Отчет по пятому лабораторному заданию «Нейросетевой разреженный автокодировщик»

Алескин Александр

5 марта 2016 г.

Содержание

1 Введение

В рамках данного задания были выполнены все требуемые пункты, в том числе вывод формулы градиента разреженного автокодировщика, исследования автокодировщика с одним и с тремя скрытыми слоями. Так же был рассмотрен бонусный пункт зависимость точности классификаци от количества обучающих патчей автокодировщика. В целях краткости были представлены только основные данные и выводы.

2 Вывод формулы градиента функции потерь

Опишем построение градиента функции при помощи алгоритма обратного распространения ошибки.

Пусть обучаем автокодировщик с K слоями: на каждом слое n_i нейронов, где $i = 0, ..., K$ (так как автокодировщик, то $n_0 = n_K$). Через L обозначим количество элементов в выборке, x^j — j-ый элемент выборки, y^j - значение целевой функции на j-ом элементе (в нашем случае $x^j = y^j$), W^i — матрица весов между $i-1$ и i слоями, а b^i — вектор смещения для нейронов на i слоем. Так как строим разряженный автокодировщик, то параметр разряженности будем обозначать через p.

Тогда общий вид функции потерь вычисляется по формуле:

$$J(W, b) = J_1(W, b) + J_2(W, b) + J_3(W, b),$$

$$J_1(W, b) = \frac{1}{L}\sum_{j=1}^{L}\frac{1}{2}||h_{W,b}(x^j) - y^j||^2 \text{ ,где } h_{W,b}(x^j) - \text{преобразование нейросети}$$

$$J_2(W, b) = \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{i=1}^{n_{k-1}}\sum_{j=1}^{n_k}(w_{ij}^k)^2 \text{ ,где } \lambda - \text{ параметр весов,}$$

$$J_3(W, b) = \beta\sum_{k=1}^{K-1}\sum_{j=1}^{n_k}\left[p\log\big(\frac{p}{\hat{p}_j^k}\big) + (1-p)\log\big(\frac{1-p}{1-\hat{p}_j^k}\big)\right],$$

Где $\hat{p}$ — среднее значение функции активации нейрона. Рассмотрим для начала функционал $J^*(W, b) = J_1(W, b) + J_3(W, b)$. Воспользуемся алгоритмом обратного распространения ошибки. Уточним следующие обозначения и свойства функций:

- В качестве функции активации используем сигмоидную функцию: $f(x) = \frac{1}{1+e^{-x}}$
- $f'(x) = \big(\frac{1}{1+e^{-x}}\big)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}+1-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\big(1 - \frac{1}{1+e^{-x}}\big) = f(x)\big(1 - f(x)\big)$
- $a_0 = X, a_i = f(z_i)$, где $z_i = a_{i-1}W^i + b^i, i = 1, ..., K$. Под матрицей X понимаем матрицу выборки размера $L \times n_0$. Заметим, что $a_K = h_{W,b}$.
- $\hat{p}_j^k = \frac{1}{L}\sum_{r=1}^{n_k} f(z_{rj}^k)$— функция активации нейрона

Вычислим градиент фунции:

1. $\sigma^{n_K} = \left(\frac{\partial J^*}{\partial z_{l,j}^{n_k}}\right)_{l,j} = \left(\frac{\partial}{\partial z_{l,j}^{n_k}} \frac{1}{2L} ||Y - h_{W,b}(X)||^2\right)_{l,j} = -\frac{1}{L}\big(Y - h_{W,b}(X)\big) f'(z^{n_k})$,

2. $\sigma^{n_i} = \left(\frac{\partial J^*}{\partial z_{l,j}^{n_i}}\right)_{l,j} = \left(\sum_{r=1}^{n_{i+1}} \frac{\partial J_1}{\partial z_r^{n_{i+1}}} \frac{\partial z_r^{n_{i+1}}}{\partial z_j^{n_i}} + \beta \frac{\partial}{\partial z_{l,j}^{n_i}} \left[p \log\left(\frac{p}{\hat{p}_j^k}\right) + (1-p) \log\left(\frac{1-p}{1-\hat{p}_j^k}\right)\right]\right)_{l,j} =$
$(W^i)^T \sigma^{n_{i+1}} f'(Z^i) + \frac{1}{L}\beta\big(-\frac{p}{\hat{p}^i} + \frac{1-p}{1-\hat{p}^i}\big) f'(Z^i) = \left[(W^i)^T \sigma^{n_{i+1}} + \frac{\beta}{L}\big(-\frac{p}{\hat{p}^i} + \frac{1-p}{1-\hat{p}^i}\big)\right] f'(Z^i)$, $i = 1, ..., K-1$

3. $\left(\frac{\partial J^*}{\partial w_{l,j}^{n_i}}\right)_{l,j} = \left(\sum_{r=1}^{L} \frac{\partial J^*}{\partial z_{rj}^{n_i}} \frac{\partial z_{rj}^{n_i}}{\partial w_{l,j}^{n_i}}\right)_{l,j} = (a^i)^T \times \sigma^{n_i}$ – "часть"градиента по элементам матрицы W^i

4. $\left(\frac{\partial J^*}{\partial b_j^{n_i}}\right)_j = \left(\sum_{r=1}^{L} \frac{\partial J^*}{\partial z_{rj}^{n_i}} \frac{\partial z_{rj}^{n_i}}{\partial b_j^{n_i}}\right)_j = \left(\sum_{r=1}^{L} \sigma_{rj}^{n_i}\right)_j$ – "часть"градиента по элементам вектора смещения b^i

Таким образом, используя алгоритм обратного распространения ошибки, находим градиент $\nabla J^*(W, b)$. Так как $J(W, b) = J^*(W, b) + J^2(W, b)$, то $\nabla J(W, b) = \nabla J^*(W, b) + \nabla J_2(W, b)$.

Тогда имеем:

- $\left(\frac{\partial J}{\partial w_{l,j}^{n_i}}\right)_{l,j} = \left(\frac{\partial J^*}{\partial w_{l,j}^{n_i}} + \frac{\partial J_2}{\partial w_{l,j}^{n_i}}\right)_{l,j} = (a^i)^T \times \sigma^{n_i} + \lambda W^i$
- $\left(\frac{\partial J}{\partial b_j^{n_i}}\right)_j = \left(\frac{\partial J^*}{\partial b_j^{n_i}}\right)_j = \left(\sum_{r=1}^{L} \sigma_{rj}^{n_i}\right)_j$ – так как вообще говоря $J_2(W, b) = J_2(W)$ (независит от b).

3 Тестирование алгоритма

На основание выше описанного алгоритма реализуем метод поиска градиента функции потерь: autoencoder_loss. Для проверки правильности выполнения воспользуемся функцией compute_gradient, в качестве аргумента которой передается функция loss_func, которая вытсляет значения функции потерь $J(W, b)$. Для проверки же compute_gradient воспользуемся набором простых функций, градиент которых не представляет трудностей найти аналитически:

- $f_1(x) = 2x + 0.5$
- $f_2(x) = 5x^2 - 0.22$
- $f_3(x, y, z) = x^2 + 2y^3 + 4z^3 + 0.5$
- $f(x, y, z, r) = \sqrt{x} + \sin y + \log(y) + e^{r^{1.5}}$

Замеряя точность этих функций в нескольких точках при помощи функции check_gradient(), убедились в правильности алгоритма. Аналогичном образом, убеждаемся в правильности работы функции autoencoder_loss. Приступим к исследовательской части.

4 Автокодировщик с одним скрытым слоем

Обучим алгоритм на предложенной выборке unlabeled.pk при помощи встроенный в библиотеку scipy функции optimize.minimize при 100 000 выделенных патчах. Как видно, на рисунке 3, почти все простые границы (вдоль прямой) выделились фильтрами. Так же выделены фильтры с основными цветами для rgb представления изображения. Такая вырожденность и большое количество зануливших фильтров является следствием большого значения параметра разреженности и того, что из большого количества независимых патчей можно выделить только

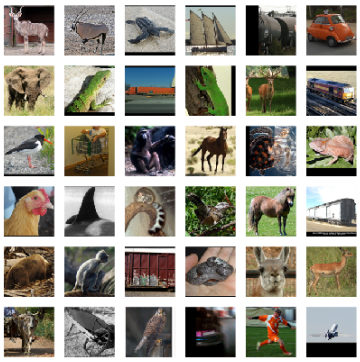

Рис. 1: Фотографии

Рис. 2: Случайные патчи

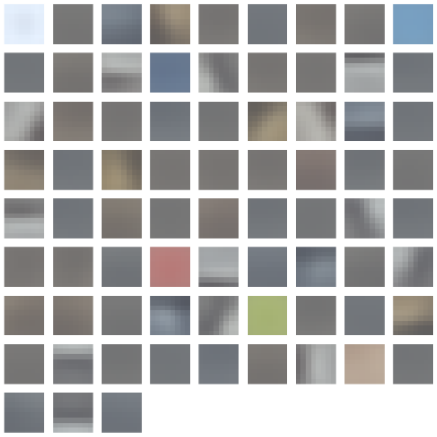

Рис. 3: Фильтры

простые границы и цвета. Поэтому гипотетически количество нейронов на скрытом слое можно было снизить почти в 2 раза, без больших потерь для точности (не проверялось). Обучался алгоритм с предложенными значениями параметров: $\lambda = 0.0001, \beta = 2, p = 0.01$.

Для того, чтобы можно было оценить целесообразность преобразования признакового пространства, оценим разницу точностей некоторых классификаторов на исходных данных и преобразованных. В качестве классификаторов возьмем Random Forest и Logistic Regression, так как эти алгоритма наиболее просто и быстро обучаемы. Оценку точности будем производить по валидационной выборке.

Патчи на изображении можно выделать разными способами. Рассмотрим несколько вариантов с разной длиной шага между соседнями патчами.

В ходе эксперимента было выяснено, что Random Forest значительно превосходит LogRegression на выборке, где в качестве признаков выступают значения интенсивностей цветных каналов изображений. Однако, преобразование данных значительно улучшает классификацию при помощи логистической регрессии, что видно из рисунка 4, и составляет 42.7%. Случайнный лес незначительно улучшает свой показатель.

При достаточном большом шаге (16-20) большая часть изображения не участвует в классификации. Однако, точность алгоритмов незначительно снижается. Вероятно, это связано с тем, что классификатор больше ориентируется на цветовые характеристики, так как выделить границы объекта на частях картинки становиться затруднительным.

5 Зависимость точности классификаци от количества обучающих патчей автокодировщика(бонус)

Так как в выше описанном исследовании размер выборки был 100 000 патчей, то для выполнения задания рассмотрим случаи при 100, 1000, 10000 элементах. Оценку классификации будем измерять теми же методами: случайнный лес и логистическая регрессия. На рисунке 5 представлены результаты. Как видно, точность классификации почти не зависит от размера выборки патчей.

6 Автокодировщик с несколькими скрытыми слоями

Проведем аналогичные испытание с тремя скрытыми слоями. Как было выяснено опытном путем, количестов патчей незначительно влияет на качество классификации, поэтому обучим автокодировщик на 10 000 патчей.

Несмотря на усложнение системы, уровень классификации заметно снизился и составил 24% для случайнного леса и 12% для логистической регрессии. Можно выделить несколько причин ухудшения. Во-первых, параметры скорее всего не являются оптимальными для данной структуры. Во-вторых, резкое увеличение параметров переобучает систему. В-третьих, на втором слое использовалось всего 25 нейронов, возможно слишком грубая аппроксимация.

7 Заключение

В данном задании были рассмотрены все обязательные пункты задания и выполнено бонусное задание "зависимость точности классификации от количества обучающих патчей для автокодировщика".

В рамках исследовательской части были получены следующие выводы:

- Использование автокодировщика для преобразования признакого пространства значительно улучшает качество линейных классификаторов и незначительно улучшает нелинейные(Random Forest).
- Разреженный автокодировщик формирует фильтры границ или цветовые фильтры.
- Многослойнный автокодировщик в силу большего количества параметров, склонен к переобучению
- Применение алгоритма обратного распространения ошибки на порядки ускоряет вычисления.
- Качество классификации слабо зависит от количества обучающих патчей для автокодировщика.

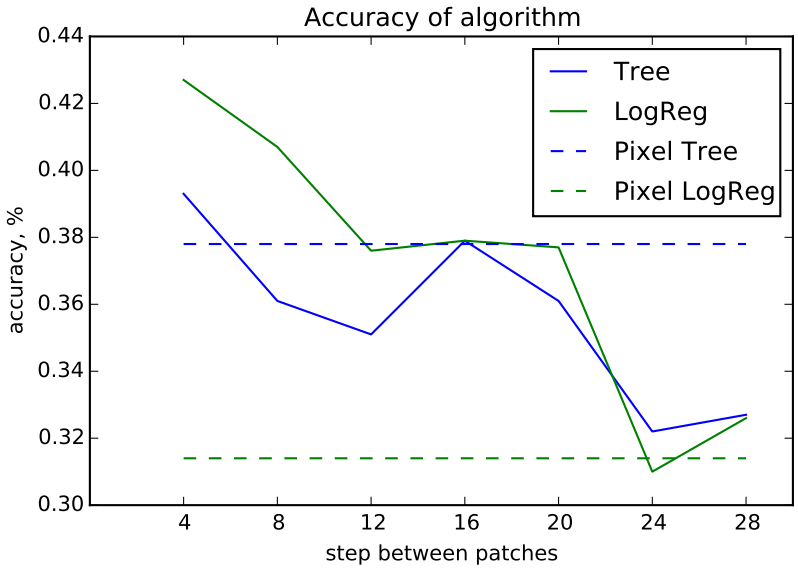

Рис. 4: Точность алгоритмов

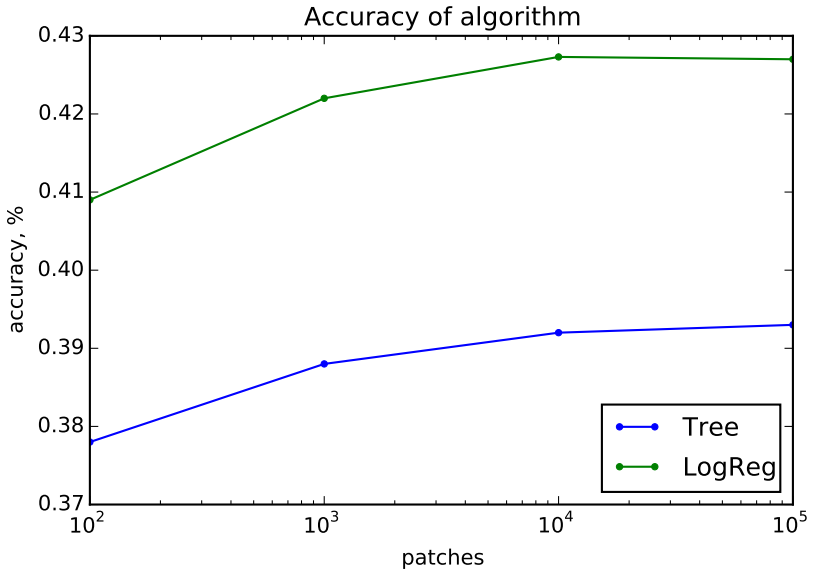

Рис. 5: Точность на разном количестве патчей