1. Каким образом происходит скалярное произведение тензоров разных рангов? Какие есть ограничения?

В NumPy есть функция *dot*, которая возвращает результат скалярного произведения векторов (также ей можно перемножать матрицы). Если у нас есть два тензора *a* (*N*-мерный) и *b* (*1*-мерный), то результатом будет сумма произведений по последней оси *a* и *b*. Если *a* (*N*-мерный) и *b* (*M*-мерный), то в результате будет сумма произведений по последней оси *a* и предпоследней оси *b*. С помощью функции *tensordot* можно задать по каким осям производить сумму произведений (можно по нескольким осям).

Длина последней оси массива *a*, а так же длина всех осей массива *b* кроме первой должны совпадать, иначе возникнет исключение *ValueError*.

2. **С повышением количества слоев в ИНС, степень нелинейности возрастает/убывает/не изменяется? Если меняется, то почему?**

Возрастает. Добавление большего количества слоев позволяет лучше представлять взаимодействия во входных данных. Например нужно научиться прогнозировать цену продажи домов с учетом следующих характеристик: количество комнат, количество окон, количество ванных и т.д. Первый уровень может изучить простые неабстрактные функции, например о том, как эти входные функции связаны с продажной ценой, такие как: больше комнат = более высокие цены продажи. По мере того как мы углубляемся в сеть, изучаемые ею функции будут становиться все более сложными маленький дом + очень старый дом = высокая цена продажи и большой дом + небольшое количество ванных комнат = низкие цены продажи. Функции, изученные на более глубоких уровнях, становятся намного более сложными и подробными, поэтому нужно чтобы в сети было много слоев, из-за которых, следовательно и увеличивается нелинейность.

3. Что такое обратное распространение ошибки? Для чего оно нужно?

Обратное распространение ошибки это практика точной настройки весов нейронной сети на основе частоты ошибок (потерь), полученных в

предыдущую эпоху (итерацию). Оно направлено на то, чтобы вернуть эту потерю назад таким образом, чтобы мы могли точно настроить вес, на основании которого функция оптимизации поможет найти веса, которые приведут к меньшим потерям в следующей итерации. Правильная настройка весов обеспечивает более низкий уровень ошибок, что делает модель надежной за счет увеличения ее обобщения.

4. На рисунке 13 в отчете, в чем измеряется средняя точность?

Относительно задачи, которую мы решаем нашей нейронной сетью, в тыс. долларах.

5. Какой по умолчанию тип доступа к полям класса в python?

Тип доступа — *public*.

6. Почему в качестве метрики используется mae?

В данной работе мы решаем задачу регрессии, т. к. понятие точности не приемлемо для регрессии, то для оценки качества часто применяется *mae* (абсолютное значение разности между истинными и прогнозируемыми значениями). Так же *mae* не чувствителен к выбросам и, учитывая несколько примеров с одинаковыми значениями входных признаков, оптимальным прогнозом будет их медианное целевое значение. Например, в отличии, от *mse*, где оптимальным прогнозом является среднее значение.

7. self.model.evaluate(val_data, val_targets, verbose=0) - что это за функция и какие параметры она принимает в данном случае?

Эта функция возвращает значения потерь и метрики для модели в тестовом режиме, расчет производится партиями (*batches*), где

val_data - входные данные

val_targets - целевые данные

verbose - режим многословия 0 или 1. 0 - ничего не покажет, 1 - индикатор выполнения. Задавая подобные 0,1, мы просто говорим, как хотим видеть прогресс в обучении для каждой эпохи.