Заметим, что для тестовых данных можно было обойтись и без использования загрузчика, но, для общности, они в алгоритм обучения попадают также через него.

Без внимания остались еще поля «train_loss» и «test_loss» класса «model_trainer». В них, по окончании каждой эпохи, сохраняются значения целевой функции на полных тренировочных и тестовых данных соответственно. Эта информация потом используется для построения кривых обучения – графиков на которых можно отследить как изменяется значение целевой функции с новыми эпохами. Эта визуализация является популярным способом диагностики состояния модели.

В этом подразделе работы были описаны основы работы с пакетом pytorch и представлены созданные программные методы для решения поставленной задачи. Еще остаются нераскрытые детали, но приведенных сведений достаточно для того чтобы начать рассматривать конкретные примеры, а прочие механизмы мы подключим по мере необходимости.

3.2 Обучение и валидация модели

3.1.1 Подбор параметров обучения

Перед тем, как использовать созданные вычислительные процедуры, придется провести еще три относительно простых преобразования данных. Они вынесены за рамки второй главы работы, так как действуют уже без использования какого-либо смысла вложенного в показатели, а рассматриваются лишь как наборы чисел.

Во первых для номинативных показателей следует провести one hot encoding (OHE) – процедуру при которой каждый номинативный показатель распадается на столько показателей, сколько он имеет уровней, каждый произведенный таким образом показатель принимает значение «1» в том случае если базовый для него показатель принимал соответствующий уровень и «0» в любом другом случае. Смысл легко понять взглянув на рисунок 3.2.

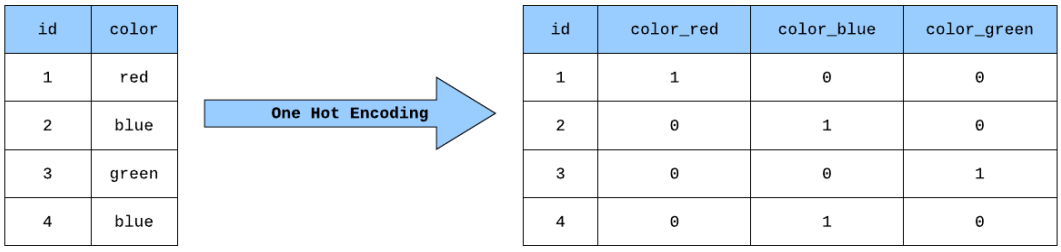

Рисунок 3.2 – Пример процедуры OHE
Примечание – Источник: [21].

Имеется таблица содержащая колонки «id» и «color». Столбец «color» принимает три уровня: «red», «blue» и «green». После проведения названного

преобразования к колонке «color» он образует три столбца, по одному для каждого из уровней.

Для проведения такой процедуры можно использовать класс «OneHotEncoder» модуля «sklearn.preprocessing».

Вторя процедура, которую мы проводили над данными – нормализация данных. Это не обязательная процедура для моделирования, но ее нередко рекомендуют проводить для увеличения эффективности процесса обучения[22 с. 112]. Опуская подробности, эта процедура оказала заметное позитивное влияние на процесс обучения. Нормализация проводилась с использованием функции «normalize» модуля «sklearn.preprocessing».

Последнее преобразование данных касается разделения данных на тренировочную и обучающую выборки. Для этого используется метод «train_test_split» модуля «sklearn.preprocessing». По умолчанию, этот метод в тестовую выборку помещает 25% процентов исходной. В именованном аргументе «stratify» мы указали столбец отклик, для того, чтобы исходное соотношение наблюдений с дефолтом и без сохранилось и внутри производных выборок. В аргументе «random_state» можно указать произвольное число, это обеспечит одинаковое разбиение при различных запусках программы.

Теперь, наконец, перейдем к описанию процесса обучения модели. Данные после ОНЕ содержат 113 столбцов, потому входной слой нейронной сети должен содержать столько же нейронов.

Изначально мы пытались использовать подход, который позволял нам рассматривать сразу некоторое множество идентификационных форм модели, но он не дал особого результата – каждая конкретная модель вела себя по разному и для их улучшения требовались уникальные настройки алгоритма обучения. Потому мы сконцентрировались на конкретной модели – неплохие результаты получались при использовании 113 же нейронов в скрытом слое.

После ряда экспериментов мы пришли к выводу, что лучше всего, во всяком случае вначале, использовать размер батча в 500 наблюдений. Примерно также мы определились с начальным learning rate в одну сотую.

На листинге К.4 представлен полный код как просто запустить модель на обучения, используя все те техники и параметры, которые мы описали выше. Заметим лишь, что под именами «X_train», «y_train», «X_test», «y_test» определены разбитые на тренировочную и тестовую выборки предикторы и столбец отклик, а запись «torch.manual_seed(0)» непосредственно перед созданием экземпляра модели обеспечит постоянство весов вне зависимости от запуска программного кода.

Кривые обучения, за первые 20 эпох обучения модели, представлены на рисунке 3.3.

Слева представлены результаты оптимизации за все 20 эпох. Первый шаг оптимизатора дает очень значимое снижение ошибки, в результате вся последующая оптимизация совсем не видна на графике, для того мы справа показали меньше эпох. Впредь на обучающих кривых будут опущены первые эпохи потому читателю стоит обращать внимание на ось ординат