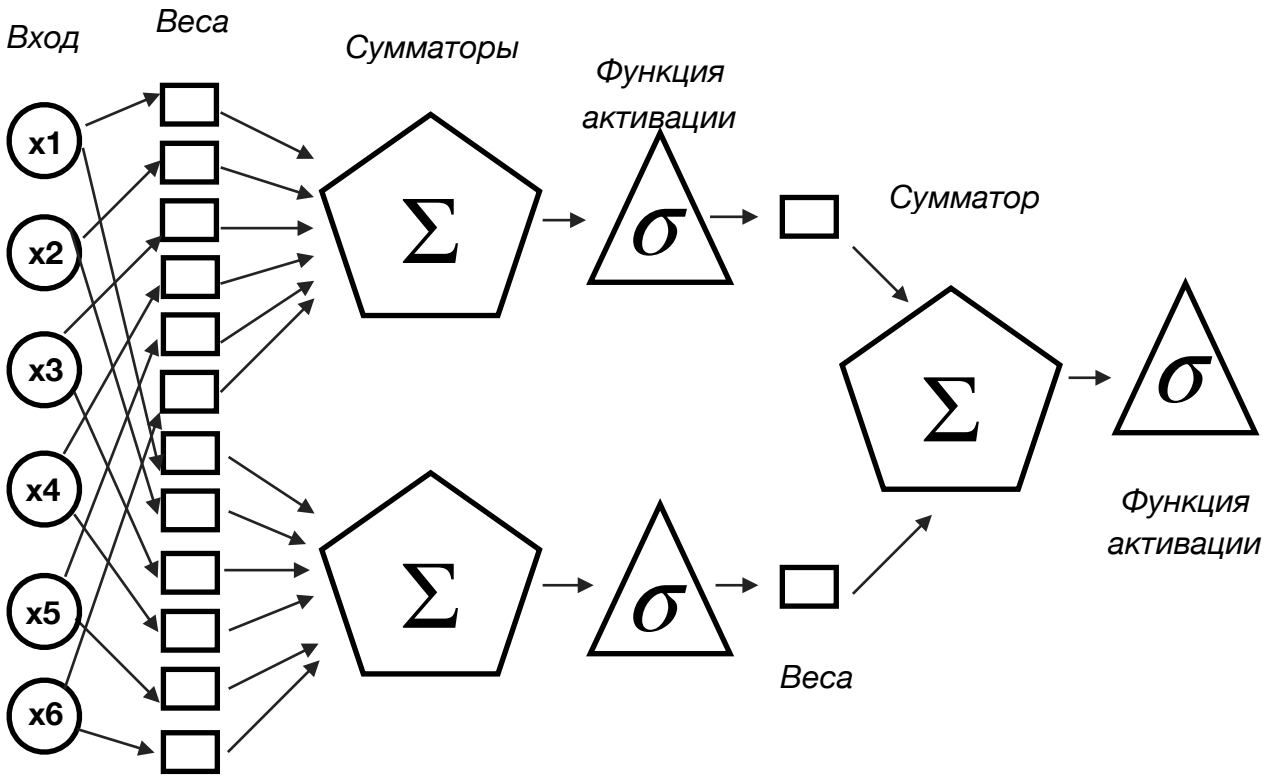
Отчёт. Нейронная сеть

Используемые методы

1. Схема, которую я реализовывала.



Как можно видеть, я использовала сигмоиду в качестве функции активации на всех слоях. Также на всех слоях я использовала сумматоры.

Входные данные:

У каждого объекта было по 6 признаков, так как в двоичном представлении цифры от 0 до 63 включительно имеют 6 разрядов.

Loss function:

$$(a(x_i \mid w) - y_i)^2/2$$

Отчёт. Нейронная сеть

Функция активации:

Сигмоида

Оптимизация:

Градиентный спуск с learning rate равным константе в программе neuronsconst2layers.py

Градиентный спуск со схемой из двух слоёв и learning rate равным производной по переменной в neuronsdiff2layers.py

Градиентный спуск со схемой из одного слоя в программе neurons1layer.py

Корректировка весов:

$$w1 += -\sigma(w_{13}\sigma(\sum_{i=1}^{6} w_i x_i) + w_{14}\sigma(\sum_{i=1}^{6} w_{i+6} x_i) w_{13}(1-\sigma(\sum_{i=1}^{6} w_i x_i))(\sigma(\sum_{i=1}^{6} w_i x_i)) x1 * learning\_rate$$

$$w13 += -\sigma(w_{13}\sigma(\sum_{i=1}^{6} w_i x_i) + w_{14}\sigma(\sum_{i=1}^{6} w_{i+6} x_i)\sigma(\sum_{i=1}^{6} w_i x_i) * learning\_rate$$

Первая формула описывает корректировку весов в первом слое, вторая формула описывает корректировку весов во втором.

Выводы

Ошибки на обучающей выборке я не заметила. Имеется в виду, что итоговое значение Loss было меньше 0.5.

Для значений функции потерь равном 0.00001 мне хватило 124197 итераций. Это случилось в программе с частными производными по переменным в качестве learning rate. А в другой программе, как мне кажется, я попала в локальный минимум, именно поэтому она зациклилась на learning rate = 0.001, то есть значение функции потерь прыгали вокруг оптимальной точки. Я попробовала сделать его равным 0.01 - меня ушло 600139 итераций на достижение порогового значения функции потерь. Теперь, руководствуясь здравым смыслом, можно сделать его равным 0.1. На этот раз ушло 968782 итераций. Теперь

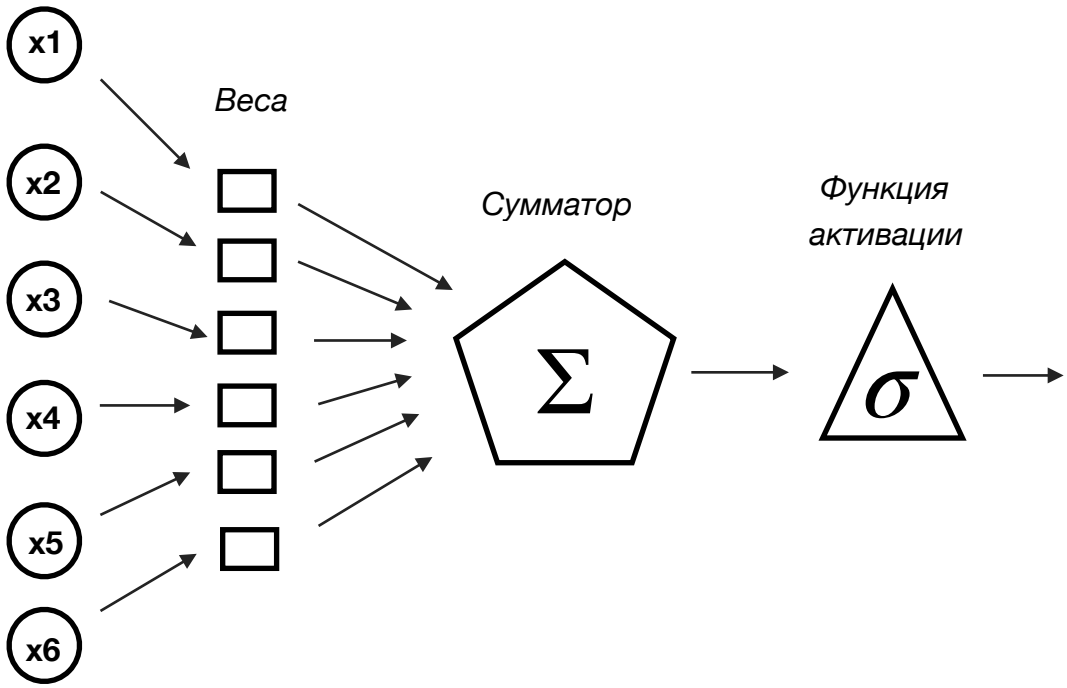
сделаем его равным 10 - 17 итераций! Увеличиваем до 100, получаем 2 итерации. На этом этапе можно сделать вывод о том, что learning rate в качестве константы - это очень гибкий алгоритм, но проблема в том, чтобы выбрать нужное значения. На лекциях нам давалась оценка в случае, если наша функция липшицева и дифференцируема.

Я попробовала изменить количество векторов в обучающей выборке. Уменьшила его до 1, при этом learning rate остался равен 0.01. Но программа зациклилась. Можно увеличить выборку, добавив в неё ещё один вектор. Но в этом случае, алгоритм тоже не сходится. Я попробовала увеличить learning rate, но это ни к чему не привело. Добавим ещё один вектор. А теперь всё замечательно сошлось.

Дело в том, что я ещё подкорректировала входные данные. Сделала их очень мало различающимися по первым переменным и имеющими отличия по последнему признаку и сеть научилась очень быстро.

Таким образом, в этом случае для сходимости хватило двух векторов в обучающей выборке.

Мы рассматривали модель состоящую из двух слоёв, а что будет с ней, если уменьшить количество слоёв до 1? Давайте посмотрим. Нарисуем схему в этом случае



Отчёт. Нейронная сеть

Корректировка весов в этом случае производятся по такой формуле:

$$w1 += -\sigma(\sum_{i=1}^{6} w_i x_i) x1 * learning\_rate$$

Если брать константу 0.01 в качестве learning rate, три вектора в обучающей выборке, то можно получить необходимое пороговое значение 10^{-5} на 2472697 итерации. Если увеличить learning rate до 0.1, можно получить нужное значение на 248146 итерации. Если увеличить до 1, получаем порог на 25732 итерации. Попробуем повторить опыт с 10 - 18 итераций. А если сделать 100, то мы получим значение уже на десятой. При learning rate = 1000, получаем на первой.

Я добилась этого результата на такой обучающей выборке:

```
object1 = [1, 1, 1, 1, 1, 0] target1 = 1
object2 = [1, 1, 1, 0, 0, 0] target2 = 1
object3 = [1, 1, 1, 1, 1, 1] target3 = 0

learning rate = 1000
```

Сравнив две модели - с одним слоем и двумя, можно сказать, что модель с двумя слоями ведёт себя более непредсказуемо, и тратит меньше итераций на определение. Также она может легко обучиться, если подать ей один вектор, например. Я положила в обучающую выборку вектор.

object1 = [1, 1, 1, 1, 1, 0] target = 1

Мне потребовалось много времени, чтобы обучить нейронную сеть - 5068548 итераций, можно было проследить за корректировкой весов и минимизации функции потерь.

```
weights =  [1.08093102 1.08093102 1.08093102 1.08093102 1.08093102
0.5       ]
Loss =  1.0014981412644546e-05

weights =  [1.08095096 1.08095096 1.08095096 1.08095096 1.08095096
0.5       ]
```

Отчёт. Нейронная сеть

```
Loss =  1.0012993724522799e-05

weights =  [1.08097089 1.08097089 1.08097089 1.08097089 1.08097089
0.5       ]
Loss =  1.0011006823470389e-05

weights =  [1.08099082 1.08099082 1.08099082 1.08099082 1.08099082
0.5       ]
Loss =  1.0009020709021146e-05

weights =  [1.08101075 1.08101075 1.08101075 1.08101075 1.08101075
0.5       ]
Loss =  1.0007035380708577e-05

weights =  [1.08103067 1.08103067 1.08103067 1.08103067 1.08103067
0.5       ]
Loss =  1.0005050838067346e-05

weights =  [1.08105059 1.08105059 1.08105059 1.08105059 1.08105059
0.5       ]
Loss =  1.0003067080629802e-05

weights =  [1.08107051 1.08107051 1.08107051 1.08107051 1.08107051
0.5       ]
Loss =  1.0001084107932434e-05

Result =  0.9955278641000689
Loss =  9.999999753726354e-06
Iteration =  5068548
```