Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Чабаненко Владислав Дмитриевич

Модификации метода стохастического градиентного спуска для задач машинного обучения с большими объемами данных

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
научный сотрудник
Д.А. Кропотов

Москва, 2016

Содержание

1 Введение

Нейронные сети способны решать широкий круг задач машинного обучения — прогнозирование временных рядов [10], распознавание речи [9], компьютерное зрение [11] и т. д.

Актуальность проблемы обучения нейронных сетей в настоящее время связана с увеличением объемов данных, а так же архитектур сетей. В данной работе рассматривается *метод стохастического градиентного спуска (SGD)* [13] — стандартный метод для обучения нейронных сетей, а так же его наиболее популярные модификации[1]. Особое внимание в работе уделяется свежей разработке [1] для ускорения обучения нейронных сетей, называемой *батч-нормализацией*. Мотивация нашего исследования состоит в том, что авторы обучали нейронную сеть с батч-нормализацией стандартным методом SGD, но не рассматривали его различные модификации.

Цели работы:

- исследовать применение батч-нормализации к различным модификациям метода SGD;
- разработать рекомендации по использованию батч-нормализации.

Работа содержит два основных раздела: теоретический и экспериментальный. Теоретический раздел состоит из трех частей. Первая часть раскрывает суть искусственных нейронных сетей, описывает различные популярные архитектуры, кратко объясняет математическую модель нейронных сетей и способ их обучения. Во второй части описывается стандартный метод для обучения нейронных сетей — метод стохастического градиентного спуска, и указывается его основной недостаток. Затем вводятся модификации стандартного метода, направленные на частичное устранение этого недостатка. Третья часть поднимает еще одну проблему, возникающую при обучении нейронных сетей, и описывает свежий подход по ее решению, называемый батч-нормализацией. Раздел, посвященный экспериментам, состоит из четырех частей. Первые две части описывают наборы данных и архитектуры нейронных сетей, используемые в работе. В третьей части подробно описываются постановки экспериментов и выдвигаемые гипотезы. Последняя часть содержит результаты экспериментов и выводы из них.

[1] http://colinraffel.com/wiki/stochastic_optimization_techniques

2 Теоретическая часть

2.1 Нейронные сети в машинном обучении

Искусственные нейронные сети в машинном обучении (artificial neural networks) — это семейство моделей, созданных по подобию центральной нервной системы у животных.

Они представляют собой систему связанных между собой нейронов, обменивающихся друг с другом сигналами (нервными импульсами). Для передачи сигнала служит синапс (место контакта между двумя нейронами). Синапсы хранят параметры нейронной сети, называемые весами, с помощью которых происходят манипуляции с сигналами.

Нейронные сети используются для аппроксимации функций, которые в общем случае неизвестны, и которые могут зависеть от большого количества признаков. С помощью нейронных сетей можно решать такие интересные и сложные задачи машинного обучения, как прогнозирование временных рядов, распознавание речи и компьютерное зрение.

Нейронная сеть может быть отображена как ориентированный граф. Первый слой нейронной сети называют *входным слоем (input layer)*, последний — *выходным слоем (output layer)*. В общем случае нейронные сети могут иметь различную структуру слоев. Самая популярная архитектура — многослойный персептрон (рис. 1), в которой промежуточные слои называются *скрытыми слоями (hidden layers)*. Благодаря наличию скрытых слоев нейронная сеть позволяет аппроксимировать очень сложные нелинейные функции.

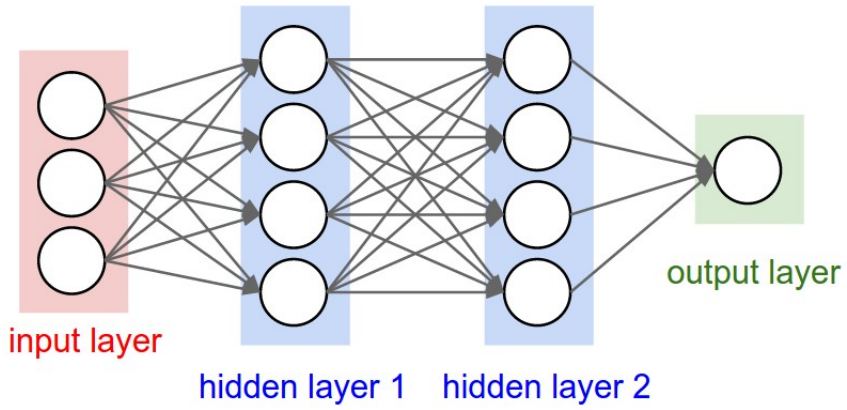

Рис. 1: Многослойный персептрон

Математически, функция нейронной сети $f(x)$ определяется как композиция функций $g_i(x)$, которые в свою очередь представляют композицию других функций. Широко распространен-

ный тип композиции — нелинейная взвешенная сумма $f(x) = \sigma(\sum_i w_i g_i(x))$, где g_i — активации нейронов с предыдущего слоя, w_i — веса линейной трансформации, а σ — некоторая заданная нелинейная функция активации. Наиболее часто используемые функции активации: *сигмоида (sigmoid)*

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{1}$$

или *выпрямленный линейный элемент (rectified linear unit)*

$$\sigma(x) = max(x, 0). \tag{2}$$

Нейронные сети по представлению графа разделяют на *сети прямого распространения (feedforward)* и *рекуррентные (recurrent)*. Первые представляют собой ациклический граф, вторые же, наоборот, соцержат циклы. Далее будут рассматриваться только сети прямого распространения.

Обычно, говоря про нейронные сети, подразумевают *полносвязные* (как на рис. 1), то есть когда в паре соседних слоев все нейроны связаны между собой.

Однако есть и другие архитектуры сетей. Большой популярностью пользуются *сверточные нейронные сети (convolutional neural networks)*. Эти нейронные сети специально спроектированы для работы с изображениями. Основная идея их применения — автоматическое выделение признаков исходного изображения. Отличия сверточной нейронной сети от полносвязной состоят в наличии так называемых *сверточных слоёв (convolutional layers)*, а также *слоёв, уменьшающих размерность (subsampling layers)*.

Свёрточный слой принимает на вход изображение X, а затем применяет к каждому окну изображения, фиксированного размера, свертку с ядром W, то есть преобразует каждый фрагмент исходного изображения в один новый пиксель Y (рис. 2, 3).

Таким образом, полученное изображение на выходе будет меньшего размера, и каждый нейрон на выходе будет связан лишь с некоторыми близлежащими нейронами входа (только с теми нейронами, которые участвуют в свертке).

Если к изображению будет применяться свертка только одним ядром, то сеть научится выделять только один вид признаков. Поэтому обычно сверточный слой состоит из нескольких изображений одинакового размера — *карт признаков (feature maps)*. Различаются карты только значениями параметров (весами свертки).

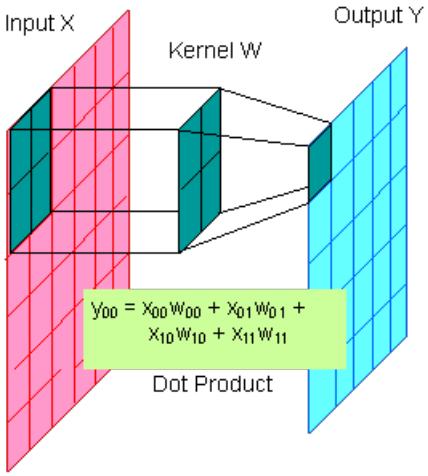

Рис. 2: Первый пиксель

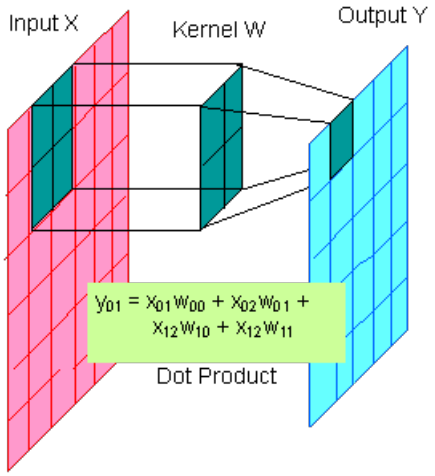

Рис. 3: Второй пиксель

Другой вид слоев *(subsampling layers)* используется для того, чтобы нейронная сеть была инвариантной к некоторым простым преобразованиям исходной картинки (небольшим сдвигам или искажениям). Простейший пример такого слоя: *макс-пулинг (max-pooling)*. В этом слое мы проходим по непересекающимся окнам $k \times k$ изображения и на выход посылаем только максимум из рассматриваемых значений. Таким образом, полученное изображение на выходе будет иметь ширину и высоту в k раз меньше.

Обычно для получения признаков более высокого порядка применяют многократно эти два слоя. Также в конце сети обычно помещают полносвязный слой, который будет принимать на вход полученные признаки и проводить классификацию изображения.

Рассмотрим вопрос обучения нейронных сетей. По данной задаче и классу функций F обучение означает использование множества наблюдений для нахождения функции $f^* \in F$, которая решает задачу в некотором оптимальном смысле.

Для определения оптимального решения вводится функционал эмпирического риска (например, как в (3)) $C : F \to \mathbb{R}$ такой, что $f^* : C(f^*) \leq C(f)\ \forall f \in F$, то есть нет других решений, для которых функция риска была бы меньше, чем для оптимального решения.

Когда решение зависит от данных, функция эмпирического риска должна быть функцией от наблюдаемых данных, иначе не будет возможности смоделировать ничего, соответствующего данным. Одна из классических задач машинного обучения — задача обучения с учителем: у нас есть набор данных X и ответы на данных Y, требуется настроить параметры модели, чтобы

как можно лучше описать наблюдаемые данные. Один из способов настройки нейронной сети на такую задачу: минимизировать отклонение предсказанных ответов на данных от реальных ответов, используя, например, среднеквадратичную функцию эмпирического риска:

$$\hat{C}(f) = \frac{1}{N} \sum_{i=1}^{N} \left(f(x_i, \theta) - y_i \right)^2 , \tag{3}$$

где оптимизация ведется по параметрам θ, если вид функции f зафиксирован.

Стандартный алгоритм для обучения нерйонных сетей — *алгоритм обратного распространений ошибки (backpropagation)* [12]. Вообще говоря, это метод для подсчета градиента по нейронной сети: от последнего слоя к первому поочередно вычисляется градиент функции риска по параметрам текущего слоя. Такое вычисление возможно благодаря тому, что нейронная сеть представляет собой композицию функций, и для нее можно использовать правило дифференцирования сложной функции[2].

2.2 Метод стохастического градиентного спуска и его модификации

Нейронные сети часто обучаются стохастически, то есть на разных итерациях используются разные части данных. Это мотивировано, как минимум, двумя причинами: во-первых, наборы данных, используемые для обучения, часто очень большие, чтобы хранить их полностью в оперативной памяти и/или производить вычисления эффективно; во-вторых, оптимизируемая функция обычно невыпуклая, таким образом, использование разных частей данных на каждой итерации может помочь от застревания модели в локальном минимуме. Кроме того, обучение нейронных сетей обычно производится с помощью градиентных методов первого порядка, так как из-за большого количества параметров в нейронной сети невозможно эффективно применять методы более высоких порядков.

Стандартным методом обучения нейронных сетей является метод стохастического градиентного спуска (SGD). Однако он может расходиться или сходиться очень медленно, если шаг обучения настроен недостаточно аккуратно. Поэтому существует много альтернативных методов с целью ускорить сходимость обучения и избавить пользователя от необходимости тщательной настройки гиперпараметров. Эти методы часто более эффективно вычисляют градиенты и

[2] https://en.wikipedia.org/w/index.php?title=Chain_rule&oldid=717805710

адаптивно изменяют шаг обучения по итерациям. Рассмотрим подробнее метод SGD и несколько его наиболее популярных модификаций.

Стохастический градиентный спуск (SGD) [13] обновляет каждый параметр, вычитая градиент оптимизируемой функции по соответствующему параметру и масштабируя его на *шаг обучения* η, являющийся гиперпараметром. Если η слишком большой, то метод будет расходиться; если слишком маленький — будет сходиться медленно. Правило пересчета:

$$\begin{aligned} i &\sim \mathcal{U}\{1, 2, \ldots, n\} \\ \theta_{t+1} &= \theta_t - \eta \nabla f_i(\theta_t), \end{aligned} \tag{4}$$

где f_i — функция, подсчитанная на i-ом *мини-батче* (части) данных, и индекс i выбирается случайным образом.

Стохастический градиентный спуск с инерцией (SGDm) [3]

В методе SGD градиент $\nabla f_i(\theta_t)$ часто быстро изменяется на каждой итерации, так как функционал вычисляется на разных данных. Это изменение частично смягчается, если использовать градиенты с прошлых итераций, масштабированные на некоторый *гиперпараметр инерции* μ по следующей формуле *(идея инерции)*:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f_i(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{5}$$

Метод адаптивного градиента (Adagrad) [4] эффективно перемасштабирует шаг обучения для каждого параметра в отдельности, учитывая историю всех прошлых градиентов для этого параметра *(идея масштабирования)*. Это делается путем деления каждого элемента в градиенте ∇f_i на квадратный корень суммы квадратов прошлых соответствующих элементов градиента. Перемасштабирование таким способом эффективно уменьшает шаг обучения для параметров, которые имеют большую величину градиента. Также метод уменьшает сам шаг обучения со временем, так как сумма квадратов увеличивается с каждой итерацией. При инициализации масштабирующего параметра $g = 0$ формула для пересчета имеет вид

$$
\begin{aligned}
g_{t+1} &= g_t + \nabla f_i(\theta_t)^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta \nabla f_i(\theta_t)}{\sqrt{g_{t+1} + \epsilon}},
\end{aligned}
\tag{6}
$$

где деление выполняется поэлементно, а ϵ — это небольшая константа, введенная для численной стабильности. Метод имеет хорошие теоретические гарантии и практические результаты [4]

Метод адаптивного скользящего среднего градиентов (RMSprop) [5] очень похож по принципу работы на метод Adagrad. Единственное его отличие в том, что шкалирующий член g_t вычисляется, как экспоненциальное скользящее среднее вместо кумулятивной суммы. Это делает g_t оценкой второго момента градиента ∇f и устраняет тот факт, что шаг обучения со временем уменьшается. Правило пересчета:

$$
\begin{aligned}
g_{t+1} &= \gamma g_t + (1 - \gamma) \nabla f_i(\theta_t)^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta \nabla f_i(\theta_t)}{\sqrt{g_{t+1} + \epsilon}}
\end{aligned}
\tag{7}
$$

Метод адаптивного шага обучения (Adadelta) [6] использует аналогичное экспоненциальное скользящее среднее для оценки второго момента градиента g_t, как и RMSprop. Также метод вычисляет скользящее среднее x_t по v_t, аналогичным инерции, но при обновлении этой величины, используется квадрат текущего шага. Обновление параметров происходит по следующим формулам:

$$
\begin{aligned}
g_{t+1} &= \gamma g_t + (1 - \gamma) \nabla f_i(\theta_t)^2 \\
v_{t+1} &= -\frac{\sqrt{x_t + \epsilon} \nabla f_i(\theta_t)}{\sqrt{g_{t+1} + \epsilon}} \\
x_{t+1} &= \gamma x_t + (1 - \gamma) v_{t+1}^2 \\
\theta_{t+1} &= \theta_t + v_{t+1}
\end{aligned}
\tag{8}
$$

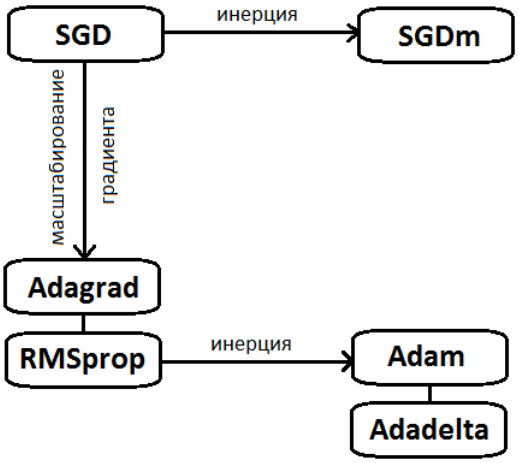

Рис. 4: Интуиция и связь модификаций метода SGD

Метод адаптивной инерции (Adam) [7] похож на каждый из трех предыдущих методов (Adagrad, Adadelta, RMSprop). Отличается он от них двумя идеями: во-первых, оценка первого момента вычисляется как скользящее среднее; во-вторых, из-за того, что оценки первого и второго моментов инициализируются нулями, используется небольшая коррекция, чтобы результирующие оценки не были смещены к нулю. Метод также инвариантен к масштабированию градиентов. При заданных гиперпараметрах $\gamma_1, \gamma_2, \lambda, \eta$ и $m_0 = 0, g_0 = 0$ правило пересчета следующее: [7]

$$\begin{aligned} m_{t+1} &= \gamma_1 m_t + (1-\gamma_1)\nabla f_i(\theta_t) \\ g_{t+1} &= \gamma_2 g_t + (1-\gamma_2)\nabla f_i(\theta_t)^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\gamma_1^{t+1}} \\ \hat{g}_{t+1} &= \frac{m_{t+1}}{1-\gamma_2^{t+1}} \\ \theta_{t+1} &= \theta_t - \frac{\eta \hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}+\epsilon}} \end{aligned} \tag{9}$$

На рис. 4 отображена связь всех рассмотренных методов. Таким образом, каждый метод использует либо идею *масштабирования градиента* (Adagrad, RMSprop), либо идею *инерции* (SGDm), либо сразу обе идеи (Adam и Adadelta).

2.3 Батч-нормализация

Помимо настройки шага для обучения нейронной сети возникает еще одна неочевидная на первый взгляд проблема. Когда обучающей системе на вход подаются данные, имеющие разную природу (например, распределение на данные изменяется со временем), система замедляется в обучении, так как ей приходится долго адаптироваться под изменяющиеся условия. В литературе такой эффект носит название *ковариационного сдвига (covariate shift)* [2]. В случае нейронных сетей такая проблема возникает на внутренних слоях. Представим, что каждый слой нейронной сети — это отдельная компонента обучающей системы. Тогда каждый слой (кроме самого первого) получает на вход данные, полученные с выхода предыдущего слоя. А так как в процессе обучения параметры сети меняются, то и распределение на данные, подаваемые на вход внутренних слоев, тоже меняется. Особенно сильно такой эффект наблюдается на более глубоких слоях. Так, даже небольшие изменения на входе нейронной сети сильно влияют на последующие слои — меняется распределение на нейроны, входящие во внутренние слои сети.

Установлено [8], что обучение сходится быстрее, если предварительно *нормализовать* (сделать нулевое матожидание и единичные диспресии) и *декоррелировать* входные данные для компоненты обучающей системы. Тогда для нейронной сети хотелось бы нормализовать данные перед входом на каждый слой. Таким образом и действует метод *батч-нормализации (batch normalization)*, предложенный в 2015 году [1].

Рассмотрим подробнее структуру батч-нормализации на примере одного внутреннего слоя.

- Пусть нормализуемы слой имеет размерность d: $x = (x_1, \ldots, x_d)$. Тогда можно нормализовать k-ое измерение x по следующей формуле (для простоты все измерения нормализуются независимо):

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}} \tag{10}$$

  Мы также должны масштабировать и сдвинуть нормализованную величину. В противном случае, лишь нормализация ограничит репрезентативную способность слоя. Например, если нормализуется вход на сигмоиду, то на выходе будет наблюдаться близкое к линейному преобразование (так как сигмоида $\sigma(x) = \frac{1}{1+\exp(-x)}$ на отрезке $[-1, 1]$ похожа на линейную функцию).

- Таким образом, нормализованная величина трансформируется в новую:

$$y^k = \gamma^k \hat{x}^k + \beta^k, \tag{11}$$

где параметры γ, β настраиваются в процессе обучения для каждой размерности.

- Более того, как в стохастическом градиентном спуске используются мини-батчи, можно так же использовать их в нормализации для оценки матожидания и дисперсии для каждого нейрона

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i^k, \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i^k - \mu_B)^2, \end{aligned} \tag{12}$$

  где $B = \{x^k_{1,\dots,m}\}$ — текущий мини-батч для k-го нейрона. Тогда вместо $\mathbb{E}[x^k]$ и $\mathrm{Var}[x^k]$ в (10) можно подставить значения μ_B и σ_B^2, являющиеся оценками соответствующих величин по мини-батчу.

- Итоговое преобразование добавляется в нейронную сеть перед нелинейной функцией активации.

В итоге батч-нормализация

- уменьшает ковариационный сдвиг во внутренних слоях нейронной сети и следовательно ускоряет обучение;
- является дифференцируемым преобразованием, то есть для обучения сети все так же можно применять метод обратного распространения ошибки;
- позволяет использовать большие шаги обучения, то есть позволяет не слишком аккуратно настраивать гиперпараметр шага;
- не производит явную декорреляцию входных данных для различных нейронов одного слоя.

В оригинальной статье [1] нейронная сеть с батч-нормализацией обучалась с помощью обычного метода SGD. В данной работе исследуется сочетание батч-нормализации и рассмотренных выше модификаций метода SGD.

3 Эксперименты

3.1 Наборы данных

В рамках работы исследования проводились для задачи многоклассовой классификации на следующих наборах данных:

- MNIST[3], рис. 5 (70 тыс. рукописных цифр — 10 классов): выборка была поделена на тренировочную (50 тыс.), валидационную (10 тыс.) и тестовую (10 тыс.)
- CIFAR-10[4], рис. 6 (60 тыс. изображений — 10 классов): выборка поделена на тренировочную (40 тыс.), валидационную (10 тыс.) и тестовую (10 тыс.)

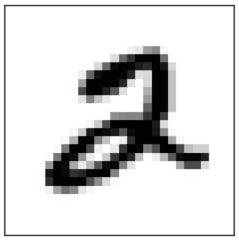

Рис. 5: Пример объекта из набора данных MNIST

Рис. 6: Примеры изображений из набора данных CIFAR-10

3.2 Архитектуры нейронных сетей

Для дальнейших экспериментов были выбраны конкретные архитектуры сетей. Опишем четыре варианта сети, над которыми проводились эксперименты:

- полносвязная сеть (multilayer perceptron, MLP): состоит из 3-х скрытых слоёв по 100 нейронов;
- сверточная сеть (convolutional neural network, CNN): состоит из 2-х сверточных слоёв (32 карты признаков со сверткой 5×5 + слой max-pooling с размером окна 2×2), затем один полносвязный слой с 256 нейронами;
- полносвязная глубокая сеть (deep MLP): состоит из 20-ти скрытых слоёв по 30 нейронов;

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

[4] https://www.cs.toronto.edu/~kriz/cifar.html

- сверточная глубокая сеть (deep CNN): состоит из 5-ти идущих подряд сверточных подсетей (3 сверточных слоя (k карт признаков с сверткой 3×3 + max-pooling с размером окна 2×2)), где для каждой следующей подсети k увеличивается в два раза: $[32, 64, 128, 256, 512]$, а затем следует один полносвязный слой с 256 нейронами;

В полносвязную сеть батч-нормализация добавляется сразу после линейной трансформации и перед функцией активации. Для сверточной сети — сразу после свертки и перед активацией.

В качестве функции активации во всех архитектурах используется выпрямленный линейный элемент (ReLU). Также на выходе каждой сети есть еще один полносвязный слой с выходом на 10 нейронов с многопеременной логистической функцией активации (softmax)[5].

3.3 Постановки экспериментов

Исследуем, как влияет добавление батч-нормализации на различные модификации метода стохастического градиентного спуска при обучении нейронных сетей.

Перед тем, как приступить к экспериментам, сформулируем несколько гипотез:

- добавление батч-нормализации в сеть увеличивает скорость сходимости обучения сети для всех методов;
- чем метод сложнее, тем батч-нормализация слабее ускоряет его сходимость;
- батч-нормализация сильнее проявляет ускорение обучения на глубоких сетях.

[5] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Влияние батч-нормализации на все методы.

Оценим, на сколько изменяется качество обучения методов при добавлении батч-нормализации. Для этого выполним следующие пункты:

1. Выберем исследуемые методы: SGD, SGDm, Adam, Adagrad, Adadelta, RMSprop.
2. Выберем наборы данных: MNIST, CIFAR-10 (описаны в 3.1).
3. Выберем архитектуры сети: MLP и CNN (описаны в 3.2).
4. Для всех исследуемых методов подберем оптимальный шаг обучения (по качеству на тестовой выборке) на всех комбинациях рассматриваемых архитектур и наборов данных. Для этого сначала грубо оценим приемлемый порядок шага обучения, перебирая значения по сетке. Затем в окрестности полученной величины переберем значение шага по более частой сетке. В результате выберем лучший шаг обучения из рассмотренных.
5. Сохраним полученные результаты работы всех методов на тренировочных и валидационных выборках по эпохам, а также результаты на тестовых выборках.
6. Составим таблицу относительных улучшений качества всех методов при добавлении в сеть батч-нормализации на тестовых выборках для всех пар наборов данных и архитектур.
7. Исследуем полученные результаты на наличие закономерностей.

Влияние батч-нормализации на методы при обучении глубоких нейронных сетей.

Проверим, что изменится для всех методов на глубоких сетях. Для этого выполним следующие пункты:

1. Возьмем исследуемые методы из предыдущего эксперимента: SGD, SGDm, Adam, Adagrad, Adadelta, RMSprop.
2. Возьмем наборы данных из предыдущего эксперимента: MNIST, CIFAR-10.
3. Выберем архитектуры для глубокой сети: deep MLP, deep CNN (описаны в 3.2).
4. Для всех исследуемых методов подберем оптимальный шаг обучения на всех комбинациях рассматриваемых глубоких архитектур и датасетов аналогично предыдущему эксперименту.

5. Сохраним полученные результаты работы всех методов на тренировочных и валидационных выборках по эпохам, а также результаты на тестовых выборках.

6. Составим таблицу относительных улучшений качества всех методов при добавлении в глубокую сеть батч-нормализации на тестовых выборках для всех пар датасетов и архитектур.

7. Сравним результаты с соответствующими результатами из предыдущего эксперимента.

Для вычисления *относительного улучшения* качества используется следующая формула:

$$\text{rel} = \frac{y - x}{100 - x}, \tag{13}$$

где x — качество метода (в процентах) до добавления батч-нормализации, y — качество после добавления батч-нормализации (заметим, что всегда $\text{rel} \leq 1$). Величина rel показывает, например, что улучшение с 50% до 51% менее значимо, чем улучшение с 90% до 91%.

3.4 Результаты экспериментов

Влияние батч-нормализации на все методы.

Из таблицы 1 видно, что все относительные улучшения положительны. Однако для методов SGDm, Adam и Adadelta для архитектуры MLP наблюдается очень слабое улучшение. Как обнаружится далее, качество этих методов может и понижаться на некоторых эпохах при добавлении в сеть батч-нормализации. Также отметим, что сильнее всего батч-нормализация улучшает качество метода SGD, как самого простого метода.

Методы	**MNIST + MLP**	**CIFAR-10 + MLP**	**MNIST + CNN**	**CIFAR-10 + CNN**
SGD	**0.24**	**0.2**	**0.35**	**0.36**
SGDm	0.06	0.1	0.29	0.25
Adam	0.05	0.09	0.13	0.14
Adagrad	0.13	0.17	0.3	0.33
Adadelta	0.03	0.11	0.19	0.2
RMSprop	0.22	0.16	0.24	0.19

Таблица 1: Улучшение качества на тестовой выборке при добавлении батч-нормализации

Попробуем выявить закономерности в полученных результатах. Посмотрим отдельно на результаты для архитектуры CNN для датасетов MNIST (табл. 2) и CIFAR-10 (табл. 3).

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.64	0.57	0.48	0.49	0.37
SGDm	0.7	0.52	0.47	0.34	0.3
Adam	**0.44**	**0.17**	**0.29**	**0.15**	**0.11**
Adagrad	0.46	0.38	0.31	**0.3**	0.33
Adadelta	**0.19**	**0.07**	**0.23**	**0.15**	**0.18**
RMSprop	**0.34**	**0.22**	**0.11**	0.36	**0.27**

Таблица 2: Улучшения качества при добавлении батч-нормализации в сеть CNN, MNIST. **Жирным** выделены топ-3 наименьшего улучшения (по столбцам)

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.32	0.4	0.3	0.34	0.32
SGDm	0.34	0.34	0.35	0.27	0.24
Adam	**0.19**	**0.22**	**0.23**	**0.2**	**0.15**
Adagrad	**0.26**	**0.26**	0.35	0.37	0.35
Adadelta	0.27	0.34	**0.17**	**0.22**	**0.22**
RMSprop	**0.23**	**0.22**	**0.26**	**0.21**	**0.19**

Таблица 3: Улучшения качества при добавлении батч-нормализации в сеть CNN, CIFAR-10. **Жирным** выделены топ-3 наименьшего улучшения (по столбцам)

В таблицах показаны улучшения для валидационных выборок. Здесь можно обнаружить следующую закономерность: слабее всего улучшаются методы Adam, Adadelta и RMSprop, которые используют для оценки второго момента градиента экспоненциальное скользящее среднее (методы, наследуемые от RMSprop на рис.4).

Теперь посмотрим на такие же результаты для архитектуры MLP для датасетов MNIST (табл. 4) и CIFAR-10 (табл. 5).

Первое, что бросается в глаза, так это отрицательная величина изменения качества (выделены **жирным красным**). Можно сделать вывод, что батч-нормализация не всегда повышает качество на промежуточных эпохах.

В таблицах 4 и 5 также указаны улучшения для валидационных выборок. Здесь наблюдаются следующие закономерности: слабее всего улучшаются методы SGDm, Adam и Adadelta, которые используют идею инерции (рис.4). Также отметим, что результаты для одинаковых архитектур получаются похожими.

Из полученных результатов можно сделать вывод, что, чем идейно сложнее методы, тем они слабее улучшаются при добавлении батч-нормализации. Так, методы Adam и Adadelta

Номер эпохи	**2**	**5**	**10**	**25**	**50**
SGD	0.67	0.63	0.42	0.25	0.22
SGDm	0.47	0.34	**0.13**	**-0.08**	**0.05**
Adam	**0.21**	**0.2**	**-0.05**	**-0.15**	**0.07**
Adagrad	0.34	0.27	0.17	0.12	0.14
Adadelta	**0.3**	**0.08**	**0.08**	**-0.0**	**0.03**
RMSprop	**0.28**	**0.27**	0.35	0.21	0.25

Таблица 4: Улучшения качества при добавлении батч-нормализации в сеть MLP, MNIST. **Жирным** выделены топ-3 наименьшего улучшения (по столбцам)

Номер эпохи	**2**	**5**	**10**	**25**	**50**
SGD	0.23	0.18	0.22	0.21	0.17
SGDm	**0.17**	**0.13**	**0.15**	**0.13**	**0.1**
Adam	**0.13**	**0.12**	**0.11**	**0.13**	**0.12**
Adagrad	0.25	0.23	0.23	0.18	0.17
Adadelta	**0.18**	**0.14**	**0.19**	**0.17**	**0.15**
RMSprop	0.2	0.18	0.21	0.22	0.16

Таблица 5: Улучшения качества при добавлении батч-нормализации в сеть MLP, CIFAR-10. **Жирным** выделены топ-3 наименьшего улучшения (по столбцам)

в схеме 4 используют идеи масштабирования градиента и инерции — и как раз для них батч-нормализация дает наименьшую прибавку.

Возникает идея, что добавление батч-нормализации в нейронную сеть слабее улучшает методы, которые и так показывают лучшее качество. Покажем, что это не совсем так.

Из таблицы 6 хорошо видно, что метод SGDm показывает довольно высокое качество, однако не входит в топ-3 методов, для которых батч-нормализация дает наименьшее улучшение.

Номер эпохи	**1**	**2**	**25**	**35**	**50**
SGD	24.26	22.14	58.95	63.23	65.45
SGDm	**30.03**	**37.98**	**64.37**	68.3	**71.54**
Adam	**43.28**	**47.3**	**68.02**	**70.79**	**72.54**
Adagrad	29.82	37.25	60.86	62.35	64.63
Adadelta	**31.31**	32.96	**67.12**	**70.56**	**70.81**
RMSprop	25.6	**39.89**	64.24	**68.96**	70.31

Таблица 6: Качество на CIFAR-10 + CNN, топ-3 лучшего качества. **Жирным** выделены методы, для которых батч-нормализация дает наименьшую прибавку (взято из табл. 3). **Жирным зеленым** выделены топ-3 лучшего качества без добавления батч-нормализации (по столбцам)

Влияние батч-нормализации на методы при обучении глубоких нейронных сетей.

Покажем, что для глубоких сетей добавление батч-нормализации играет большую роль, чем для неглубоких сетей. В таблице 7 указаны улучшения для глубокой и обычной архитектур MLP. Для лучшей наглядности посмотрим на их разность в таблице 8. Видно, что для глубоких архитектур улучшение заметно сильнее.

Номер эпохи	**2**	**3**	**10**	**25**	**50**
SGD	0.84	0.9	0.94	0.93	0.7
SGDm	0.8	0.83	0.32	0.19	0.03
Adam	0.72	0.66	0.58	0.4	0.28
Adagrad	0.85	0.84	0.45	0.37	0.37
Adadelta	0.58	0.69	0.27	0.18	0.14
RMSprop	0.72	0.82	0.45	0.39	0.37

Номер эпохи	**1**	**3**	**15**	**35**	**50**
SGD	0.67	0.63	0.42	0.25	0.22
SGDm	0.47	0.34	0.13	-0.08	0.05
Adam	0.21	0.2	-0.05	-0.15	0.07
Adagrad	0.34	0.27	0.17	0.12	0.14
Adadelta	0.3	0.08	0.08	-0.0	0.03
RMSprop	0.28	0.27	0.35	0.21	0.25

Таблица 7: Улучшения качества при добавлении батч-нормализации в сеть MLP, MNIST. Слева таблица для глубокой архитектуры (deep MLP), справа — для обычной (MLP)

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.17	0.27	0.52	0.68	0.48
SGDm	0.33	0.49	0.19	0.27	-0.02
Adam	0.51	0.46	0.63	0.55	0.21
Adagrad	0.51	0.57	0.28	0.25	0.23
Adadelta	0.28	0.61	0.15	0.18	0.11
RMSprop	0.44	0.55	0.1	0.18	0.12

Таблица 8: Разница в улучшении качества deep MLP и MLP для датасета MNIST

Посмотрим на аналогичные таблицы 9 для архитектуры CNN и опять для наглядности вычислим их разности. Видно, что опять зеленый цвет превалирует — то есть для глубокой архитектуры батч-нормализация повышает качество сильнее. На самом деле так происходит из-за того, что без батч-нормализации на очень глубоких сетях методы совсем не обучаются или обучаются очень медленно (см. приложение).

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.42	0.63	0.71	0.78	0.68
SGDm	0.44	0.65	0.71	0.69	0.42
Adam	0.36	0.57	0.6	0.61	0.52
Adagrad	0.21	0.47	0.59	0.65	0.63
Adadelta	0.41	0.59	0.68	0.67	0.65
RMSprop	0.13	0.49	0.63	0.67	0.61

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.32	0.4	0.3	0.34	0.32
SGDm	0.34	0.34	0.35	0.27	0.24
Adam	0.19	0.22	0.23	0.2	0.15
Adagrad	0.26	0.26	0.35	0.37	0.35
Adadelta	0.27	0.34	0.17	0.22	0.22
RMSprop	0.23	0.22	0.26	0.21	0.19

Таблица 9: Улучшения качества при добавлении батч-нормализации в сеть CNN, CIFAR-10. Слева таблица для глубокой архитектуры (deep CNN), справа — для обычной (CNN)

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	0.1	0.23	0.41	0.44	0.34
SGDm	0.1	0.31	0.36	0.42	0.18
Adam	0.17	0.35	0.37	0.41	0.37
Adagrad	-0.05	0.21	0.24	0.28	0.28
Adadelta	0.14	0.25	0.51	0.45	0.43
RMSprop	-0.1	0.27	0.37	0.46	0.42

Таблица 10: Разница в улучшении качества deep CNN и CNN для датасета CIFAR-10

Комбинация батч-нормализации с методом Adam.

При проведении экспериментов неоднократно возникали проблемы при обучении нейронной сети с батч-нормализацией методом Adam. Качество по эпохам начинало непредсказуемо прыгать даже при малом шаге обучения (рис. 7). Здесь показано качество метода Adam на тренировочной и валидационной выборках усложненного набора данных MNIST (cluttered MNIST) [14].

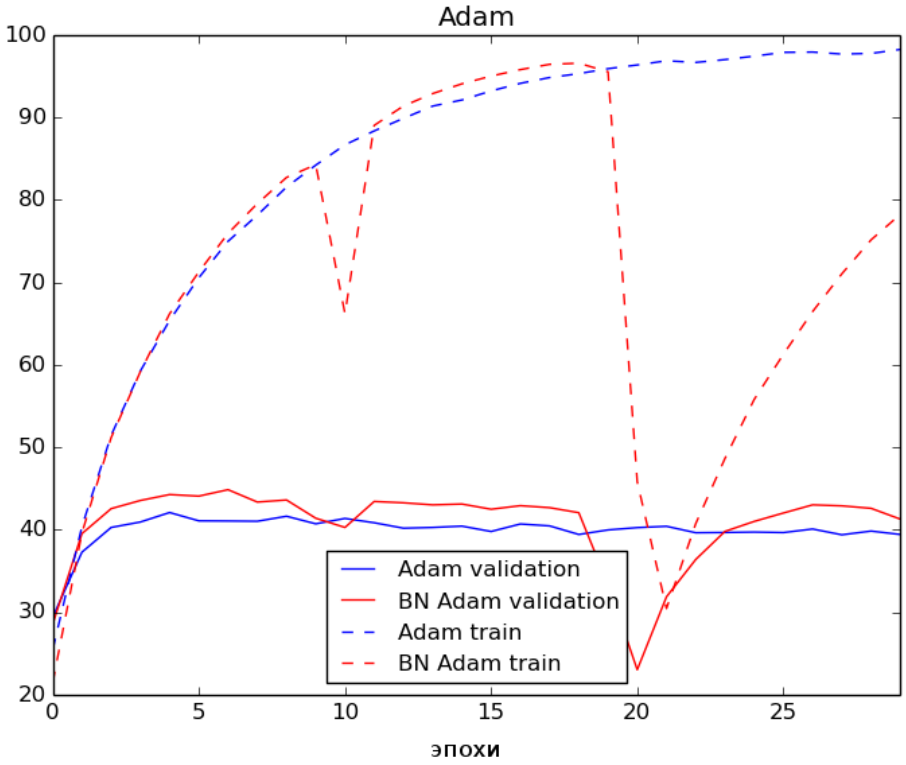

Рис. 7: Прыжки метода Adam при добавлении батч-нормализации

Проблему окончательно изучить не удалось, однако на основе проведенных экспериментов удалось вывести несколько рекомендаций по использованию батч-нормализации.

4 Заключение

Батч-нормализация является прорывной идеей в области обучения нейронных сетей. Однако она не всегда повышает качество работы рассмотренных методов. Были сформулированы некоторые рекомендации по ее применению:

- Для полносвязной неглубокой архитектуры сети батч-нормализацию стоит применять к более простым методам. Например, методы, использующие инерцию, такие как SGDm, Adam, Adadelta, покажут хорошее качество и без батч-нормализации. А с ней они работают дольше и могут показать качество хуже.
- Для глубоких сетей всегда нужно использовать батч-нормализацию, так как иначе методы могут совсем не обучаться.
- Если время или количество эпох ограничено и очень мало, то обязательно стоит добавить в сеть батч-нормализацию, так как она сильнее всего помогает именно на первых эпохах.
- Для метода Adam с батч-нормализацией нужно быть аккуратным: чтобы не возникло проблем при обучении, требуется аккуратно подобрать параметры метода.

Список литературы

[1] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift."arXiv preprint arXiv:1502.03167 (2015).

[2] Shimodaira, Hidetoshi. "Improving predictive inference under covariate shift by weighting the log-likelihood function."Journal of statistical planning and inference 90.2 (2000): 227-244.

[3] Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning."Proceedings of the 30th international conference on machine learning (ICML-13). 2013.

[4] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization."The Journal of Machine Learning Research 12 (2011): 2121-2159.

[5] Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude."COURSERA: Neural Networks for Machine Learning 4 (2012): 2.

[6] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method."arXiv preprint arXiv:1212.5701 (2012).

[7] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization."arXiv preprint arXiv:1412.6980 (2014).

[8] LeCun, Yann A., et al. "Efficient backprop."Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

[9] Waibel, Alexander, et al. "Phoneme recognition using time-delay neural networks."Acoustics, Speech and Signal Processing, IEEE Transactions on 37.3 (1989): 328-339.

[10] Kolarik, Thomas, and Gottfried Rudorfer. "Time series forecasting using neural networks."ACM Sigapl Apl Quote Quad. Vol. 25. No. 1. ACM, 1994.

[11] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks."Advances in neural information processing systems. 2012.

[12] Hecht-Nielsen, Robert. "Theory of the backpropagation neural network."Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, 1989.

[13] Amari, Shunichi. "A theory of adaptive pattern classifiers."Electronic Computers, IEEE Transactions on 3 (1967): 299-307.

[14] Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention."Advances in Neural Information Processing Systems. 2014.

Приложение

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	10.64	10.64	19.65	47.38	90.55
BN SGD	85.64	91.27	95.44	96.55	97.14
Улучшение	0.84	0.9	0.94	0.93	0.7
SGDm	20.0	33.78	91.54	95.29	96.19
BN SGDm	83.79	88.86	94.22	96.2	96.32
Улучшение	0.8	0.83	0.32	0.19	0.03
Adam	68.01	81.55	91.83	95.02	95.86
BN Adam	91.12	93.65	96.57	97.01	97.03
Улучшение	0.72	0.66	0.58	0.4	0.28
Adagrad	39.39	53.41	91.85	94.06	94.69
BN Adagrad	90.72	92.73	95.48	96.26	96.63
Улучшение	0.85	0.84	0.45	0.37	0.37
Adadelta	41.63	61.98	93.58	95.83	96.16
BN Adadelta	75.59	88.06	95.32	96.56	96.71
Улучшение	0.58	0.69	0.27	0.18	0.14
RMSprop	48.66	65.59	92.09	95.36	95.85
BN RMSprop	85.86	93.67	95.67	97.15	97.37
Улучшение	0.72	0.82	0.45	0.39	0.37

Таблица 11: Качество на MNIST, DEEP MLP

Номер эпохи	**2**	**5**	**20**	**35**	**50**
SGD	9.67	9.43	10.33	10.33	48.46
BN SGD	47.77	66.79	74.13	80.41	83.61
Улучшение	0.42	0.63	0.71	0.78	0.68
SGDm	9.43	9.43	17.47	32.8	71.01
BN SGDm	49.14	68.27	75.76	79.4	83.15
Улучшение	0.44	0.65	0.71	0.69	0.42
Adam	21.33	38.71	51.72	59.16	71.44
BN Adam	50.04	73.53	80.62	84.2	86.37
Улучшение	0.36	0.57	0.6	0.61	0.52
Adagrad	10.11	27.44	40.39	47.71	58.87
BN Adagrad	28.84	61.41	75.4	81.64	84.67
Улучшение	0.21	0.47	0.59	0.65	0.63
Adadelta	9.67	11.77	13.23	34.67	48.87
BN Adadelta	47.1	63.65	72.35	78.35	82.19
Улучшение	0.41	0.59	0.68	0.67	0.65
RMSprop	12.97	20.47	38.2	47.24	66.04
BN RMSprop	23.97	59.22	77.36	82.63	86.79
Улучшение	0.13	0.49	0.63	0.67	0.61

Таблица 12: Качество на CIFAR-10, DEEP CNN