1. Что такое L2 регуляризация весов?

Регуляризация — это один из способов предотвратить переобучение модели. Для борьбы с переобучением можно увеличить количество данных, уменьшить количество признаков или ограничить веса. Когда модель переобучается, веса у признаков становятся большими по модулю и разными по знаку. Ограничением их по модулю занимается регуляризация. Существуют L_1 и L_2 регуляризаторы. L_2-регуляризатор добавляет к функционалу потерь слагаемое, равное сумме квадратов весов в линейной модели с множителем λ.

$$J_{RIDGE} = \sum_{i=1}^{N} (y_n - \widehat{y}_n)^2 + \lambda \|w\|_2^2,$$

L_1-регуляризатор отличается только тем, что добавляет вместо суммы квадратов сумму модулей весов с тем же множителем λ.

$$J_{LASSO} = \sum_{i=1}^{N} (y_n - \widehat{y}_n)^2 + \lambda \|w\|_1.$$

Регрессия с L_2-регуляризатором называется ридж-регрессией или гребневой регрессией, а с L_1-регуляризатором – регрессией лассо. Использование регуляризации позволяет уменьшить матожидание квадрата ошибки! Матожидание квадрата ошибки дисперсии состоит из суммы смещения, дисперсии и шума (с ним ничего нельзя сделать). МНК (метод наименьших квадратов) оценки имеют нулевое смещение. Переобучение приводит к тому, что смещения нет, а дисперсия большая. Регуляризация приводит к тому, что мы получаем смещенные оценки коэффициентов модели, но суммарная ошибка может быть меньше за счет меньшей дисперсии этих оценок коэффициентов. Это похоже на стреляние в мишень не совсем в цель (регуляризация дала смещение), но более точно (уменьшилась дисперсия). А с нулевым смещением и большой дисперсией попаданий было бы гораздо меньше.

2. При какой скорости обучения больше вероятность попасть в локальный минимум, при маленькой или большой?

Есть такой алгоритм обучения, как градиентный спуск, он применяется почти в каждой модели машинного обучения. Его суть как раз и представляет

то, как обучаются модели. Он является методом нахождения минимального значения функции потерь. Минимизация функции – это поиск самой глубокой впадины в этой функции, ведь локальных минимумов может быть много, а нам нужен самый минимальный. Найдя такой минимум, мы получим наименьшую возможную ошибку или повысим точность модели. Градиентному спуску нужен градиент (производная функции потерь относительно одного веса, выполненная для всех весов) и функция потерь.

Функция потерь отслеживает ошибку с каждым примером обучения, а нам нужно найти, куда сместить вес, чтобы минимизировать эту функцию потерь для данного примера обучения. Для этого мы используем производную функции относительного одного веса. Нам надо знать, насколько сильно надо скорректировать вес, для этого мы используем коэффициент скорости обучения. Его значения – от 0 до 1, где 0 значит, что корректировка веса производиться не будет. Коэффициент скорости обучения можно рассматривать как «шаг в правильном направлении», где направление задаст градиент! При помощи производной мы находим направление, а с помощью скорости делаем шаг в этом направлении размером, зависящим от величины скорости. Таким образом, чем выше скорость (ее еще называют гипер-параметр), тем выше вероятность промахнуться, шагнув мимо минимума. Из этого я делаю вывод, что при маленькой скорости обучения больше вероятность попасть в локальный минимум и вообще в нем застрять и так и не найти самый минимальный локальный минимум для успешной минимизации.

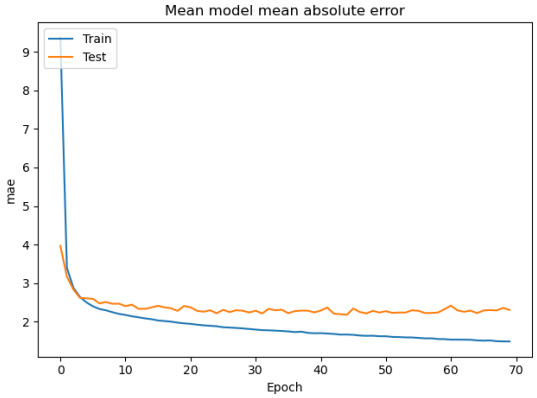
3. Верно ли утверждение, что сверточная сеть не является сетью прямого распространения?

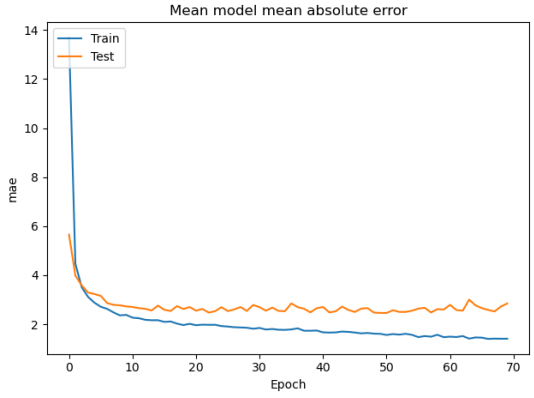
Да, верно. В сети прямого распространения сигнал распространяется только от входного слоя к выходному, никогда не обратно. СНС использует метод обратного распространения ошибки: проводит прямое распространение, применяет функцию потери, проводит обратное распространение и обновляет

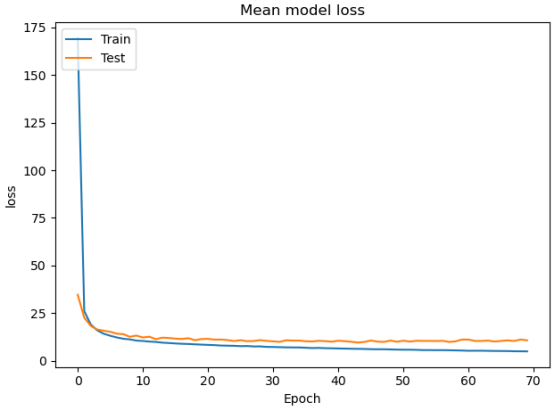
вес. Обратное распространение помогает СНС найти веса, оказавшие большее влияние на потери, чтобы их настроить и потери уменьшить.

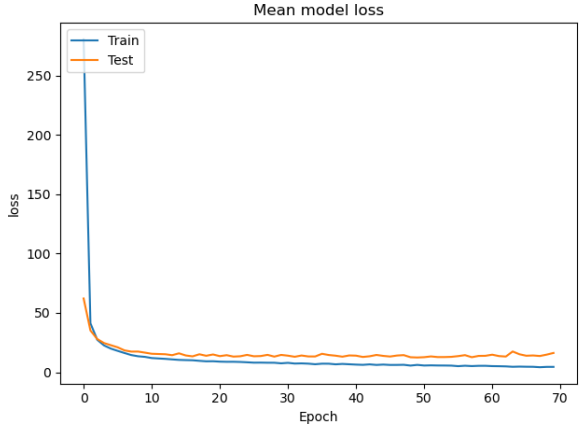
4. Какая в итоге архитектура сети оказалось наилучшей?

От увеличения количества промежуточных слоев результаты (значения точности и потерь) на тестах становились только хаотичнее, так что я оставила как было, один с 64 нейронами и функцией активации relu. Увеличение количества эпох только увеличивало общий наклон графика значения mean absolute error, а уменьшение количества эпох в среднем показывало результаты с меньшей mae, т.е. средней ошибкой, чем для бОльшего кол-ва эпох, поэтому я решила, что лучше оставлю его уменьшенным со 100 до 70. Особой разницы между увеличенным количеством блоков и уменьшенным я не заметила. Я бы даже сказала, что меньше блоков – лучше, основываясь на своих средних результатах, но вдруг они случайные... Слева k = 2, справа k = 20. Так что я оставила k = 4.









5. Для чего нужна test_data?

Хранить данные для тестирования, которые загружаются в нее из библиотеки. Больше никак не используется, в работе используются train_data и train_targets.

6. В какой строчке я получаю значение mae на обучающей выборке?

В строчке 60. А mae на тестах в строчке 59.

7. Сколько нейронов сдвига в моей модели?

По умолчанию в функции Dense параметр use_bias всегда True, если его не изменить. В этом можно убедиться на сайте документации keras https://keras.io/layers/core/ или ctrl+ЛКМ по функции Dense в PyCharm. `activation(dot(input, kernel) + bias)`, а на последнем слое функции активации нет, поэтому выходит, что всего у меня нейрон сдвига один, но используется для балансировки весов 64-х нейронов.