менее, по результатам этого раздела мы уже пришли к заметным результатам так на момент завершения раздела имеется модель с показателем AUC в 0,7792 на тренировочной выборке, что уже более чем допустимо. Но, как будет доказано далее, это не предел.

3.2.2 Более тонкая подгонка и валидация финальной модели

В предыдущем пункте мы остановились на том, что после определенной итерации процесс оптимизации теряет стабильность в результате слишком большого шага оптимизации. Позже это вовсе может привести к тому, что целевая функция вообще не может оптимизироваться и даже может показаться системный рост. Но вместе с этим решения связанные с радикальным понижением learning rate нам так же подходили в виду слишком медленного процесса оптимизации, кроме того, алгоритм становиться более чувствительным к локальным минимумам. Проблемы связанные с большей вероятностью попадая в локальный минимум можно конечно обойти с помощью меньшего размера батча, но это также приводит к повышению вычислительных затрат – в каждую эпоху будет включено больше шагов.

Описанные проблемы можно решить динамическим изменением learning rate в зависимости от стадии обучения. Конечно, можно было бы самостоятельно описать логику, но в pytorch реализован специальный набор инструментов – планировщики learning rate. Более подробное описание можно найти в [16] в разделе «How to adjust learning rate».

Для его использования нужно создать экземпляр одного из классов-оптимизаторов или, даже, создать собственный наследуя от одного из уже созданных. При создании, первым обязательным аргументом указывается оптимизатор вместе с которым участвует планировщик. Для задействования оптимизатора в алгоритме рекомендуется сразу после вызова метода «step» оптимизатора вызвать тот-же метод у планировщика.

При проектировании класса-учителя модели мы сразу заложили возможность использования планировщика, достаточно лишь в конструктор именованным аргументом передать созданный планировщик.

Так же достаточно трудоемко вести оптимизацию в полу ручном режиме, потому был предусмотрен алгоритм остановки оптимизации на случай остановки уменьшения целевой функции. Тут напомним случай рисунка 3.2 справа, несмотря на то, что функция возрастает начиная с 8-ой эпохи, к 12-й оптимизация продолжается и к 20-й достигает куда более хороших результатов. Представленный пример описывает, что просто контролировать чтобы значение убывало по сравнению с предыдущей итерацией недостаточно.

Мы предлагаем следующий подход – будем контролировать не по предыдущей эпохе, но через произвольно выбираемое число эпох. Для указания числа эпох через которое начинается осуществляться проверка у метода «model_trainer.fit» предусмотрен именованный аргумент «check_epochs». И так теперь более подробно, если на каждой конкретной эпохе значение целевой

функции на тестовых данных оказывается больше нежели «check_epochs» эпох назад то алгоритм прерывается, не взирая на то, сколько эпох указано для обучения модели.

Кроме того предусмотрено сохранение лучшей, в смысле целевой функции на тестовых данных, модели и советующей ей эпохи в полях «best_model» и «best_epoch». Нужно это потому, что приведенный критерий остановки в общем случае допускает повышение целевой функции и таким образом мы отловим лучшую модель.

Начнем описание конкретной реализации всех описанных идей. Мы будем использовать планировщик «optim.lr_scheduler.ExponentialLR» этот планировщик в конструктор принимает именованный аргумент «gamma» который показывает во сколько раз после каждой эпохи уменьшается learning rate, понятно, что нужно указывать число меньше единицы.

Вот пример, как может быть создан планировщик:

lr_scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma = 0.99).

Расскажем о неудачных попытках применения этой технологии. Изначально выставили «gamma» 0,95 и за 300 эпох было модель не достигла значимого результата. В таблице М.1 последняя ситуация представлена в строке соответствующей тремя сотням итераций и 0,95 «gamma» – результаты даже хуже, чем без использования нововведений. Объяснение такого поведения объясняет рисунок М.1. Там алгоритм останавливается досрочно после выхода на плато, мы сделали вывод, что такое поведение связано со слишком быстрым уменьшением learning rate.

В результате параметр «gamma» в планировщике было решено увеличить до 0,99. Результаты модели обученной при таких параметрах представлены в строке с 300 эпох и 0,99 «gamma». Уже достаточно хорошо и то, что функция обучающей выборки постепенно отрывается от тестовой, свидетельствует о постепенно появляющемся переобучении модели. Но, после изучения поля «best_epoch» облучателя модели, по значению 300 стало понятно, что не выполнилось выставленное требование к остановке модели, потому, скажем доучивать модель до тысячи эпох в надежде что алгоритм остановиться.

После около десяти минут вычислений на персональном компьютере базовой комплектации, алгоритм остановился. Характеристика качества полученного классификатора представлена в таблице М.1 в строке соответствующей тысяче эпох и параметру «gamma» 0,99.

Для того, чтобы удостовериться, что мы действительно вышли на плато на рисунке 3.6 рассмотрим полученную кривую обучения. По рисунку, во первых, можно понять, что мы действительно вышли из алгоритма по критерию остановки на 516-й эпохе, во вторых, что скорее всего мы выжали все возможное из этой модели – последние три эпохи нет совершенно никакого улучшения значения целевой функции ни по тестовым ни по тренировочным данным.