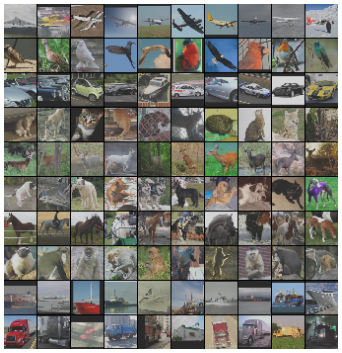
ВМК

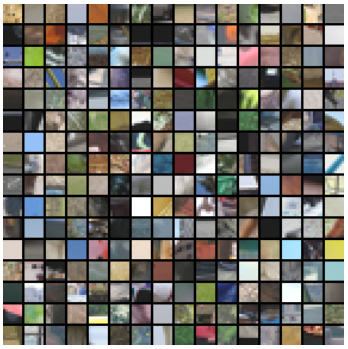
Задание 5. Нейросетевой разреженный автокодировщик

Практикум 317 группы, весна 2015

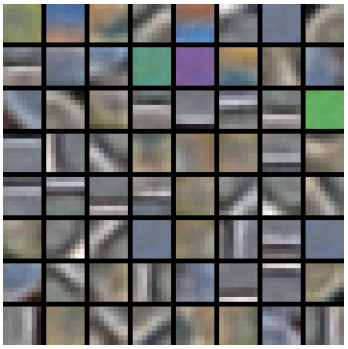
Начало выполнения задания: 16 февраля 2015 года.
Срок сдачи: **3 марта 2015 года, 23:59.**
Среда для выполнения задания: Python 3.4 (желательно) / 2.7 (при использовании GPU).
Текст задания последний раз обновлялся 21 февраля 2015 г.



(a) Исходные изображения



(b) Выделенные патчи



(c) Обученные фильтры

Содержание

1 Ликбез

В последние годы в задачах анализа звука и изображений большую популярность приобрели методы глубинного обучения. Основная их идея заключается в том, что из данных извлекается иерархия признаков: признаки первого уровня подсчитываются непосредственно по объекту, признаки второго уровня — из признаков первого уровня, и так далее. На последнем уровне иерархии получается ответ решаемой задачи. Несмотря на то, что подобные методики применялись с начала 90-х годов, настоящим прорывом стала в 2012-м году работа Алекса Крижевского [1], который смог эффективно применить парадигму глубинного обучения к задаче классификации изображений и уменьшить ошибку на 20% по сравнению с лучшими классическими методами. Глубинная свёрточная сеть, которую использовал Крижевский, имеет десятки миллионов настраиваемых параметров. Более новые сети, предназначенные для решения той же задачи, уже сейчас больше ещё на порядок.

Задача настройки глубинных сетей довольно сложна. Функции, оптимизируемые при их обучении, невыпуклы, содержат плато и множество локальных минимумов, что зачастую ведет к переобучению на фоне большого числа параметров. Найти глобальный минимум такой функции практически невозможно, поэтому на практике стремятся найти хороший локальный минимум. При этом результат обучения сильно зависит от инициализации и используемого метода оптимизации. Чтобы избежать переобучения, можно сначала сократить размерность данных (игнорируя метки классов), а затем уже обучаться на данных низкой размерности. Первый этап может заменять ручное извлечение признаков, поэтому эту идею называют обучением признаков (feature learning). Автокодировщик — это нейросеть, которая преобразует данные высокой размерности в данные низкой размерности, как и метод главных компонент (PCA), но при этом преобразование является нелинейным.

[1] Kryzhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

Искусственная нейронная сеть

В данном пункте будет приведено краткое напоминание того, что такое нейронная сеть, а также будут введены используемые в дальнейшем обозначения. При необходимости более подробный материал по нейронным сетям можно посмотреть в соответствующей лекции К. В. Воронцова.

Предположим, что мы решаем классическую задачу обучения с учителем, имея в распоряжении набор объектов вместе с соответствующими метками $\{x^{(i)}, y^{(i)}\}$. Нейронные сети позволяют строить сложные нелинейные алгоритмы для настройки на данные. Минимальная структурная единица нейронной сети — нейрон, который схематически можно отобразить следующим образом:

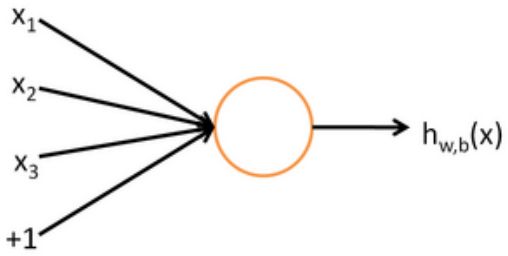


Рис. 2: Структура нейрона.

В данном примере нейрон — это «вычислительный узел», принимающий на вход значения x_1, x_2, x_3 (а также константный единичный вход), а на выходе выдающий значение $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, где функция $f : \mathbb{R} \to \mathbb{R}$ называется функцией активации. Везде далее мы будем использовать *сигмоидную* функцию активации:

$$f(z) = \frac{1}{1 + \exp(-z)}.$$

Таким образом, одинокий нейрон в точности отвечает преобразованию из логистической регрессии.

Нейронная сеть в общем случае строится как соединение множества нейронов, объединенных в *слои* так, что выходы одного слоя являются входами следующего:

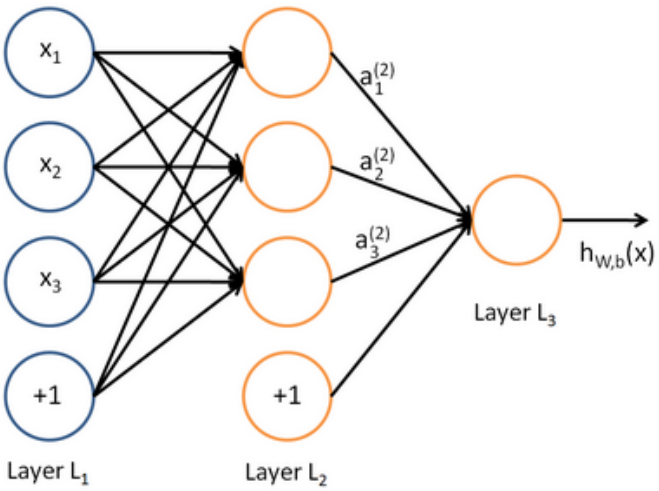


Рис. 3: Структура нейросети.

Самый левый слой сети называется входным, самый правый — выходным (на рисунке он состоит из одного нейрона), остальные слои называют *скрытыми*, потому что их «правильные» значения отсутствуют в обучающем наборе. Таким образом, данная сеть содержит 3 входных нейрона, 3 скрытых и 1 выходной.

Нейросеть параметризуется значениями $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots)$, где под $W_{ij}^{(l)}$ мы понимаем параметр, или вес, который отвечает соединению между $j-$м нейроном в слое l и $i-$м нейроном в слое $l+1$. А за $b_i^{(l)}$ будем обозначать смещение или же вес, который связан с константными единичными входами на каждом слое сети. Таким образом на рисунке выше $W^{(1)} \in \mathbb{R}^{3\times3}$ и $W^{(2)} \in \mathbb{R}^{1\times3}$. Еще раз отметим, что в константные узлы не входят никакие связи, а выходы они подают всегда $+1$.

Мы будем обозначать $a_i^{(l)}$ значение активации (число на выходе) нейрона i в слое l. Для $l = 1$ положим $a_i^{(1)} = x_i$, отвечающее i-му входу. Для заданных параметров W, b нейросеть определяет вещественное преобразование

$h_{W,b}(x)$. В частности, для сети выше:

$$\begin{aligned} a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}); \\ a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}); \\ a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}); \\ h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}). \end{aligned}$$

Далее под $z_i^{(l)}$ будем записывать общую взвешенную сумму входов в нейрон i в слое l вместе с константным узлом — к примеру, $z_i^{(2)} = \sum_{j=1}^n W_{ij}^{(1)}x_j + b_i^{(1)}$, где $a_i^{(l)} = f(z_i^{(l)})$. За счет этого можно перейти к более простой записи выражений выше:

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)}; \\ a^{(2)} &= f(z^{(2)}); \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)}; \\ h_{W,b}(x) &= a^{(3)} = f(z^{(3)}). \end{aligned}$$

Эти преобразования называются прямым проходом или прямым распространением (forward propagation). В общем случае если под $a(1)$ понимать x из входного слоя, то активация каждого следующего слоя получается по формуле

$$\begin{aligned} z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)}; \\ a^{(l+1)} &= f(z^{(l+1)}). \end{aligned}$$

Алгоритм обратного распространения ошибки

Напомним, что мы решаем классическую задачу обучения с учителем, имея в распоряжении набор объектов вместе с соответствующими метками $\{x^{(i)}, y^{(i)}\}$, всего m объектов. Для обучения нейронной сети воспользуемся градиентным спуском. Для каждого отдельного объекта (x, y) мы определяем функцию потерь как

$$J(W, b; x, y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2.$$

Общая функция потерь для всех объектов может быть вычислена как

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^m J(W, b; x^{(i)}, y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^m \left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2. \end{aligned}$$

Первая часть функционала — усредненная квадратичная ошибка по всем объектам, вторая часть — регуляризация (или контроль угасания весов), которая контролирует порядок весов и не дает переобучиться. Параметр λ, контролирующий угасание весов, регулирует относительную важность двух частей функционала.

Нашей целью теперь стоит минимизация $J(W, b)$ как функции от W и b. Для обучения нейронной сети мы сначала инициализируем параметры $W_{ij}^{(l)}$ и $b_i^{(l)}$ маленькими случайными числами, близкими к 0 (скажем, из нормального распределения $N(0, \varepsilon^2)$ для достаточно малого ε), а затем применяем алгоритм оптимизации. Отметим важность инициализации случайными числами, а не нулями — в последнем случае нейроны на средних слоях будут обучаться на подсчет одних и тех же преобразований, тогда как при случайной инициализации происходит т.н. *нарушение симметрии* (symmetry breaking).

Одна итерация градиентного спуска обновляет параметры W, b по правилу:

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b), \\ b_i^{(l)} &= b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b), \end{aligned}$$

где α — темп обучения при оптимизации. Главной сложностью здесь является вычисление соответствующих частных производных — для этого и применяется алгоритм обратного распространения ошибки:

1. Произвести прямой проход — вычислить функции активации для всех слоев $L_2, L_3, \ldots$ до выходного L_{n_l}.
2. Для каждого узла i на выходе (в слое L_{n_l}) вычислить

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}}\ \frac{1}{2}\left\|y - h_{W,b}(x)\right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)}).$$

3. Для $l = n_l - 1, n_l - 2, \ldots, 2$ для каждого узла i в соответствующем слое вычислить

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}).$$

4. Вычилсить соответствующие частные производные как

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) &= a_j^{(l)} \delta_i^{(l+1)} \\ \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) &= \delta_i^{(l+1)}. \end{aligned}$$

Везде в приведенном алгоритме соответствующие циклы («для каждого узла») могут (и должны быть) оптимизированы с помощью матричных вычислений. Введенные переменные $\delta_i^{(l)}$ можно понимать как «вклад», который вносит соответствующий $i-$й нейрон на $l-$м слое сети в общую ошибку при варьировании соответствующего входа из предыдущего слоя.

Автокодировщик

Предположим, у нас есть набор неразмеченных данных $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots\}$, где $x^{(i)} \in \mathbb{R}^n$. Автокодировщиком называется нейросеть, целевой вектор которой полагается равным её входу, т.е. $y^{(i)} = x^{(i)}$. Автокодировщик тем самым строит приближение функции $h_{W,b}(x) \approx x$, т.е. фактически приближение для тождественной функции. Хоть тождественная функция и не выглядит разумной целью для аппроксимации сама по себе, тем не менее накладывая ограничения, к примеру, на число нейронов в скрытых слоях нейронной сети, мы можем обнаружить структурные закономерности в наших данных. Применение автокодировщика также полезно в ситуации, когда мы работаем с частично размеченными данными — при этом он используется для обучения сети, которая затем генерирует признаки для стандартных алгоритмов обучения с учителем (в качестве признаков затем выступают значения активации нейронов одного из внутренних слоев).

Разреженный автокодировщик

Помимо использования малого числа нейронов на скрытом слое, можно накладывать и другие ограничения для обнаружения скрытых закономерностей. Одно из таких ограничений — ограничение разреженности на нейроны скрытого слоя. Если под активным нейроном понимать нейрон со значением функции активации, близком к 1, а под неактивным — близком к 0 (для случая сигмоидной функции активации), то можно наложить ограничение, при котором каждый из нейронов большую часть времени был бы неактивен.

Пусть $\hat{\rho}_j$ — среднее значение функции активации нейрона j по всем объектам выборки. Тогда мы будем требовать приближенного выполнения ограничения $\hat{\rho}_j = \rho$, где ρ — параметр, отвечающий за разреженность, обычно достаточно малая величина, близкая к 0 (порядка 0.05). Чтобы достичь этого, мы добавим дополнительно ограничение для функции потерь нашей нейросети, а именно добавим такое слагаемое:

$$\sum_{j=1}^{h} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{(1 - \rho)}{(1 - \hat{\rho}_j)},$$

где h — количество скрытых нейронов. В терминах КЛ-дивергенции этот регуляризатор можно формально (т.к. вектора на самом деле не являются распределениями) записать как

$$\sum_{j=1}^{h} \mathrm{KL}(\rho || \hat{\rho}_j).$$

Общая функция потерь тем самым будет равна

$$J_{\text{sparse}}(W, b) = J(W, b) + \beta \sum_{j=1}^{h} \mathrm{KL}(\rho || \hat{\rho}_j),$$

где β — гиперпараметр, т.е. параметр, задающий модель.

2 Задание

Ваша задача состоит в том, чтобы реализовать обучение разреженного автокодировщика и показать, как он обнаруживает, что границы объектов и цветовые переходы — одно из лучших представлений для естественных изображений. Краткое описание автокодировщика дано выше в ликбезе. Общий план процесса обучения автокодировщика и его использования для генерации признаков приведен ниже. Вам необходимо будет реализовать необходимый функционал в виде python-модулей, спецификации которых будут приведены в конце документа.

Подготовка

1. В этом задании обучение разреженного кодировщика будет происходить с помощью метода оптимизации L-BFGS на патчах (участках исходных изображений) размера 8×8, поэтому необходимо реализовать модуль для генерации соответствующей выборки на основе исходных изображений. Подробное описание предлагаемых к использованию данных приведено в следующем пункте. При генерации каждому случайному патчу соответствует случайныо выбранное исходное изображение и случайно выбранное место «выреза».

2. После генерации набора патчей их необходимо нормализовать — в каждом канале следует отбросить интенсивности, выходящие за 3 стандартных отклонения в обе стороны, а затем перевести линейным преобразованием в отрезок $[0.1, 0.9]$ — под отбрасыванием понимается срезка в смысле функционального анализа — если обозначить $f_\tau^+(x) = \max(x, \tau)$ и $f_\tau^-(x) = \min(x, \tau)$, то над векторами интенсивности необходимо сделать преобразование
$$X := f^+_{\mu-3\sigma}(f^-_{\mu+3\sigma}(X)),$$
где μ и σ — мат. ожидания и стандартные отклонения по столбцам X.

3. В дальнейшем вам необходимо будет реализовать метод обратного распространения ошибки с учетом использования регуляризатора разреженности. **Распишите** в вашем отчете то, как при этом должны измениться формулы при подсчете градиентов вообще и подсчет для $\delta_i^{(l)}$ в частности.

Обучение разреженного автокодировщика

4. Реализуйте функцию потерь разреженного однослойного автокодировщика и соответствующий ей градиент с сигмоидной функции активации. Напоминаем, что целевая функция потерь должна в себя включать квадратичную ошибку, регуляризатор разреженности, и контроль угасания весов. При выводе формул мы использовали матричную нотацию для некоторых весов, при реализации же для возможности дальнейшей оптимизации сторонними функции стоит передавать и возвращать вытянутые в вектора параметры. Значение функционала и градиент **считаются по всем объектам выборки одновременно** (градиентный спуск не стохастический), чего можно добиться, используя по максимуму матричные произведения.

5. Для проверки правильности вычислений градиента функции потерь автокодировщика реализуйте функцию приближенного вычисления градиента произвольной функции на основе разностной аппроксимации ($\frac{\partial f(x)}{\partial x_i} \approx \frac{f(\dots, x_i+\varepsilon, \dots) - f(\dots, x_i-\varepsilon, \dots)}{2\varepsilon}$), чтобы можно было сравнить с ним подсчитанное «аналитически» значение, а её в свою очередь проверьте на корректность с помощью подсчета градиента простых функций вроде многочленов небольших степеней. Таким образом должна использоваться двухэтапная схема проверки:

   (a) Проверить, что `compute_gradient` для объявленной отдельно простой функции выдает именно то, что и ожидается (проверка осуществляется функцией `check_gradient`);

   (b) Убедившись, что `compute_gradient` делает корректные приближения, проверить с помощью него, что аналитические градиенты в `autoencoder_loss` считаются правильно.

6. Реализовав функцию потерь, минимизацию можно будет провести с помощью библиотеки `scipy.optimize` и метода оптимизации L-BFGS, сделав такой вызов в коде:

   `scipy.optimize.minimize(J, theta, method='L-BFGS-B', jac=True, ...)`,

   где `J` и `theta` — функция и точка старта метода оптимизации, далее идет выбор метода оптимизации (можно оставлять неизменным), дополнительно также можно указать количество итераций и наличие дебаг-вывода при минимизации.

   В качестве начального приближенного приближения для b можно использовать 0, а W_{ij} генерировать из равномерного распределения на интервале
$$\left[-\sqrt{\frac{6}{n_{in}+n_{out}+1}}, \sqrt{\frac{6}{n_{in}+n_{out}+1}}\right],$$
   где n_{in} — число входов в нейрон, а n_{out} — число выходов из нейрона.

Визуализация

7. Для подтверждения гипотезы о том, что наилучшее представление изображений — представление в виде ребер и цветовых переходов, нужно реализовать модуль для визуализации весов на скрытом слое обученного автокодировщика — каждый нейрон следует представлять отдельным изображением того же размера, что и исходные патчи. В отчет необходимо включить изображения, которые у вас получились. Также включите иллюстрацию того, как ухудшается визуализация при изменении каждого из гиперпараметров. Пример визуализации правильно обученного скрытого слоя представлен в заголовке документа.

Параметры по умолчанию для проверки правильности реализации

Напоминаем, что перед использованием параметров

- Количество входных нейронов — 192;
- Количество нейронов на одном скрытом слое — 75;
- $\rho = 0.01$;
- $\lambda = 0.0001$;
- $\beta = 3$.

Обучение классификатора на данных сокращённой размерности

Далее для классифкации мы будем использовать следующую процедуру — сначала на основе десятков тысяч патчей, выделенных из неразмеченных данных (`unlabeled.pk`), нужно обучить параметры автокодировщика. Затем на основе признаков, выделенных автокодировщиком, должна производиться оценка точности (accuracy) модели, обученной на обучающей выборке (`train.pk`), по тестовой выборке (`test.pk`).

8. Используя модули библиотеки `scikit-learn`, обучите несколько различных моделей классификации, используя в качестве признаков сначала значения интенсивностей уветных каналов изображений, а затем признаки, выделенные из изображений с помощью автокодировщика. Для получения признаков с помощью автокодировщика следует использовать (возможно, перекрывающиеся) участки исходных изображений с равномерным шагом — так, из изображения размера 12×16 можно получить 6 патчей размера 8×8 с шагом 4. Сравните результаты классификации для использования разного шага выделения патчей.

9. Исследуйте, как влияет изменение структуры скрытых слоев на качество классификации — сравните сеть с одним скрытым слоем с сетью с тремя скрытыми слоями, где число нейронов на первом и третьем скрытых слоях совпадают, и оно больше числа нейронов на среднем слое.

Бонус

За выполнение нижеприведенных бонусных заданий можно получить дополнительно в сумме до 1 балла.

10. (0.5 балла) Исследуйте, как влияет увеличение используемой выборки (10^2–10^5 патчей) для обучения автокодировщика на итоговое качество классификации. Задействуйте при обучении данные без меток (`unlabeled.pk`) для значительного увеличения исходной выборки (на несколько порядков). Получилось ли улучшить результат? Добавьте соответствующие графики и выводы.

11. (0.5 балла) Реализуйте подсчет функции потерь и градиента для разреженного автокодировщика с помощью библиотек, задействующих вычисления на `GPU` — например, `CUDAMat` иди `gnumpy`. Включите в отчет данные о полученном приросте скорости вместе с соответствующими графиками.

12. (1 балл) Попробуйте использовать в качестве функции активации Rectified Linear Unit — выведите соответствующие формулы для обратного распространения ошибки и проверьте, что его использование ускоряет сходимость при минимизации функции потерь. Как это влияет на качество?

3 Данные

Предлагается использовать датасет STL-10[2], данные в котором получены на основе еще большего популярного датасета ImageNet[3], откуда были выбраны только 10 метаклассов. Большая часть изображений (>90%) приведена без меток классов, что позволяет в полной мере задействовать парадигму предобучения без учителя. Примеры изображений приведены в заголовке документа. В исходном виде данные представляют из себя либо mat-файлы, либо бинарные файлы. Для удобства по адресу https://yadi.sk/d/tHsZGxKIeSNsL данные были выложены в виде следующих pickle-файлов:

- `train.pk` (8000 размеченных изображений размера 96×96),
- `test.pk` (5000 размеченных изображений размера 96×96),
- `unlabeled.pk` (100000 неразмеченных изображений размера 96×96).

[2] http://www-cs-faculty.stanford.edu/~acoates/stl10/
[3] http://image-net.org/

Каждый pickle-файл содержит словарь (`dict`) с некоторыми из полей (какие из них имеются в наличии, можно узнать с помощью метода `.keys()`) `X, y, class_names, fold_indices` — они соотвествуют матрице с изображениями, меткам классов, соответствию классов и их имен и индексам для разбивки файла train (на сайте приведена особая процедура валидации алгоритмов для этого датасета). Вы можете использовать и другие наборы данных (например, CIFAR-10) для проведения дополнительных экспериментов.

UPD. Исходные pickle-файлы перенесены в поддиректорию «34/» на Яндекс.Диске, которую и рекомендуется использовать. Дополнительно также создана поддиректория «27/», в которой есть аналогичные файлы `train.pk` и `test.pk`, сохраненные с использованием устаревшего протокола, которые можно загружать с помощью python 2.7. Там же файл `unlabeled.pk` разбит на 5 промежуточных файлов «X*.pk», каждый из которых содержит по 20000 неразмеченных изображений в виде numpy-матриц.

4 Требования к реализации

При работе разрешается использовать сторонние пакеты `numpy`, `scipy`, `matplotlib` и `Pillow`. При реализации на `GPU` можно воспользоваться библиотеками `CUDAMat` иди `gnumpy`. Постарайтесь уделить особое внимание оптимизации кода, используйте векторизацию и матричные вычисления, где это возможно. При выделении патчей из изображения не подгружайте лишний раз изображения для выделения каждого нового патча.

Для сдачи задания необходимо предоставить:

1. Отчет в формате pdf (оформленный в системе LaTeX) с описанием всех проведенных исследований со всеми графиками и выводами.
2. IPython notebook с кодом для воспроизведения всех результатов из отчета: таблиц, графиков и пр.
3. Python модули со всеми требуемыми функциями в соответствии со спецификациями, приведенными ниже.

Спецификация

Среди предоставленных файлов должны быть следующие модули и функции в них:

1. Модуль `sample_patches.py` для получения выборки из патчей изображений с функциями:
   (a) `normalize_data(images)`
   Описание параметров:
   - `images` — переменная типа `numpy.ndarray`, матрица размера $N \times D$, содержащая N изображений или патчей в своих строчках, каждое изображение — вектор каждое изображение — вектор, который соответствующем решейпом можно привести к виду $d \times d \times 3$. Например, это может быть матрица размера 100×192 для нормализации 100 скрытых нейронов или патчей 8×8 или матрица размера 100×27648 для нормализации 100 исходных изображений из STL-10.

   Функция возвращает матрицу того же размера с изображениями, нормализованными способом, указанным во втором пункте задания.

   (b) `sample_patches_raw(images, num_patches=10000, patch_size=8)`
   Описание параметров:
   - `images` — `numpy.ndarray`-матрица с исходными изображениями по строкам.
   - `num_patches` — количество патчей для генерации.
   - `patch_size` — длина/ширина одного патча.

   Функция возвращает матрицу размера $N \times D$ с изображениями по строчкам аналогично исходным данным, где $N =$ `num_patches`, а D можно вычислить как $D = 3 \times$ `patch_size`^{2} по размеру патчей и количеству каналов. Для генерации патчей используются приведенный выше датасет.

   Особо отметим, что количество строк в `images` и количество строк N на выходе **не равны** друг другу — теоретически мы можем набрать 10000 патчей и из 10 картинок, но чем больше картинок, тем разнообразней получаются патчи.

   (c) `sample_patches(images, num_patches=10000, patch_size=8)`
   Функция аналогична приведенной выше за исключением того, что патчи должны быть дополнительно нормализованы функцией `normalize_data`.

2. Модуль `gradient.py` для численного подсчета градиента с функциями:
   (a) `compute_gradient(J, theta)`
   Описание параметров:

- `J` — переменная типа `function`, целевая функция J.
- `theta` — переменная типа `numpy.ndarray` из домена J', точка, в которой необходимо подсчитать градиент.

Функция возвращает приближенное разностным отношением значение градиента в указанной точке в виде переменной типа `numpy.ndarray` той же размерности.

(b) `check_gradient()`

Функция должна проверять корректность реализации предыдущей функции. В частности, можно реализовать дополнительно функцию, для которой заранее можно подсчитать точное значение градиента и сравнить с численным.

3. Модуль `autoencoder.py` с реализацией автокодировщика с функциями:

(a) `initialize(hidden_size, visible_size)`

Описание параметров:

- `hidden_size` — переменная типа `numpy.ndarray`, симметричный одномерный вектор, описывающий количество нейронов на скрытых слоях нейросети.
- `visible_size` — переменная типа `int`, количество нейронов на выходном/выходном слоях.

Функция возвращает `numpy.ndarray`-вектор с проинициализированными начальными параметрами для обучения автокодировщика, вытянутыми в один вектор-параметр. «Вытягивание» можно произвести за счет перевода каждой матрицы в вектор с помощью `np.reshape`, а затем объединения всех векторов в один с помощью `np.concatenate` — к примеру, сначала будут идти «вытянутые» $W^{(l)}$, а затем соответствующие $b^{(l)}$.

(b) `autoencoder_loss(theta, visible_size, hidden_size, lambda_, sparsity_param, beta, data)`

Описание параметров:

- `theta` — переменная типа `numpy.ndarray`, вектор-параметр автокодировщика, аналогичный выходу функции `initialize`.
- `visible_size, hidden_size` — то же, что и на входе функции `initialize`.
- `lambda_, sparsity_param, beta` — переменные типа `float`, параметры функции потерь и обучения автокодировщика.
- `data` — переменная типа `numpy.ndarray`, объекты выборки, матрица размера $N \times D$, содержащая N объектов в своих строчках, в нашем случае — каждый объект суть вектор длины $8 \times 8 \times 3$ (т.е. $D = 192$) по размеру патча и количеству каналов. Количество патчей будет меняться ($N = 1000, 10000$ и т.п.)

Функция возвращает значение функции потерь и её градиент в виде пары (`tuple`) переменных типа `float` и `numpy.ndarray` соответственно.

(c) `autoencoder_transform(theta, visible_size, hidden_size, layer_number, data)`

Описание параметров:

- `theta, visible_size, hidden_size, data` — аналогично предыдущей функции
- `layer_number` — номер слоя, на основе которого получать приближение.

Функция преобразует входные данные на основе обученных параметров автокодировщика.

4. Модуль `display_layer.py` для визуализации исходных патчей и скрытых нейронов сети с функциями:

(a) `display_layer(X, filename='layer.png')`

Описание параметров:

- `X` — переменная типа `numpy.ndarray`, матрица размера $N \times D$, содержащая N изображений, их частей или значений активации в своих строчках, каждое изображение — вектор, который соответствующем решейпом можно привести к виду $d \times d \times 3$. Например, это может быть матрица размера 100×192 для визуализации 100 скрытых нейронов или патчей 8×8 или матрица размера 16×27648 для визуализации 16 исходных изображений из STL-10.
- `filename` — путь для сохранения визуализации.

Функция сохраняет визуализацию переданных патчей/фильтров в отдельном изображении. Пример визуализации см. в заголовке документа.

Дополнительный необходимый функционал вы также можете включать в отдельные модули, либо можно описывать функции отдельными клетками в IPython-ноутбуках.

5 История изменений

21 февраля

1. Перезалиты файлы данных, большая неразмеченная выборка разбита на мелкие файлы.
2. Добавлено указание по нормализации данных.
3. Добавлено замечание по поводу «нестохастичности» подсчета градиента.
4. Добавлены указания по численной аппроксимации градиента и её проверке.
5. Вынесены отдельно указания для проверки правильности реализации.
6. Исправлено описание функции `sample_patches` — размер выхода задается *вторым* параметром.