ВМК МГУ

Задание 1. Автоматическое дифференцирование для автокодировщика

Курс: Глубинное обучение, осень 2017

Начало выполнения задания: 22 сентября
Срок сдачи: **8 октября (воскресенье), 23:59.**
Среда для выполнения задания – PYTHON 3.

Содержание

1 Необходимая теория

1.1 Модель автокодировщика

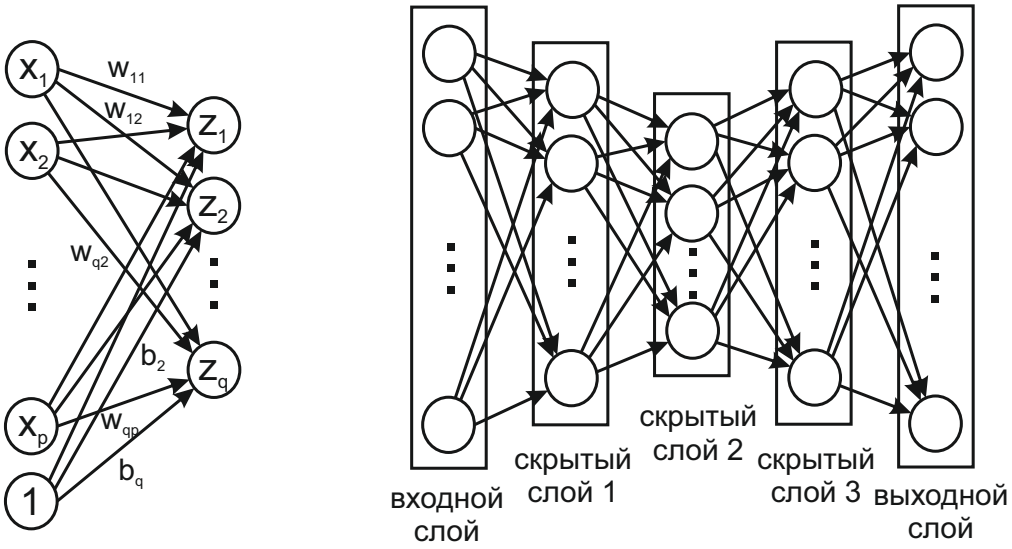
Рассмотрим задачу понижения размерности в данных. Пусть имеется выборка из n объектов $X = \{\boldsymbol{x}_i\}_{i=1}^N$, где $\boldsymbol{x}_i \in \mathbb{R}^D$. Задача состоит в том, чтобы найти представление для выборки X в пространстве меньшей размерности с минимальной потерей информации, т.е. найти $T = \{\boldsymbol{t}_i\}_{i=1}^n$, где $\boldsymbol{t}_i \in \mathbb{R}^d$ и $d \ll D$.

Одним из способов решения этой задачи является обучение модели автокодировщика. Для задания модели автокодировщика рассмотрим сначала полносвязный слой нейронной сети, показанный на рис. 1, слева. Данный слой принимает на вход вектор $\boldsymbol{x} \in \mathbb{R}^p$ размерности p и возвращает вектор $\boldsymbol{z} \in \mathbb{R}^q$ размерности q. При этом каждый выходной элемент (нейрон) z_i вычисляется с помощью 1) линейной комбинации входящих в него элементов x_j с весами w_{ij} с добавлением сдвига b_i и 2) применения к результату линейной комбинации некоторой одномерной нелинейной функции активации g:

$$\boldsymbol{z} = g(W\boldsymbol{x} + \boldsymbol{b}) = g(\overline{W}\bar{\boldsymbol{x}}),$$

где $W \in \mathbb{R}^{q\times p}$, $\boldsymbol{b} \in \mathbb{R}^q$ и $\overline{W} = [W, \boldsymbol{b}]$, $\bar{\boldsymbol{x}} = [\boldsymbol{x}^T, 1]^T$ – расширенные матрица весов и вектор входа. В качестве операции g может использоваться как тождественное преобразование, так и простая нелинейная функция, например, одна из функций из следующего списка:

1. $\sigma(s) = \frac{1}{1+\exp(-s)}$ – сигмоида,
2. $\tanh(s) = 2\sigma(2z) - 1$ – гиперболический тангенс,
3. $\mathrm{ReLU}(s) = \max(0, z)$ – rectified linear unit,
4. $\mathrm{LeakyReLU}(s) = \max(z, \alpha z),\ 0 < \alpha < 1.$



(a) (b)

Рис. 1: Слева: полносвязный слой нейронной сети, справа: пример автокодировщика с тремя скрытыми слоями

Модель автокодировщика состоит из нескольких полносвязных слоёв (см. рис. 1, справа). Входной и выходной слои состоят из D нейронов, симметричные скрытые слои имеют одинаковое число нейронов, а средний слой состоит из d нейронов. Обозначим через $\boldsymbol{z}^l$ – выход l-го скрытого слоя, где $l = 0, 1, 2, \ldots, L$, $\boldsymbol{z}^0 = \boldsymbol{x}$, $\boldsymbol{z}^L$ – выход последнего слоя. Обучение автокодировщика осуществляется путём решения следующей задачи оптимизации:

$$F(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i, \boldsymbol{\theta})) \to \min_{\boldsymbol{\theta}}. \tag{1}$$

Здесь в качестве функции потерь выступает $L(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|^2$, а через $\boldsymbol{\theta}$ обозначены параметры всех скрытых слоёв $\{\overline{W}^l\}_{l=1}^L$. Таким образом, автокодировщик обучается восстанавливать свой вход, пропуская вектора размерности D через узкий слой нейронов размерности d. После обучения выход нейронов среднего слоя автокодировщика для объекта $\boldsymbol{x} \in \mathbb{R}^D$ играет роль его представления меньшей размерности $\boldsymbol{t} \in \mathbb{R}^d$. Как правило, на среднем слое автокодировщика используется тождественная функция активации.

1.2 Вычисление градиента

Для решения задачи оптимизации (1) с помощью численных методов оптимизации необходимо уметь вычислять для текущего значения параметров $\boldsymbol{\theta}$ значение функции F, а также её градиент $\nabla_{\boldsymbol{\theta}} F$. Рассмотрим вычисление этих величин с помощью процедуры обратного распространения ошибки.

Для простоты ограничимся одним объектом выборки $\boldsymbol{x}$ и рассмотрим вычисление $L(\boldsymbol{x}, \boldsymbol{z}^L(\boldsymbol{x}, \boldsymbol{\theta}))$ вместе с $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}, \boldsymbol{z}^L(\boldsymbol{x}, \boldsymbol{\theta}))$. Для удобства представим модель автокодировщика в виде графа вычислений (см. рис. 2), в котором все параметры и вычисляемые величины являются вершинами. Тогда, зная $\boldsymbol{x}$ и параметры $\boldsymbol{\theta} = \{\overline{W}^l\}_{l=1}^L$, значение функции потерь L может быть вычислено с помощью следующего **прохода вперёд**:

$$\begin{aligned}
&\boldsymbol{z}^0 = \boldsymbol{x};\\
&\textbf{для } l = 1, \ldots, L:\\
&\quad \boldsymbol{u}^l = \overline{W}^l \bar{\boldsymbol{z}}^{l-1};\\
&\quad \boldsymbol{z}^l = g(\boldsymbol{u}^l);\\
&L = L(\boldsymbol{x}, \boldsymbol{z}^L).
\end{aligned} \tag{2}$$

Теперь рассмотрим вычисление $\nabla_{\overline{W}^l} L(\boldsymbol{x}, \boldsymbol{z}^L)$ для произвольного слоя l. Заметим, что функция потерь L зависит от параметров $\overline{W}^l$ только через $\boldsymbol{u}^l$. Тогда, рассматривая функцию потерь как сложную функцию

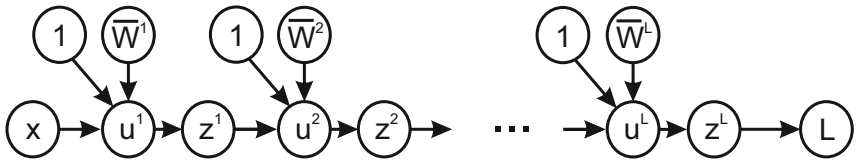


Рис. 2: Граф вычислений для автокодировщика

$L(\boldsymbol{u}^l(\overline{W}^l, \boldsymbol{z}^{l-1}))$, требуемую производную можно вычислить как

$$\frac{\partial L}{\partial \overline{W}^l_{ij}} = \frac{\partial L}{\partial u^l_i}\frac{\partial u^l_i}{\partial \overline{W}^l_{ij}} = \frac{\partial L}{\partial u^l_i}\bar{z}^{l-1}_j \;\Rightarrow\; \nabla_{\overline{W}^l} L = (\nabla_{\boldsymbol{u}^l} L)(\bar{\boldsymbol{z}}^{l-1})^T,$$
$$\frac{\partial L}{\partial z^{l-1}_i} = \sum_j \frac{\partial L}{\partial u^l_j}\frac{\partial u^l_j}{\partial z^{l-1}_i} = \sum_j \frac{\partial L}{\partial u^l_j} W^l_{ji} \;\Rightarrow\; \nabla_{\boldsymbol{z}^{l-1}} L = (W^l)^T \nabla_{\boldsymbol{u}^l} L.$$

Аналогично заметим, что функция L зависит от $\boldsymbol{u}^l$ только через $\boldsymbol{z}^l$, следовательно,

$$\frac{\partial L}{\partial u^l_i} = \frac{\partial L}{\partial z^l_i}\frac{\partial z^l_i}{\partial u^l_i} = \frac{\partial L}{\partial z^l_i} g'(u^l_i) \;\Rightarrow\; \nabla_{\boldsymbol{u}^l} L = \nabla_{\boldsymbol{z}^l} L \odot g'(\boldsymbol{u}^l).$$

Здесь через $\odot$ обозначено покомпонентное умножение. Таким образом мы получили, что требуемая производная $\nabla_{\overline{W}^l} L$ выражается через $\nabla_{\boldsymbol{z}^l} L$, которая, в свою очередь, выражается через $\nabla_{\boldsymbol{z}^{l+1}} L$ и так далее. Последняя производная $\nabla_{\boldsymbol{z}^L} L$ вычисляется непосредственно $\nabla_{\boldsymbol{z}^L} L = \boldsymbol{z}^L - \boldsymbol{x}$. В результате мы получаем схему вычисления всех производных $\nabla_{\overline{W}^l} L$ с помощью **прохода назад**:

$$\begin{aligned}
&\nabla_{\boldsymbol{z}^L} L = \boldsymbol{z}^L - \boldsymbol{x};\\
&\nabla_{\boldsymbol{u}^L} L = \nabla_{\boldsymbol{z}^L} L \odot g'(\boldsymbol{u}^L);\\
&\nabla_{\overline{W}^L} L = (\nabla_{\boldsymbol{u}^L} L)(\bar{\boldsymbol{z}}^{L-1})^T;\\
&\textbf{для } l = L-1, L-2, \ldots, 1:\\
&\quad \nabla_{\boldsymbol{z}^l} L = (W^{l+1})^T \nabla_{\boldsymbol{u}^{l+1}} L;\\
&\quad \nabla_{\boldsymbol{u}^l} L = \nabla_{\boldsymbol{z}^l} L \odot g'(\boldsymbol{u}^l);\\
&\quad \nabla_{\overline{W}^l} L = (\nabla_{\boldsymbol{u}^l} L)(\bar{\boldsymbol{z}}^{l-1})^T.
\end{aligned} \tag{3}$$

Для применения прохода назад необходимо хранить величины $\{\boldsymbol{u}^l, \boldsymbol{z}^l\}_{l=1}^L$, вычисленные при проходе вперёд.

Для сокращения количества настраиваемых параметров в автокодировщике зачастую используется приём зацепления соответствующих весов. Для автокодировщика на рис. 1, справа зацепление весов происходит по правилу $\overline{W}^3 = (\overline{W}^1)^T$, т.е. веса симметричных слоёв совпадают между собой. Рассмотрим, как зацепление весов влияет на процедуру прохода назад для вычисления градиентов функции потерь L. Построим для этого случая граф вычислений (см. рис. 3). По данному графу очевидно, что

$$\frac{\partial L}{\partial \overline{W}^1_{ij}} = \frac{\partial L}{\partial u^1_i}\frac{\partial u^1_i}{\partial \overline{W}^1_{ij}} + \frac{\partial L}{\partial u^3_i}\frac{\partial u^3_i}{\partial \overline{W}^1_{ij}} \;\Rightarrow\; \nabla_{\overline{W}^1} L = (\nabla_{\boldsymbol{u}^1} L)(\bar{\boldsymbol{z}}^0)^T + \bar{\boldsymbol{z}}^2(\nabla_{\boldsymbol{u}^3} L)^T.$$

При этом все производные $\nabla_{\boldsymbol{u}^l} L$ последовательно вычисляются для $l = L, L-1, \ldots, 1$, как в описанном выше проходе назад (3).

Описанная модификация эквивалентна запуску стандартного прохода назад (3) и вычислению производной функции L по весам $\overline{W}^1$ как суммы $\nabla_{\overline{W}^1} L$ и $(\nabla_{\overline{W}^3} L)^T$.

1.3 Вычисление произведения гессиана на вектор

В современных методах оптимизации из семейства т.н. безгессианных методов (Hessian-free optimization, см. [1]) помимо вычисления значения функции F и её градиента $\nabla_{\boldsymbol{\theta}} F$ требуется также уметь вычислять $\nabla^2 F(\boldsymbol{\theta})\boldsymbol{p}$ –

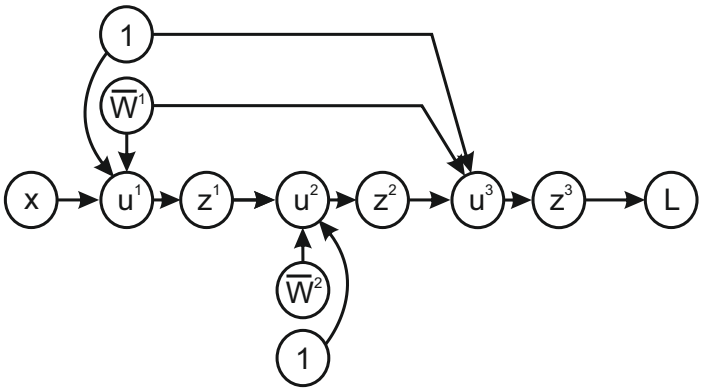


Рис. 3: Граф вычислений для автокодировщика на рис. 1, справа, с зацепленными весами

произведение гессиана функции на произвольный вектор $\boldsymbol{p}$. Рассмотрим модификацию проходов вперёд и назад для организации данного вычисления (подробнее см. [2]). Для этого снова рассмотрим один объект $\boldsymbol{x}$ и функцию потерь на этом объекте $L(\boldsymbol{x}, \boldsymbol{z}^L(\boldsymbol{x}, \boldsymbol{\theta}))$. Введём следующие операторы $R_{\boldsymbol{p}}\{\cdot\}$:

$$R_{\boldsymbol{p}}\{u_i^l\} = (\nabla_{\boldsymbol{\theta}} u_i^l)^T \boldsymbol{p}, \quad R_{\boldsymbol{p}}\{z_i^l\} = (\nabla_{\boldsymbol{\theta}} z_i^l)^T \boldsymbol{p}.$$

Смысл операторов $R_{\boldsymbol{p}}\{\cdot\}$ – значение производной функции-аргумента по заданному направлению $\boldsymbol{p}$. $R_{\boldsymbol{p}}\{\cdot\}$ – это оператор дифференцирования, следовательно, к нему применимы стандартные правила вычисления производных. Вектор параметров автокодировщика $\boldsymbol{\theta}$ состоит из $\{\overline{W}^1, \ldots, \overline{W}^L\}$. Разобьём вектор $\boldsymbol{p}$ по аналогичным компонентам $\{\overline{P}^1, \ldots, \overline{P}^L\}$. Здесь $\overline{P}^l$ – матрицы тех же размеров, что и $\overline{W}^l$. Тогда для вычисления всех величин $R_{\boldsymbol{p}}\{\boldsymbol{u}^l\}$ и $R_{\boldsymbol{p}}\{\boldsymbol{z}^l\}$ проход вперёд (2) модифицируется следующим образом:

$$\begin{aligned}
&R_{\boldsymbol{p}}\{\boldsymbol{z}^0\} = \boldsymbol{0};\\
&\textbf{для } l = 1, \ldots, L:\\
&\quad R_{\boldsymbol{p}}\{\boldsymbol{u}^l\} = \overline{W}^l R_{\boldsymbol{p}}\{\bar{\boldsymbol{z}}^{l-1}\} + \overline{P}^l \bar{\boldsymbol{z}}^{l-1};\\
&\quad R_{\boldsymbol{p}}\{\boldsymbol{z}^l\} = g'(\boldsymbol{u}^l) \odot R_{\boldsymbol{p}}\{\boldsymbol{u}^l\};\\
&R_{\boldsymbol{p}}\{L\} = \nabla_{\boldsymbol{z}^L} L(\boldsymbol{x}, \boldsymbol{z}^L)^T R_{\boldsymbol{p}}\{\boldsymbol{z}^L\}.
\end{aligned} \tag{4}$$

Здесь $R_{\boldsymbol{p}}\{\bar{\boldsymbol{z}}^l\} = [R_{\boldsymbol{p}}\{\boldsymbol{z}^l\}^T, 0]^T$. Данная схема получается путём применения оператора дифференцирования $R_{\boldsymbol{p}}\{\cdot\}$ к каждой строчке алгоритма прохода вперёд (2).

Величина $\nabla^2_{\boldsymbol{\theta}} L(\boldsymbol{x}, \boldsymbol{z}^L(\boldsymbol{x}, \boldsymbol{\theta}))\boldsymbol{p}$ представляет собой вектор, который может быть разбит по компонентам по аналогии с $\boldsymbol{\theta}$. Нетрудно показать, что эти компоненты вычисляются как $\{R_{\boldsymbol{p}}\{\nabla_{\overline{W}^1} L\}, \ldots, R_{\boldsymbol{p}}\{\nabla_{\overline{W}^L} L\}\}$. Все требуемые здесь величины могут быть вычислены с помощью применения оператора $R_{\boldsymbol{p}}\{\cdot\}$ к каждой строчке алгоритма прохода назад (3). Заметим, что для применения данного алгоритма необходимо предварительно провести алгоритмы (2), (3) и (4).

1.4 Гаусс-ньютоновская аппроксимация гессиана

При прямом применении метода Ньютона и его безгессианной версии к задачам невыпуклой оптимизации возникают трудности в ситуациях, когда гессиан $\nabla^2 F(\boldsymbol{\theta})$ в текущей точке не является положительно-определённой матрицей (подробнее см. [3], глава 3). Для избежания подобных трудностей внутрь метода Ньютона, как правило, добавляется специальная процедура, позволяющая для текущей точки $\boldsymbol{\theta}$ скорректировать гессиан до положительно-определённой матрицы. Одной из наиболее популярных процедур подобного вида в машинном обучении является использование вместо матрицы гессиана её т.н. гаусс-ньютоновской аппроксимации [2].

Рассмотрим задачу оптимизации (1):

$$F(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i, \boldsymbol{\theta})) \to \min_{\boldsymbol{\theta}}.$$

Вычислим значение градиента и гессиана функции F:

$$\frac{\partial F}{\partial \theta_q} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D}\frac{\partial L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}))}{\partial z_j^L}\frac{\partial z_j^L}{\partial \theta_q},$$

$$\frac{\partial^2 F}{\partial \theta_p \partial \theta_q} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D}\left[\sum_{k=1}^{D}\frac{\partial^2 L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}))}{\partial z_k^L \partial z_j^L}\frac{\partial z_k^L}{\partial \theta_p}\frac{\partial z_j^L}{\partial \theta_q} + \frac{\partial L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}))}{\partial z_j^L}\frac{\partial^2 z_j^L}{\partial \theta_p \partial \theta_q}\right].$$

Последнее выражение для гессиана в векторном виде можно представить как

$$\nabla^2_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T \nabla^2_{\boldsymbol{z}^L} L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}))\left(\frac{\partial \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right) + \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D}\frac{\partial L(\boldsymbol{x}_i, \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}))}{\partial z_j^L}\nabla^2_{\boldsymbol{\theta}} z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta}).$$

Здесь через $J_i = \frac{\partial \boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left\{\frac{\partial z_j^L}{\partial \theta_k}\right\}_{j,k=1}^{D,M}$ обозначен якобиан преобразования выходов автокодировщика на i-ом объекте выборки по всем параметрам $\boldsymbol{\theta}$, M – длина вектора $\boldsymbol{\theta}$. С учётом того, что $L(\boldsymbol{x},\boldsymbol{z}) = \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2$, получаем $\nabla_{\boldsymbol{z}} L(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{z}-\boldsymbol{x}$, $\nabla^2_{\boldsymbol{z}} L(\boldsymbol{x},\boldsymbol{z}) = I$. Следовательно, окончательно гессиан F может быть записан как

$$\nabla^2 F(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} J_i^T J_i + \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D}(z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta}) - x_{ij})\nabla^2_{\boldsymbol{\theta}} z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta}). \tag{5}$$

Назовём **гаусс-ньютоновской аппроксимацией гессиана** первое слагаемое в выражении (5):

$$\nabla^2_{GN} F(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} J_i^T J_i.$$

Заметим, что введённая аппроксимация по построению всегда является неотрицательно-определённой матрицей. Кроме того, если предположить, что при оптимальных параметрах автокодировщика $\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta}) \approx \boldsymbol{x}_i$, второе слагаемое в выражении (5) обнуляется, и гаусс-ньютоновская аппроксимация в пределе переходит в гессиан. Также в ситуации автокодировщика с одним слоем и линейной функцией активации на нём преобразование $\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})$ является линейным по параметрам $\boldsymbol{\theta}$. Следовательно, $\nabla^2_{\boldsymbol{\theta}} z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta}) = \mathbf{0}$, и гаусс-ньютоновская аппроксимация становится точным выражением для гессиана.

На практике в безгессианном методе Ньютона для обучения автокодировщика [1] вместо гессиана F на каждом шаге используется его гаусс-ньютоновская аппроксимация. Соответственно, для работы метода оптимизации здесь необходимо реализовать процедуру умножения гаусс-ньютоновской аппроксимации на произвольный вектор $\boldsymbol{p}$:

$$\nabla^2_{GN} F(\boldsymbol{\theta})\boldsymbol{p} = \frac{1}{N}\sum_{i=1}^{N} J_i^T J_i \boldsymbol{p}.$$

Для реализации данной формулы достаточно уметь вычислять произведение якобиана J_i на вектор $\boldsymbol{p}$, а затем произведение транспонированного якобиана J_i^T на вектор $\boldsymbol{q}_i = J_i\boldsymbol{p}$. Обе эти операции могут быть реализованы с помощью соответствующих проходов вперёд-назад.

Для реализации первой операции можно заметить, что

$$(J_i\boldsymbol{p})_j = \sum_{k=1}^{M}\frac{\partial z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta})}{\partial \theta_k}p_k = \nabla z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta})^T\boldsymbol{p} = R_{\boldsymbol{p}}\{z_j^L\},\ \Rightarrow\ J_i\boldsymbol{p} = R_{\boldsymbol{p}}\{\boldsymbol{z}^L\}.$$

Таким образом, для вычисления произведения якобиана на вектор $\boldsymbol{p}$ достаточно выполнить алгоритм (4).

Для реализации второй операции проведём аналогичные вычисления:

$$(J_i^T\boldsymbol{q})_k = \sum_{j=1}^{D}\frac{\partial z_j^L(\boldsymbol{x}_i,\boldsymbol{\theta})}{\partial \theta_k}q_j = \frac{\partial}{\partial \theta_k}\sum_{j=1}^{D} z_k^L(\boldsymbol{x}_i,\boldsymbol{\theta})\theta_k,\ \Rightarrow\ J_i^T\boldsymbol{q} = \nabla_{\boldsymbol{\theta}}\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})^T\boldsymbol{q}.$$

Окончательно,

$$\frac{1}{N}\sum_{i=1}^{N} J_i^T\boldsymbol{q}_i = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})^T\boldsymbol{q}_i = \nabla_{\boldsymbol{\theta}}\left[\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})^T\boldsymbol{q}_i\right].$$

В результате здесь достаточно выполнить проход назад (3) для новой функции $G(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{z}^L(\boldsymbol{x}_i,\boldsymbol{\theta})^T\boldsymbol{q}_i$.

1.5 Численный контроль градиентов

На практике для проверки корректности вычисления функции F и её градиента $\nabla_{\boldsymbol{\theta}} F$ удобно пользоваться следующим результатом:

$$\nabla F(\boldsymbol{\theta})^T \boldsymbol{p} \approx \frac{F(\boldsymbol{\theta} + \varepsilon \boldsymbol{p}) - F(\boldsymbol{\theta})}{\varepsilon}. \tag{6}$$

Здесь ε – достаточно маленькое положительное число. Рекомендуемое значение $\varepsilon = \sqrt{\varepsilon_{mach}}$, где ε_{mach} – машинная точность. При таком выборе ε разница между значением выражения $\nabla F(\boldsymbol{\theta})^T \boldsymbol{p}$ и его разностной аппроксимацией должно быть порядка $C\sqrt{\varepsilon_{mach}}$, где C – это оценка сверху на абсолютное значение функции F и $\|\nabla^2 F(\boldsymbol{\theta})\|$ в малой окрестности $\boldsymbol{\theta}$. Аналогичный разностный результат можно использовать для проверки корректности вычисления произведения гессиана на вектор:

$$\nabla^2 F(\boldsymbol{\theta}) \boldsymbol{p} \approx \frac{\nabla F(\boldsymbol{\theta} + \varepsilon \boldsymbol{p}) - \nabla F(\boldsymbol{\theta})}{\varepsilon}.$$

В том случае, если для используемой функции F константа C оказывается достаточно большой, оценка корректности реализации вычислений по формуле (6) может оказаться затруднительной. Альтернативным методом здесь является использование специальной комплексной точки [4]:

$$\nabla F(\boldsymbol{\theta})^T \boldsymbol{p} \approx \frac{\mathrm{Im}[F(\boldsymbol{\theta} + i\varepsilon \boldsymbol{p})]}{\varepsilon}. \tag{7}$$

В данной формуле i – это мнимая единица, $\mathrm{Im}[\cdot]$ – мнимая часть аргумента. В методе (7) за счёт отсутствия операции вычитания в числителе допускается использование ε порядка 10^{-12} и меньше, а разница между $\nabla F(\boldsymbol{\theta})^T \boldsymbol{p}$ и его оценкой (7) практически не зависит от констант типа C и приближается к ε_{mach}.

2 Формулировка задания

Для выполнения задания необходимо:

1. Для модели автокодировщика реализовать процедуру вычисления функции $F(\boldsymbol{\theta})$ (1) с помощью прохода вперёд (2), а также её градиента с помощью прохода назад (3). Проверить корректность вычислений с помощью разностной аппроксимации (6) или (7);

2. Выписать формулы для прохода назад для оператора $R_{\boldsymbol{p}}\{\cdot\}$, полученные формулы вставить в отчёт. Реализовать процедуру вычисления произведения гессиана функции $F(\boldsymbol{\theta})$ на произвольный вектор $\boldsymbol{p}$. Проверить корректность вычислений с помощью разностной аппроксимации;

3. Реализовать процедуру вычисления произведения гаусс-ньютоновской аппроксимации гессиана функции $F(\boldsymbol{\theta})$ на произвольный вектор $\boldsymbol{p}$. Для реализованного кода проверить корректность вычислений;

4. Реализовать набор стохастических методов оптимизации: 1) стохастический градиентный спуск (SGD) [5], 2) SGD+momentum [6], 3) RMSprop [6] и 4) ADAM [7].

5. Провести исследование с реализованной моделью автокодировщика и методами стохастической оптимизации. Рассмотреть набор данных `MNIST`[1], а также несколько архитектур автокодировщика, в которых на среднем слое находится два нейрона и используется линейная функция потерь. Для различных методов оптимизации построить графики сходимости значения функции потерь на валидационной выборке в зависимости от числа эпох[2]. Прокомментировать использованный способ настройки длины шага и размера мини-батча. Какой из методов оптимизации показывает наилучшие результаты? Визуализировать значения выходов со среднего слоя (там, где два нейрона) для начальных значений параметров автокодировщика и для параметров, найденных в результате оптимизации.

6. Составить отчёт в формате PDF о результатах всех проведённых исследований.

[1] http://yann.lecun.com/exdb/mnist/

[2] для ускорения процедур обучения можно использовать подмножество выборки MNIST

3 Спецификация

Вместе с заданием выдаётся набор базовых классов, которые следует использовать для собственной реализации 1) функций активации, 2) полносвязных слоёв, 3) нейронной сети прямого распространения и 4) модели автокодировщика. Ниже представлено описание основных функций. Полные прототипы см. в прилагаемых к заданию py-файлах.

3.1 Функции активации

Модуль: activations.py
Базовый класс BaseActivationFunction. Все функции активации должны быть реализованы в виде отдельных классов, наследованных от базового.

Вычисление значения функции активации на заданном наборе входов:

```
def val(self, X)
```

- `X` — набор входов, переменная типа `numpy.array` (число, вектор или матрица).

Функция возвращает массив того же размера, что и `X`.

Функции `deriv` и `second_deriv` вычисляют первую и вторую производную функции активации соответственно. Их прототипы аналогичны `val`.

Если в реализуемой функции активации есть параметры (например, параметр α в LeakyReLU), то их следует вносить в конструктор класса.

При реализации различных функций активации рекомендуется проводить тестирование первой и второй производной с помощью разностных методов из раздела 1.5.

3.2 Полносвязные слои

Модуль: layers.py
Базовый класс BaseLayer. Полносвязный слой должен быть реализован в виде отдельного класса FCLayer, наследованного от базового.

Проход вперёд для заданного батча входных данных:

```
def forward(self, inputs)
```

- `inputs` — набор входов, переменная типа `numpy.array` размера число_входов × размер_батча.

Функция возвращает массив типа `numpy.array` размера число_выходов × размер_батча. Предполагается, что все вычисляемые величины, которые понадобятся для прохода назад и других проходов по сети, сохраняются в память слоя.

Проход назад для заданных производных функции потерь по выходам слоя. Предполагается, что проход вперёд для данного слоя уже проведён.

```
def backward(self, derivs)
```

- `derivs` — набор производных функции потерь по выходам слоя, переменная типа `numpy.array` размера число_выходов × размер_батча.

Функция возвращает два элемента: 1) производные функции потерь по входам слоя в виде массива типа `numpy.array` размера число_входов × размер_батча и 2) производные функции потерь по весам слоя в виде вектора типа `numpy.array` длины число_параметров_слоя. Все вычисляемые величины, которые понадобятся для R_p прохода назад, сохраняются в память слоя.

Функции `Rp_forward` и `Rp_backward` вычисляют R_p выходы слоя и R_p производные функции потерь по входам слоя. Их прототипы аналогичны `forward` и `backward`. При вызове `Rp_forward` предполагается, что проход вперёд для данного слоя уже проведён. При вызове `Rp_backward` предполагается, что для данного слоя уже проведены проход вперёд, R_p проход вперёд и проход назад.

3.3 Сеть прямого распространения

Модуль: ffnet.py. Этот модуль предоставляется в полностью реализованном виде. Собственной реализации здесь не требуется. Модуль реализует класс FFNet, задающий произвольную нейронную сеть прямого распространения.

Конструктор класса:

```
def __init__(self, layers)
```

- `layers` — набор слоёв сети, список из экземпляров класса, наследованного от BaseLayer.

Установка параметров сети. Данная функция принимает одномерный вектор из всех значений параметров сети и присваивает его соответствующие элементы параметрам каждого слоя сети, преобразованных к требуемой форме (например, матрице).

```
def set_weights(self, w)
```

- `w` — параметры сети, одномерный массив типа `numpy.array` длины num_params.

Считывание текущих параметров сети.

```
def get_weights(self)
```

Установка вектора направления для его дальнейшего умножения на гессиан или его гаусс-ньютоновскую аппроксимацию. Функция аналогична `set_weights`.

```
def set_direction(self, p)
```

- `p` — вектор направления для параметров сети, одномерный массив типа `numpy.array` длины num_params.

Проход вперёд по сети:

```
def compute_outputs(self, inputs)
```

- `inputs` — входной набор данных, матрица типа `numpy.array` размера num_inputs × num_objects.

Функция возвращает значения выходов сети для входных данных, матрица типа `numpy.array` размера num_outputs × num_objects.

Проход назад по сети:

```
def compute_loss_grad(self, derivs)
```

- `derivs` — значения производной скалярной функции потерь по всем выходам сети для всех объектов, матрица типа `numpy.array` размера num_outputs × num_objects.

Функция возвращает значения производных функции потерь по всем параметрам сети, вектор типа `numpy.array` длины num_params.

Функции `compute_Rp_outputs` и `compute_loss_Rp_grad` осуществляют R_p проходы вперёд и назад по сети. Их прототипы аналогичны функциям `compute_outputs` и `compute_loss_grad`.

Считывание активаций с заданного слоя. Предполагается, что проход вперёд по сети уже произведён.

```
def get_activations(self, layer_number)
```

- `layer_number` — номер слоя в сети.

Функция возвращает матрицу типа `numpy.array` размера layer_num_outputs × num_objects.

3.4 Автокодировщик

Модуль: autoencoder.py.
Класс Autoencoder, реализующий модель автокодировщика.

Конструктор класса:
```
def __init__(self, layers)
```

- `layers` — набор слоёв сети, список из экземпляров класса, наследованного от BaseLayer.

Вычисление функции потерь автокодировщика и её градиента для заданного мини-батча данных:
```
def compute_loss(self, inputs)
```

- `inputs` — входные данные, матрица типа `numpy.array` размера num_features × num_objects.

Функция возвращает 1) значение функции потерь, число и 2) значение градиента функции потерь, вектор типа `numpy.array` длины num_params.

Вычисление произведения гессиана функции потерь на произвольный вектор:
```
def compute_hessvec(self, p)
```

- `p` — произвольный вектор, вектор типа `numpy.array` длины num_params.

Функция возвращает результат – вектор типа `numpy.array` длины num_params.

Функция `compute_gaussnewtonvec` возвращает произведение гаусс-ньютоновской аппроксимации на произвольный вектор. Прототип аналогичен функции `compute_hessvec`.

Запуск метода оптимизации ADAM.
```
def run_adam(self, inputs, step_size=0.01, num_epoch=200, minibatch_size=100, l2_coef=1e-5,...
            test_inputs=None, display=False)
```

- `inputs` — входные данные, матрица типа `numpy.array` размера num_features × num_objects;
- `step_size` — длина шага;
- `num_epoch` — число эпох для оптимизации;
- `minibatch_size` — размер минибатча;
- `l2_coef` — коэффициент L_2-регуляризации;
- `test_inputs` — валидационная выборка, матрица типа `numpy.array` размера num_features × num_test_objects;
- `display` — флаг отображения номера текущей эпохи, текущего качества на обучении и тесте и проч.

Функция возвращает словарь с следующими полями:

- train_loss — значение оптимизируемого функционала на обучающей выборке в конце каждой эпохи, список;
- train_grad — норма градиента оптимизируемого функционала на обучающей выборке в конце каждой эпохи, список;
- test_loss — значение оптимизируемого функционала на валидационной выборке в конце каждой эпохи, список;
- train_grad — норма градиента оптимизируемого функционала на валидационной выборке в конце каждой эпохи, список.

4 Оформление задания

Выполненное задание следует загрузить в систему *anytask*. Присланный вариант задания должен содержать в себе:

- Текстовый файл в формате PDF с указанием ФИО и номера группы, содержащий необходимые формулы и описание всех проведённых исследований.
- Все исходные коды с необходимыми комментариями.

Список литературы

[1] J. Martens. Deep learning via Hessian-free optimization // ICML, 2010.

[2] N. Schraudolph. Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent // Neural Computation, V.14, 2002, pp. 1723–1738.

[3] J. Nocedal, S. Wright. Numerical optimization. Springer, 2006.

[4] W. Squire, G. Trapp. Using Complex Variables to Estimate Derivatives of Real Functions // SIAM Review, V.40, 1998, pp. 110–112.

[5] L. Bottou, F. Curtis, J. Nocedal. Optimization Methods for Large-Scale Machine Learning // ArXiv preprint 1606.04838, 2016.

[6] https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[7] D. Kingma, J. Ba. Adam: A Method for Stochastic Optimization // ArXiv preprint 1412.6980, 2014.