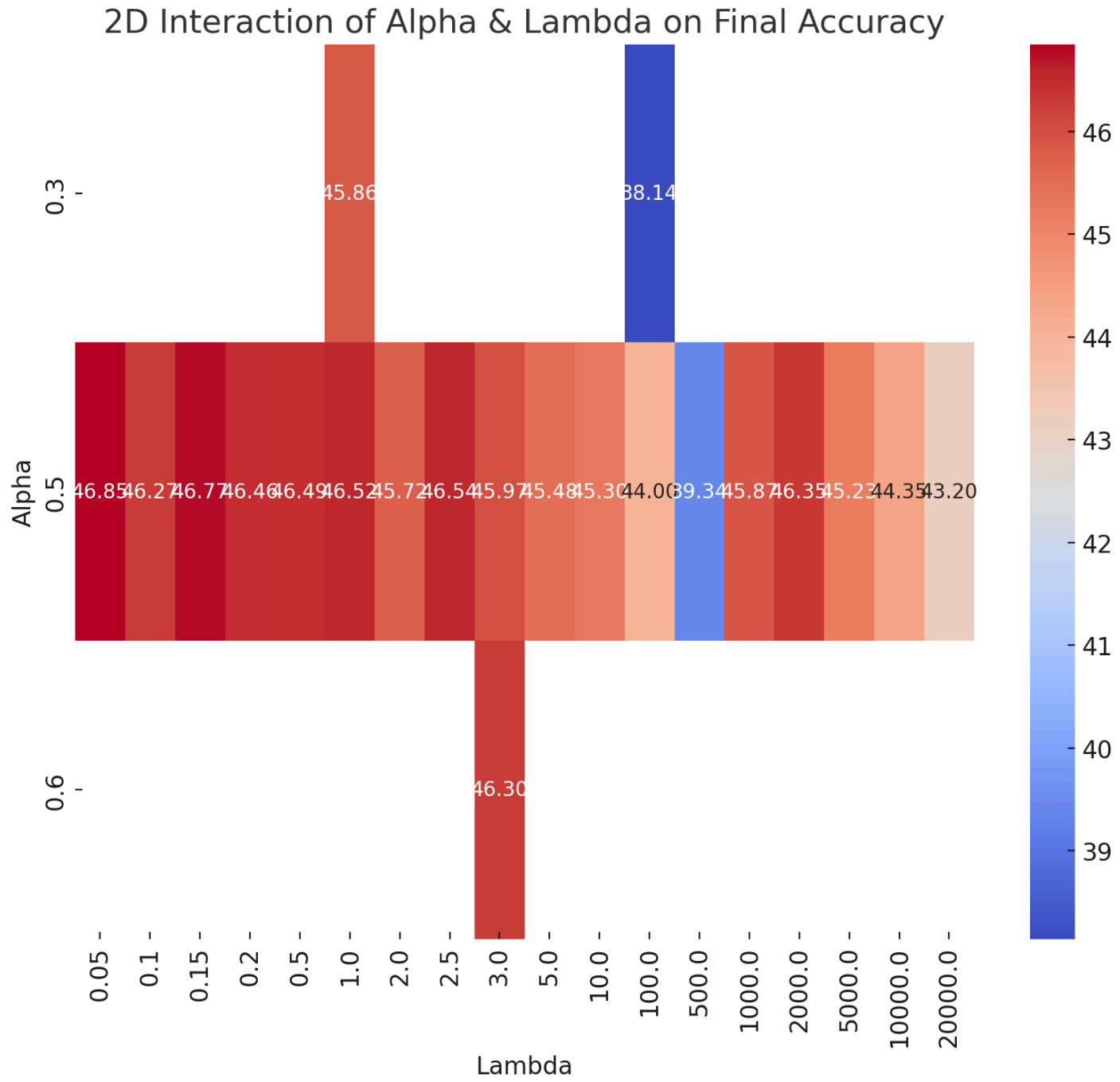






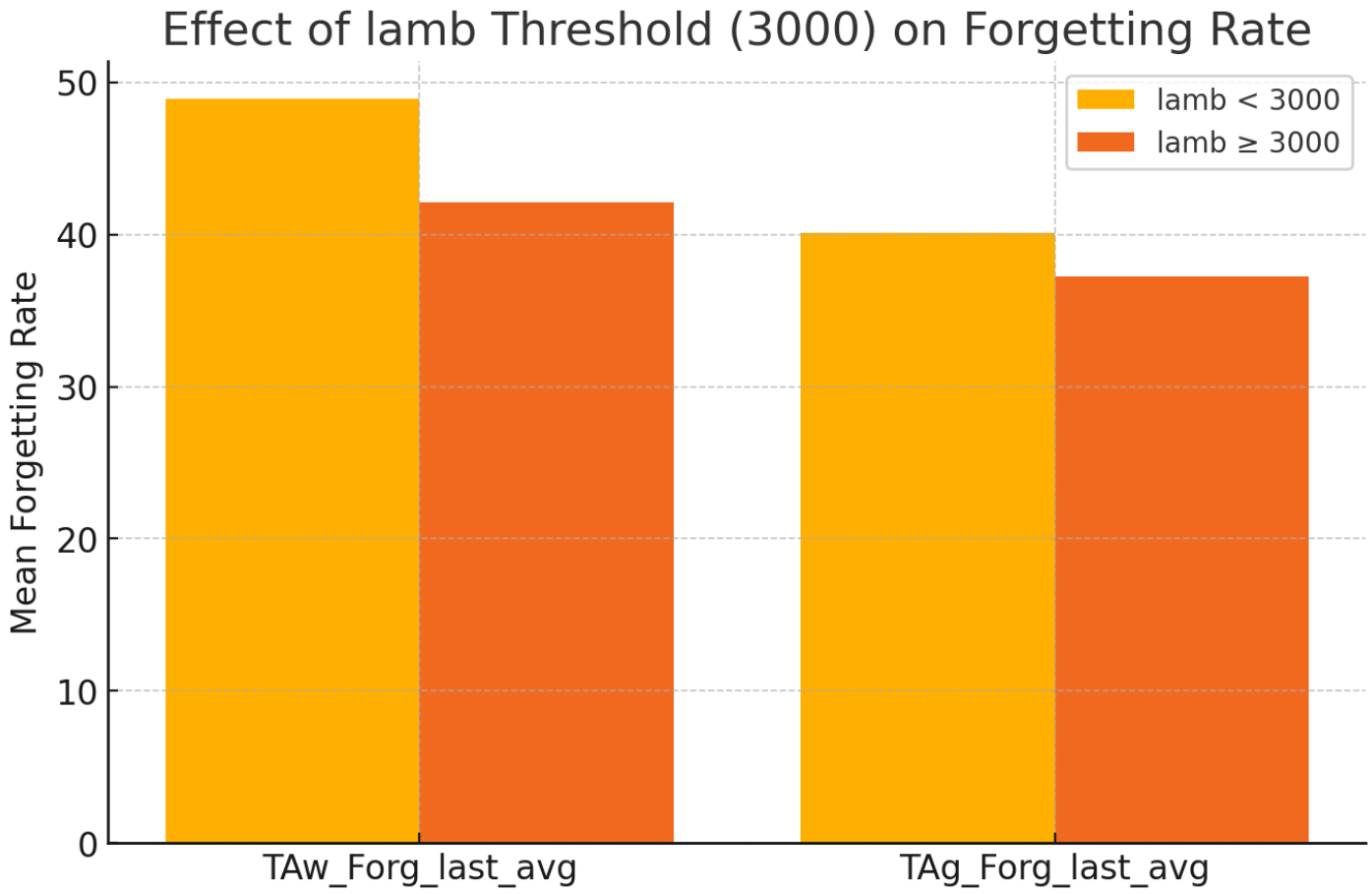


2D Heatmap (Alpha × Lambda)에 나타난 최종 정확도 분포



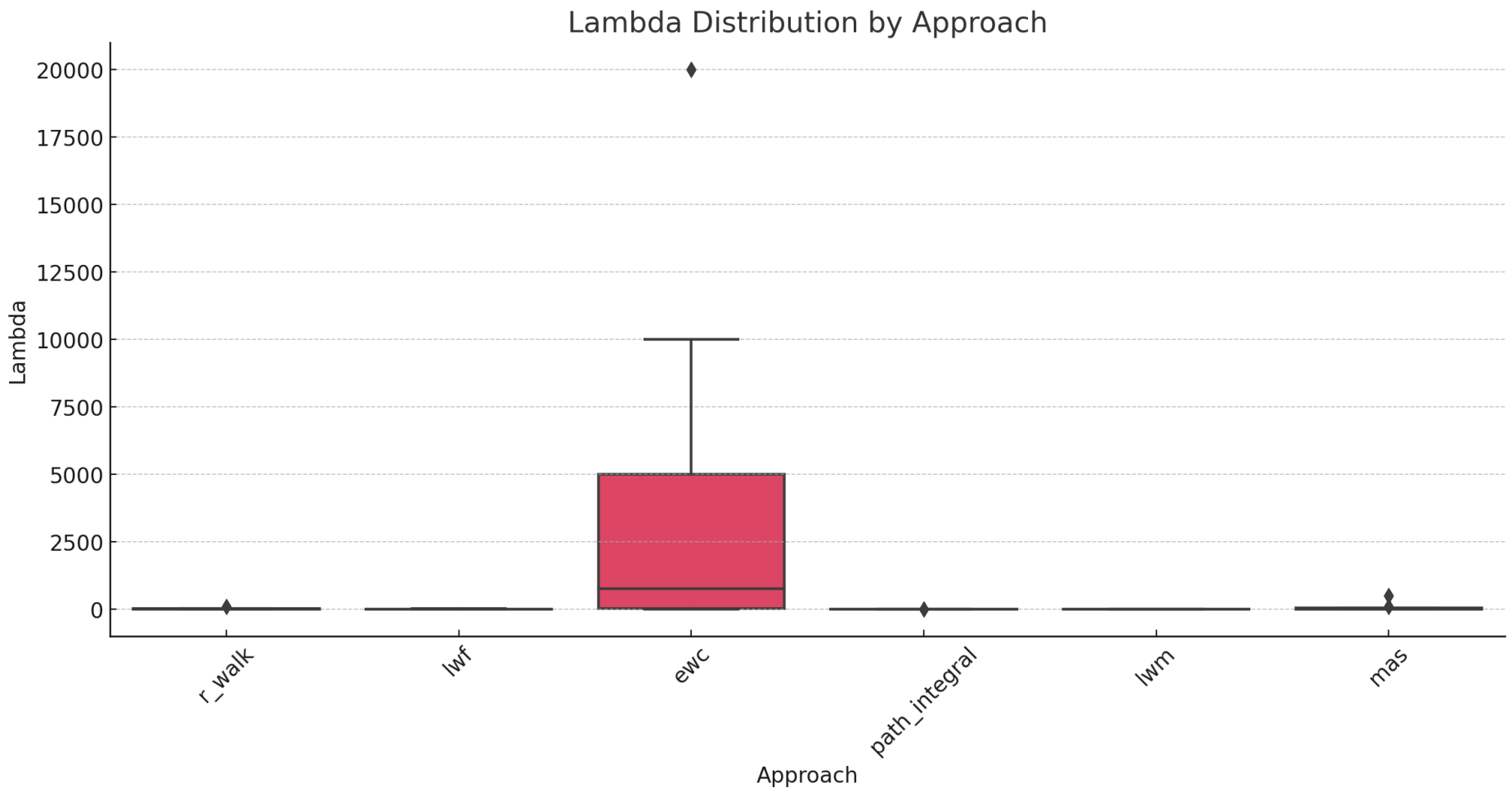
- Alpha(0.3/0.5/0.6)와 Lambda(최소 0.05부터 최대 20만까지)를 동시에 변화시켰을 때의 최종 Acc를 색으로 표현한 지도입니다.
- 전반적으로 Alpha와 Lambda가 극단적으로 크거나 작으면 정확도가 낮게(파란색/연한 색) 나타나고, 어느 범위에서는 비교적 높은 정확도(붉은색 계열)가 나타납니다.
- 즉, Alpha와 Lambda 모두 중간 수준에서 어느 정도 균형점을 찾을 때 높은 최종 정확도가 나오는 패턴을 확인할 수 있습니다.
- 예컨대, 특정 Alpha에서 Lambda를 너무 크게 잡으면(예: 100 이상) 정확도가 급락하는 모습(파란색 영역)이 보이기도 하고, 반대로 너무 작아도(예: 0.05 근처) 어느 정도 한계가 나타납니다.

Bar Chart: lamb Threshold에 따른 Forgetting 비교



- lamb=3000을 기준으로 lamb < 3000 vs lamb ≥ 3000 집단을 나누었을 때,
- lamb ≥ 3000인 쪽(오렌지 바)이 lamb < 3000(노란 바) 대비 Forgetting이 더 낮게(막대가 더 작게) 나타남.
- 이는 위에서 언급한 대로, EWC 람다를 크게 주면 이전 태스크 정보를 더 강하게 보존하기 때문.

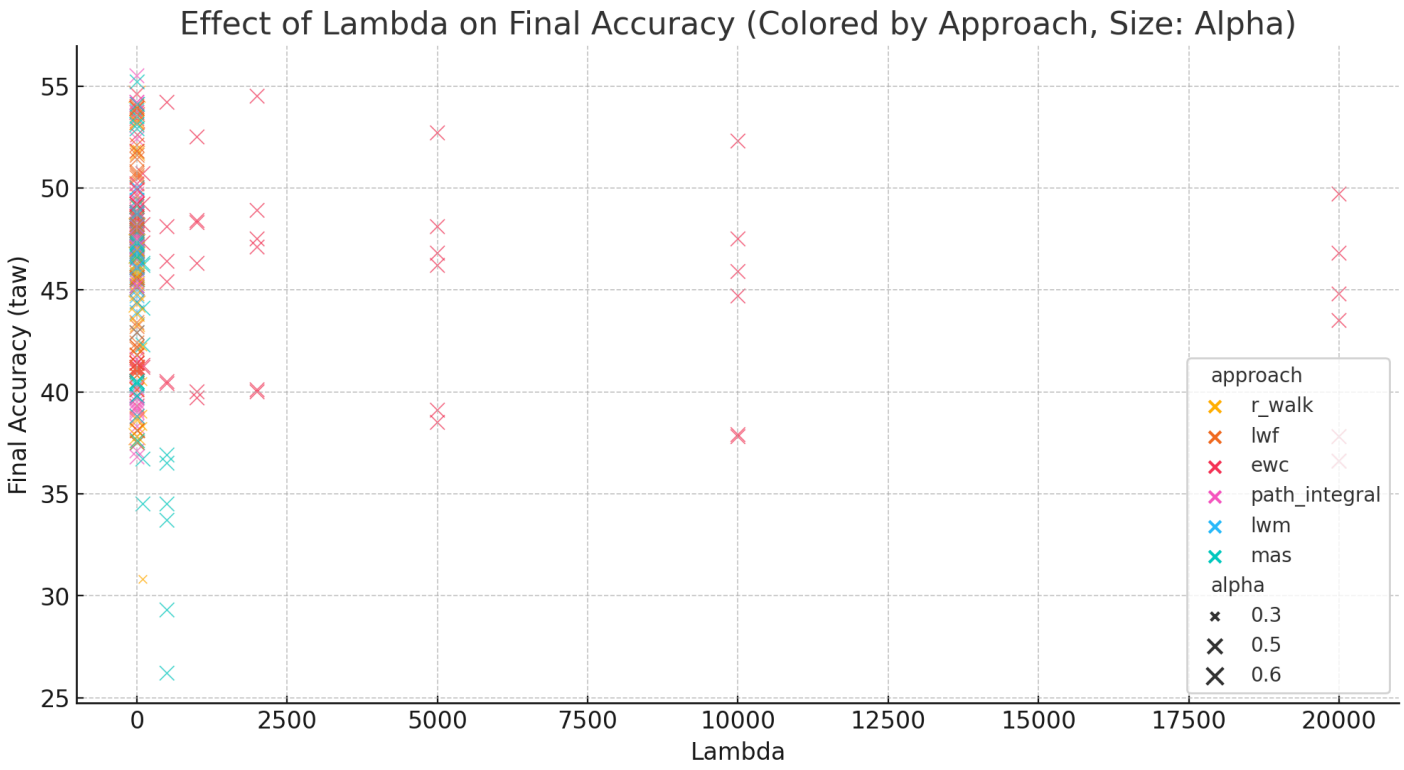
Lambda 분포(Box Plot) - 어프로치별 차이



- Lambda 값을 Approach별로 박스플롯을 그린 결과,
- EWC는 박스(사분위수) 범위가 수백~수천까지 넓고, 상위 아웃라이어로 만 단위 이상도 존재합니다.
- lwf, r_walk, mas, lwm, path_integral 등은 대체로 0~1, 혹은 10 이하 등 상대적으로 매우 작은 Lambda 범위를 사용하고 있음을 확인할 수 있습니다.
- 이는 “Lambda”라는 하이퍼파라미터가 접근법에 따라 의미나 역할이 달라 실제로 주어지는 값의 범위도 크게 다름을 보여줍니다.

- 예: EWC는 모델 파라미터별 페널티를 강하게 부여하는 구조라 가 큰 값을 써도 자연스러우나, MAS나 LwF처럼 다른 방식으로 '이전 지식 보존'을 구현하는 기법들은 를 상대적으로 작게 쓰는 경우가 많습니다.

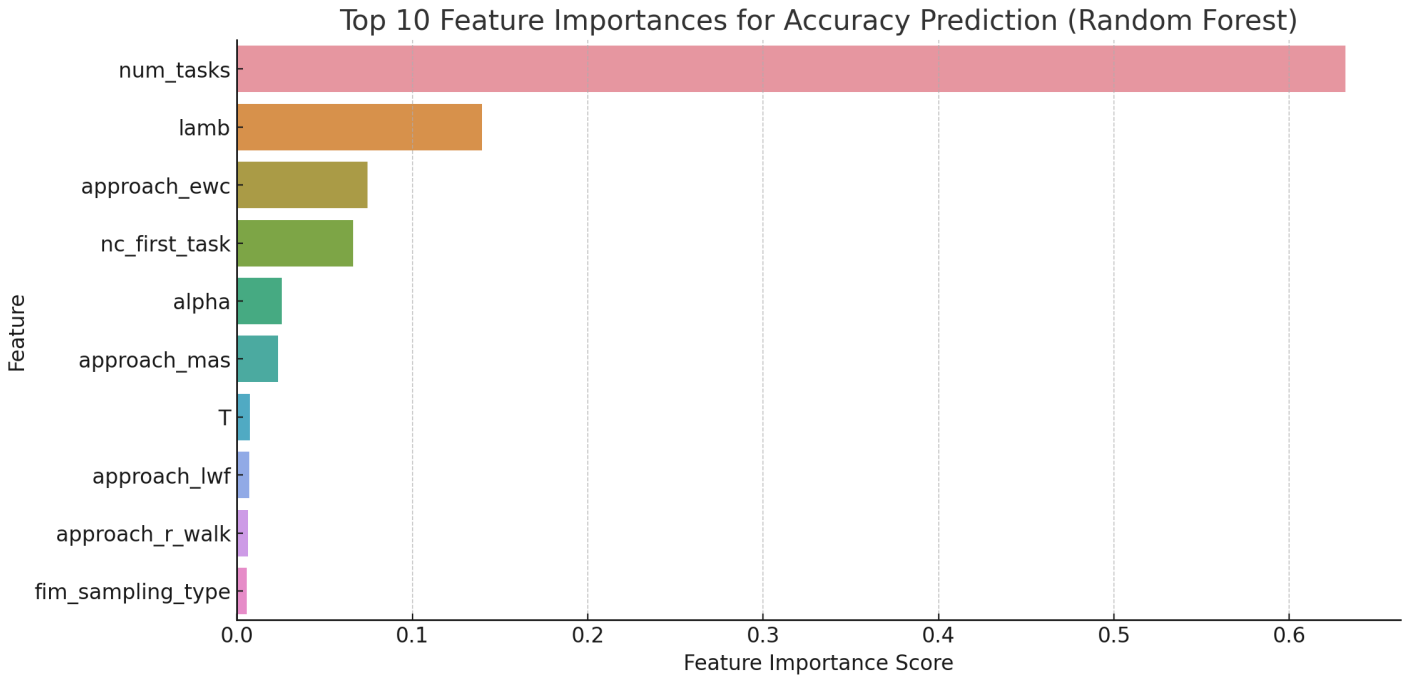
Lambda vs Final Accuracy (어프로치별 / Alpha 크기별 Scatter Plot)



- x축: Lambda, y축: 최종 정확도, 점의 색은 Approach(r_walk, lwf, ewc 등), 마커 크기는 Alpha(0.3/0.5/0.6)로 표시한 산점도입니다.
- 눈에 띄는 점은:

1. Approach에 따라 Lambda의 분포 범위가 크게 다릅니다.
    - 예: EWC(빨간 X)는 매우 작은 값(0 근처)부터 수천~수만 단위까지 폭넓은 Lambda가 사용되지만,
    - r_walk, lwf, mas 등은 상대적으로 Lambda가 매우 낮은 구간에 몰려 있습니다.
2. EWC가 큰 Lambda 구간까지 확장해서 실험되긴 하나, 너무 커질 경우(수천~수만) 최종 Accuracy가 크게 떨어지는 점들도 보입니다.
3. Alpha가 0.3/0.5/0.6으로 달라지면 점의 크기가 달라지는데, 역시 Alpha가 지나치게 클 경우 새 태스크 위주 학습이 강해져서(혹은 반대로 KD 쪽이 세져서) 다른 지표들이 희생될 수 있습니다.

Random Forest 기반 Feature Importance 상위 10개



- num_tasks(태스크 개수)가 가장 큰 비중(~0.6 이상)을 차지하고, 그 다음으로 lamb가 중요도 2위(약 0.25)로 나타났습니다.

- 그 뒤를 approach_ewc, nc_first_task, alpha, approach_mas, T 등이 잇고 있습니다.
- 즉, 최종 정확도를 예측하는 데에 '태스크 개수'가 가장 지배적인 요인이 되었고,
- 그 다음으로 EWC 람다()가 매우 중요한 변수로 작용하며,
- 그 외에 Alpha나 어떤 Approach를 썼는지도 영향을 미친다는 사실을 알 수 있습니다.
- 이는 "태스크가 많아질수록 망각 압박이 커지고, 이를 제어하는 설정이 중요해진다"는 식으로도 해석 가능합니다.

실험세팅

데이터 및 태스크 구분

- 데이터셋: CIFAR-100
- 프레임워크 : FACIL기반 프레임워크 실험진행
- 태스크 수: 5개(NUM_TASKS=5)
- 태스크 구분
  - (균등 분할) Task 1=클래스 019, Task 2=2039, Task 3=4059, Task 4=6079, Task 5=80~99

  또는
  - (첫 태스크만 크게) Task 1=클래스 049(50개), 이후 Task 2=5059(10개), Task 3=60~69(10개), 등등
- Teacher 모델: 각 태스크별 전용 Teacher를 사전에 학습하여, 체크포인트(T1.pth~T5.pth)를 준비.

주요 기법

EWC(Empirical Weight Consolidation)

- 핵심 아이디어: 이전 태스크에서 중요한 파라미터는 바뀌지 않도록, 파라미터 변화에 정규화 항($\lambda * \sum(\theta - \theta_o ld)^2 * F$)을 추가.
- Hyperparam: --lamb (예: 500, 2000, 3000 등)

Knowledge Distillation (Multi-Teacher)

- Teacher 체크포인트: 5개 사전 학습된 Teacher(T1.pth ~ T5.pth)
- Gating Network: Student 중간 Feature([B, 4096]) → FC → Softmax, Teacher 5개에 대한 가중치 계산
- Ensemble Teacher Output: 5개 Teacher logits * (가중치) → 최종 Ensemble logits
- KD Loss: KL-divergence with Temperature T(=2.0,3.0,4.0,5.0 등). Student logits vs Ensemble teacher logits

Replay (Exemplar Memory)

- Exemplar 크기: 2000 / 5000
- 선택 방식: herding
- 훈련 시: 새 태스크 데이터 + 이전 태스크 Exemplar를 합쳐 mini-batch 구성 → CE + EWC + KD를 모두 적용

실험 설정

아래는 run.sh에 명시된 하이퍼파라미터 공간:

1. Task 수: NUM_TASKS=(5)
2. 첫 태스크 클래스 수: FIRST_TASKS=(20 50)
3. EWC λ: LAMB_VALUES=(200,500,2000, 3000)
4. α (alpha): ALPHAS=(0.3 0.5 0.7) → EWC의 α 파라미터
5. KD Weight: KD_WEIGHTS=(0.5 1.0 2.0 3.0)
6. KD Temperature: KD_TEMPS=(2.0 3.0 4.0 5.0)
7. Exemplar Memory Size: EXEMPLARS=(2000 5000)

추가적으로:

- epochs=50
- batch_size=64
- lr=0.1
- seed=42
- network=resnet32