МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

Кафедра прикладной математики

Направление подготовки: Прикладная математика и информатика

Магистерская программа: Системное программирование

Отчет по лабораторной работе
«Реализация метода обратного распространения ошибки для двухслойной полностью связанной нейронной сети»

Выполнил:

студент группы 381603м4

Семичев Ю.А.

Нижний Новгород

2017

Содержание

Постановка задачи

Целью данной работы было изучение метода обратного распространения ошибки для обучения многослойных глубоких нейронный сетей на примере двухслойной (то есть с одним скрытым слоем) полностью связанной сети. В качестве тестовой задачи использовалась задача распознавания цифр по изображениям из набора данных MNIST.

Задачи:

- Изучить общую схему метода обратного распространения ошибки
- Вывести математические формулы для вычисления градиентов функции ошибки по параметрам нейронной сети и формул коррекций весов
- Проектирование и разработка программной реализации метода
- Тестирование программной реализации

Задача обучения нейронной сети

Пусть $y = f(x), x \in R^N, y \in R^m$ – некоторая неизвестная функций и нам известен набор её значений для некоторых аргументов. Пусть такой набор имеет размер L. Тогда все аргументы функции можно построчно сложить в матрицу X, а все её значения в матрицу Y.

Обучить нейронную сеть – значит подобрать такие веса всех её нейронов, что при подаче ей на вход строчек матрицы X будут получаться выходы, наиболее близкие к строчкам матрицы Y. Также ожидается, что при подаче ей на вход любых других аргументов x, на выходе будут вектора, близкие к истинному значению $f(x)$. Функция, являющаяся мерой близости между выходом нейронной сети и ожидаемым значением называется ***функцией ошибки*** $E(w)$ (считаем, что X и Y фиксированы и таким образом, ошибка зависит только от параметров сети). Таким образом, ***задача обучения нейронной сети*** выглядит так:

$$E(w) \to \min_w$$

Для решения ***задачи регрессии*** (то есть когда $f(x)$ – непрерывная функция), обычно использую ***евклидову ошибку***:

$$E(w) = \frac{1}{2}\sum_{k=1}^{L}\sum_{j=1}^{M}\left(y_j^k - u_j^k\right)^2$$

Здесь y^k – k-ая строчка матрицы Y, а u^k – выход сети для входа x^k – k-ой строчки матрицы X.

Для решения ***задачи классификации*** (то есть когда $f(x)$ принимает конечный набор из K значений, называемые ***классами***) матрица Y (который в этом случае имеет вид вектора-столбца длины L) преобразуется в бинарную матрицу $L \times K$, где в каждой строке стоит только одна 1 по индексу, равному номеру класса для соответствующей строчки матрицы X. После этого в качестве функции ошибки используется функция, которая называется ***кросс-энтропией***:

$$E(w) = -\sum_{k=1}^{L}\sum_{j=1}^{M} y_j^k \ln\left(u_j^k\right)$$

Метод обратного распространения ошибки

Метод обратного распространения ошибки является градиентным методом минимизации функции ошибки для полностью связанной нейронной сети. То есть веса нейронной сети меняются по закону:

$$w(k+1) = w(k) + \eta p(w)$$

Здесь η – ***скорость обучения***, $p(w)$ – направление сдвига в пространстве параметров сети. В классической реализации метода обратного распространения ошибки:

$$p(w) = -\nabla E(w)$$

Общая схема метода:

1. Прямой проход по сети. Во время него вычисляется выход сети для какого-то входа путём послойного вычисления значений в телах нейронов. Также, вычисляются производные от функций активации нейронов для вычисленных значений.
2. Вычисление функции ошибки и её производных по весам выходного слоя.
3. Обратный проход по сети. Во время него считаются градиенты функции ошибки на слоях и корректируются веса

Шаги 1-3 повторяются до выполнения ***критерия остановки*** (по числу итераций или по достигнутой точности)

Теперь распишем необходимые действия более подробно для сети с одним скрытым слоем и для евклидовой функции ошибки. Распишем функцию ошибки для одного примера следующим образом:

$$E(w) = \frac{1}{2}\sum_{j=1}^{M}\left(y_j - u_j\right)^2 = \frac{1}{2}\sum_{j=1}^{M}\left(y_j - \varphi^{(2)}\left(\sum_{s=0}^{K} w_{js}^{(2)}\varphi^{(1)}\left(\sum_{i=0}^{N} w_{si}^{(1)} x_i\right)\right)\right)^2$$

Здесь $w_{ij}^{(k)}$ – вес, связывающий i нейрон $(k-1)$-го слоя с j-ым нейроном k-го слоя, $\varphi^{(k)}$ – функция активации всех нейронов k-го слоя. Отсюда:

$$\frac{\partial E}{\partial w_{js}^{(2)}} = \left(y_j - u_j\right)\frac{\partial u_j}{\partial w_{js}^{(2)}} = \left[v_s = \varphi^{(1)}\left(\sum_{i=0}^{N} w_{si}^{(1)} x_i\right)\right] = \left[g_i = \sum_{s=0}^{K} w_{js}^{(2)} v_s\right] =$$

$$= \left(y_j - u_j\right)\frac{d\varphi^{(2)}(g_i)}{dg_i}\frac{\partial g_i}{\partial w_{js}^{(2)}} = \left(y_j - u_j\right)\frac{d\varphi^{(2)}(g_i)}{dg_i} v_s = \frac{\partial E}{\partial g_i} v_s = \delta_j^{(2)} v_s$$

$$\frac{\partial E}{\partial w_{si}^{(1)}} = \sum_{j=1}^{M} (y_j - u_j) \frac{d\varphi^{(2)}(g_j)}{dg_j} \frac{dg_j(v_s)}{dv_s} \frac{d\varphi^{(1)}(f_s)}{df_s} x_i \, , f_s = \sum_{i=0}^{N} w_{si}^{(1)} x_i$$

$$\frac{\partial E}{\partial w_{si}^{(1)}} = \sum_{j=1}^{M} (y_j - u_j) \frac{d\varphi^{(2)}(g_j)}{dg_j} w_{js}^{(2)} \frac{d\varphi^{(1)}(f_s)}{df_s} x_i = \delta_s^{(1)} x_i, \delta_s^{(1)} = \frac{\partial E(w)}{\partial f_s}$$

Заметим, что

$$\delta_s^{(1)} = \sum_{j=1}^{M} (y_j - u_j) \frac{d\varphi^{(2)}(g_j)}{dg_j} w_{js}^{(2)} \frac{d\varphi^{(1)}(f_s)}{df_s} = \sum_{j=1}^{M} w_{js}^{(2)} \frac{d\varphi^{(1)}(f_s)}{df_s} \delta_j^{(2)}$$

Таким образом, при прямом проходе подсчитываются значения в телах нейронов первого слоя (f_s) и второго слоя (g_s), а при обратном проходе с последнего на первый слой переносится «ошибка» $\delta_i^{(s)}$. Эти формулы обобщаются и на случай многослойной нейронной сети.

Если в качестве активационной функции на всех слоях используется сигоидальная функция (т.е. $\varphi^{(1)}(u) = \varphi^{(2)}(u) = \frac{1}{1+e^{-u}}$), то

$$\frac{\partial E}{\partial w_{js}^{(2)}} = (y_j - u_j) \frac{e^{-g_i}}{(1 + e^{-g_i})^2} v_s = \delta_j^{(2)} v_s,$$

$$\frac{\partial E}{\partial w_{si}^{(1)}} = \sum_{j=1}^{M} (y_j - u_j) \frac{e^{-g_i - f_s}}{(1 + e^{-g_i})^2 (1 + e^{-f_s})^2} w_{js}^{(2)} x_i = \delta_s^{(1)} x_i,$$

Теперь выведем аналогичные формулы для задачи классификации (кросс-энтропия в качестве функции ошибки и softmax в качестве функции активации на последнем слое)

$$E(w) = -\sum_{j=1}^{M} y_j \ln(u_j) = -\sum_{j=1}^{M} y_j \ln\left(\varphi^{(2)}\left(\sum_{s=0}^{K} w_{js}^{(2)} \varphi^{(1)}\left(\sum_{i=0}^{N} w_{si}^{(1)} x_i\right)\right)\right)$$

$$\frac{\partial E}{\partial w_{js}^{(2)}} = -\frac{y_j}{u_j} \frac{\partial u_j}{\partial w_{js}^{(2)}} = -\frac{y_j}{u_j} \frac{d\varphi^{(2)}(g_i)}{dg_i} \frac{\partial g_i}{\partial w_{js}^{(2)}} = -\frac{y_j}{u_j} \frac{d\varphi^{(2)}(g_i)}{dg_i} v_s = \delta_j^{(2)} v_s$$

$$\frac{\partial E}{\partial w_{si}^{(1)}} = -\sum_{j=1}^{M} \frac{y_i}{u_j} \frac{d\varphi^{(2)}(g_j)}{dg_j} \frac{dg_j(v_s)}{dv_s} \frac{d\varphi^{(1)}(f_s)}{df_s} x_i = \delta_s^{(1)} x_i$$

Таким образом, принципиально алгоритм в этом случае не меняется, меняется только формула вычисления «ошибки» на последнем слое $\delta_j^{(2)} = \frac{\partial E}{\partial g_i}$

Если $\varphi^{(1)}(u) = \frac{1}{1+e^{-u}}$ (сигмоидальная функция), а $\varphi^{(2)}(u_i) = \frac{e^{u_i}}{\sum_{k=1}^{M} e^{u_k}}$ (softmax), то:

$$\frac{\partial E}{\partial w_{js}^{(2)}} = -\frac{y_j}{u_j}\frac{e^{g_j}\left(\sum_{k=1}^{M} e^{g_k} - e^{g_j}\right)}{\left(\sum_{k=1}^{M} e^{g_k}\right)^2} v_s = \delta_j^{(2)} v_s$$

$$\frac{\partial E}{\partial w_{si}^{(1)}} = -\sum_{j=1}^{M}\frac{y_i}{u_j}\frac{e^{g_j - f_s}\left(\sum_{k=1}^{M} e^{g_k} - e^{g_j}\right)}{\left(\sum_{k=1}^{M} e^{g_k}\right)^2 (1+e^{-f_s})^2} w_{js}^{(2)} x_i = \delta_s^{(1)} x_i$$

Матричные вычисления в методе обратного распространения ошибки

Пусть $x_i^{(k)}$ – значение на выходе i-го нейрона k-го слоя ($x_i^{(0)} = x_i$), $\mathrm{N_k}$ – число нейронов k-го слоя ($N_0 = N$, $N_S = M$, S – число слоёв). Тогда во время прямого прохода эти значения вычисляются так:

$$x_i^{(k)} = \varphi^{(k)}\left(\sum_{j=1}^{N_{k-1}} w_{ij}^{(k)} x_j^{(k-1)}\right)$$

Пусть теперь $x^{(k)}$ – вектор-строка значений на выходе нейрона k-го слоя, $W^{(k)}$ – матрица весов связей между $(k-1)$-ым и k-ым слоями, а функция $\varphi^{(k)}$ принимает на вход вектор и для каждого его компонента вычисляет скалярную функцию активации. Тогда:

$$x^{(k)} = \varphi^{(k)}\left(W^{(k)} x^{(k-1)}\right)$$

Таким образом, домножения на матрицы весов и взятие векторной функции активации на каждом слое, позволяет провести прямой проход для одного входа. Однако, если X – матрица, в которой по строчкам лежат разные входы, то на с использованием матричных операций можно вычислить все соответствующие им выходы за один проход. Действительно, если $X^{(k)}$ – матрица, в которой по строчкам лежат значения на выходе нейронов при «прогоне» соответствующих строк матрицы X через сеть, то:

$$X^{(k)} = \varphi^{(k)}\left(W^{(k)} X^{(k-1)}\right)$$

Аналогично, к матричным операциям можно свести и обратный проход. Пусть $\delta^{(k)} = \frac{\partial E}{\partial x^{(k)}}$, тогда:

$$\delta_i^{(k)} = \sum_{j=1}^{M} w_{ji}^{(k+1)} \frac{d\varphi^{(k)}\left(x_i^{(k)}\right)}{dx_i^{(k)}} \delta_j^{(k+1)}$$

И поэтому ($*$ - покомпонентное умножение векторов):

$$\delta^{(k)} = W^{(k+1)^T} \delta^{(k+1)} * \frac{d\varphi^{(k)}\left(x^{(k)}\right)}{x^{(k)}}$$

Если $\Delta^{(k)}$ – матрица ошибок для нескольких примеров: $\Delta^{(k)} = W^{(k+1)^T} \Delta^{(k+1)} * \frac{d\varphi^{(k)}\left(X^{(k)}\right)}{X^{(k)}}$

Программная реализация

Был реализован пакетный и последовательный режимы для обучения многослойной нейронной сети, решающей задачу классификации и регрессии. Реализация была произведена на языке ***Python 3.6*** с использованием библиотеки ***numpy***.

Интерфейс класса ***MyMLPClassifier***, решающего задачу обучения нейронной сети и предсказания ответа на тестовых примерах был сделан по аналогии с интерфейсом класса ***NeuralNetwork.MLPClassifier*** библиотеки ***scikit-learn***.

Все параметры нейронной сети передаются в конструктор класса ***MyMLPClassifier*** при создании объекта. Параметры конструктора:

- ***activation*** – строка, название функции активации на всех слоях кроме последнего слоя при решении задачи классификации (возможные значения: ‘logistic’, ‘relu’)
- ***task*** – строка, тип решаемой задачи (возможные значения: ‘classification’, ‘regression’)
- ***hidden_layer_sizes*** – кортеж, содержащий количества нейронов во внутренних слоях. Число внутренних слоёв определяется величиной кортежа.
- ***max_iter*** – максимальное число итераций метода для одного примера обучающей выборки (или для одной пачке в пакетном режиме)
- ***random_state*** – целое число, инициализатор внутреннего генератора случайных чисел, который используется для начальной инициализации весов.
- ***tol* –** float, точность (величина функции ошибки) при достижении которого прекращается обучение сети на текщем примере (пачке)
- ***batch_size*** – размер одной пачки. Должен быть делителем числа примеров в обучающей выборке.
- ***num_epochs*** – число эпох обучения
- ***learn_rate*** – скорость обучения
- ***verbose*** – логическое значение. Если True, то во время обучения выводится дополнительная отладочная информация
- ***show_epoch_progress*** – логическое значение. Если True, то во время обучения показывается прогресс прохождения одной эпохи в %.

После создания объекта класса ***MyMLPClassifier*** можно вызвать у него метод ***fit*** для обучения нейронной сети на примерах из обучающей выборки.

Ей на вход подаются матрица ($np.array$) признаков X и матрица выходов Y (для решения задачи классификации – вектор y с метками классов).

После обучения, можно вызвать метод ***predict***, который принимает матрицу X из тестовой выборки и предсказывает по ней выход Y с помощью обученной сети.

Вспомогательные методы класса:

- ***fit_batch*** – производит обучение сети на одной пачке
- ***calc_predict_error*** – считает ошибку предсказания для обучающей выборки. Нужна, чтобы выводить текущую ошибку предсказания в конце эпохи.

Вспомогаетльные функции:

- ***logistic, relu, softmax*** – функции активации (вместе с их производными)
- ***euclid_error, cross_entropy_error*** – функции ошибок (вместе с их производными
- ***synch_shuffle*** – возвращает случайную перестановку элементов выборки (нужна для перемешивания примеров в начале каждой эпохи)

Тестовый набор данных

Программная реализация метода обратного распространения ошибки тестировалась на задаче распознавания рукописных цифр по ***набору данных MNIST***. Набор данных представляет с собой 70000 одноканальных изображений 28×28 с нарисованными от руки цифрами. Для каждого изображения известно, какая цифра на нём изображена. Изображения изначально разделены на обучающую (60000 примеров) и тестовую (10000 примеров) выборки.

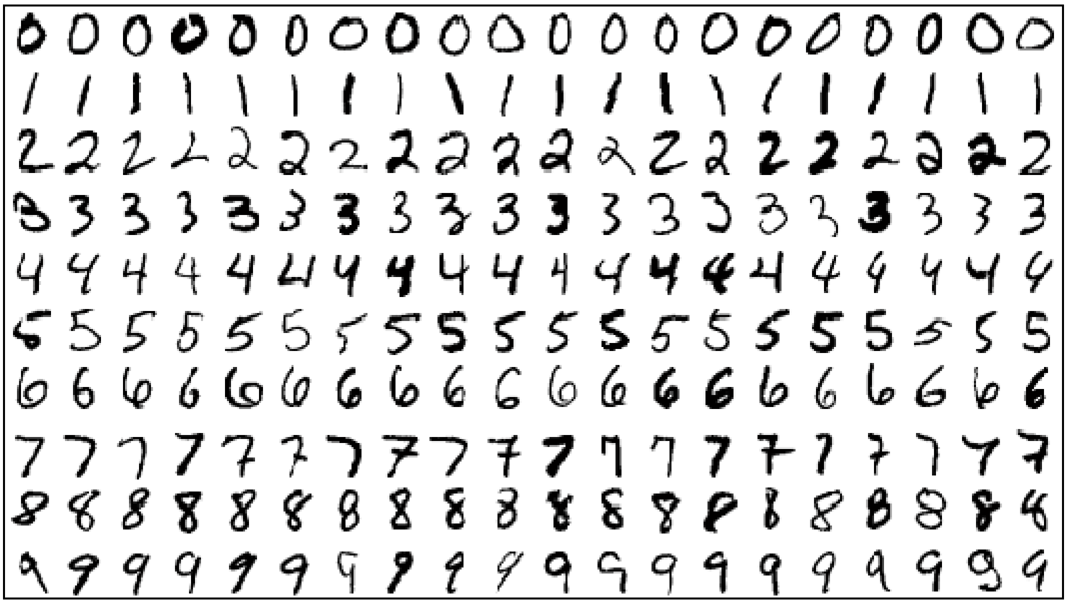

Рис 1. Примеры изображений из набора данных MNIST

Набор данных загружался с помощью отдельного скрипта ***download_mnist.py***

Полученные результаты

Наилучший результат был получен на следующем наборе параметров:

- Сигмоидальная функция активации
- 800 нейронов на одном внутреннем слое
- Последовательный режим обучения (размер пачки – 1)
- Число итераций обучения на одном примере – 1
- Число эпох – 30
- Скорость обучения – 0.01

Точность (отношение числа правильных предсказаний к числу всех предсказаний) на тренировочной выборке составила 0.08875, точность на тестовой выборке – 0.0894