# **Описание работы нейронной сети**

Программа начинает работу с четырьмя папками, в которых лежат верные и ошибочные данные для обучения и верные и ошибочные данные для теста. С помощью модуля MFCC мы преобразуем звук в список из 10 признаков. В переменные ITR, ITW, IOR, IOW соответственно загружаются списки из преобразованных раннее звуков для тренировки и для теста, а в ytrain и ytest – 1, если является хлопком, 0 - если нет.

ITR, yrainR = mfcc.get\_features("./data/input\_train\_right/", ytrainR, [1])  
ITW, ytrainW = mfcc.get\_features("./data/input\_train\_wrong/", ytrainW, [0])  
IOR, ytest = mfcc.get\_features("./data/input\_test\_right/", ytest, [1])  
IOW, ytest = mfcc.get\_features("./data/input\_test\_wrong/", ytest, [0])

Далее мы получаем два списка для теста: ITest, в котором лежат сначала верные данные для теста, а затем ошибочные, и ytest, в котором лежат сначала 1, потом 0, соответствующие тому, являются ли данные в ITest хлопком или нет. Затем с помощью функции shuffle мы параллельно перемешиваем данные в списках, не нарушая соответствия каждому элементу из ITest его истинности из ytest.

ITest = IOR + IOW  
ITest, ytest = shuffle(ITest, ytest)

Переходим к структуре сети nn\_structure = [11, 15, 10], первый параметр - входные нейроны (10 признаков звука + 1 нейрон смещения), второй - скрытый слой (методом проб и ошибок установлено, что их должно быть 8 – 20, поэтому их 15), третий - выходной слой, 10 значений, соответствующие 10 признакам после обработки. Так же создаем пустые списки для весов - W, и для нейронов смещения - b.

nn\_structure = [11, 15, 10]

W = list()  
b = list()

Теперь непосредственно переходим к самому обучению. Так как изначально у нас имеется неравное количество верных и ошибочных данных для обучения, мы проводим выборку по 200 с каждого списка ITR и ITW, соединяем в список ITrain и перемешиваем, параллельно с ytrain. Таким образом, за одну эпоху будет обрабатываться по 400 данных

part\_of\_ITR, part\_of\_ytrainR = take\_part\_of\_array(ITR, 200, ytrainR)  
part\_of\_ITW, part\_of\_ytrainW = take\_part\_of\_array(ITW, 200, ytrainW)  
ITrain = part\_of\_ITR + part\_of\_ITW  
ytrain = part\_of\_ytrainR + part\_of\_ytrainW

Реализация сети, которую мы используем, на выходе работает с 10 признаками, поэтому нужно преобразовать список с ответами в массив из векторов с 10-ю параметрами.

yvtrain = convert\_y\_to\_vect(ytrain)

Это необходимо для вычисления выходной дельты

В функцию обучения поступают структура сети, массивы с признаками и с проверочными значениями и списки с весами. В самой функции задаются число итераций и константа альфа. Возвращает функция измененный список весов и функцию средней стоимости.

W, b, avg\_cost\_func = train\_nn(nn\_structure, ITrain, yvtrain, W, b)

На каждой эпохе мы рисуем график изменения функции средней стоимости.

plt.plot(avg\_cost\_func)  
plt.ylabel('Average J')  
plt.xlabel('Iteration number')  
plt.show()

После прохождения 10 эпох мы вычисляем % точности на тестовых данных.

y\_pred = predict\_y(W, b, ITest, 3)  
print(' - Prediction accuracy is {}%'.format(accuracy\_score(ytest, y\_pred) \* 100))

**Изменение параметров:**

Для 10 эпох нам вполне хватит 600 итераций. Если у нас одна эпоха, и сеть сразу работает со всеми данными без выборки, число итераций должно составлять где-то 1000-1400, далее % точности не меняется.

От константы альфа зависит то, на сколько изменятся веса. По графику функции средней стоимости видно, что чем меньше альфа, тем плавнее сходимость функции к нулю. Если поставить слишком маленькую константу, то будет необходимо провести довольно большое число итераций.

За функцию активации у нас отвечает гиперболический тангенс. Он выдает тот же % правильности ответа, что и сигмоида, но для нее необходимо в три раза больше итераций.

Было установлено, что слой должен быть только один, иначе % точности становится случайным, то есть с каждым прогоном его точность может быть от 0 до 90 %.

**Заключение:**

Нам удалось добиться более-менее стабильных 95-98% точности за 5 минут времени выполнения программы, а ответ может допустимо колебаться с в пределах +-2%. Так же было замечено, что 95 и 97 % программа выдает реже, чем 97 и 96.