Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Информационная безопасность (ИУ8)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №6 на тему:

«Изучение алгоритма обратного распространения ошибки (метод Back Propagation)»

Вариант 8

Выполнил: Песоцкий А. А.,
студент группы ИУ8-61

Проверила: Коннова Н.С.,
доцент каф. ИУ8

Москва 2020 г.

Цель работы

Исследовать функционирование многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки (англ. Back Propagation — BP).

Постановка задачи

На примере МНС архитектуры N – J – M (рис. 1) реализовать её обучение методом BP, проведя настройку весов нейронов скрытого ($w_{ij}^{(1)}(k),\ i = \overline{0,N}$, $j = \overline{1,J}$) и выходного ($w_{jm}^{(2)}(k),\ j = \overline{0,J}$, $j = \overline{1,M}$) слоёв, где индексы $i,j = 0$ соответствуют нейронам смещения; $k = 1,2, \ldots$ — номер эпохи обучения.

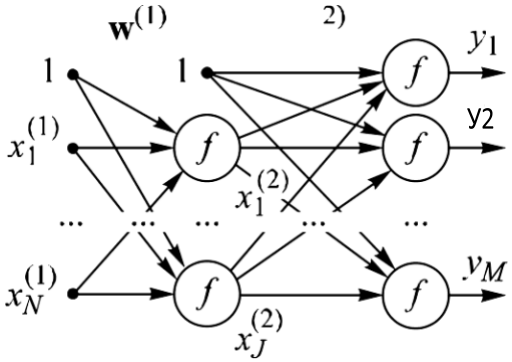

Рисунок 1. Многослойная НС

Алгоритм обратного распространения ошибки. Обозначения:

$x_i^{(1)}(k), x_j^{(2)}(k)$ — входные сигналы нейронов скрытого и выходного слоёв;

$net_j^{(1)}(k), net_m^{(2)}(k)$ — комбинированные входы нейронов скрытого и входного слоёв;

$out_j^{(1)}(k), out_m^{(2)}(k)$ — выходные сигналы нейронов скрытого и выходного слоёв;

$\delta_j^{(1)}(k), \delta_m^{(2)}(k)$ — ошибки скрытого и выходного слоёв.

Начальные веса $w_{ij}^{(1)}(0), w_{j_ь}^{(12)}(0)$ принять произвольными.

Функция активации нейронов скрытого и выходного слоев

$$f(net) = \frac{1 - \exp(-net)}{1 + \exp(-net)} \qquad (-1,1).$$

Ее производная выражается через значения самой функции как

$$\frac{df(net)}{dnet} = \frac{1}{2}[1 - f^2(net)].$$

На *первом этапе* следует рассчитать по заданному входному сигналу x_i, $i = \overline{0,N}$ выход МНС $y_m(k)$:

1) $x_i^{(1)} \equiv x_i\,, i = \overline{0,N}\,;$

2) $net_j^{(1)}(k) = \sum_{i=1}^{N} w_{ij}^{(1)}(k)x_i^{(1)} + w_{0j}^{(1)},\ j = \overline{1,J};$

3) $x_j^{(2)}(k) \equiv out_j^{(1)}(k) = f\left[net_j^{(1)}(k)\right],\ j = \overline{1,J};$

4) $net_m^{(2)}(k) = \sum_{j=1}^{J} w_{jm}^{(2)}(k)x_j^{(3)} + w_{0m}^{(3)},\ m = \overline{1,M};$

5) $y_m(k) \equiv out_m^{(2)}(k) = f\left[net_m^{(2)}(k)\right],\ m = \overline{1,M}.$

На втором этапе по известному желаемому выходу tm оценивают ошибки выходного и скрытого слоев (обратное распространение ошибки):

1) $\delta_m(k) \equiv\ \delta_m^{(2)}(k) = \frac{df\left[net_m^{(2)}(k)\right]}{dnet_m^{(2)}(k)}[t_m - y_m(k)]\,,\ m = \overline{1,M};$

2) $\delta_j^{(1)}(k) = \frac{df\left[net_j^{(1)}(k)\right]}{dnet_j^{(1)}(k)}\sum_{m=1}^{M} w_{jm}^{(2)}(k)\delta_m(k),\ j = \overline{1,J}.$

После этого на третьем этапе производят настройку весов:

1) $w_{ij}^{(1)}(k+1) = w_{ij}^{(1)}(k) + \Delta w_{ij}^{(1)}(k),\ \Delta w_{ij}^{(1)}(k) = \eta x_i \delta_j^{(1)}(k);$

2) $w_{jm}^{(2)}(k+1) = w_{jm}^{(2)}(k) + \Delta w_{jm}^{(2)}(k),\ \Delta w_{jm}^{(2)}(k) = \eta x_j^{(2)}(k)\delta_m(k);$

Норму обучения следует принять $\eta \in (0,1]$.

Затем по тому же входному сигналу x_i выполняется расчёт нового выходного вектора $y_i(k+1)$ и далее аналогично вплоть до достижения эпохи $k = K$, на которой суммарная среднеквадратичная ошибка не превысит некоторого порога $0 < \varepsilon < 1$:

$$E(k) = \sqrt{\sum_{j=1}^{M}\left[t_j - y_j(k)\right]^2} \le \varepsilon.$$

Ход работы

МНС архитектура	1 – 1 – 3 (N – 1, J – 1, M – 3)
Входной вектор	$\mathbf{x} = (\mathbf{1}, -\mathbf{2})$
Целевой вектор	$\mathbf{10t} = (\mathbf{2}, \mathbf{1}, \mathbf{3})$, t = (0.2, 0.1, 0.3)
Погрешность	$1 \cdot 10^{-4}$
Норма обучения	$\boldsymbol{\eta} = \mathbf{1}$

Начальные веса возьмём нулевыми:

$$w_{ij}^{(1)}(0) = 0, \qquad i = \overline{0, N}, \qquad j = \overline{1, J};$$

$$w_{jm}^{(2)}(0) = 0, \qquad j = \overline{0, J}, \qquad m = \overline{1, M}.$$

Параметры НС на последовательных эпохах:

Epoch	Weights (hidden)	Weights (external)	Y	E
0	[[0.0, 0.0]]	[[0.0, 0.0], [0.0, 0.0], [0.0, 0.0]]	[0.0, 0.0, 0.0]	0.37417
1	[[0.0, 0.0]]	[[0.1, 0.0], [0.05, 0.0], [0.15, 0.0]]	[0.049958, 0.024995, 0.07486]	0.28076
2	[[0.013074, 0.0]]	[[0.17483, 0.0], [0.087479, 0.0], [0.26194, 0.0]]	[0.087195, 0.043712, 0.13023]	0.21146
3	[[0.030124, 0.0]]	[[0.23081, 0.0], [0.11557, 0.0], [0.34539, 0.0]]	[0.11489, 0.057721, 0.171]	0.16023
4	[[0.046997, 0.0]]	[[0.2728, 0.0], [0.13664, 0.0], [0.408, 0.0]]	[0.13556, 0.068213, 0.20122]	0.12215
5	[[0.062051, 0.0]]	[[0.30443, 0.0], [0.15246, 0.0], [0.45539, 0.0]]	[0.15105, 0.076082, 0.22384]	0.09364
6	[[0.074822, 0.0]]	[[0.32834, 0.0], [0.16435, 0.0], [0.49156, 0.0]]	[0.16271, 0.08199, 0.24095]	0.07212
7	[[0.085358, 0.0]]	[[0.34649, 0.0], [0.17329, 0.0], [0.51938, 0.0]]	[0.17153, 0.08643, 0.254]	0.05577
8	[[0.093906, 0.0]]	[[0.36031, 0.0], [0.18003, 0.0], [0.54089, 0.0]]	[0.17823, 0.089771, 0.26404]	0.04326
9	[[0.10077, 0.0]]	[[0.37085, 0.0], [0.1851, 0.0], [0.55762, 0.0]]	[0.18333, 0.092287, 0.2718]	0.03365
10	[[0.10624, 0.0]]	[[0.3789, 0.0], [0.18892, 0.0], [0.57067, 0.0]]	[0.18722, 0.094182, 0.27784]	0.02624
11	[[0.11059, 0.0]]	[[0.38507, 0.0], [0.19181, 0.0], [0.5809, 0.0]]	[0.19019, 0.095611, 0.28255]	0.02049
12	[[0.11403, 0.0]]	[[0.3898, 0.0], [0.19398, 0.0], [0.58893, 0.0]]	[0.19247, 0.096688, 0.28624]	0.01603
13	[[0.11675, 0.0]]	[[0.39342, 0.0], [0.19562, 0.0], [0.59525, 0.0]]	[0.19421, 0.0975, 0.28914]	0.01256
14	[[0.11889, 0.0]]	[[0.39621, 0.0], [0.19686, 0.0], [0.60022, 0.0]]	[0.19555, 0.098113, 0.29142]	0.00985
15	[[0.12058, 0.0]]	[[0.39835, 0.0], [0.19779, 0.0], [0.60415, 0.0]]	[0.19658, 0.098576, 0.29321]	0.00773
16	[[0.12191, 0.0]]	[[0.39999, 0.0], [0.1985, 0.0], [0.60725, 0.0]]	[0.19737, 0.098925, 0.29463]	0.00608
17	[[0.12295, 0.0]]	[[0.40125, 0.0], [0.19903, 0.0], [0.60971, 0.0]]	[0.19798, 0.099189, 0.29575]	0.00478
18	[[0.12378, 0.0]]	[[0.40223, 0.0], [0.19943, 0.0], [0.61165, 0.0]]	[0.19844, 0.099387, 0.29663]	0.00376
19	[[0.12443, 0.0]]	[[0.40297, 0.0], [0.19974, 0.0], [0.61318, 0.0]]	[0.1988, 0.099538, 0.29733]	0.00296
20	[[0.12493, 0.0]]	[[0.40355, 0.0], [0.19997, 0.0], [0.6144, 0.0]]	[0.19908, 0.099651, 0.29789]	0.00233
21	[[0.12534, 0.0]]	[[0.40399, 0.0], [0.20014, 0.0], [0.61536, 0.0]]	[0.19929, 0.099736, 0.29833]	0.00184
22	[[0.12565, 0.0]]	[[0.40433, 0.0], [0.20027, 0.0], [0.61612, 0.0]]	[0.19945, 0.099801, 0.29867]	0.00145
23	[[0.1259, 0.0]]	[[0.40459, 0.0], [0.20037, 0.0], [0.61673, 0.0]]	[0.19958, 0.09985, 0.29895]	0.00114
24	[[0.12609, 0.0]]	[[0.40479, 0.0], [0.20044, 0.0], [0.61721, 0.0]]	[0.19968, 0.099887, 0.29917]	0.0009
25	[[0.12625, 0.0]]	[[0.40495, 0.0], [0.2005, 0.0], [0.61759, 0.0]]	[0.19975, 0.099914, 0.29934]	0.00071
26	[[0.12637, 0.0]]	[[0.40507, 0.0], [0.20054, 0.0], [0.61789, 0.0]]	[0.19981, 0.099935, 0.29948]	0.00056
27	[[0.12646, 0.0]]	[[0.40516, 0.0], [0.20057, 0.0], [0.61813, 0.0]]	[0.19985, 0.099951, 0.29958]	0.00044
28	[[0.12654, 0.0]]	[[0.40523, 0.0], [0.2006, 0.0], [0.61832, 0.0]]	[0.19989, 0.099963, 0.29967]	0.00035
29	[[0.1266, 0.0]]	[[0.40528, 0.0], [0.20061, 0.0], [0.61847, 0.0]]	[0.19991, 0.099972, 0.29974]	0.00028
30	[[0.12664, 0.0]]	[[0.40533, 0.0], [0.20063, 0.0], [0.61858, 0.0]]	[0.19993, 0.099979, 0.29979]	0.00022
31	[[0.12668, 0.0]]	[[0.40536, 0.0], [0.20064, 0.0], [0.61868, 0.0]]	[0.19995, 0.099984, 0.29984]	0.00017
32	[[0.12671, 0.0]]	[[0.40538, 0.0], [0.20065, 0.0], [0.61875, 0.0]]	[0.19996, 0.099988, 0.29987]	0.00014
33	[[0.12673, 0.0]]	[[0.4054, 0.0], [0.20065, 0.0], [0.61881, 0.0]]	[0.19997, 0.099991, 0.2999]	0.00011
34	[[0.12675, 0.0]]	[[0.40542, 0.0], [0.20066, 0.0], [0.61886, 0.0]]	[0.19998, 0.099993, 0.29992]	9e-05

Выходной Y = [0.19998, 0.099993, 0.29992]
Ошибка E = 9e-05

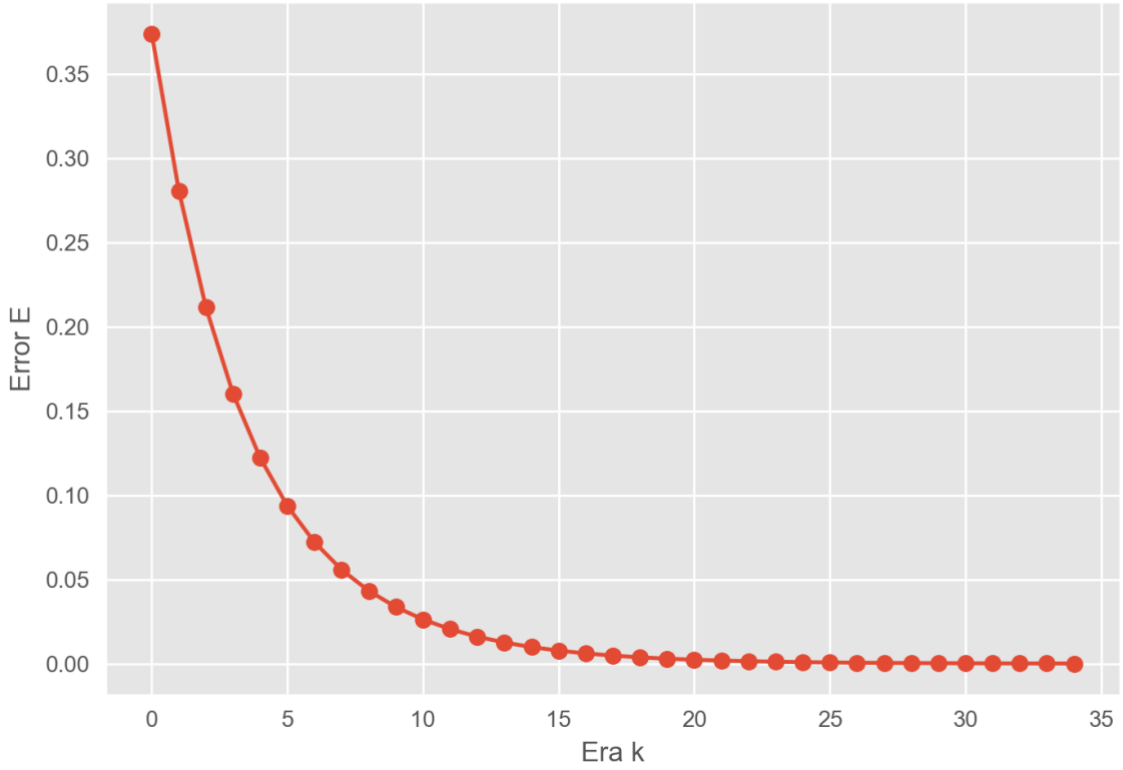

Рисунок 2. График зависимости квадратичной ошибки от эпохи обучения

Выводы

В ходе выполнения работы было изучено функционирование МНС с архитектурой N – J – M, и было выполнено её обучение согласно алгоритму обратного распространения ошибки (метод Back Propagation).

Основная идея этого алгоритма состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы.

Несмотря на многочисленные успешные применения обратного распространения, оно не является универсальным решением. Больше всего неприятностей приносит неопределённо долгий процесс обучения.

Приложение А.

Файл ‘mnn.py’.

```
from computation import *
from prettytable import PrettyTable

'''
Функция обучения
:param N: параметр N
:param J: параметр J
:param M: параметр M
:param vector_x: входной вектор
:param vector_t: целевой вектор
:param eps: погрешность
'''
def learn_mnn(N, J, M, vector_x, vector_t, eps):

    #генерируем начальные веса скрытых нейронов
    hnw = list()    # hidden neuron weights
    for i in range(0, J):
        hnw.append([0.0] * (N + 1))
    #генерируем начальные веса внешних нейронов
    enw = list()    # external neuron weights
    for i in range(0, M):
        enw.append([0.0] * (J + 1))

    epoch_list = list()
    error_list = list()
    epoch = 0
    epoch_list.append(epoch)

    y = [0.0] * (M)
    error = count_error(y, vector_t)
    error_list.append(error)

    pt = PrettyTable(['Epoch','Weights (hidden)', 'Weights (external)', 'Y', 'E'])
    pt.add_row([epoch,
                [[float('{:.5}'.format(hw)) for hw in hnw[i]] for i in range(len(hnw))],
                [[float('{:.5}'.format(ew)) for ew in enw[j]] for j in range(len(enw))],
                [float('{:.5}'.format(y)) for y in y], round(error, 5)])

    while error > eps:
```

```
        #ПЕРВЫЙ ЭТАП (1)
        #получаем значения нейронов скрытого и внешнего слоёв
        h_nets, h_outs, e_nets, e_outs \
            = get_neuron_values(hnw, enw, vector_x)

        #ВТОРОЙ ЭТАП (2)
        #внешние дельта
        e_deltas = get_external_deltas(e_nets, e_outs, vector_t)
        #скрытые дельта
        h_deltas = get_hidden_deltas(J, enw, e_deltas, h_nets)

        #ТРЕТИЙ ЭТАП (3)
        #пересчёт скрытых весов
        hnw = update_hidden_weights(hnw, J, N, h_deltas, vector_x)
        #пересчёт внешних весов
        enw = update_external_weights(enw, M, J, e_deltas, h_outs)

        #пересчитыаем Y
        h_nets, h_outs, e_nets, \
        y = get_neuron_values(hnw, enw, vector_x)

        error = count_error(y, vector_t)
        epoch += 1
        epoch_list.append(epoch)
        error_list.append(error)

        pt.add_row([epoch,
                    [[float('{:.5}'.format(hw)) for hw in hnw[i]] for i in range(len(hnw))],
                    [[float('{:.5}'.format(ew)) for ew in enw[j]] for j in range(len(enw))],
                    [float('{:.5}'.format(y)) for y in y], round(error, 5)])

    graph_plot(epoch_list, error_list)
    return pt


if __name__ == "__main__":
    X = [1, -2]
    T = [0.2, 0.1, 0.3]
    test = learn_mnn(1, 1, 3, X, T, 0.0001)
    print(test)
```

Файл ‘computation.py’.

```
from math import exp, sqrt
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.style as style

nu = 1

'''
Функция подсчитывает net
:param vector_x: входной вектор
:param vector_w: вектор весов
:param return: значение net
'''
def get_net(vector_x, vector_w):
    net = 0
    for i in range(1, len(vector_w)):
        net += vector_x[i] * vector_w[i]
    net += vector_w[0]
    return net


'''
Функция подсчитывает f(net)
:param net: net
:param return: значение f(net)
'''
def function(net):
    return (1 - exp(-net)) / (1 + exp(-net))


'''
Функция подсчитывает производную функции
:param net: net
:param return: значение производной
'''
def derivative(net):
    return 0.5 * (1 - function(net)**2)
```

```
'''
Функция рассчитывает дельты внешних нейронов
:param e_nets: net'ы внешних нейронов
:param e_outs: выходы внешних нейронов
:param vector_t: целевой вектор
:param return: набор дельт
'''
def get_external_deltas(e_nets, e_outs, vector_t):
    #внешние дельта
    external_deltas = list()
    for (net, out, t) in zip(e_nets, e_outs, vector_t):
        external_deltas.append(derivative(net) * (t - out))  #считаем внешние delta
    return external_deltas


'''
Функция рассчитывает дельты скрытых нейронов
:param J: параметр J
:param enw: веса внешних нейрнонов
:param e_deltas: дельты внешних нейронов
:param h_nets: net'ы скрытых нейронов
:param return: набор дельт
'''
def get_hidden_deltas(J, enw, e_deltas, h_nets):
    #вычисление подсумм
    sums = list()
    for j in range(0, J):
        temp = 0
        for (weight_m, delta) in zip(enw, e_deltas):
            temp += weight_m[j] * delta
        sums.append(temp)

    #скрытые дельта
    hidden_deltas = list()
    for (net, s) in zip(h_nets, sums):
        hidden_deltas.append(derivative(net) * s)  #считаем скрытые delta
    return hidden_deltas


'''
Функция расчёта net и выходов скрытых и внешних нейронов
:param hnw: веса скрытых нейрнонов
```

```
:param enw: веса внешних нейрнонов
:param vector_x: входной вектор
:param return: net'ы и выходы
'''
def get_neuron_values(hnw, enw, vector_x):

    #скрытые выходы
    hidden_nets = list()  #net'ы скрытого слоя
    hidden_outs = list()  #выходы скрытого слоя
    hidden_outs.append(1)  #добавляем х смещения
    for hidden_weights in hnw:  #для каждого набора весов
        net = get_net(vector_x, hidden_weights)  #считаем net
        hidden_nets.append(net)
        hidden_outs.append(function(net))  #считаем выход скрытого слоя

    #внешние выходы
    external_nets = list()  #net'ы внешнего слоя
    external_outs = list()  #выходы внешнего слоя
    for external_weights in enw:  #для каждого набора весов
        net = get_net(hidden_outs, external_weights)  #считаем net
        external_nets.append(net)
        external_outs.append(function(net))  #считаем выход внешнего слоя

    return hidden_nets, hidden_outs, external_nets, external_outs


'''
Функция обновления весов скрытых нейронов
:param hnw: веса скрытых нейрнонов
:param J: параметр J
:param N: параметр N
:param h_deltas: дельты скрытых нейронов
:param vector_x: входной вектор
:param return: обновлённые веса
'''
def update_hidden_weights(hnw, J, N, h_deltas, vector_x):
    #пересчёт скрытых весов
    for j in range(0, J):
        for i in range(0, N):
            hnw[j][i] += nu * vector_x[i] * h_deltas[j]
    return hnw
```

```
'''
Функция обновления весов внешних нейронов
:param enw: веса внешних нейрнонов
:param M: параметр M
:param J: параметр J
:param e_deltas: дельты внешних нейронов
:param h_outs: выходы скрытых нейронов
:param return: обновлённые веса
'''
def update_external_weights(enw, M, J, e_deltas, h_outs):
    #пересчёт внешних весов
    for m in range(0, M):
        for j in range(0, J):
            enw[m][j] += nu * h_outs[j] * e_deltas[m]
    return enw


'''
Функция подсчёта квадратичной ошибки
:param y: вектор y
:param vector_t: целевой вектор
:param return: квадратичная ошибка
'''
def count_error(y, vector_t):
    error = 0
    for (out, t) in zip(y, vector_t):
        error += (t - out) ** 2
    error = sqrt(error)
    return error


'''
Функция строит график E(k)
:param epoch_list: список эпох
:param error_list: список ошибок
'''
def graph_plot(epoch_list, error_list):
    style.use('seaborn')
    style.use('ggplot')
    plt.grid(True)
    plt.plot(epoch_list, error_list)
```

```
plt.xlabel('Era k')
plt.ylabel('Error E')
plt.scatter(epoch_list, error_list)
mpl.style.use('bmh')
plt.show()
```