Федеральное Государственное бюджетное учреждение

высшего профессионального образования

"Заполярный Государственный Университет имени Н. М. Федоровского"

Кафедра «Информационные системы и технологии»

Нейронные сети

Отчет по лабораторной работе №2

Тема: Сеть обратного распространения ошибки

Выполнил:

студент группы ИС-21

Сидельников Максим Эдуардович

Проверила:

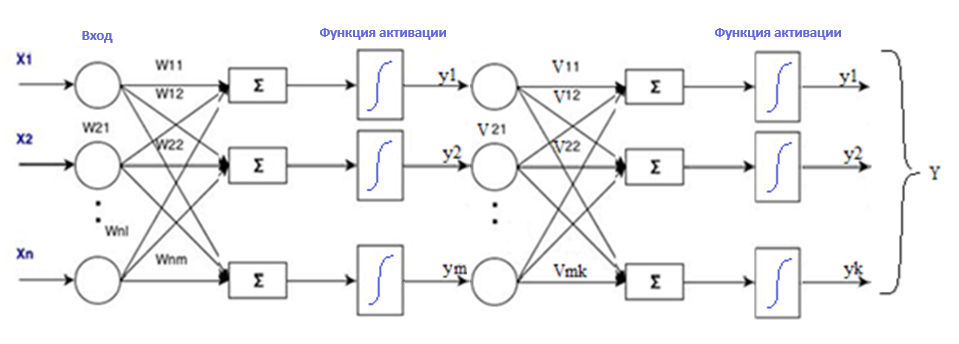
Дыптан Елена Арнольдовна

Норильск 2024

**Цель:** сформировать обучающую и тестовые выборки, реализовать алгоритм обучения и функционирования «Сети обратного распространения ошибки», обучить и протестировать сеть.

**Архитектура сети**

Архитектура сети представляет собой многослойную нейронную сеть с одним скрытым слоем следующего вида:

****

Где Xn: 300 (количество входов)

Yn: 10 (количество выходов)

Активационная функция каждого нейрона Yn – логистическая (сигмоидальная) и имеет вид:

Yn = 1/(1+e^(-as) )

где Sn = ∑(xi \* wni)

a=1 (наклон)

На входе нейронная сеть получает вектор X = [X0,X1,…X300]

На выходе нейронная сеть возвращает вектор Y = [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Y9,]

Ошибка считается по алгоритму вычисления функции потерь с использованием кросс-энтропии:

1. Определение epsilon = 1e-10.
2. Ограничение значения y\_pred (в данном случае изменения не произойдет, так как все значения уже в пределах допустимого интервала).
3. Нахождение индексов истинных классов: [0, 1].
4. Выбор предсказанных вероятностей для истинных классов: [0.7, 0.6].
5. Вычисление отрицательных логарифмов: [-log(0.7), -log(0.6)] ≈ [0.357, 0.511].
6. Нахождение среднего значения этих логарифмов: (0.357 + 0.511) / 2 ≈ 0.434.

Алгоритм обучения

1. Инициализировать весовую матрицу W и V небольшими случайными значениями от -0.3 до 0.3. Задать начальное значения a, и max кол-во эпох обучения, а также коэффициент точности Ꜫ
2. Вычисления текущего выходного сигнала, использовали функцию активации типа сигмоид, как на скрыто, так и на внешнем слоях
3. Настройка синаптических весов по формулам:
4. Для выходного слоя:

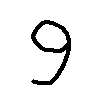
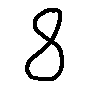
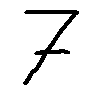
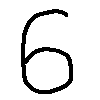
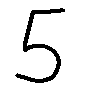
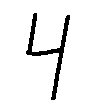
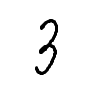
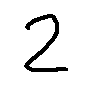
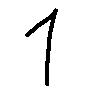
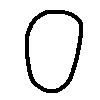
V\_jk^N=V\_jk^N-α ∂E/(∂V\_jk^N ); ∂E/(∂V\_jk^N )=(y\_k-y)\*z\_j

1. Для скрытого слоя:

W\_ij^N=W\_ij^N-α ∂E/(∂W\_ij^N ); ∂E/(∂W\_ij^N )=(y\_k-y)\*z\_j\*x\_i

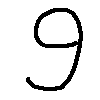
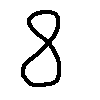
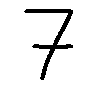
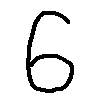
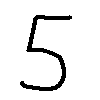
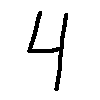
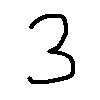
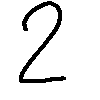
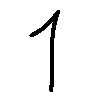
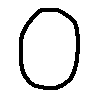
1. Шаги 2- 3 повторяются для всех обучающих векторов. Обучение заканчивается, когда для каждого образа обучающие выборки в течении 1 эпохи ошибка не превышает Ꜫ или когда превышено max кол-во эпох.

**Обучающая выборка**

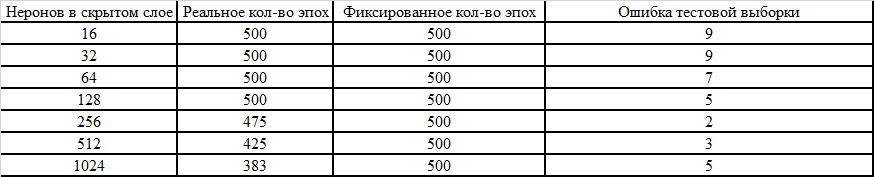
****

Размер обучающей выборки:

**Тестовая выборка**



Размер тестовой выборки:

****

**Вывод:** была разработана и протестирована сеть обратного распространения ошибки, состоящая из одного скрытого и одного внешнего слоя. В ходе работы была реализована архитектура сети, определены параметры активационных функций и алгоритмы вычисления ошибок с использованием кросс-энтропии. Проведено обучение сети на выборке из 300 образцов и тестирование на 100 образцах. Настроены весовые коэффициенты для уменьшения ошибки по обучающим данным. Сеть успешно обучена до достижения заданного уровня точности или максимального количества эпох.

**Листинг программы:**

def \_\_init\_\_(self, folder):

self.folder = folder # Папка с изображениями

self.learning\_rate = 0.01 # Скорость обучения

self.input\_size = 10000 # Размер входного слоя (100x100 изображение -> 10000 пикселей)

self.hidden\_size = 256 # Размер скрытого слоя

self.num\_classes = 10 # Количество классов (цифр 0-9)

# Инициализация весов и смещений случайными значениями

self.W1 = np.random.randn(self.input\_size, self.hidden\_size) \* 0.01

self.b1 = np.zeros((1, self.hidden\_size))

self.W2 = np.random.randn(self.hidden\_size, self.num\_classes) \* 0.01

self.b2 = np.zeros((1, self.num\_classes))

# Загрузка изображений и меток

self.images, self.labels = self.load\_images(folder=self.folder)

# Обучение модели

self.train(self.images, self.labels)

def load\_images(self, folder=None):

if folder is None:

folder = self.folder

images = []

labels = []

for filename in os.listdir(folder):

img\_path = os.path.join(folder, filename)

try:

with Image.open(img\_path) as img:

img = img.convert("L")

img = img.resize((100, 100))

images.append(np.asarray(img).flatten() / 255.0)

label = int(filename[0])

labels.append(label)

except Exception as e:

print(f"Ошибка загрузки {img\_path}: {e}")

return np.array(images), np.array(labels)

def forward(self, X):

Z1 = np.dot(X, self.W1) + self.b1

A1 = np.maximum(0, Z1)

Z2 = np.dot(A1, self.W2) + self.b2

A2 = self.softmax(Z2)

return Z1, A1, Z2, A2

def softmax(self, x):

e\_x = np.exp(x - np.max(x))

return e\_x / e\_x.sum(axis=1, keepdims=True)

def backward(self, X, Y, Z1, A1, Z2, A2):

m = Y.shape[0]

dZ2 = A2 - Y

dW2 = np.dot(A1.T, dZ2) / m

db2 = np.sum(dZ2, axis=0, keepdims=True) / m

dA1 = np.dot(dZ2, self.W2.T)

dZ1 = dA1 \* (Z1 > 0)

dW1 = np.dot(X.T, dZ1) / m

db1 = np.sum(dZ1, axis=0, keepdims=True) / m

return dW1, db1, dW2, db2

def update\_parameters(self, dW1, db1, dW2, db2):

self.W1 -= self.learning\_rate \* dW1

self.b1 -= self.learning\_rate \* db1

self.W2 -= self.learning\_rate \* dW2

self.b2 -= self.learning\_rate \* db2

def guess(self, image):

\_, \_, \_, A2 = self.forward(image)

return A2, np.argmax(A2, axis=1)[0]

def train(self, images, labels):

Y = np.eye(self.num\_classes)[labels]

epoch = 0

max\_epochs = 500

while epoch < max\_epochs:

Z1, A1, Z2, A2 = self.forward(images)

loss = self.cross\_entropy\_loss(Y, A2)

dW1, db1, dW2, db2 = self.backward(images, Y, Z1, A1, Z2, A2)

self.update\_parameters(dW1, db1, dW2, db2)

print(f"Эпоха {epoch}, потери: {loss}")

if loss < 0.1:

break

epoch += 1

print(f"Обучение завершено после {epoch} эпох")

def cross\_entropy\_loss(self, y\_true, y\_pred):

epsilon = 1e-10

y\_pred = np.clip(y\_pred, epsilon, 1. - epsilon)

n\_samples = y\_true.shape[0]

log\_p = -np.log(y\_pred[np.arange(n\_samples), y\_true.argmax(axis=1)])

loss = np.sum(log\_p) / n\_samples

return loss