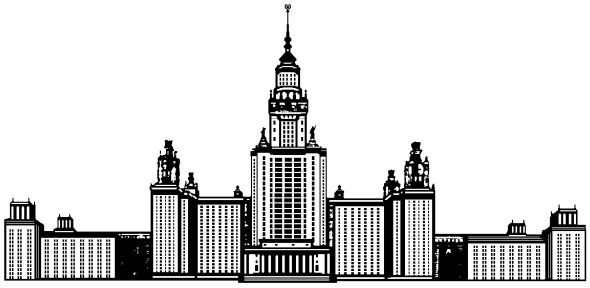


Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Януш Виктор Янович

# Обучение методов оптимизации с использованием нейросетевых моделей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**Научный руководитель:**
научный сотрудник ВМК МГУ
Д. А. Кропотов

Москва, 2018

**Аннотация**

Глубинное обучение достигло больших успехов в автоматическом выделении признаков. Тем не менее, методы оптимизации до сих пор придумываются вручную. Однако, несколько подходов к обучению оптимизаторов с использованием рекуррентных нейронных сетей уже было предложено. Один из них был предложен в [1]. Другие работы (например, [2, 3]) рассматривают задачу обучения метода оптимизации как задачу обучения с подкреплением [4]. В данной выпускной квалификационной работе показано сравнение с имеющимися методами другого подхода, использующего методы обучения с подкреплением. Полученный метод имеет более высокую обобщающую способность чем имеющиеся методы и по качеству работы не уступает стандартным методам, обладая тем преимуществом, что не требует настройки.

# Содержание

# 1 Введение

Современные подходы к машинному обучению предполагают, что алгоритмы, которые учатся на определенных данных, должны лучше справляться с задачей, чем такие, которые построены на придуманных человеком правилах. Такой подход принес заметные плоды во многих областях: компьютерное зрение, обработка сигналов, медицинские анализы, генерация данных. Тем не менее, не все области оказались охвачены машинным обучением в полной мере. Так, этот подход, до недавнего времени, практически не применялся к улучшению методов самого машинного обучения. Эта идея, названная мета-обучением [5–7], стала достаточно широко распространена лишь в последнее время. Она существует во многих различных формах, но в том или ином виде представляет собой настройку процесса обучения базовых моделей. В частности, процесс обучения зачастую представляет собой стохастическую градиентную оптимизацию [8].

В данной работе предлагается рассмотреть обучаемые модели оптимизаторов, на основе нейросетевых моделей. Предлагается изучить преимущества и недостатки двух подходов к настройке методов оптимизации: с помощью обучения с учителем и обучения с подкреплением, использующего технику репараметризации. Будет произведено сравнение двух данных подходов к обучению между собой, а также сравнение между собой обученных и стандартных методов стохастической оптимизации. На данный момент, таковыми являются SGD, SGD+Momentum, Adam [9].

# 2 Постановка задачи

Исследуем задачу обучения метода оптимизации. Рассмотрим произвольный метод оптимизации первого порядка, а также функцию $f(\theta)$, где $\theta$ — параметры, по которым производится оптимизация. Введем следующие обозначения: $t$ — номер текущего шага, $\theta_t$ — параметры на шаге $t$, $f_t = f(\theta_t)$ — значение функции в момент времени $t$, $\nabla_t = \nabla f(\theta_t)$ — градиент в момент времени $t$.

Методы градиентной оптимизации первого порядка работают следующим образом:

$$\theta_t = \theta_{t-1} + g_t,$$
$$(g_t, h_t) = G(\nabla_0, \ldots, \nabla_t, h_{t-1}, \phi).$$

Здесь $G$ — некоторая функция принимающая на вход градиенты (возможно стохастические) с предыдущих и текущего шага, некоторое внутреннее состояние $h_t$, а также параметры оптимизатора — $\phi$, и выдает на выходе следующий шаг, а также обновляет внутреннее состояние. Для иллюстрации приведем несколько примеров.

Для стохастического градиентного спуска:

$$\phi = \{\alpha_0, \gamma\}$$
$$h_t = \{\} \text{ — Нет памяти}$$
$$G(\nabla_0, \ldots, \nabla_t, h_t, \phi) = -\frac{\alpha_0}{(t+1)^\gamma}\nabla_t$$

Для стохастического градиентного спуска с инерцией:

$$\phi = \{\alpha_0, \gamma, \mu\}$$
$$h_t = \text{ направление сдвига на шаге } (t-1)$$
$$G(\nabla_0, \ldots, \nabla_t, h_t, \phi) = (\mu h_t - \frac{\alpha_0}{(t+1)^\gamma}\nabla_t, \mu h_t - \frac{\alpha_0}{(t+1)^\gamma}\nabla_t)$$

Данные примеры показывают, что для того чтобы задать оптимизатор — необходимо задать функцию $G$ и начальное состояние $h_0$. Обучить оптимизатор, соответственно, означает найти такие параметры $\phi$, что алгоритм будет выдавать разумные шаги. В работе [1] было предложено в качестве функции $G$ взять рекуррентную нейронную сеть:

$$G(\nabla_t, h_t, \phi) = \text{RNN}(\nabla_t, h_t)$$

Это означает, что данный оптимизатор работает следующим образом: считается градиент $\nabla_t$ на текущей итерации, затем подается в рекуррентную сеть, которая на выходе выдает $g_t, h_t$. После чего шаг $g_t$ прибавляется к $\theta_{t-1}$. Существует несколько причин выбрать в качестве оптимизатора именно рекуррентную сеть. Во-первых, данный тип нейронных сетей наиболее приспособлен для обработки последовательностей. Во-вторых, в таких сетях естественным образом возникает внутреннее состояние — память.

# 3 Обзор методов обучаемой оптимизации

Авторы [1] пытаются решить задачу обучаемых методов оптимизации с помощью рекуррентных нейронных сетей. Они ставят задачу обучить оптимизатор так, чтобы была высокая обобщающая способность на задачах из того же класса, что и обучающая выборка, использованная для настройки оптимизатора. Также, они попробовали перенести свою модель на задачи из другого класса, но в основном такие попытки ни к чему не привели.

В [2,3] предлагается использовать обучение с подкреплением для обучения оптимизатора. Конкретнее, для этого они используют алгоритм обучения с подкреплением под названием Guided Policy Search (GPS). Этот метод хорошо подходит для задач с непрерывным пространством действий, а так же является довольно эффективным по скорости обучения. В данной работе предлагается рассмотреть также другие подходы к обучению методов оптимизации с использованием обучения с подкреплением.

В работе [10] модель из [1] модифицируется таким образом, чтобы ее можно было применять к задаче так называемого few-shot обучения. Они обучают оптимизатор таким образом,

чтобы подстроить его под определенный класс обучающих выборок. При этом в качестве функции потерь для оптимизатора используется не ошибка на обучающей выборке, а тестовая ошибка. Также, они выучивают хорошую инициализацию для сети, чтобы потом ее можно было обучить быстро и с малым размером обучающей выборки.

В работе [11] авторы пытаются применить рекуррентные нейронные сети для обучения методов оптимизации в применении к задачам обучения с подкреплением, а также различным задачам оптимизации без доступа к структуре оптимизируемой функции, таким как задача обучения гауссовского процесса в форме бандита [12], подбор гиперпараметров нейросети, а также задачи простого управления.

В [13] исследуются различные приемы, которые могут позволить увеличить обобщающую способность модели из [1]. Например, одним из таких приемов является случайное перемасштабирование пространства параметров, которое мешает переобучиться на масштаб задачи. Это довольно логичное изменение, так как логично, что хороший оптимизатор должен работать одинаково независимо от аффинных преобразований параметров. Стандартным примеров такого адаптивного поведения является Adam [9].

Авторы [14] прибегают к иному взгляду на поиск оптимального метода оптимизации. Предлагается использовать рекуррентную нейронную сеть — контроллер. Такой контроллер будет выдавать строчку, описывающую оптимизатор. Эта строчка будет состоять из различных операндов, таких как градиент, скользящее среднее градиентов и т.д., а также операции над этими операндами. Затем, с помощью обучения с подкреплением такой контроллер будет обучаться выдавать хорошие оптимизаторы. Однако, их метод требует большого количества вычислительных ресурсов.

# 4 Существующие подходы

## 4.1 Обучение оптимизатора

Для обучения модели необходимо выбрать функцию потерь. В работе [1] было предложено использовать следующую функцию потерь:

$$L(\phi) = \mathbb{E}_{f,\theta_0}\left[\frac{1}{T}\sum_{t=1}^{T} w_t f_t\right]$$

Здесь $T$ — количество шагов, на которые запущен оптимизационный процесс. Математическое ожидание берется, неформально говоря, по определенному классу функций. Выбор такой функции потерь выглядит довольно разумным, с учетом того, что, вообще говоря, необходимо получить не только низкое значение $f(\theta_T)$, но и поощрить высокую скорость сходимости на промежуточных итерациях. Для простоты положим все $w_t = 1$. Данный функционал имеет определенный геометрический смысл — он пропорционален площади под графиком значения функции в зависимости от итерации. В итоге, метод обучения оптимизатора

состоит в решении следующей задачи:

$$L(\phi) = \mathbb{E}_f \left[ \frac{1}{T} \sum_{t=1}^{T} f(\theta_t) \right]$$
$$\phi^* = \arg\min_{\phi} L(\phi)$$

Для решения этой оптимизационной задачи можно использовать стохастический градиентный спуск — стандартный метод для минимизации функционалов в форме математического ожидания. Для того, чтобы сделать это, необходимо получить реализацию $\hat{f}$ случайной функции $f$, а затем посчитать $\hat{L}(\hat{f}, \phi) = \frac{1}{T} \sum_{t=1}^{T} \hat{f}(\theta_t)$ и $\frac{d\hat{L}}{d\phi}(\hat{f}, \phi)$. Посчитанный градиент затем можно подать в любой метод оптимизации первого порядка, например, в Adam [9], или даже в какой-нибудь обученный оптимизатор, наподобие данной модели.

Единственное, что осталось выяснить — как считать градиент $\frac{d\hat{L}}{d\phi}$. Для этого достаточно запустить метод обратного распространения ошибки на графе 1. На нем: optimizee — оптимизируемая функция, optimizer — метод оптимизации, $m$ — функция, принимающая на вход градиенты и выдающая шаг.

Стоит отметить, что есть одна существенная деталь. Заметим, что $\nabla_t$ зависит от $\phi$ так как $\nabla_t$ зависит от $\theta_t$, который в свою очередь зависит от оптимизатора. Это значит, что в процессе вычисления градиента потребуется вычисление $\frac{\partial \nabla_t}{\partial \phi}$, что может быть нежелательно, так как требует вычисления второй производной исходной функции. В частности, если $\theta$ представляет собой набор параметров достаточно большой нейросети (например, с $10^5$ параметров), то гессиан будет иметь квадратичный размер, что крайне нежелательно. В связи с этим замечанием, авторы [1] предлагают не пропускать градиент через $\nabla_t$, что и показано на вычислительном графе прерывистой линией. Также, следует отметить, что это может служить в качестве регуляризации модели, так как не позволяет подстраивать получаемые на вход градиенты определенным образом и уменьшает переобучение оптимизатора.

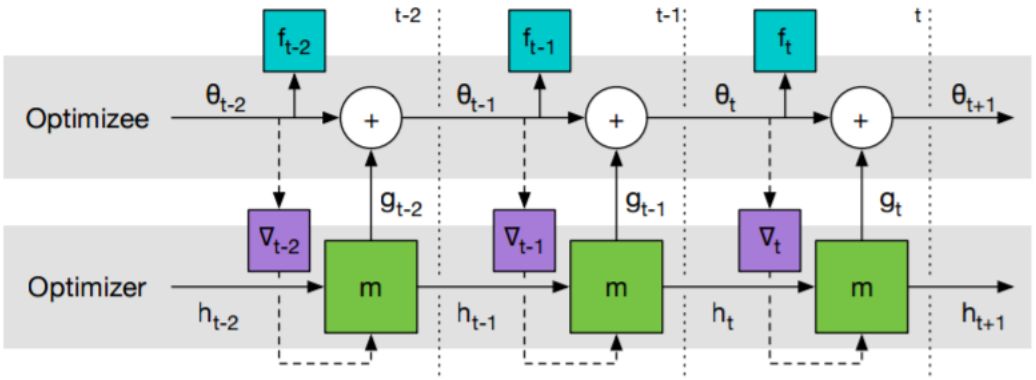



Рис. 1: Вычислительный граф, соответствующий модели оптимизации. Здесь: optimizee — оптимизируемая функция, optimizer — метод оптимизации, $m$ — функция, принимающая на вход градиенты и выдающая шаг

## 4.2 Покоординатный оптимизатор

Существующие алгоритмы стохастической оптимизации часто применяются к нейронным сетям. Отличительная особенность таких задач оптимизации состоит в том, что данные модели имеют огромное количество параметров: от нескольких тысяч до нескольких десятков миллионов. Типичная рекуррентная сеть содержит в себе полносвязный слой — линейное преобразование из пространства одной размерности в пространство другой размерности в композиции с какой-нибудь нелинейной функцией. Дело в том, что если мы будем принимать весь вектор в качестве входа в нейросеть, то параметров у метода оптимизации будет еще больше, чем у самой модели. Ясно, что такую модель будет довольно сложно не просто обучить, но и уместить в память. Другой недостаток такого подхода в том, что мы не можем изменить число параметров оптимизируемой функции без переобучения метода оптимизации.

Все эти соображения приводят к одному решению: метод оптимизации должен обрабатывать каждую координату отдельно от других. Это приводит к фиксированному числу параметров у оптимизатора, а также к возможности применять его к произвольному числу параметров оптимизируемой функции. Такое решение приводит к возможности обучить оптимизатор на одних задачах, а затем обобщить его на другие, например, на задачу обучения глубинных нейронных сетей.

Рисунок 2 показывает как работает такая модель на практике. Весь вектор градиента приходит на вход в рекуррентную нейронную сеть. Этот вектор трактуется как батч входов, что позволяет обрабатывать каждую координату независимо. Такое решение приводит к тому, что каждой координате $i$ соответствует свое внутреннее состояние $h_{ti}$ и свой выход $g_{ti}$. Затем эти выходы объединяются в вектор $g_t$ и обновляют вектор параметров $\theta_t$.

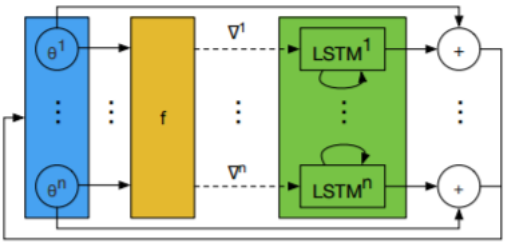



Рис. 2: Схема работы покоординатного оптимизатора

# 5 Предлагаемый подход

## 5.1 Недостатки существующих подходов

Предыдущий подход к решению задачи обучения методов оптимизации имеет несколько недостатков, которые проявляются на практике. Основной проблемой является довольно

низкая обобщающая способность таких моделей. Если же обобщающая способность находится на приемлемом уровне, то зачастую качество не слишком сильно выше, чем у стандартных методов стохастической оптимизации. В связи с этим, предлагается сместить точку зрения и посмотреть на данную задачу с позиций обучения с подкреплением. В данном разделе будет дано описание того, что это такое, а затем будут представлены некоторые методы решения данной задачи.

## 5.2 Постановка в форме задачи обучения с подкреплением

Задача обучения с подкреплением — задача обучения взаимодействия *агента* со *средой* таким образом, чтобы *награда* получаемая агентом была наибольшей, или, равнозначно, суммарная стоимость действия была минимальной. Обычно такая постановка задачи формализуется с помощью марковского процесса принятия решения (Markov decision process — MDP).

Марковский процесс принятия решений — кортеж $(\mathcal{S}, \mathcal{A}, p_0, p, r, \gamma)$, где

- $\mathcal{S}$ — множество состояний,
- $\mathcal{A}$ — множество действий,
- $p_0 : \mathcal{S} \to \mathbb{R}^+$ — распределение над начальными состояниями,
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ — вероятностное распределение переходов,
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ — награда получаемая при совершении данного действия в данном состоянии,
- $\gamma$ — коэффициент дисконтирования.

Задача выучить стохастическую *политику* $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$, которая задает условное распределение над действиями при условии состояния. Оптимальная политика задается следующим образом:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s_0, a_0, \ldots, s_T, a_T} \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$$

Покажем, как свести задачу обучения метода оптимизации к задаче обучения с подкреплением. Для этого нужно задать соответствующий марковский процесс принятия решений. Введем следующий MDP:

- $D$ — количество координат,
- $\mathcal{S} = \mathbb{R}^{2D}$ — значение параметров и градиент (в соответствующей точке),
- $\mathcal{A} = \mathbb{R}^D$ — значение шага,
- $p_0(s)$ — соответствует исходной задаче,

- $p(s' \mid s, a) = \delta(s' = s + a)$,
- $r(s, a) = -f(s + a)$,
- $\gamma = 1$.

Как легко видеть, в такой постановке задача сводится к исходной. Важное следствие такого представления в том, что мы можем попытаться применить методы обучения с подкреплением для решения данной задачи.

Существенное замечание к этой модели состоит в том, что она не подходит для стохастических оптимизаторов. Дело в том, что во время стохастической оптимизации мы не можем посчитать полный градиент оптимизируемой функции, у нас есть лишь возможность получить несмещенную оценку на него (либо можем посчитать и полностью, но это может занять очень большое время, и поэтому так делать не получается на практике). Для работы с такой задачей в постановке обучения с подкреплением нужно ввести *частично наблюдаемый марковский процесс принятия решений*. Основное отличие в том, что мы не обозреваем состояние $s \in \mathcal{S}$, лишь наблюдения $o \in \mathcal{O}$. В таком случае необходимо учитывать то, что мы не работаем с истинными состояниями и корректировать применяемые алгоритмы.

## 5.3 Обучение методами обучения с подкреплением

Запишем еще раз функционал, который оптимизируется в задаче обучения с подкреплением:

$$L(\pi) = \mathbb{E}_{s_0, a_0, \ldots, s_T, a_T} \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$$

Существует большое количество алгоритмов решения данной задачи. В работах [2, 3] предлагается использовать алгоритм под названием Guided Policy Search (GPS) [15] для обучения оптимизаторов.

Одним из основных подходов к обучению с подкреплением является так называемый Policy Gradients (градиенты по политике) подход. В этом подходе предполагается набирать эпизоды взаимодействия со средой с использованием текущей политики, накапливать градиент, а затем обновлять политику. Градиент по политике вычисляется следующим образом. Пусть

$$\tau = \{s_0, a_0, \ldots, s_T, a_T\},$$
$$R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t).$$

Задача максимизировать

$$L(\phi) = \mathbb{E}_{\pi(\tau|\phi)} \left[ R(\tau) \right] \to \max_{\phi},$$
$$\pi(\tau \mid \phi) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t, \phi) p(s_{t+1} \mid s_t, a_t).$$

Как известно, в таком случае

$$\frac{\partial L(\phi)}{\partial \phi} = \mathbb{E}_{\pi(\tau|\phi)} \left[ R(\tau) \nabla_\phi \log \pi(\tau) \right]$$

Соотвественно, для подсчета стохастического градиента ценности политики, необходимо разыграть эпизод $\tau$, а затем подсчитать $R(\tau)\nabla_\phi \log \pi(\tau \mid \phi)$ и подать в любой метод стохастической оптимизации первого порядка.

Эта формула работает в общем случае, и дает несмещенную оценку на градиент ценности политики $\pi$. Тем не менее, у данного подхода есть несколько серьезных недостатков. Во-первых, этот подход никак не использует дифференцируемость награды в том случае, если она дифференцируема. Во-вторых, хотя эта оценка на градиент действительно является несмещенной — дисперсия такого стохастического градиента крайне высока [16].

Для того, чтобы решить эти проблемы, можно попытаться воспользоваться техникой репараметризации. В общем виде, эта техника выглядит так. Необходимо решить задачу:

$$h(\theta) = \mathbb{E}_{p(\xi|\theta)} \left[ f(\xi) \right] \to \max_\theta$$

Если существует случайная величина $\varepsilon \sim p(\varepsilon)$ и детерминированная функция $g(\varepsilon, \theta)$ такие, что $\xi = g(\varepsilon, \theta) \sim p(\xi \mid \theta)$, тогда можно переписать исходную задачу в виде:

$$h(\theta) = \mathbb{E}_{p(\varepsilon)} \left[ f(g(\varepsilon, \theta)) \right] \to \max_\theta$$

Посчитать градиент $\nabla_\theta h(\theta)$ стало гораздо проще так как достаточно его просто пробросить через матожидание, получая следующую формулу для стохастического градиента:

$$\hat{\nabla}_\theta h(\theta) = \nabla_\theta f(g(\hat{\varepsilon}, \theta)),\ \hat{\varepsilon} \sim p(\varepsilon)$$

Конкретно в нашем случае, достаточно выбрать политику такой, чтобы было легко выполнить репараметризацию. В дальнейшем будет показано как это сделать.

В общем виде подсчет градиента будет выглядеть следующим образом:

$$\begin{aligned}
\nabla_\phi L(\phi) &= \nabla_\phi \mathbb{E}_{p(\varepsilon)} \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t(s_t, \varepsilon_t)) \right] \\
&= \mathbb{E}_{p(\varepsilon)} \nabla_\phi \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t(s_t, \varepsilon_t)) \right] \\
&= \mathbb{E}_{p(\varepsilon)} \left[ \sum_{t=0}^{T} \gamma^t \nabla_\phi r(s_t, a_t(s_t, \varepsilon_t)) \right] \\
&= \mathbb{E}_{p(\varepsilon)} \left[ \sum_{t=0}^{T} \gamma^t \nabla_\phi a_t(s_t, \varepsilon_t) \nabla_a r(s_t, a) \Big|_{a=a_t(s_t,\varepsilon_t)} \right]
\end{aligned}$$

Остается лишь подставить сюда функцию награды:

$$\begin{aligned}
r(s, a) &= -f(s + a) \\
\nabla_a r(s, a) &= -\nabla f(s + a) \\
\nabla_\phi L(\phi) &= -\mathbb{E}_{p(\varepsilon)} \left[ \sum_{t=0}^{T} \gamma^t \nabla_\phi a_t(s_t, \varepsilon_t) \nabla f(s_t + a_t(s_t, \varepsilon_t)) \right]
\end{aligned}$$

Следует отметить, что обычно коэффициент дисконтирования $\gamma < 1$ вводят потому, что длина горизонта $T = +\infty$ и необходимо обеспечить существование $R(\tau)$ (ряд обязан сходиться). В нашей задаче горизонт имеет конечную длину и не имеет смысла делать меньший штраф на более поздних шагах. В связи с этим логично положить $\gamma = 1$.

# 6 Эксперименты

Для начала опишем детали моделей, которые использовались для обучения. Их было две: одна обучена с использованием обучения с подкреплением, а другая нет.

Обе модели состоят из двухслойной LSTM [17] сети, на вход которой подается следующий вектор признаков:

$$[\nabla_t, \nabla_t^2, m_t, v_t, s_t, \mathrm{norm}(\nabla_t), \mathrm{norm}(\nabla_t)^2, m_t^{\mathrm{norm}}, v_t^{\mathrm{norm}}, s_t^{\mathrm{norm}}]$$

Здесь $\mathrm{norm}(x) = \frac{x}{\|x\|}$, $x^2$ означает поэлементное возведение в квадрат, а также используются следующие обозначения:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_t \qquad m_t^{\mathrm{norm}} = \beta_1 m_{t-1}^{\mathrm{norm}} + (1-\beta_1)\mathrm{norm}(\nabla_t)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_t^2 \qquad v_t^{\mathrm{norm}} = \beta_2 v_{t-1}^{\mathrm{norm}} + (1-\beta_2)\mathrm{norm}(\nabla_t)^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{m}_t^{\mathrm{norm}} = \frac{m_t^{\mathrm{norm}}}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \hat{v}_t^{\mathrm{norm}} = \frac{v_t^{\mathrm{norm}}}{1-\beta_2^t}$$

$$s_t = \frac{\hat{m}_t}{\hat{v}_t + \varepsilon} \qquad s_t^{\mathrm{norm}} = \frac{\hat{m}_t^{\mathrm{norm}}}{\hat{v}_t^{\mathrm{norm}} + \varepsilon}$$

По сути, эти входы — просто величины, используемые в тех или иных методах оптимизации. Конкретно здесь они взяты из метода Adam. Далее, модели различаются.

**Модель на основе обучения с учителем.**
Здесь используется модифицированная модель из [1]. На выходе LSTM выдает новое внутреннее состояние и два выхода: $d_t, \log\gamma_t$. Шаг конструируется по формуле:

$$g_t = \alpha_t \frac{d_t}{\|d_t\|}$$

$$\alpha_t = \alpha_{t-1}\gamma_t$$

Здесь $\alpha_t$ — часть внутреннего состояния оптимизатора. По смыслу это текущая длина шага, которая обновляется на каждом шаге с помощью мультипликативной прибавки $\gamma_t$. Такая схема заставляет оптимизатор понимать, что ему нужно сделать с длиной шага: уменьшить, увеличить, или оставить прежней. Если бы длина шага выдавалась напрямую, оптимизатор

мог бы подобрать подходящую длину шага переобучившись на тренировочной выборке. $\log \alpha_0$ выбирается из равномерного распределения $U[-6, -2]$. Назовем эту модель — **BP-модель**.

**Модель с использованием обучения с подкреплением.**
На выходе LSTM выдает новое внутреннее состояние и два выхода: $d_t, \log \alpha_t$. Шаг конструируется по формуле:

$$g_t = \alpha_t \gamma_t \frac{d_t}{\|d_t\|}$$
$$\gamma_t \sim Gamma(2, 1)$$

Здесь длина шага $\alpha_t$ выдается оптимизатором напрямую. Тут имеет смысл так делать, поскольку есть шум, который не позволяет переобучиться на длину шага. Шум обязателен, так как нужна стохастическая политика. На стадии тестирования может иметь или не иметь смысл отключать этот шум. В экспериментах шум отключается. Так как добавленный шум — непараметрический, то легко будет сделать трюк репараметризации. Назовем эту модель — **RL-модель**.

## 6.1 Эксперименты на задачах нестохастической оптимизации

Рассмотрим задачи нестохастической оптимизации. В этих условиях будем подавать на вход моделям лишь градиенты $\nabla_t$. Этого должно быть достаточно для тех случаев, в которых градиенты не зашумлены. Покажем, что модель, основанная на простом обучении с учителем, хорошо запоминает обучающую выборку и достигает неплохого качества на ней. Для этого возьмем следующую задачу:

$$f(x) = \frac{1}{D}\|Wx - b\|^2 \to \min_x$$
$$W \in \mathbb{R}^{D \times D}, b \in \mathbb{R}^D$$
$$W_{ij} \sim \mathcal{N}(W_{ij} \mid 0, 0.1)$$
$$b_i \sim \mathcal{N}(b_i \mid 0, 0.1)$$
$$x_0 \sim \mathcal{N}(x_0 \mid 0, 0.1)$$

Будем использовать тренировочное количество итераций $T_{\text{train}} = 100$, а тестовое — $T_{\text{test}} = 1000$. Для того чтобы сгенерировать очередную функцию, достаточно сгенерировать параметры $W, b$, а также начальную инициализацию $x_0$. Сгенерируем $N = 2000$ таких функций, для каждой сделаем шаг градиентной оптимизации по функции $L(\phi)$ с помощью метода Adam и посмотрим на тестовое качество оптимизации. Для этого запустим оптимизаторы на $M = 100$ функциях и усредним значение функции в зависимости от итерации. Получим следующую картинку 3:

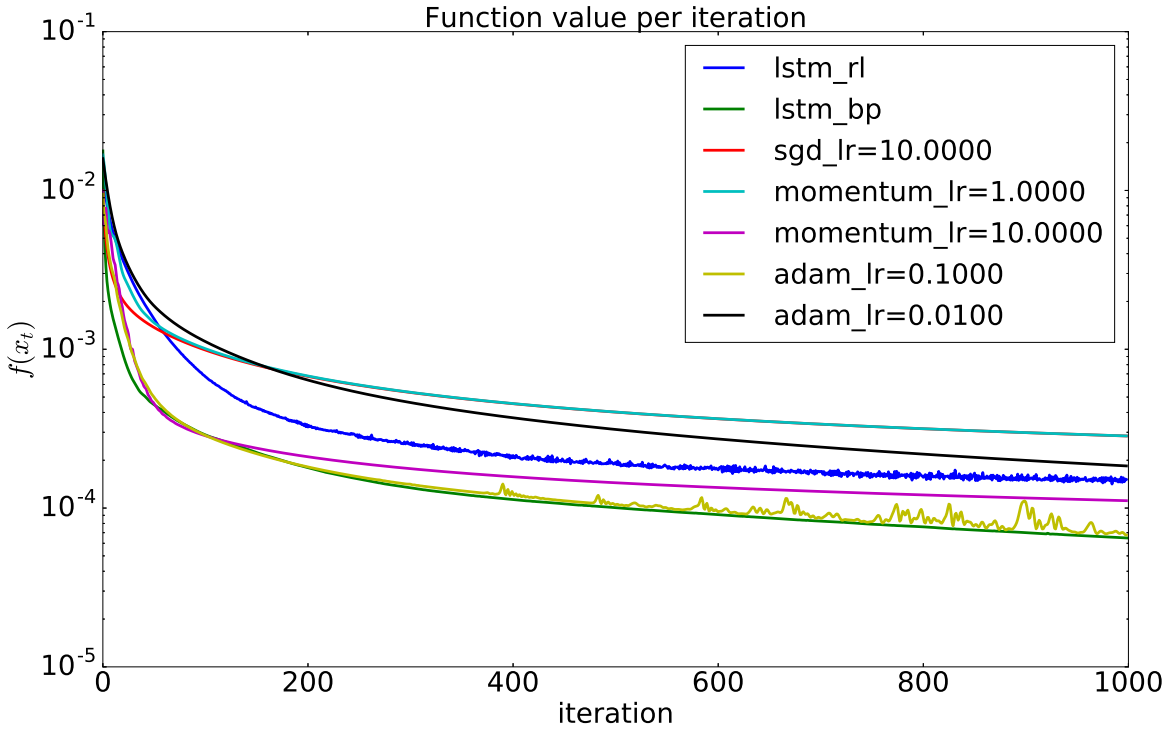



Рис. 3: График квадратичной $f(x)$ в зависимости от итерации для обеих моделей

Как видно, обучаемый BP-оптимизатор работает не хуже остальных, при этом не нужно подбирать его гиперпараметры, такие как длина шага, расписание изменения длины шага, различные коэффициенты, и так далее.

Попробуем изменить тип задачи на такой, который отличается от того что, использовался для обучения оптимизатора. Запустим тот же самый эксперимент на функции Розенброка. Функция Розенброка выглядит так:

$$f(x,y) = \sum_{i=1}^{D} (x_i - a_i)^2 + b_i(y_i - x_i^2)^2$$

$$a \sim \mathcal{N}(a \mid 0, I)$$

$$b \sim \mathrm{LogU}[0, 3]$$

$$x_0, y_0 \sim \mathcal{N}(x_0, y_0 \mid 0, I)$$

На этот раз мы увидим другую ситуацию: 4

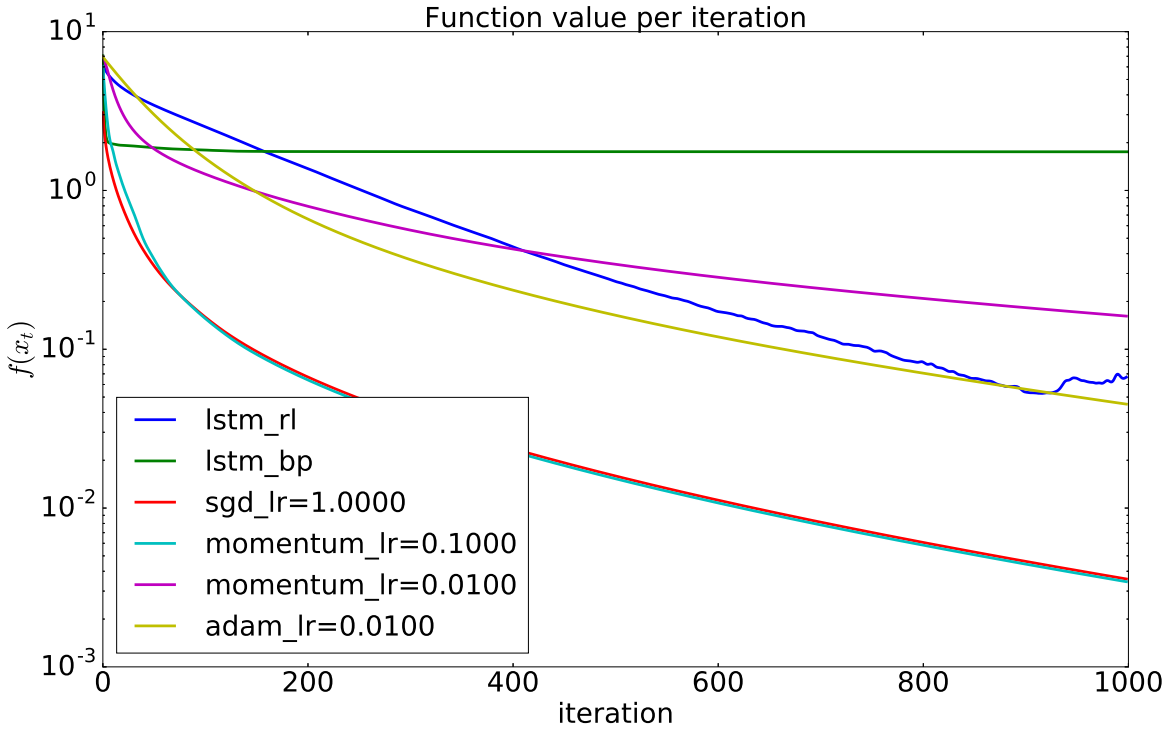



Рис. 4: График значения функции Розенброка в зависимости от итерации для обеих моделей

Как видно, BP-оптимизатор *переобучился* на тип задачи, использовавшийся для обучения. С другой стороны, если посмотреть на то, как справился на обоих задачах RL-метод, то видно, что хотя у него ниже качество на исходной задаче, метод, основанный на обучении с подкреплением имеет более высокую обобщающую способность. BP-метод на обучении имеет выше качество вследствие переобучения.

## 6.2 Эксперименты на задачах стохастической выпуклой оптимизации

Теперь проверим как работают методы, если применять их к задачам стохастической оптимизации. Для начала, попробуем применить их к задачам выпуклой оптимизации, так как они проще. Разумно предположить, что и здесь сохранится та же ситуация: BP-метод будет переобучаться, а RL-метод будет показывать более хорошую обобщающую способность.

В качестве обучающей задачу выберем обучение линейной регрессии:

$$f(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

Так как нам нужно иметь возможность генерировать различные функции, желательно иметь возможность генерировать обучающие выборки для линейной регрессии. Для этого можно воспользоваться следующей процедурой:

1. сгенерируем $X \sim \mathcal{N}(X \mid 0, \sigma^2 I)$, где $\sigma = 0.1$,

2. сгенерируем $\hat{w} \sim \mathcal{N}(\hat{w} \mid 0, I)$,

3. выберем $y$ следующим образом: $y_i = \hat{w}^T x_i + \varepsilon_i$, где $\varepsilon_i \sim \mathcal{N}(\varepsilon_i \mid 0, \sigma^2), \sigma = 0.01$.

Обучив модели на данной задаче, построим график 5 для количества итераций $T = 100$:

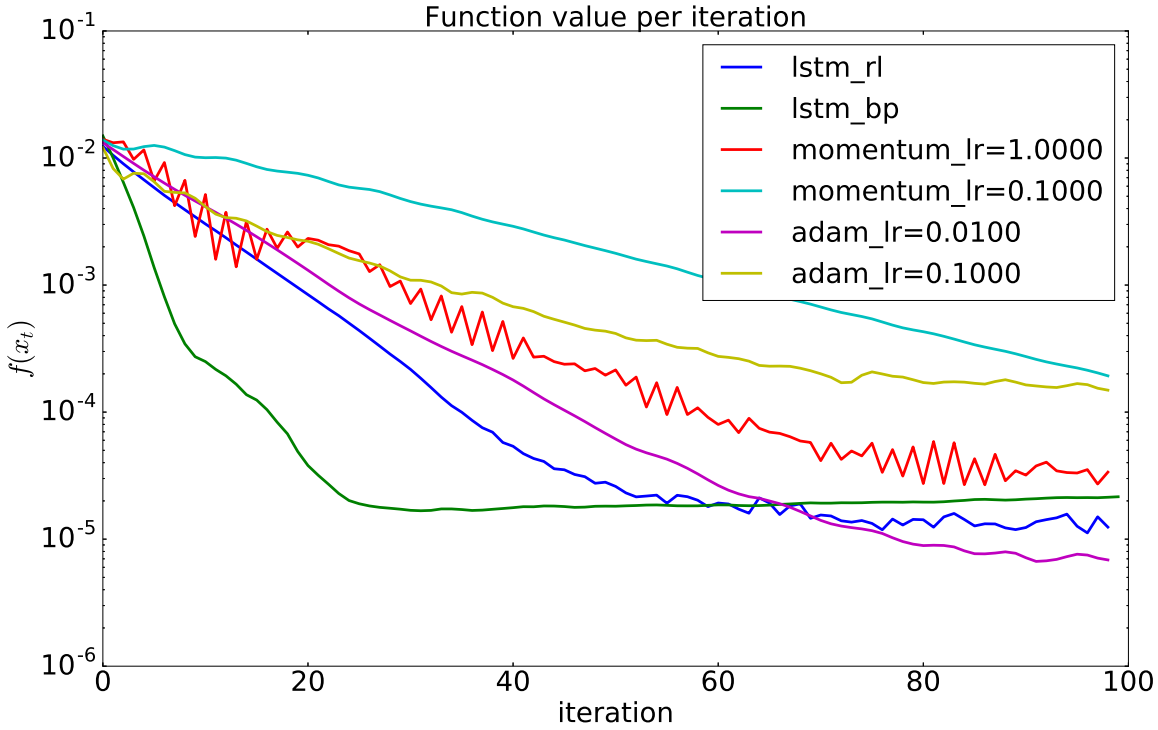



Рис. 5: График значения функции в зависимости от итерации на линейной регрессии (100 итераций)

Из графика легко видеть, что BP-модель отлично запомнила обучающую задачу и показывает на ней лучшую скорость сходимости. Посмотрим теперь, что будет, если увеличить число итераций в 10 раз, т.е. сделать $T = 1000$. Результат отображен на графике 6

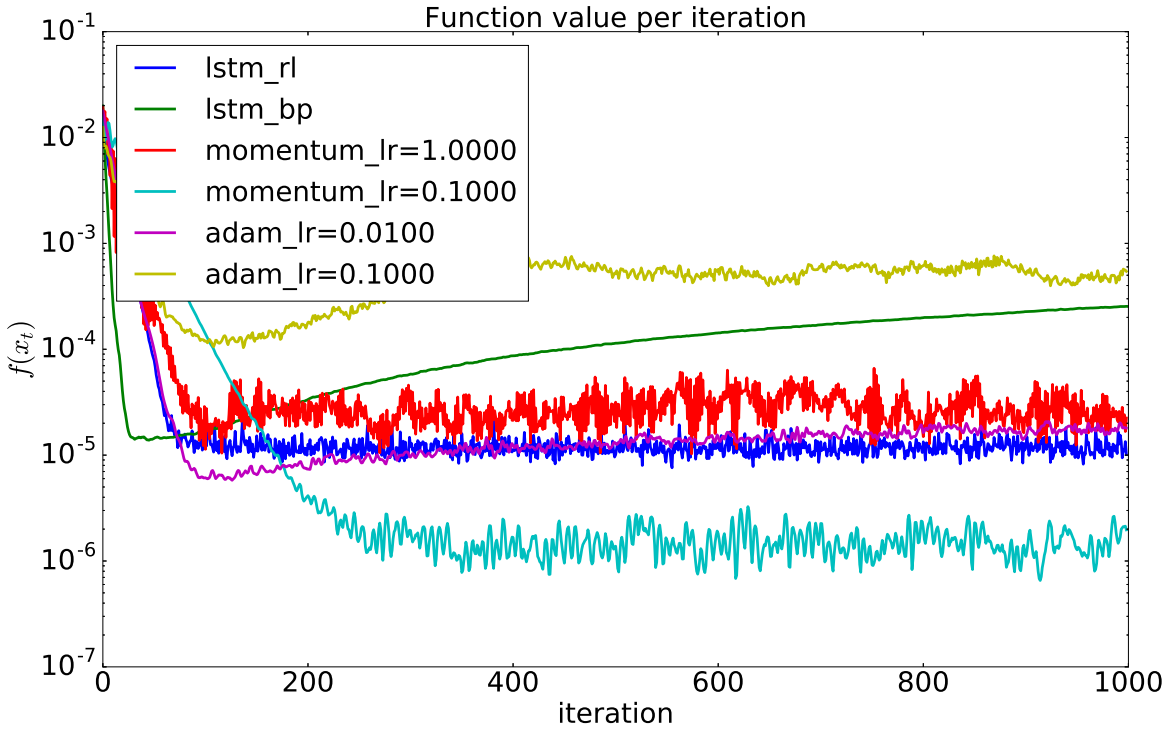



Рис. 6: График значения функции в зависимости от итерации на линейной регрессии (1000 итераций)

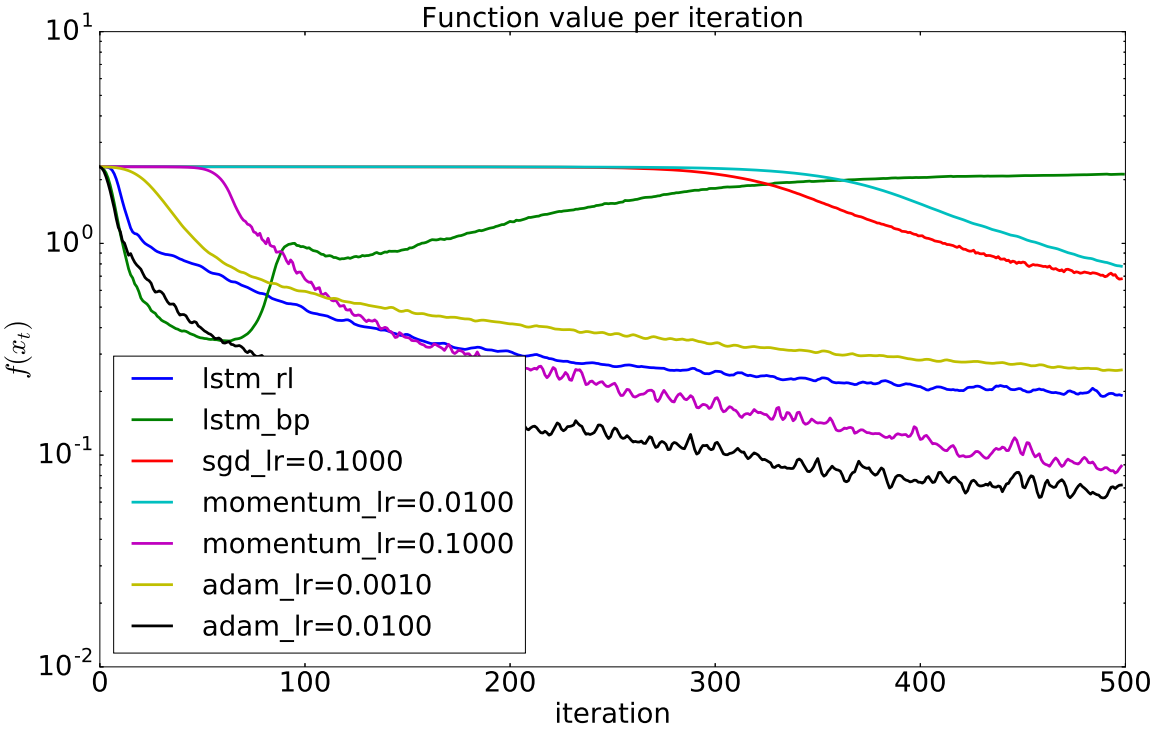
Как видно, BP-модель разошлась при превышении количества итераций $T_{\text{train}} = 100$. Таким образом, наша гипотеза подтвердилась: обобщающая способность модели, использующей обучение с подкреплением, выше.

## 6.3 Эксперименты на нейросетях

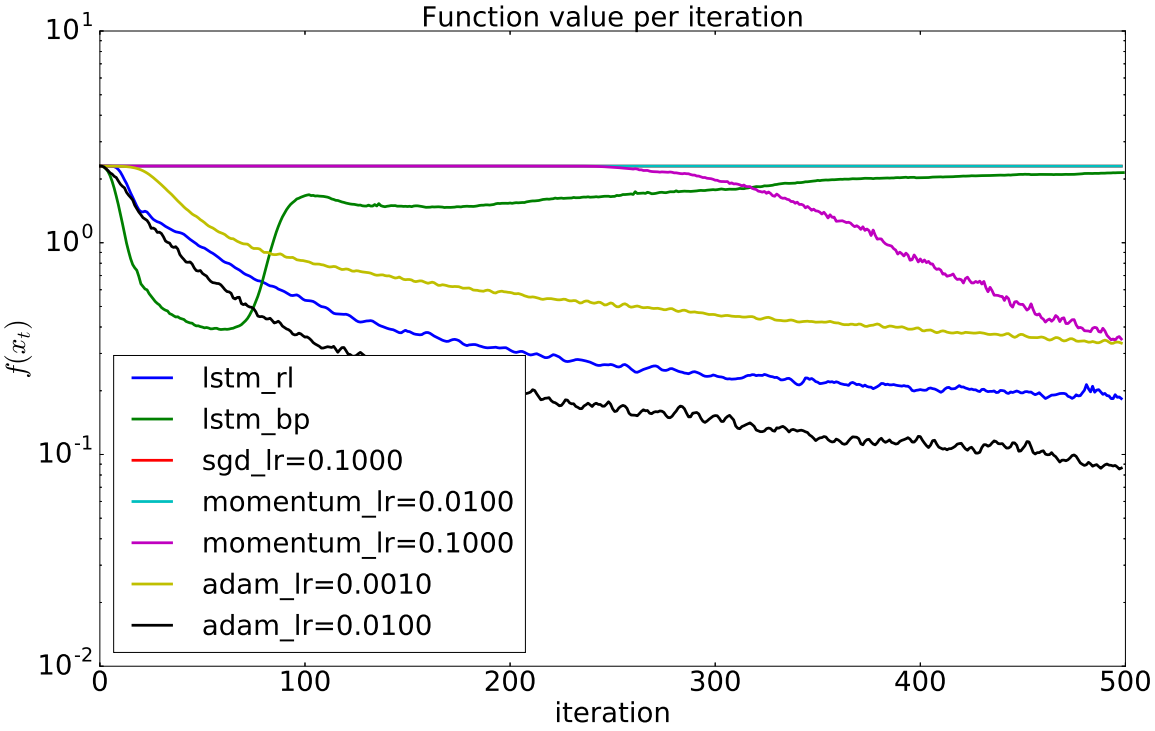
Наконец, посмотрим на результаты, полученные при обучении различных нейросетей. Основные эксперименты с нейросетями были произведены на полносвязных сетях с различным количеством слоев. Эти сети были двух типов: классификаторы и автокодировщики. Данные, которые использовались для обучения — датасет MNIST [18]. Для проверки обобщающей способности менялись следующие параметры: количество слоев, количество нейронов в слоях, использовалась или нет батч-нормализация [19, 20].

Мета-обучение производилось на стохастических задачах оптимизации. Это были задачи обучения следующих моделей: линейная регрессия, логистическая регрессия, однослойный полносвязный классификатор, однослойный автокодировщик. Тестирование производилось на обучении классификаторов и автокодировщиков, в которых было от 1 до 10 слоев, а так же была включена или отключена батч-нормализация. На этапе обучения было выбрано количество итераций $T_{\text{train}} = 50$. На тесте использовалось $T_{\text{test}} = 500$. Несколько показательных графиков представлены на рисунках 7 и 8. Легко видеть, что до достижения числа итераций $T_{\text{train}}$, BP-модель быстро уменьшает функцию потерь. При этом, после превышения трениро-

вочного числа итераций BP-модель начинает довольно скоро расходиться, что свидельствует о переобучении метода. RL-модель, напротив, стабильно уменьшает функцию потерь даже при десятикратном превышении числа итераций, которое было на обучении. Это показывает, что RL-метод выучил и обобщил правило, позволяющее уменьшать значение функции потерь и далее.

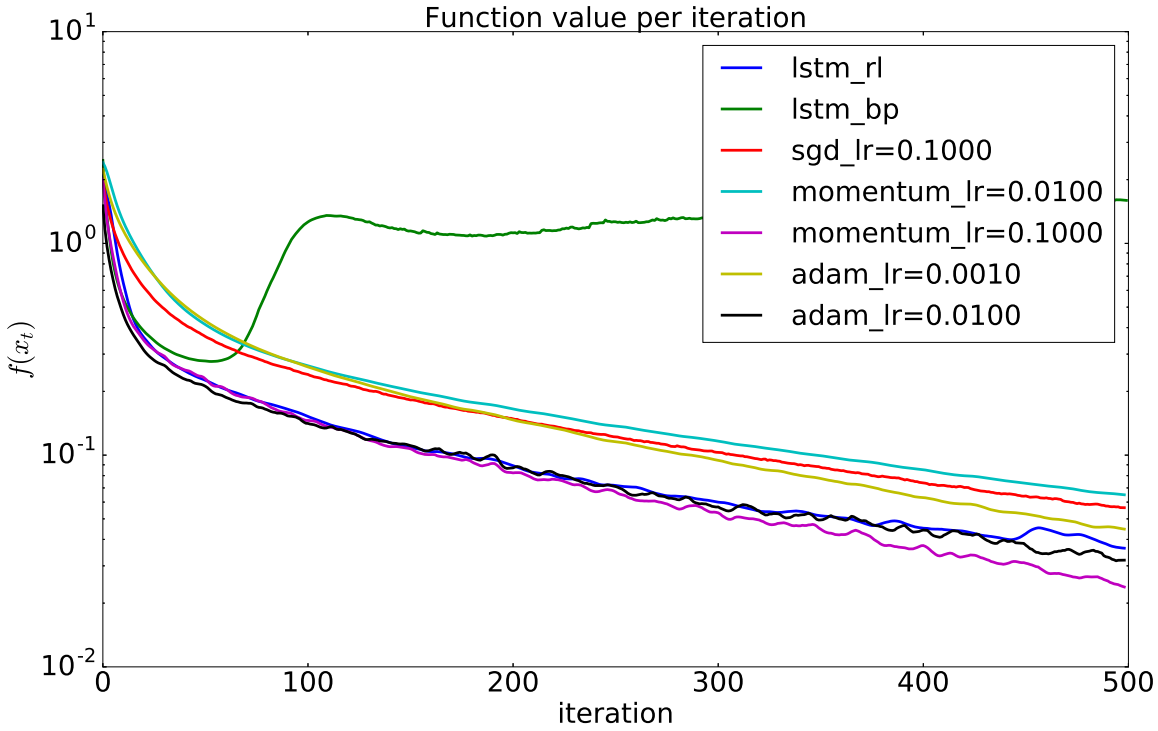



(a) 2 слоя, 100 нейронов в каждом

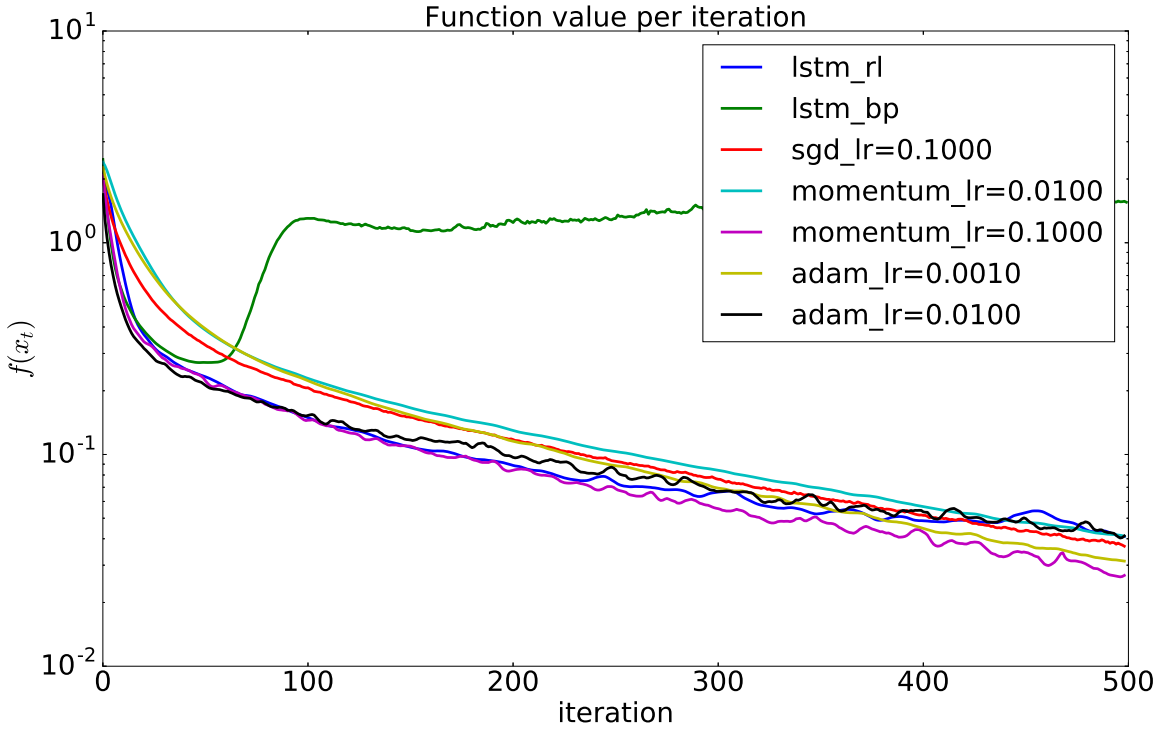



(b) 3 слоя, 100 нейронов в каждом

Рис. 7: График значения функции в зависимости от итерации для нейронных сетей




(a) 2 слоя, 100 нейронов в каждом




(b) 3 слоя, 100 нейронов в каждом

Рис. 8: График значения функции в зависимости от итерации для нейронных сетей с батч-нормализацией

# 7 Выводы

Обучаемые методы оптимизации являются следующим шагом в области способов настройки моделей машинного обучения. Данные методы в ряде случаев позволяют добиться более высокой скорости сходимости, а также не требуют отдельной настройки параметров. Ранее известные методы, которые основаны на обучении с учителем, при прямом применении, к сожалению, показывают склонность к переобучению на те или иные параметры оптимизационного процесса, такие как количество итераций на обучении. Полученный в данной работе метод, основанный на обучении с подкреплением, напротив, показывает более высокую обобщающую способность. Это позволяет с большей уверенностью использовать обучаемые методы оптимизации в применении к тем задачам, которые не встречались в обучающей выборке.

# 8 Заключение

В данной работе были рассмотрены модели обучаемых оптимизаторов на основе рекуррентных нейронных сетей. Было рассмотрено два метода обучения подобных моделей: уже известный — на основе обучения с учителем, а также новый — на основе обучения с подкреплением с использованием той специфики, что награда дифференцируема по действиям. Были проанализированы сильные и слабые стороны таких способов обучения, а также экспериментально проверена их работоспособность и обобщающая способность. Также, было проведено сравнение со стандартными методами, использующимися в машинном обучении. Показано, что методы на основе обучения с учителем имеют более высокую склонность к переобучению, нежели предлагаемый в данной работе метод.

# 9 На защиту выносится

- Новый метод обучения оптимизаторов на основе обучения с подкреплением
- Экспериментальная проверка повышенной обобщающей способности метода на основе обучения с подкреплением

# Список литературы


[1] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

[2] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[3] K. Li and J. Malik, "Learning to optimize neural nets," *arXiv preprint arXiv:1703.00441*, 2017.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[5] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv preprint arXiv:1703.03400*, 2017.

[6] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[7] T. Munkhdalai and H. Yu, "Meta networks," *arXiv preprint arXiv:1703.00837*, 2017.

[8] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International Conference on Learning Representations*, vol. 1, p. 6, 2017.

[11] Y. Chen *et al.*, "Learning to learn for global optimization of black box functions," *arXiv preprint arXiv:1611.03824*, 2016.

[12] E. Mathieu, "Gaussian process bandits,"

[13] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," *arXiv preprint arXiv:1703.03633*, 2017.

[14] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," *arXiv preprint arXiv:1709.07417*, 2017.

[15] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, pp. 1–9, 2013.

[16] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Advances in Neural Information Processing Systems*, pp. 1942–1950, 2017.