Graph-less MLP

Блок 1: разработка

1. Metrics
   a. Accuracy
      Есть дисбаланс в классах, но не очень сильный, поэтому в целом accuracy использовать можно, реализованы ещё многоклассовые precision и recall (с помощью макроусреднения), чтобы ориентироваться и на них, но полученные выводы не отличаются в зависимости от метрик, поэтому две дополнительные служат лишь подтверждением.
   b. Выбор лучшей модели по статистикам на валидации. Результаты нескольких экспериментов
   c. Необходимые зависимости от параметра seed
      Добавлена функция set_seed, плюс зависимости от сида при сохранении.
   d. Сравнение моделей, выводы

Модель\метрика (test)	accuracy	precision	recall
teacher	0.777 +- 0.031	0.767 +- 0.063	0.739 +- 0.034
mlp	0.674 +- 0.034	0.809 +- 0.02	0.569 +- 0.038
student_mlp	0.673 +- 0.033	0.78 +- 0.074	0.566 +- 0.042

      GNN (teacher) показывает значительно лучшие результаты, чем две другие модели, MLP и GLNN (student_mlp) дают очень близкие результаты. При этом GNN ощутимо дольше учится, чем другие модели.
2. Data
   a. Какого размера датасет?

      В датасете содержится граф с 2708 вершинами.

   b. Сколько фичей у элементов датасета?

      У каждой из вершин есть вектор бинарных фичей длиной 1433 элементов. Также есть матрица смежности графа, то есть для каждой вершины есть список других вершин, к которым от этой вершины проведено направленное ребро, такой список может быть пустым.

      Таком образом, можно считать, что у вершины 1434 фичи: 1433 бинарных и 1 набор вершин.

3. Class Dataset
   a. Что возвращает graph_collate_fn?
      Пусть размер батча равен B, размерность пространства признаков (бинарных) равна D=1433.

      graph_collate_fn возвращает словарь с 4 ключами: labels, h, h_nn, nn_lenghts.

      - labels: одномерный тензор с метками классов объектов, размер: B
      - h: двумерный тензор с фичами объектов, содержащихся в батче, размер: B*D
      - h_nn: трёхмерный тензор с фичами объектов, размер: B*L*D

        Каждой вершине графа в батче соответствует двумерный тензор размера L*D, содержащий фичи других вершин, к которым проведены рёбра от этой вершины.
        Если для какого-то объекта таких вершин больше, чем nn_max_size (заданная константа), то берутся фичи только первых nn_max_size вершин, остальные отбрасываются. Если таких вершин нет, то делается одна “искусственная” с нулевым вектором признаков. Таким образом, количество векторов признаков для каждого отдельного объекта
            Li = max(1, min(число исходящих ребер, nn_max_size))
        Далее список фичей каждого объекта паддится нулями до максимальной длины и получается размерность L = max(Li).
      - nn_lenghts: одномерный тензор с количеством исходящих ребер для каждой вершины (с учётом добавленной “искусственной” вершины для вершин, не имеющих исходящих рёбер), размер: B.
   b. Реализация node_student_collate_fn
   c. Какую информацию о графе из исходных данных использует student_mlp модель?

Для каждой вершины используется только вектор фичей для этой вершины, связи с другими вершинами и какая-либо информация о них не используется.

4. Class MLP
   a. Сколько в сети обучаемых параметров?

      В случае с input_dim = 1433, output_dim = 7, num_layers=1, use_norm=False:

      1433 * 7 + 7 = 10038

      В общем случае:

      BN = 1 if use_norm else 0, ID = input_dim, HD = hidden_dim, OD = output_dim

      Для num_layers == 1

      (2*ID*BN) + (ID*OD + OD)

      Для num_layers > 1

      (2*ID*BN) + (ID*HD + HD) + (HD*HD + HD)*(num_layers - 1) + (HD*OD + OD)

   b. Стоит ли установить num_layers=2?

Модель\метрика	accuracy	precision	recall
mlp (2 layers)	0.254 +- 0.127	0.074 +- 0.05	0.215 +- 0.054

      Если кроме количества слоёв ничего менять не будем, то точно не стоит.

   c. Стоит ли установить dropout_p=0.5?

Модель\метрика	accuracy	precision	recall
mlp (dropout_p = 0.5)	0.698 +- 0.024	0.786 +- 0.024	0.607 +- 0.036

      Модель не так сильно переобучается под трейн, качество на валидации хуже не становится, модель получается более устойчивой, поэтому дропаут добавить стоит.

5. Categorical features

   Так как все фичи являются бинарными, такой подход кажется не очень удачным. Появилось несколько вариантов реализации такой идеи:

   1) Сделать один эмбеддинг слой, преобразующий все фичи, равные 0 и 1, в новые числа вне зависимости от того, какая это фича (первая, вторая, …, сотая, …)
   2) Сделать для каждой фичи свой отдельный эмбеддинг слой, аналогичный первому пункту, то есть теперь 0 и 1 для разных признаков трансформируются в разные значения.
   3) Сделать эмбеддинг для всей первой половины вектора фичей одним из способов: сделать вместо первой половины признаков одну категориальную фичу из 2^((1433+1) // 2) категорий, и сделать эмбеддинг для неё или сделать такую же фичу, но с N категориями, где N – число различных векторов длины (1433+1) // 2, встретившихся во всём датасете

   Примеры первых двух подходов реализованы в модуле embeddings.py и могут использоваться как первые слои в любой из рассмотренных архитектур (на вход в форварде получают (x['h'])). Количество обучаемых параметров и время обучения при таком подходе увеличиваются, размерность не снижается, качество не улучшается.

   Третий подход имеет очевидные недостатки: необходим большой размер эмбеддингов (в первом случае очень большой), из-за небольшого объёма выборки большинство категорий не встретится / встретится один раз, в случае второго подхода (кодировать только встретившиеся вектора) модель не сможет работать на новых данных, то есть такой подход возможен только при трансдуктивном обучении.

6. Bugs
   1. Не знаю, можно ли считать это багом, но я бы заменила строчки
      criterion = StudentLoss(1)
      и
      self.coeff = nn.Parameter(torch.FloatTensor(coeff), requires_grad=False)

на

criterion = StudentLoss(0.0)

и

self.coeff = nn.Parameter(torch.FloatTensor([coeff]), requires_grad=False)

чтобы можно было менять соотношения лоссов в зависимости от коэффициента при дальнейших экспериментах.

2. Добавила дропаут слой в forward MLP.

Блок 2: анализ литературы

1) Основные вклады статьи

Авторы статьи предлагают более быструю на инференсе альтернативу графовым нейронным сетям, которая представляет собой MLP, усиленный с помощью дистилляции знаний от классической GNN. Их выводы заключаются в том, что, обучая MLP с использованием софтмакс меток объектов, полученных от графовой сети, можно получить модель, которая не будет / будет слабо уступать графовым сетям по качеству (на многих датасетах) и при этом значительно быстрее работать на инференсе. Суть подхода состоит в том, чтобы обучать MLP только на фичах вершин, не используя информацию о ребрах графа и соседних вершинах, в то время как графовые сети эту информацию используют.

2) Отличия в коде

1. Отличается архитектура MLP: у авторов отсутствует relu после последнего слоя, вообще не используется нормировка, у нас отсутствует дропаут в форварде.
2. Авторы статьи стараются сбалансировать выборки по количеству объектов в классе, у нас происходит рандомный сплит.
3. У авторов статьи присутствует L2 регуляризация в оптимизаторе, у нас – нет, также отличаются некоторые другие гиперпараметры.
4. У нас лучшей моделью считается та, которая даёт минимальный лосс на трейне, авторы статьи проверяют качество модели на валидационной выборке, измеряя метрику качества.

Если исправить эти пункты (особенно 3-4), это должно улучшить качество.

3) Идеи

В первую очередь бы сделала анализ и более тщательную обработку фичей с учетом их смысла, например, построение различных статистик.
Далее имеет смысл провести эксперименты с более тщательным подбором коэффициента в лоссе (конкретно для этого датасета получались лучшие результаты, чем в статье, при coef=0.3 и добавлении регуляризации), таже далее стоит попробовать другие архитектуры в качестве student-модели.
Также я бы рассмотрела подход с использованием ответов teacher-модели не для сравнения с ответами student-модели, а для её инициализации, описанный в Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons (https://arxiv.org/pdf/1811.03233.pdf) и попробовала модифицировать его для данного случая.

4) Использование скрытых состояний сети вместо softlabels

Это хорошая идея, так как скрытые состояния содержат намного больше информации (как минимум за счёт размерности), чем просто вероятности принадлежности классу.
Можно попробовать сделать MLP с num_layers > 2 и размером скрытого состояния, соответствующим размеру скрытого состояния графовой сети. Далее вычислять лосс как взвешенную сумму

a*loss_1(log_preds_mlp, target) + (1 – a)*loss_2(last_hidden_mlp, graph_hidden)

при этом важно брать a > 0, чтобы последний слой mlp обучался, ориентируясь на ошибку student_mlp, остальные слои будут обучаться, ориентируясь на сумму ошибок. В качестве loss_2 можно взять какой-нибудь лосс для регрессии, например, MSE.