МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Отчет по тестовому заданию «Сравнительный анализ реализации и оптимизации нейронной сети в PyTorch и TensorFlow»

Отчет выполнил
Броничев Александр Русланович

Москва
2025

Содержание

Введение

Постановка задачи

Провести анализ производительности простой модели с использованием различных инструментов. Исследуются показатели между фрэймворками PyTorch и TensorFlow. Так же изучается влияние графовых оптимизаций на скорость обучения. Обучается трехслойный перцептрон имеющий входной слой (по размеру признаков), скрытый слой (128 нейронов, ReLU) и выходной слой (1 нейрон). Для выполнения задачи нужно реализовать данную модель на обоих фрэймворках и исследовать скорость обучения, используемую память, точность обучения на некотором датасете.

Описание датасета

Для решения задачи используется датасет **California Housing Dataset**. Этот датасет содержит информацию о недвижимости в Калифорнии (цены на жильё и характеристики районов) и часто используется для задач регрессии (прогнозирования стоимости домов). Источником является библиотека sklearn (в реализации на TF и PT из sklearn.datasets берется fetch_california_housing). Размер используемого датасета состовляет 20 640 записей, а каждая запись имеет 8 признаков. Целевой переменной является средняя стоимость дома в сотнях тысяч долларов. Каждая строка - это один район Калифорнии. Рассмотрим признаки датасета:

- Средний доход жителей района (float, 0.5 - 15)
- Средний возраст домов района (float, 1 - 52)
- Среднее количество комнат в доме (float, 0.8 - 141)
- Среднее количество спален в доме (float, 0.3 - 34)
- Население района (float, 3 - 35 682)
- Среднее количество жильцов в доме (float, 0.7 - 1 243)
- Широта расположения района (float, 32.5 - 41.9)
- Долгота расположения района (float, -124.3 - -114.3)
- ЦЕЛЬ: Средняя стоимость дома (float, 0.15 - 5)

В датасете нет пропусков. Числовые признаки взяты небольшими, отчего удобны. Однако масштаб разный, поэтому была проведена нормализация.

Методы

Описание архитектуры

В задаче изучается обучение трехслойного перцептрона. **Перцептрон** - это простейший вид нейронных сетей, в основе которого лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов. Опишем принцип работы данной архитектуры:

1. Первыми в работу включаются элементы-сенсоры.
2. Далее сенсоры передают сигналы ассоциативным элементам
3. Если сумма сигналов на ассоциативный элемент привысила какой-то порог, то этот элемент передает сигнал реагирующему элементу
4. Реагируюшие элементы получают сигнал с некоторым весом и в зависимости от некоторого порога дают ответ

В выполняемой задаче рассматривается перцептрон, у которого:

- Количество сенсоров зависит от количества признаков датасета (в случае California Housing Dataset 8) (входной слой)
- 128 ассоциативных нейронов ReLU (скрытый слой)
- 1 реагирующий элемент (выходной слой)

Примененные оптимизации

В реализации на TensorFlow в качестве оптимизации используется декоратор @tf.function. С его помощью функция Python преобразуется в граф TensorFlow, что значительно ускоряет ее выполнение при работе с большими вычислениями, которые встречаются при обучении моделей. Рассмотрим основные параметры:

- *input_signature* принимает ожидаемые размеры и типы данных входных тензоров, позволяет избежать лишних ретрассировок при изменении формы входных данных
- *autograph* автоматически конвертирует условия и циклы в граф, если равен True, иначе использует только операции TensorFlow, теряя часть оптимизации
- *jit_compile* включает компиляцию XLA
- *reduce_retracing* уменьшает количество ретрассировок при изменении входных данных
- *experimental_relax_shapes* разрешает более гибкое поведение при изменении форм входных данных
- *experimental_follow_type_hints* автоматически приводит типы данных к указанным в аннотациях Python

В работе все параметры выставлены по умолчанию.

В реализации на PyTorch используется оптимизация torch.compile. Эта функция ускоряет выполнение моделей за счет компиляции графов вычислений. Она автоматически оптимизирует код, уменьшая накладные расходы интерпретатора Python. Рассмотрим режимы torch.compile, которые включаются через mode= после параметра модели:

- *mode="default"* баланс между скоростью компиляции и производительностью
- *mode="reduce-overhead"* уменьшает накладные расходы интерпретатора Python
- *mode="max-autotune"* максимально агрессивная оптимизация, включая автоподбор лучших ядер для операций
- *mode="no-compile"* отключает компиляцию, оставляя оригинальный PyTorch-код

В работе используется режим max-autotune

Результаты

Испытания были проведены на двух различных устройствах (на одном на cpu, на втором на gpu). Было взято различное количество эпох (10, 20 и 30). Размер батча составляет 64. На обучение взято 80% датасета, оставшиеся 20% используются для проверки точности обучения.

Таблицы сравнения производительности

Приведу таблицы времени и потерь для испытаний TensorFLow на cpu

Номер эпохи	Время (сек)	Потери MAE
1	11.69	0.5515
2	12.62	0.4595
3	11.98	0.4385
4	11.17	0.4262
5	11.33	0.4176
6	10.91	0.4102
7	11.06	0.4052
8	12.33	0.4002
9	9.57	0.3971
10	9.46	0.3934

Таблица 1: Таблица обучения TensorFlow без оптимизации, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	12.85	0.5556
2	9.72	0.4606
3	9.83	0.4365
4	10.48	0.4258
5	9.48	0.4187
6	10.40	0.4119
7	13.37	0.4087
8	12.26	0.4035
9	13.98	0.4000
10	14.25	0.3985
11	12.38	0.3953
12	10.77	0.3936
13	10.26	0.3906
14	10.63	0.3897
15	10.28	0.3882
16	11.14	0.3872
17	10.37	0.3858
18	9.73	0.3841
19	9.67	0.3826
20	8.99	0.3810

Таблица 2: Таблица обучения TensorFlow без оптимизации, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	9.60	0.5568
2	8.91	0.4637
3	8.69	0.4408
4	9.03	0.4294
5	9.45	0.4231
6	9.36	0.4163
7	9.57	0.4130
8	10.48	0.4090
9	10.46	0.4052
10	9.88	0.4020
11	11.88	0.4009
12	10.50	0.3979
13	11.05	0.3948
14	9.00	0.3927
15	9.69	0.3912
16	9.14	0.3900
17	9.49	0.3877
18	9.13	0.3872
19	9.70	0.3845
20	9.89	0.3847
21	9.58	0.3848
22	9.35	0.3830
23	10.06	0.3809
24	9.36	0.3800
25	8.97	0.3802
26	9.08	0.3788
27	10.21	0.3776
28	9.79	0.3764
29	8.75	0.3748
30	10.07	0.3746

Таблица 3: Таблица обучения TensorFlow без оптимизации, 30 эпох

Номер эпохи	Время	Потери MAE
1	3.47	0.7682
2	2.97	0.5069
3	2.91	0.4617
4	3.07	0.4481
5	3.01	0.4313
6	3.00	0.4334
7	3.33	0.4299
8	3.49	0.4245
9	2.91	0.4201
10	3.02	0.4163

Таблица 4: Таблица обучения TensorFlow с оптимизацией, 10 эпох

Номер эпохи	Время	Потери MAE
1	2.67	0.5511
2	3.36	0.4586
3	3.71	0.4435
4	4.05	0.4325
5	3.10	0.4250
6	3.67	0.4192
7	3.38	0.4139
8	3.00	0.4084
9	3.13	0.4057
10	3.08	0.4020
11	2.55	0.3979
12	2.69	0.3955
13	2.33	0.3930
14	2.74	0.3907
15	2.48	0.3888
16	2.93	0.3875
17	3.03	0.3858
18	2.42	0.3871
19	2.93	0.3857
20	2.70	0.3828

Таблица 5: Таблица обучения TensorFlow с оптимизацией, 20 эпох

Номер эпохи	Время	Потери MAE
1	2.97	0.5497
2	2.72	0.4584
3	2.36	0.4400
4	2.44	0.4271
5	2.31	0.4192
6	2.51	0.4122
7	2.33	0.4064
8	2.32	0.4018
9	2.34	0.3993
10	2.29	0.3954
11	2.41	0.3932
12	2.69	0.3919
13	2.40	0.3900
14	2.37	0.3876
15	2.35	0.3865
16	2.35	0.3856
17	2.33	0.3853
18	2.34	0.3842
19	2.32	0.3818
20	2.35	0.3817
21	2.33	0.3804
22	2.35	0.3795
23	2.34	0.3790
24	2.33	0.3780
25	2.32	0.3780
26	2.33	0.3771
27	2.33	0.3766
28	2.32	0.3754
29	2.33	0.3738
30	2.48	0.3750

Таблица 6: Таблица обучения TensorFlow с оптимизацией, 30 эпох

Приведу таблицы времени и потерь для испытаний PyTorch на cpu:

Номер эпохи	Время (сек)	Потери MAE
1	0.48	0.8085
2	0.43	0.5329
3	0.42	0.4800
4	0.43	0.4616
5	0.42	0.4514
6	0.43	0.4465
7	0.42	0.4403
8	0.43	0.4355
9	0.43	0.4306
10	0.42	0.4251

Таблица 7: Таблица обучения PyTorch без оптимизации, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	0.49	0.7494
2	0.46	0.5117
3	0.42	0.4668
4	0.46	0.4514
5	0.42	0.4414
6	0.42	0.4357
7	0.43	0.4293
8	0.43	0.4246
9	0.43	0.4221
10	0.46	0.4167
11	0.44	0.4138
12	0.44	0.4116
13	0.44	0.4065
14	0.44	0.4054
15	0.44	0.4013
16	0.45	0.4030
17	0.45	0.3976
18	0.45	0.3943
19	0.45	0.3926
20	0.45	0.3906

Таблица 8: Таблица обучения PyTorch без оптимизации, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	0.50	0.8783
2	0.49	0.5589
3	0.44	0.4899
4	0.67	0.4652
5	0.54	0.4529
6	0.59	0.4456
7	0.83	0.4406
8	0.55	0.4354
9	0.41	0.4304
10	0.41	0.4263
11	0.43	0.4226
12	0.43	0.4169
13	0.42	0.4150
14	0.43	0.4115
15	0.44	0.4092
16	0.49	0.4053
17	0.48	0.4038
18	0.44	0.4018
19	0.44	0.4062
20	0.44	0.3976
21	0.46	0.3957
22	0.45	0.3954
23	0.44	0.3930
24	0.44	0.3911
25	0.44	0.3904
26	0.44	0.3884
27	0.45	0.3900
28	0.44	0.3864
29	0.45	0.3893
30	0.45	0.3849

Таблица 9: Таблица обучения PyTorch без оптимизации, 30 эпох

Номер эпохи	Время (сек)	Потери MAE
1	0.66	0.7858
2	0.57	0.5302
3	0.51	0.4808
4	0.76	0.4634
5	0.65	0.4522
6	0.67	0.4443
7	0.53	0.4377
8	0.56	0.4322
9	0.59	0.4284
10	0.56	0.4237

Таблица 10: Таблица обучения PyTorch с оптимизацией, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	0.49	0.8521
2	0.54	0.5226
3	0.62	0.4764
4	0.58	0.4590
5	0.56	0.4500
6	0.84	0.4433
7	0.65	0.4406
8	0.55	0.4340
9	0.75	0.4305
10	0.63	0.4268
11	0.56	0.4230
12	0.55	0.4187
13	0.67	0.4159
14	0.56	0.4136
15	0.55	0.4093
16	0.54	0.4051
17	0.56	0.4030
18	0.56	0.4001
19	0.64	0.3984
20	0.57	0.3977

Таблица 11: Таблица обучения PyTorch с оптимизацией, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	0.51	0.8159
2	0.49	0.5332
3	0.50	0.4849
4	0.49	0.4627
5	0.62	0.4511
6	0.60	0.4443
7	0.50	0.4389
8	0.51	0.4339
9	0.50	0.4300
10	0.51	0.4270
11	0.52	0.4213
12	0.52	0.4195
13	0.52	0.4151
14	0.73	0.4121
15	0.52	0.4095
16	0.53	0.4045
17	0.53	0.4029
18	0.53	0.4023
19	0.55	0.4009
20	0.54	0.3988
21	0.55	0.3997
22	0.55	0.3936
23	0.54	0.3928
24	0.54	0.3898
25	0.55	0.3894
26	0.55	0.3907
27	0.66	0.3880
28	0.58	0.3874
29	0.58	0.3851
30	0.57	0.3838

Таблица 12: Таблица обучения PyTorch с оптимизацией, 30 эпох

Приведу таблицы времени и потерь для испытаний TensorFlow на gpu:

Номер эпохи	Время (сек)	Потери MAE
1	10.89	0.5655
2	10.75	0.4639
3	11.00	0.4411
4	10.81	0.4262
5	10.67	0.4200
6	10.62	0.4120
7	10.66	0.4074
8	10.84	0.4034
9	11.06	0.3991
10	11.13	0.3966

Таблица 13: Таблица обучения TensorFlow без оптимизации, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	12.02	0.5532
2	11.33	0.4601
3	10.70	0.4356
4	10.64	0.4222
5	10.82	0.4152
6	10.94	0.4091
7	10.62	0.4035
8	10.59	0.4012
9	10.63	0.3957
10	10.61	0.3945
11	12.16	0.3921
12	11.83	0.3908
13	11.38	0.3887
14	11.15	0.3861
15	11.28	0.3867
16	12.08	0.3848
17	12.07	0.3842
18	11.76	0.3836
19	10.89	0.3820
20	10.66	0.3825

Таблица 14: Таблица обучения TensorFlow без оптимизации, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	10.59	0.5563
2	10.70	0.4650
3	10.65	0.4434
4	10.64	0.4300
5	10.62	0.4211
6	10.64	0.4144
7	11.32	0.4097
8	11.66	0.4041
9	11.60	0.4008
10	11.69	0.4021
11	11.51	0.3982
12	11.61	0.3924
13	12.15	0.3903
14	11.15	0.3887
15	11.38	0.3868
16	11.49	0.3863
17	11.48	0.3851
18	11.35	0.3839
19	11.36	0.3831
20	10.89	0.3826
21	11.24	0.3805
22	11.99	0.3799
23	11.70	0.3800
24	11.84	0.3775
25	11.80	0.3783
26	11.99	0.3794
27	11.93	0.3773
28	12.25	0.3755
29	11.19	0.3757
30	11.50	0.3742

Таблица 15: Таблица обучения TensorFlow без оптимизации, 30 эпох

Номер эпохи	Время (сек)	Потери MAE
1	3.84	0.8204
2	2.63	0.5125
3	2.63	0.4660
4	2.62	0.4514
5	2.62	0.4406
6	2.64	0.4348
7	2.66	0.4299
8	2.62	0.4260
9	2.64	0.4211
10	2.62	0.4174

Таблица 16: Таблица обучения TensorFlow с оптимизацией, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	2.62	0.5626
2	2.64	0.4612
3	2.66	0.4430
4	2.65	0.4334
5	2.64	0.4264
6	2.69	0.4195
7	2.64	0.4151
8	2.65	0.4103
9	2.65	0.4058
10	2.65	0.4036
11	2.65	0.4012
12	2.64	0.3975
13	2.64	0.3949
14	2.67	0.3920
15	2.67	0.3898
16	2.67	0.3905
17	2.68	0.3889
18	2.66	0.3867
19	2.66	0.3861
20	2.67	0.3839

Таблица 17: Таблица обучения TensorFlow с оптимизацией, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	2.66	0.5584
2	2.64	0.4571
3	2.66	0.4359
4	2.63	0.4254
5	2.63	0.4185
6	2.63	0.4121
7	2.64	0.4093
8	2.63	0.4048
9	2.63	0.4012
10	2.63	0.3969
11	2.63	0.3950
12	2.63	0.3929
13	2.63	0.3913
14	2.63	0.3892
15	2.63	0.3880
16	2.63	0.3861
17	2.63	0.3855
18	2.63	0.3838
19	2.64	0.3844
20	2.63	0.3808
21	2.63	0.3807
22	2.63	0.3822
23	2.63	0.3796
24	2.63	0.3803
25	2.64	0.3804
26	2.65	0.3781
27	2.69	0.3768
28	2.64	0.3772
29	2.68	0.3760
30	2.67	0.3758

Таблица 18: Таблица обучения TensorFlow с оптимизацией, 30 эпох

Приведу таблицы времени и потерь для испытаний PyTorch на gpu:

Номер эпохи	Время (сек)	Потери MAE
1	2.01	0.8380
2	1.01	0.5341
3	1.00	0.4791
4	1.04	0.4590
5	0.98	0.4484
6	1.03	0.4417
7	1.03	0.4345
8	1.03	0.4296
9	1.03	0.4239
10	1.07	0.4206

Таблица 19: Таблица обучения PyTorch без оптимизации, 10 эпох

Номер эпохи	Время (сек)	Потери MAE
1	1.05	0.7924
2	0.99	0.5135
3	1.01	0.4724
4	0.99	0.4586
5	1.01	0.4486
6	1.06	0.4413
7	1.09	0.4384
8	1.08	0.4312
9	1.05	0.4271
10	1.10	0.4224
11	1.00	0.4168
12	1.08	0.4141
13	1.10	0.4114
14	1.08	0.4067
15	1.06	0.4027
16	1.08	0.4002
17	1.07	0.3992
18	1.02	0.3966
19	1.15	0.3923
20	1.09	0.3920

Таблица 20: Таблица обучения PyTorch без оптимизации, 20 эпох

Номер эпохи	Время (сек)	Потери MAE
1	1.09	0.7945
2	1.14	0.5169
3	1.09	0.4708
4	0.90	0.4556
5	0.96	0.4475
6	0.95	0.4400
7	0.92	0.4353
8	0.94	0.4304
9	0.92	0.4250
10	0.93	0.4232
11	0.94	0.4185
12	0.90	0.4151
13	0.94	0.4114
14	0.93	0.4080
15	0.91	0.4059
16	0.95	0.4027
17	0.91	0.4015
18	0.96	0.3997
19	0.92	0.3980
20	0.90	0.3948
21	0.95	0.3939
22	0.93	0.3916
23	0.94	0.3904
24	0.93	0.3923
25	0.91	0.3873
26	0.94	0.3874
27	0.99	0.3861
28	0.97	0.3844
29	0.97	0.3854
30	0.93	0.3832

Таблица 21: Таблица обучения PyTorch без оптимизации, 30 эпох

К сожалению на gpu не получилось запустить PyTorch с оптимизацией, поскольку видеокарта слишком старая и не поддерживает triton, необходимый для компиляции.

Графики обучения

Графики обучения на TensorFlow (cpu):

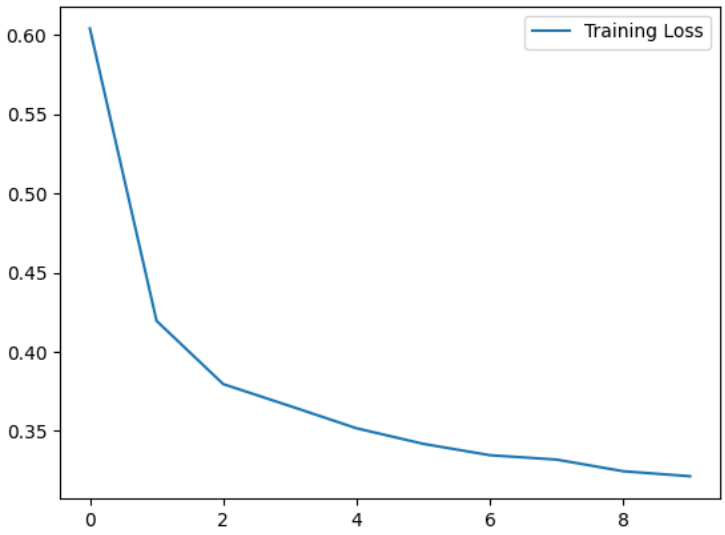

Рис. 1: График обучения к таблице 1

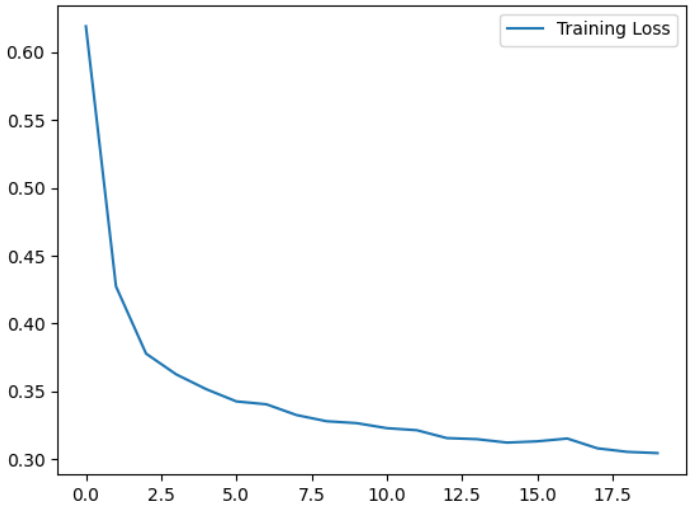

Рис. 2: График обучения к таблице 2

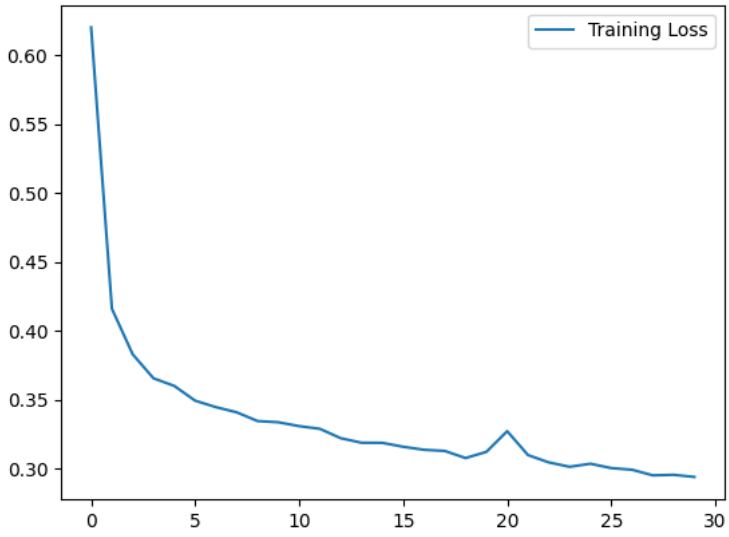

Рис. 3: График обучения к таблице 3

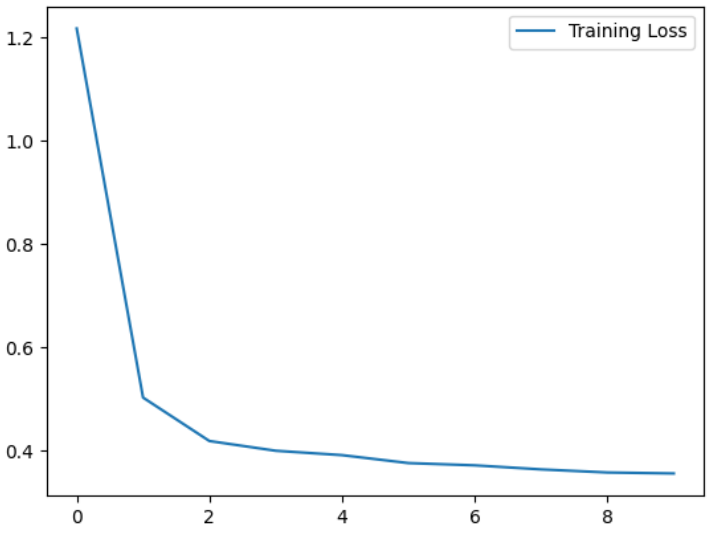

Рис. 4: График обучения к таблице 4

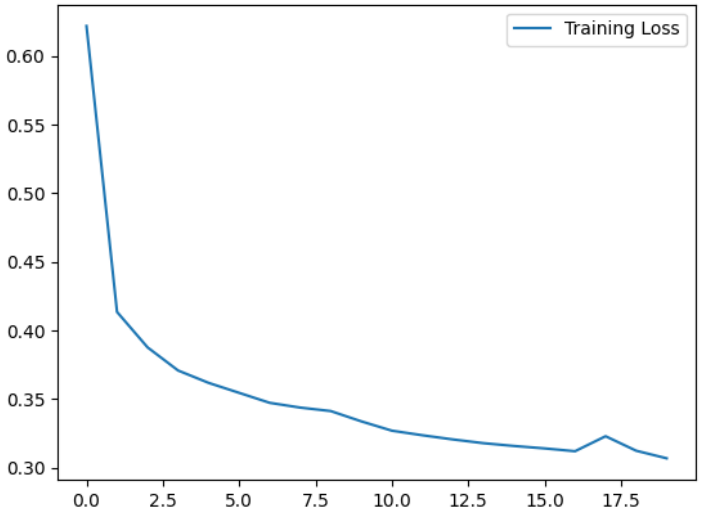

Рис. 5: График обучения к таблице 5

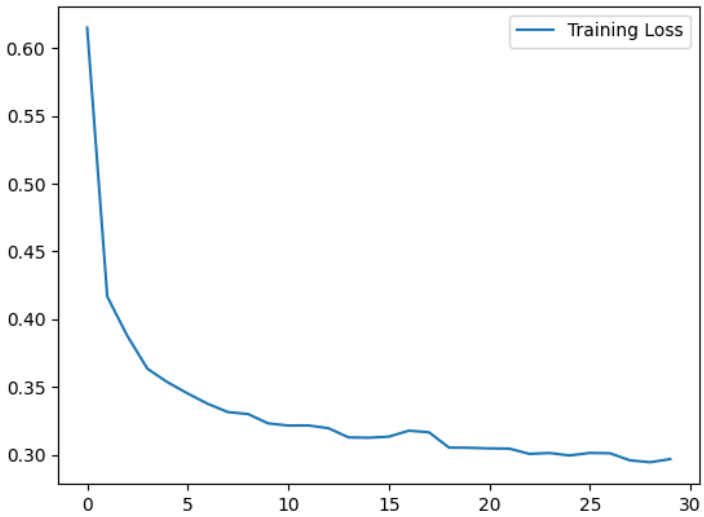

Рис. 6: График обучения к таблице 6

Графики обучения на PyTorch (cpu):

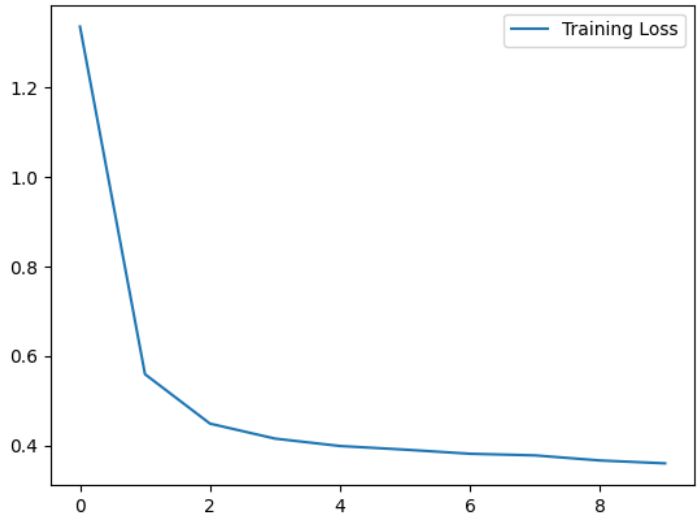

Рис. 7: График обучения к таблице 7

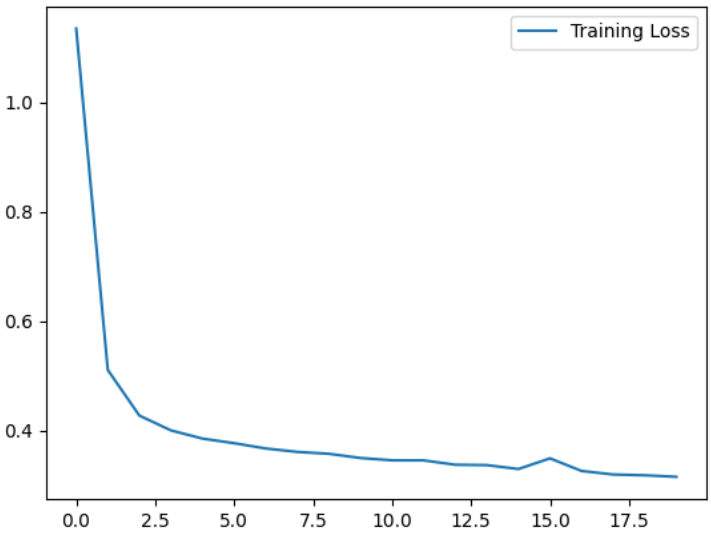

Рис. 8: График обучения к таблице 8

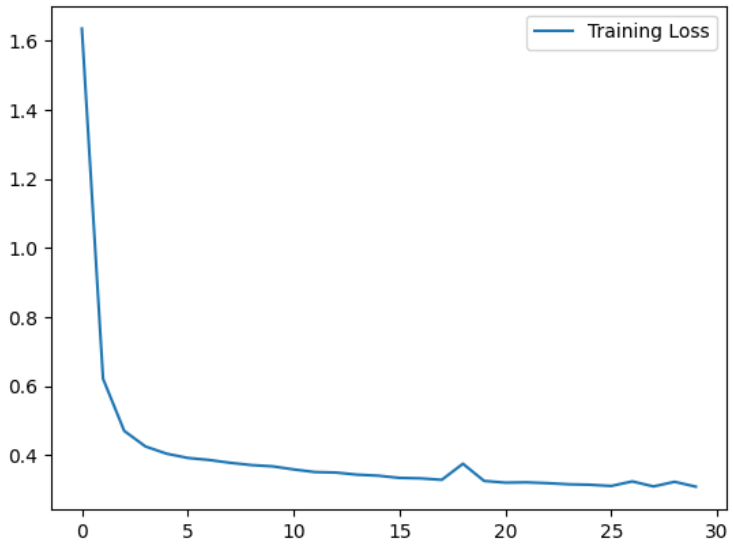

Рис. 9: График обучения к таблице 9

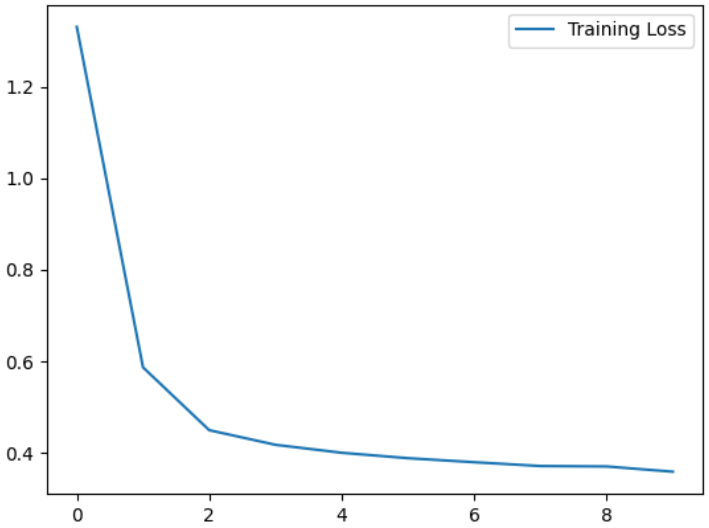

Рис. 10: График обучения к таблице 10

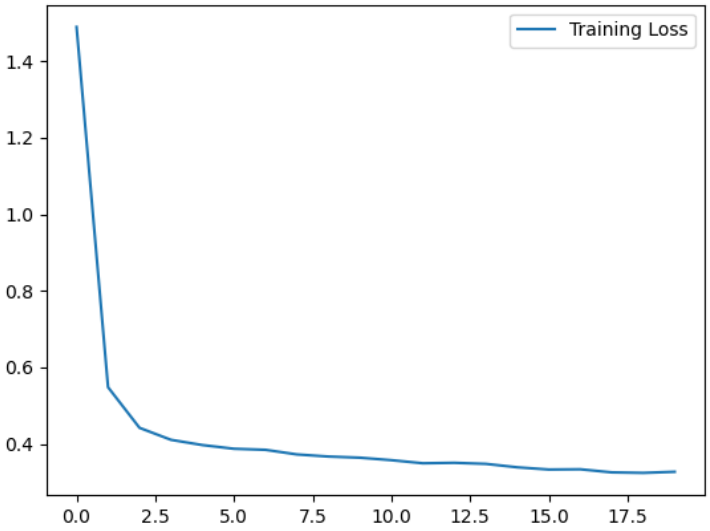

Рис. 11: График обучения к таблице 11

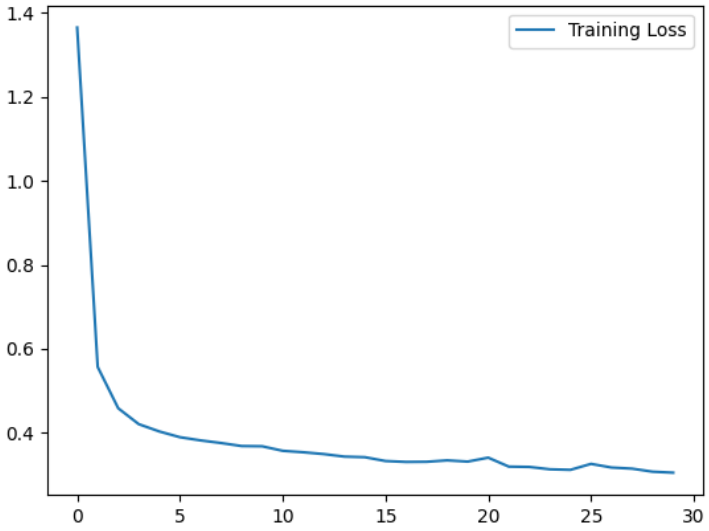

Рис. 12: График обучения к таблице 12

Графики обучения на TensorFlow (gpu):

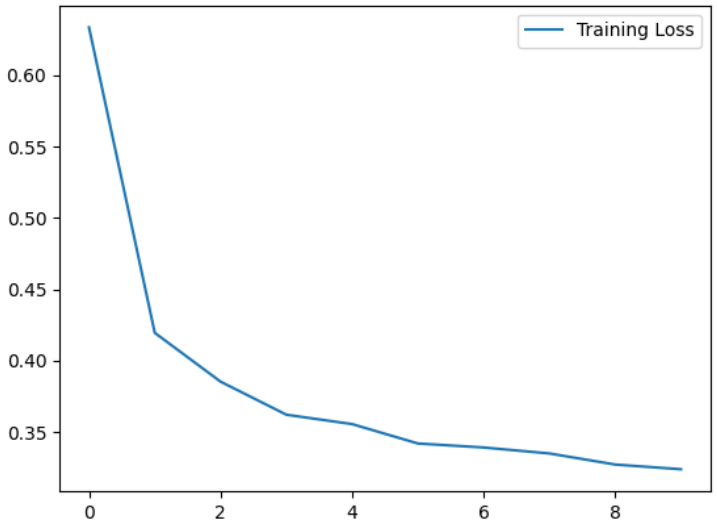

Рис. 13: График обучения к таблице 13

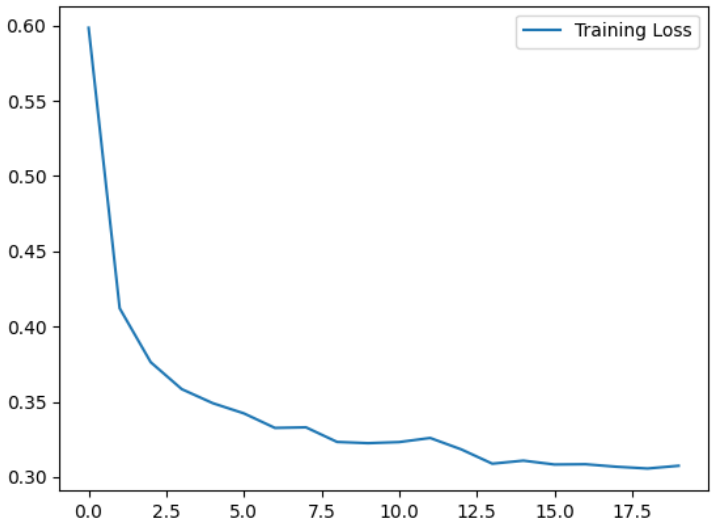

Рис. 14: График обучения к таблице 14

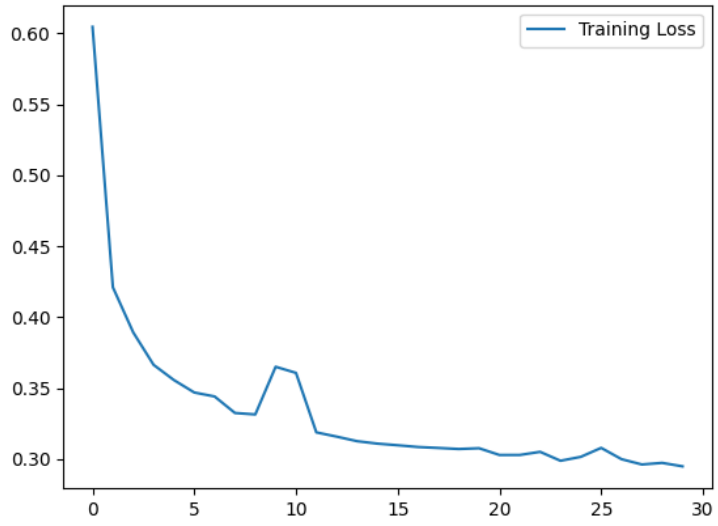

Рис. 15: График обучения к таблице 15

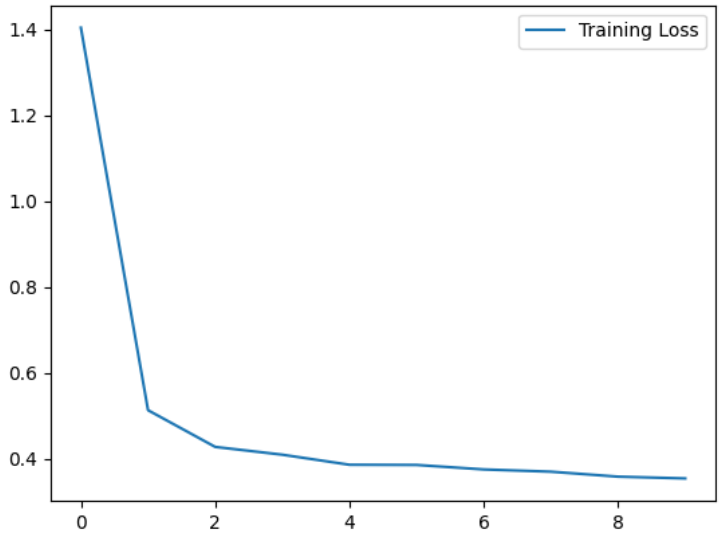

Рис. 16: График обучения к таблице 16

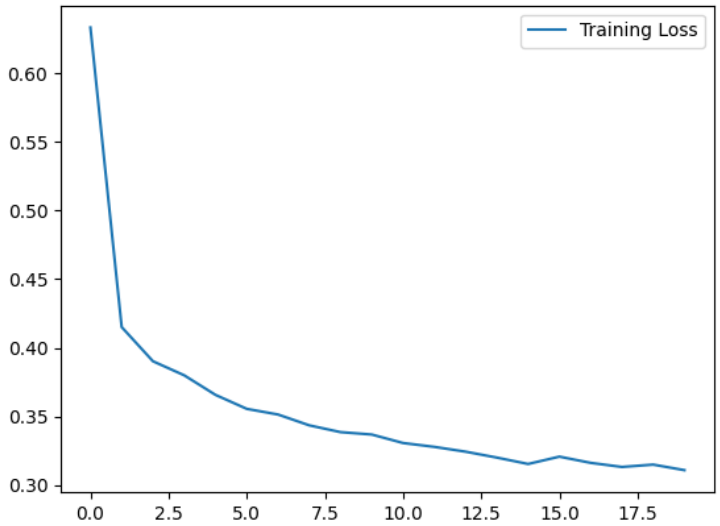

Рис. 17: График обучения к таблице 17

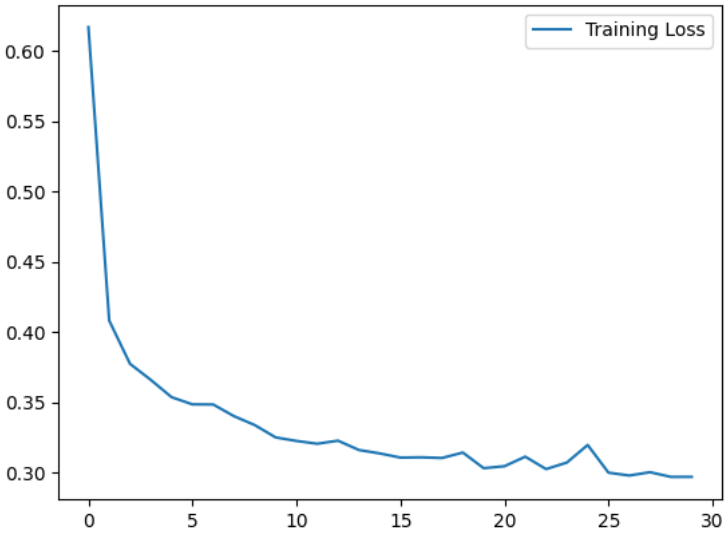

Рис. 18: График обучения к таблице 18

Графики обучения на PyTorch (gpu):

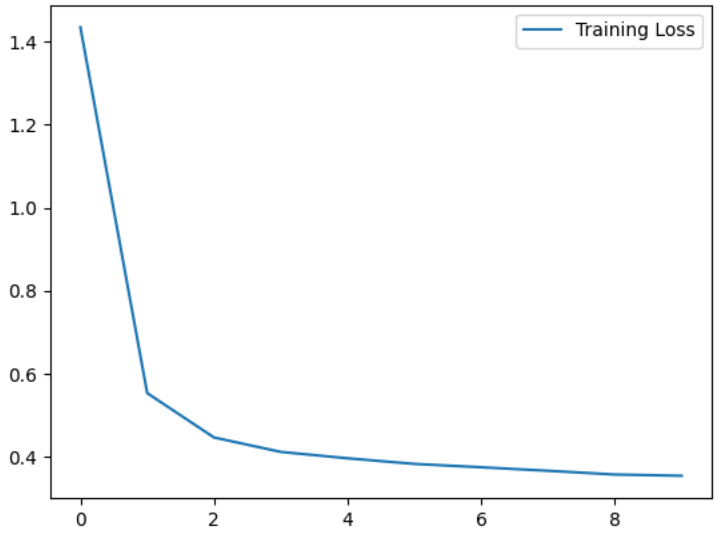

Рис. 19: График обучения к таблице 19

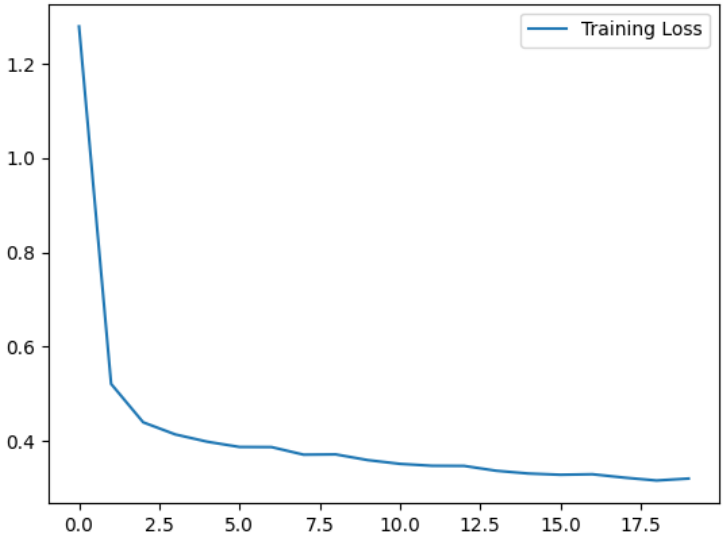

Рис. 20: График обучения к таблице 20

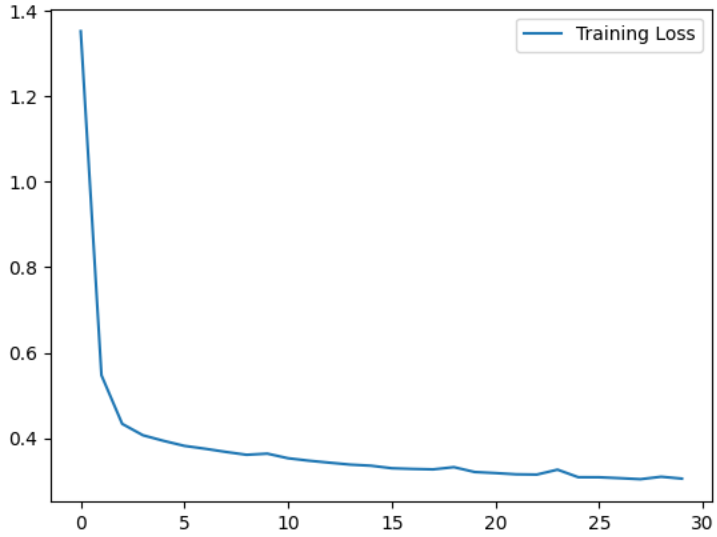

Рис. 21: График обучения к таблице 21

Анализ trade-off «скорость-память»

Количество эпох	Время	максимальная память
10	112.17 (11.21 на эпоху)	0.66 MB
20	220.91 (11.04 на эпоху)	0.26 MB
30	290.17 (9.67 на эпоху)	0.21 MB

Таблица 22: Таблица использования памяти и времени(TensorFlow без оптимизации cpu)

Количество эпох	Время	максимальная память
10	31.26 (3.12 на эпоху)	2.93 MB
20	60.94 (3.04 на эпоху)	0.47 MB
30	72.06 (2.40 на эпоху)	0.19 MB

Таблица 23: Таблица использования памяти и времени(TensorFlow с оптимизацией cpu)

Количество эпох	Время	максимальная память
10	4.30 (0.43 на эпоху)	0.71 MB
20	8.88 (0.44 на эпоху)	0.61 MB
30	14.34 (0.48 на эпоху)	0.60 MB

Таблица 24: Таблица использования памяти и времени(PyTorch без оптимизации cpu)

Количество эпох	Время	максимальная память
10	6.06 (0.61 на эпоху)	1.34 MB
20	11.96 (0.60 на эпоху)	1.00 MB
30	16.40 (0.55 на эпоху)	0.70 MB

Таблица 25: Таблица использования памяти и времени(PyTorch с оптимизацией cpu)

Количество эпох	Время	максимальная память
10	108.47 (10.84 на эпоху)	0.24 MB
20	224.23 (11.21 на эпоху)	0.25 MB
30	341.99 (11.40 на эпоху)	0.24 MB

Таблица 26: Таблица использования памяти и времени(TensorFlow без оптимизации gpu)

Количество эпох	Время	максимальная память
10	27.57 (2.75 на эпоху)	1.01 MB
20	53.17 (2.65 на эпоху)	0.70 MB
30	79.26 (2.64 на эпоху)	0.23 MB

Таблица 27: Таблица использования памяти и времени(TensorFlow с оптимизацией gpu)

Количество эпох	Время	максимальная память
10	11.22 (1.12 на эпоху)	0.66 MB
20	21.17 (1.06 на эпоху)	0.67
30	28.55 (0.95 на эпоху)	0.67 MB

Таблица 28: Таблица использования памяти и времени(PyTorch без оптимизации gpu)

Анализ точности обучения

Исследования CPU:

За 10 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3601
- MAE: 0.4175

За 20 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3407
- MAE: 0.3952

За 30 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3122
- MAE: 0.3839

За 10 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3382
- MAE: 0.4050

За 20 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3191
- MAE: 0.3935

За 30 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3080
- MAE: 0.3812

За 10 эпох обучения на PyTorch с оптимизацией на тестовой выборке:

- MSE: 0.3649
- MAE: 0.4289

За 20 эпох обучения на PyTorch с оптимизацией на тестовой выборке:

- MSE: 0.3466
- MAE: 0.4043

За 30 эпох обучения на PyTorch с оптимизацией на тестовой выборке:

- MSE: 0.3262
- MAE: 0.3881

За 10 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3649
- MAE: 0.4282

За 20 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3252
- MAE: 0.3939

За 30 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3173
- MAE: 0.3900

Исследования GPU:

За 10 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3564
- MAE: 0.4195

За 20 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3233
- MAE: 0.3932

За 30 эпох обучения на TensorFlow с оптимизацией на тестовой выборке:

- MSE: 0.3084
- MAE: 0.3789

За 10 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3360
- MAE: 0.4057

За 20 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3324
- MAE: 0.3938

За 30 эпох обучения на TensorFlow без оптимизации на тестовой выборке:

- MSE: 0.3141
- MAE: 0.3804

За 10 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3637
- MAE: 0.4260

За 20 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3344
- MAE: 0.3994

За 30 эпох обучения на PyTorch без оптимизации на тестовой выборке:

- MSE: 0.3167
- MAE: 0.3846

Выводы

Сравнение API фрэймворков

Начну с простоты использования. Для себя я определенно выделю в простоте TensorFlow, так как существует автоматическое обучение. Однако для оптимизации с помощью @tf.function нужно было писать обучение вручную, что делает удобтво и простоту примерно на одном уровне с PyTorch. Реализация перцептрона на обоих фрэймворках достаточно проста.

Расскажу так же о выводах сделанных после эксперименатов. На небольшой и простой модели трехслойного перцептрона PyTorch показал себя заметно лучше по скорости обучения, нежели TensorFlow (примерно 0.5 секунд на эпоху у PyTorch против 10 без @tf.funcion и 2-3 с указанной оптимизацией у TensorFlow), причем, как с оптимизацией, так и без, как на cpu, так и на gpu. В точности при тестах на выборке из датасета, не учавствовавшей в обучении результат получен неоднозначный. Выделить очевидного лидера в качестве обучения реализованной модели я не смог.

Рассмотрим полученную информацию по оптимизациям. @tf.function дает прирост скорости примерно в 5 раз при обучении взятой модели. Точность обучения при проведении эксперементов TensorFlow отличалась, но показатели потерь MSE и MAE были примерно одинаковы, что говорит либо о малом влиянии на точность, либо о его отсутствии. В это время оптимизация torch.compile показала менее очевидный результат. Проверка была сделана только на cpu, где реализация с оптимизацией показала результат хуже, чем реализация без. Отличие было в примерно 0.1-0.15 секунд, что составляет примерно 20% разницы.

Различия в реализациях на cpu и gpu была не в пользу последнего устройства как на TensorFlow, так и на PyTorch. Связываю это с простотой модели и маленьким размером батча, отчего видеокарта нагружается не целиком при сохранении накладных расходов при передаче данных на графический процессор.

В большинстве случаев PyTorch сильнее нагружал память. Нагрузка на память была незначительной, поскольку выбранное количество батчей не велико (64).

Таким образом, я могу порекомендовать PyTorch для обучения простых моделей, не требующих большого количества сложных вычислений из-за очень большого преимущества в скорости, а так же советую обучать простые модели на cpu в связи с образованием накладных расходов при обучении на gpu.

Рекомендации по использованию оптимизаций

При проведении эксперементов выяснил, что для простых моделей на TensorFlow использование @tf.function обязательно. Преобразование действий в граф и сохранение последовательноти действий выгодно увеличивает скорость обучения. Предполагаю, что на более сложных моделях выигрыш может быть меньше в связи с усложнением графа, но считаю, что он все еще будет.

С оптимизацией torch.compile все не так однозначно. Сравнение было только на cpu, поэтому буду писать именно о нем. torch.compile анализирует граф, использует некоторые

компиляции, хорошие для больших моделей, однако исследуемая модель мала и нужные оптимизации не только тормозят выполнение из-за самой компиляции, но и выбираемые оптимизации тормозят обучение модели, что было выявленно на всех испытаних (10, 20, 30 эпох). Для малых моделей рекомендую не использовать torch.compile при обучении на cpu.

Еще одна рекомендация: перед началом обучения пропустить модель сквозь датасет без обучения при испоьзовании PyTorch. "Разогрев"таким образом модель дальнейшее обучение будет точнее и быстрее. Первые эпохи обучения без выполнения рекомендации будут очень неточны и медленны.

Ограничения методов

Для обучения трехслойного перцептрона написание кода на PyTorch выглядит избыточным, поскольку требуется прописания многих вещей (например функцию обучения и функцию оценки) вручную, в то время как в TensorFlow присутствует model.fit(), служащий для обучения модели без излишнего кода.

TensorFlow кажется избыточным решением для простой модели из-за времени обучения, которое должно быть меньше у моделей сложнее.

По рзультатам исследований незначительна, но заметна большая скорость обучения простых моделей на cpu.