Задание № Б-2.2. Метод обратного распространения ошибки

Введение

В этом упражнении вы реализуете алгоритм обратного распространения ошибки для обучения нейронной сети, которая будет использоваться для распознавания рукописных цифр.

Описание файлов задания:

- ex4.m – главный скрипт для выполнения упражнения,
- ex4data1.mat – обучающая выборка с картинками рукописных цифр,
- ex4weights.mat – значения весов для нейронной сети,
- displayData.m – функция для визуализации обучающей выборки,
- fmincg.m – минимизирующий функционал,
- sigmoid.m - сигмоида,
- computeNumericalGradient.m – численное вычисление значений градиента,
- checkNNGradients.m – функция, помогающая проверить градиент,
- debugInitializeWeights.m – функция для инициализации весов,
- predict.m – функция прогнозирования нейронной сети,
- * sigmoidGradient.m – вычисляет градиент сигмоиды,
- * randInitializeWeights.m – случайным образом инициализирует веса,
- * nnCostFunction.m – функция стоимости для нейронной сети.

(*) – файлы, с которыми вам необходимо работать.

1. Нейронные сети

В предыдущем упражнении вы реализовали так называемое прямое распространение для нейронной сети, во время которого вычисляется значение гипотезы $h_\theta(x)$. Затем эта сеть была использована для распознавания рукописных цифр, при этом значения весов уже были вам предоставлены. В этом упражнении вы запрограммируете алгоритм обратного распространения ошибки, используемого для обучения нейронной сети, и найдете эти веса самостоятельно.

Предоставленный скрипт `ex4.m` поможет вам выполнить это упражнение.

1.1 Визуализация данных

Скрипт ex4.m начнет с того, что загрузит обучающую выборку, выберет из нее случайным образом 100 строк и передаст их функции `displayData` для отображения на картинке, подобной Рисунку 1.

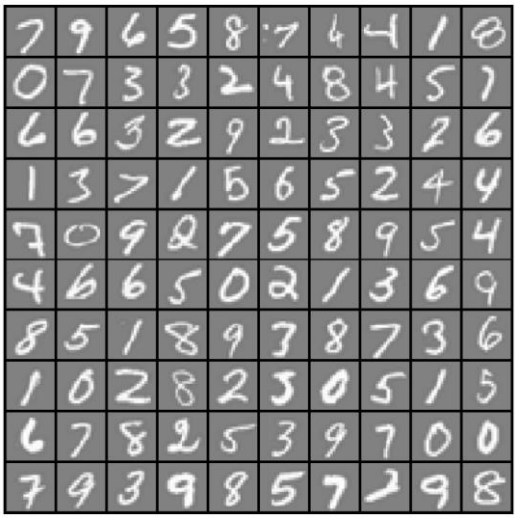


Рисунок 1: Подмножество обучающей выборки

Это та же самая обучающая выборка, которая использовалась вами в предыдущем упражнении. Напомним, что в файле `ex4data1.mat` содержится 5000 обучающих примеров, каждый из которых представляет собой изображение цифры в оттенках серого размером 20 на 20 пикселей. Каждый пиксель представлен числом с плавающей точкой, обозначающим интенсивность оттенка серого. Сетка пикселей размером 20 на 20 «разворачивается» в 400-мерный вектор. В результате мы получаем матрицу X размером 5000 на 400, где каждая строка является обучающим примером рукописного изображения цифры.

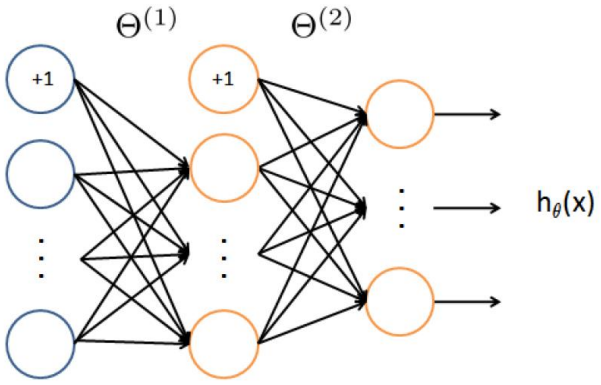
$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

Вторая часть обучающей выборки представляет собой 5000-мерный вектор y с метками. В целях удобства работы с индексацией Octave / MATLAB, начинающейся с единицы, мы присвоили цифре ноль метку со значением десять. Следовательно, цифра «0» помечается как «10», в то время как цифры от «1» до «9» помечаются как «1» до «9» в их естественном порядке.

1.2 Представление модели

Наша нейронная сеть показана на Рисунке 2. Она состоит из 3 слоев (входной, скрытый и выходной). Напомним, что входные данные представляют собой значения пикселей изображений. Поскольку изображения имеют размер 20×20, это дает нам 400 узлов входного слоя (не считая специального узла для смещения, который всегда выдает +1). Как и прежде, обучающие данные будут загружены в переменные X и y.

Вам даются матрицы весов $\Theta^{[1]}$ и $\Theta^{[2]}$ уже обученной нейронной сети. Они хранятся в файле `ex4weights.mat` и загружаются скриптом `ex4.m` в переменные `Theta1` и `Theta2`. Матрицы имеют размерности, соответствующие нейронной сети с 25 нейронами скрытого слоя и 10 нейронами выходного слоя.



$z^{[1]} = \Theta^{[1]} x$ $\quad$ $z^{[2]} = \Theta^{[2]} a^{[1]}$

$a^{[1]} = g(z^{[1]})$ $\quad$ $a^{[2]} = g(z^{[2]}) = h_\theta(x)$

добавляем $a_0^{[1]} = 1$ $\quad$ добавляем $a_0^{[2]} = 1$

Рисунок 2: Модель нейронной сети и прямого распространения для вектора x

1.3 Прямое распространение и функция стоимости

Сначала вам необходимо реализовать функцию стоимости и ее градиент для нейронной сети. Допишите код в файле `nnCostFunction.m`, чтобы он возвращал значение функции стоимости.

Напомним, что функция стоимости для нейронной сети (без регуляризации) равна[1]

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)_k\right) - \left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)_k\right)\right],$$

где $h_\theta\left(x^{(i)}\right)$ вычисляется по формулам, указанным на Рисунке 2, а $K = 10$ – количество классов. Заметьте, что $h_\theta\left(x^{(i)}\right)_k = a_k^{[2]}$ – значение функции активации k-го нейрона выходного слоя. Также напомним, что в то время, как значениями исходных меток (значениями переменной y) были 1, 2, ..., 10, для целей обучения нейронной сети нам нужно перекодировать метки в векторы, содержащие только 0 или 1, так что

$$y = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \ldots \text{или} \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}.$$

Например, если $x^{(i)}$ является изображением цифры 5, то соответствующий ему $y^{(i)}$ (который используется в функции стоимости) должен быть 10-мерным вектором с $y_5^{(i)} = 1$ и $y_k^{(i)} = 0, k \neq 5$.

Вы должны реализовать прямое распространение, которое вычисляет значения $h_\theta\left(x^{(i)}\right)$ для всех примеров i=1,…,m и затем суммирует все полученные ошибки согласно формуле выше. **Ваш код должен работать для обучающей выборки любого размера с любым количеством классов** (при этом вы можете предполагать, что всегда существует не менее $K \geq 3$ классов). Не забудьте про то, что вам нужно добавлять столбец из единиц к матрицам весов.

Как только вы закончите, `ex4.m` вызовет вашу функцию `nnCostFunction`, используя загруженный набор весов `Theta1` и `Theta2`. Вы должны получить значение функции стоимости около 0,287629.

1.4 Регуляризованная функция стоимости

Функция стоимости для нашей нейронной сети с регуляризацией задается формулой

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)_k\right) - \left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)_k\right)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}\left(\Theta_{j,k}^{[1]}\right)^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}\left(\Theta_{j,k}^{[2]}\right)^2\right].$$

Хотя вы можете предполагать, что нейронная сеть имеет только 3 слоя (входной, скрытый и выходной), ваш код при этом должен работать для любого количества узлов указанных слоев. В формуле выше мы для примера явно указали в формуле индексы для $\Theta^{[1]}$ и $\Theta^{[2]}$.

Напомним, что вам не следует регуляризовать веса, соответствующие смещениям ($\Theta_{j,0}^{[1]}$ и $\Theta_{j,0}^{[2]}$). Добавьте регуляризацию в свою функцию стоимости. Заметьте, что формула выше состоит из двух независимых слагаемых, поэтому вы можете просто прибавить к значению функции стоимости, которую вы реализовали в п.1.3, регуляризационный терм.

После того, как вы закончите, `ex4.m` вызовет функцию `nnCostFunction`, используя загруженный набор весов `Theta1`, `Theta2` и $\lambda = 1$. Вы должны получить значение около 0,383770.

---

[1] С помощью θ в $J(\theta)$ будем обозначать всю совокупность параметров (весов) нейронной сети, задаваемых в нашем примере матрицами $\Theta^{[1]}$ и $\Theta^{[2]}$.

2. Обратное распространение ошибки

В этой части упражнения вы реализуете алгоритм обратного распространения ошибки для вычисления градиента функции стоимости нейронной сети. Вам нужно будет дописать код в `nnCostFunction.m`, чтобы он возвращал корректное значение `grad`. После этого вы сможете обучить нейронную сеть, минимизировав функцию затрат $J(\theta)$ с помощью оптимизатора `fmincg`.

Вы начнете, как обычно, с нерегуляризованного случая, а после того, как убедитесь в корректности вычисления градиента, добавите в него регуляризацию.

2.1 Градиент логистической функции

Сначала напишите код для вычисления градиента функции сигмоиды. Как вы помните, он вычисляется по формуле:

$$g'(z) = \frac{d}{dz} g(z) = g(z)\big(1 - g(z)\big),$$

где

$$g(z) = \frac{1}{1 + e^{-z}}.$$

После того, как закончите, протестируйте полученную функцию на случайных значениях, вызвав функцию `sigmoidGradient(z)` из командной строки Octave/Matlab. Для больших значений z (как положительных, так и отрицательных) градиент должен быть близок к нулю. Когда $z = 0$, градиент должен равняться в точности 0.25. Ваш код должен корректно работать также для векторного и матричного входа. Для матриц функция должна вычислять градиент сигмоиды для каждого элемента.

2.2 Случайная инициализация

При обучении нейронной сети очень важно случайным образом инициализировать начальные значения весов, чтобы избежать проблемы симметрии. Одна из эффективных стратегий инициализации заключается в том, чтобы значения $\Theta^{[l]}$ выбирать случайным образом на отрезке $[-\varepsilon_{init}, \varepsilon_{init}]$. Для целей настоящего упражнения возьмите значение $\varepsilon_{init} = 0.12$. Этот диапазон гарантирует, что значения параметров будут оставаться небольшими, что сделает процесс обучения более эффективным.

Ваша задача – завершить код в файле `randInitializeWeights.m`, который инициализирует θ; допишите в файл следующие строки:

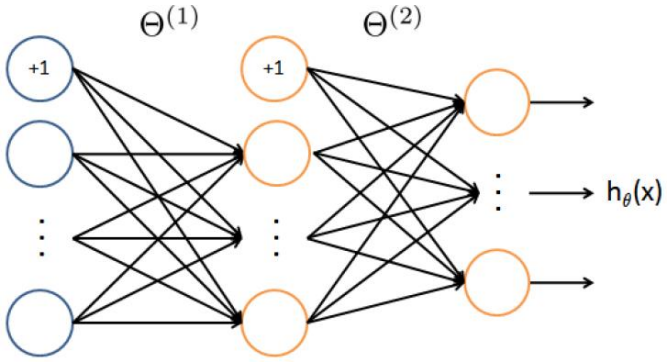
```
% инициализируем веса случайными маленькими значениями
epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init – epsilon_init;
```

2.3 Обратное распространение ошибки

Теперь вы реализуете алгоритм обратного распространения ошибки. Напомним, что интуиция, лежащая в основе алгоритма, заключается в следующем. Для обучающего примера $(x^{(t)}, y^{(t)})$ мы сначала делаем прямой проход, во время которого вычисляем значения всех функций активации по всей сети, включая выходное значение гипотезы $h_\theta(x)$. Затем для каждого узла j в слое l мы вычисляем «ошибку» $\delta_j^{[l]}$, которая показывает, насколько этот узел «несет ответственность» за итоговую ошибку работы всей сети на указанном примере.

Для выходного нейрона мы можем напрямую измерить разницу между его активацией и истинным целевым значением и использовать это в качестве значения $\delta_j^{[2]}$ (в нашем примере слой 2 является выходным). Для нейронов скрытого слоя вычислим $\delta_j^{[1]}$ как средневзвешенное значение ошибок нейронов второго слоя.

Более детально алгоритм обратного распространения заключается в следующем (также изображен на Рисунке 3). Вы должны выполнять шаги 1-4 в цикле, который обрабатывает по одному входному примеру за раз. То есть вы должны реализовать цикл for для t = 1:m и поместить шаги 1-4, описанные ниже, внутрь этого цикла, который на t-й итерации выполняет вычисление на t-м обучающем примере $(x^{(t)}, y^{(t)})$ обучающей выборки. Шаг 5 разделит накопленное значение градиентов на m и получит итоговые значения градиента функции стоимости нейронной сети.



$\delta^{[1]} = \left(\Theta^{[1]}\right)^T \delta^{[2]} .* g'\left(z^{[1]}\right) \quad \delta^{[2]} = a^{[2]} - y_j$

удаляем $\delta_0^{[1]}$

Рисунок 3: Обновления весов во время обратного распространения

Алгоритм, описанный ниже, можно векторизовать, в результате чего прямой ход и обратное распространение будут вычисляться одновременно для всех примеров из обучающей выборки. Однако, мы рекомендуем вам сначала попробовать реализовать этот код с помощью цикла, и только после этого, если есть желание и уверенность в своих силах, постараться переписать его в виде нескольких формул с матричными операциями.

Шаг 1. Выполните прямой ход нейронной сети по формулам из Рисунка 2, вычислив значения взвешенных сумм и функций активации – $z^{[1]}, a^{[1]}, z^{[2]}, a^{[2]}$. Помните, что вы должны добавить в x и $a^{[1]}$ единичные значения.

Шаг 2. Для каждого k-го нейрона выходного слоя вычислите ошибку по формуле

$$\delta_k^{[2]} = \left(a_k^{[2]} - y_k\right),$$

где $y_k \in \{0,1\}$ означает, принадлежит ли рассматриваемый пример классу k ($y_k = 1$), или нет ($y_k = 0$). Как и в предыдущем упражнении, для решения этой задачи вы можете найти полезными логические массивы.

Шаг 3. Для скрытого слоя вычислите

$$\delta^{[1]} = \left(\Theta^{[1]}\right)^T \delta^{[2]} .* g'\left(z^{[1]}\right).$$

Шаг 4. Аккумулируйте значение градиента, полученное на этом учебном примере, используя следующую формулу. Обратите внимание, что теперь вам нужно убрать значение $\delta_0^{[1]}$:

$$\Delta^{[2]} = \Delta^{[2]} + \delta^{[2]}\left(a^{[1]}\right)^T,$$
$$\Delta^{[1]} = \Delta^{[1]} + \delta^{[1]}x^T.$$

Также заметим, что в предыдущих двух формулах вектор столбец умножается на вектор строку, в результате чего получается матрица в точности той же размерности, что и матрица весов соответствующего слоя.

Шаг 5. Получите значение нерегуляризованного градиента функции стоимости нейронной сети, поделив аккумулированные значение на m:

$$\frac{\partial}{\partial\Theta_{ij}^{[l]}}J(\theta) = D_{ij}^{[l]} = \frac{1}{m}\Delta_{ij}^{[l]}.$$

После того, как вы закончите программировать алгоритм обратного распространения ошибки, скрипт `ex4.m` запустит численную проверку вашей функции.

2.4 Проверка градиента

Один из приемов, позволяющих убедиться, что вы корректно вычисляете градиент – численно посчитать его приближенное значение в нескольких конкретных точках. Делается это следующим образом. «Развернем» матрицы весов $\Theta^{[1]}$ и $\Theta^{[2]}$ в один длинный вектор. Далее, представим, что функция $f_i(\theta)$ должна вычислять $\frac{\partial}{\partial\theta_i}J(\theta)$, и вы хотите проверить, так ли это. Пусть

$$\theta^{(i+)} = \theta + \begin{bmatrix}0\\0\\\vdots\\\epsilon\\\vdots\\0\end{bmatrix} \text{ и } \theta^{(i-)} = \theta - \begin{bmatrix}0\\0\\\vdots\\\epsilon\\\vdots\\0\end{bmatrix}.$$

То есть $\theta^{(i+)}$ совпадает с θ везде, кроме i-го значения, которое увеличено на ϵ. То же касается и $\theta^{(i-)}$, в котором i-тое значение уменьшено на величину ϵ. Теперь вы можете численно проверить $f_i(\theta)$, вычислив для каждого i и убедившись, что:

$$f_i(\theta) \approx \frac{J\left(\theta^{(i+)}\right) - J\left(\theta^{(i-)}\right)}{2\epsilon}.$$

Точность, с которой это отношение должно выполняться, будет зависеть от функции J. Но, взяв $\epsilon = 10^{-4}$, вы, как правило, будете видеть, что левая и правая части приведенного выше отношения будут совпадать как минимум в 4 значащих цифрах.

В файле `computeNumericalGradient.m` вы найдете уже написанную функцию для численного вычисления градиента. Хотя вам не требуется изменять файл, мы настоятельно рекомендуем вам взглянуть на код, чтобы понять, как он работает.

На следующем шаге `ex4.m` с помощью функции `checkNNGradients.m` создаст небольшую нейронную сеть и обучающую выборку, которые будут использоваться для проверки ваших градиентов. Если ваша реализация обратного распространения ошибки верна, вы должны увидеть относительную разницу меньше 1e-9.

Практический совет. При выполнении проверки градиента, как правило, используют небольшую нейронную сеть с относительно небольшим количеством элементов входного и скрытого слоев, что приводит к относительно небольшому количеству параметров. В функции `checkNNGradients` наш код создает небольшую случайную модель и набор данных, которые используются с `computeNumericalGradient` для проверки градиента. После того, как вы убедитесь, что ваши

вычисления градиента верны, вы должны отключить проверку градиента перед запуском алгоритма обучения.

2.5 Регуляризованная нейронная сеть

Теперь добавим регуляризацию в нейронную сеть. Оказывается, сделать это очень просто, так как для этого не нужно менять формулу, которую вы только что запрограммировали. Вы можете добавить регуляризационный терм после того, как вычислили значение градиента с помощью обратного распространения ошибки:

$$\frac{\partial}{\partial \Theta_{i0}^{[l]}} J(\theta) = D_{i0}^{[l]} = \frac{1}{m} \Delta_{i0}^{[l]},$$
$$\frac{\partial}{\partial \Theta_{ij}^{[l]}} J(\theta) = D_{ij}^{[l]} = \frac{1}{m} \Delta_{ij}^{[l]} + \frac{\lambda}{m} \Theta_{ij}^{[l]}, \text{ для } j \geq 1.$$

В очередной раз напоминаем, что вы не должны регуляризовать первые столбцы $\Theta^{[l]}$, соответствующие смещениям.

После того, как вы закончите, скрипт `ex4.m` приступит к выполнению численной проверки вашего градиента. Если все верно, вы получите относительную разницу не более 1e-9.

2.6 Поиск параметров с помощью `fmincg`

На следующем шаге скрипт `ex4.m`, используя `fmincg`, обучит нейронную сеть.

После завершения обучения скрипт `ex4.m` сообщит о точности вашего классификатора на учебной выборке, вычислив процент правильно классифицированных примеров. Если ваша реализация верна, вы должны увидеть точность около 95,3% (плюс/минус 1% из-за случайной инициализации). Если использовать большее количество итераций обучения, то можно добиться большей точности. Мы рекомендуем вам попробовать обучить нейронную сеть на большем количестве итераций (например, установить значение `MaxIter` равным 400), а также изменить параметр регуляризации. При соответствующих настройках обучения можно добиться того, чтобы нейронная сеть будет распознавать примеры из обучающей выборки со 100% точностью.

3. Визуализация скрытого слоя

Один из способов понять, чему учится ваша нейронная сеть – это визуализировать веса нейронов на скрытом слое. Обратите внимание, что для обученной вами нейронной сети i-я строка матрицы весов $\Theta^{[1]}$ есть 401-мерный вектор, представляющий собой параметры для i-го нейрона скрытого слоя. Если мы отбросим смещение, то получим 400-мерный вектор с весами для каждого пикселя входного изображения.

Таким образом, один из способов визуализировать «представление», которому обучился нейрон скрытого слоя, это преобразовать 400-мерный вектор в изображение 20×20.[2] На следующем шаге скрипт `ex4.m` сделает это с помощью функции `displayData`, которая покажет вам изображение (аналогично Рисунку 4) с 25 блоками, каждый из которых соответствует одному нейрону скрытого слоя в сети.

Вы можете заметить, что скрытые нейроны можно условно интерпретировать как детекторы, которые ищут штрихи и другие паттерны во входных данных.

[2] Оказывается, это соответствует изображению входного значения (картинки), которое приводит к максимальной активации данного нейрона, с учетом нормирующего ограничения $\|x\|_2 \leq 1$.

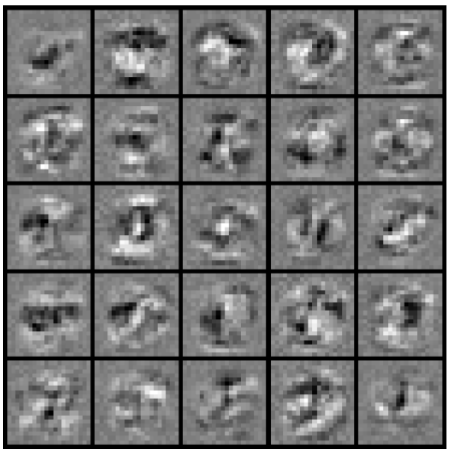


Рисунок 4: Визуализация нейронов скрытого слоя

3.1 Эксперименты

Попробуйте поэкспериментировать с количеством итераций обучения и коэффициентом регуляризации λ, и посмотрите, как в зависимости от этого меняется визуализация весов нейронов скрытого слоя.