Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Физический факультет

Кафедра вычислительной физики

Поляков Даниил Николаевич

Выбор гиперпараметров нейронной сети для решения задач математической физики

Бакалаврская работа

03.03.01 «Прикладные математика и физика»

Научный руководитель:

к.ф.-м.н., доц. **Степанова Маргарита Михайловна**

Рецензент:

к.ф.-м.н. **Шефов Константин Сергеевич**

Санкт-Петербург

2023

Оглавление

Введение

Самыми распространёнными классами методов численного решения дифференциальных уравнений являются метод конечных разностей (МКР) и метод конечных элементов (МКЭ). Данные методы глубоко развивались в течение десятилетий, и в результате были разработаны мощные алгоритмы моделирования различных физических процессов [1].

В этих методах производится дискретизация области определения уравнения, т. е. построение сетки на расчётной области. Подбор правильной сетки сам по себе является нетривиальной задачей. В связи с этим данным методам свойственны ошибки дискретизации, а при повышении размерности задачи экспоненциально растут потребности в вычислительных ресурсах.

За последние годы с помощью искусственных нейронных сетей решается огромное количество теоретических и прикладных задач. Современные методы глубокого обучения уже хорошо зарекомендовали себя в моделировании физических явлений. Например, в работе [2] продемонстрирована способность глубоких нейронных сетей моделировать ливни частиц в коллайдерах, в работе [3] показано, что они могут предоставить быстрые эмуляторы подсеточных процессов в моделировании климата, а в работе [4] предложили общий метод поиска архитектуры нейронной сети, способный ускорить моделирование для широкого круга научных задач. Глубокое обучение также оказывает большое влияние в области механики жидкости [5].

В частности, поскольку нейронная сеть является универсальным аппроксиматором [6], с её помощью можно аппроксимировать функции и решать дифференциальные уравнения. В работе [7] вводится класс PINN-сетей, ориентированных на решение дифференциальных уравнений с частными производными. Его идея заключается в том, чтобы использовать решаемое уравнение в составе целевой функции нейронной сети. При этом производная от целевой функции по параметрам сети может быть вычислена аналитически, с использованием автоматического дифференцирования, на котором основано обучение нейронных сетей. Таким образом, сеть может быть обучена без обучающей выборки.

Преимуществом нейронных сетей над МКР и МКЭ является их способность к генерализации, т. е. предсказанию решения в области вне обучающей выборки. В работе [8] представлено решение двумерного волнового уравнения с помощью нейронной сети, обученной по выборке, взятой в начальный момент времени; при этом полученная сеть даёт хорошее приближение решения в последующие моменты времени.

Для получения наилучшего результата перед обучением модели необходимо правильно подобрать конфигурацию её гиперпараметров. Под гиперпараметрами нейронной сети понимают параметры, значения которых не меняются в ходе обучения модели. Это могут быть параметры, связанные с архитектурой модели, такие как количество скрытых слоёв, ширина слоёв и функция активации, или параметры, относящиеся к процессу обучения модели, такие как скорость обучения, размер батча, оптимизирующий алгоритм и его возможные параметры. От гиперпараметров зависит скорость сходимости модели и её максимально возможная точность.

Процесс подбора конфигурации модели часто оказывается очень трудоёмким. В связи с этим в настоящее время ведутся исследования и разработки в области автоматической оптимизации гиперпараметров (*Hyper-Parameter Optimization*, HPO) [9]. Средства автоматической оптимизации гиперпараметров позволяют уменьшить затраты человеческих ресурсов, но взамен, как правило, возникает потребность в большом количестве вычислительных ресурсов, особенно в случае одновременной оптимизации большого количества гиперпараметров.

В работе [10] представлен пример процедуры оптимизации гиперпараметров при решении двумерного уравнения Гельмгольца.

В настоящей работе исследуются процессы ручной и автоматической оптимизации гиперпараметров нейронной сети для решения уравнения Гельмгольца в пространствах больших размерностей (до 8). Построение и тренировка модели проводится на базе фреймворка PyTorch без использования специальных библиотек для PINN-сетей, а автоматическая оптимизация гиперпараметров проводится с использование фреймворка Ray Tune.

Цели, поставленные в текущей работе:

1. изучить влияние гиперпараметров на сходимость нейронной сети к решению;
2. сравнить методы автоматического подбора гиперпараметров;
3. исследовать возможность решения уравнения Гельмгольца в пространстве большой размерности.

1. Теория

Далее приведены общие сведения о нейронных сетях, об их применении в решении уравнений, гиперпараметрах нейронных сетей и алгоритмах оптимизации гиперпараметров.

1.1. Искусственная нейронная сеть

В настоящей работе ведётся работа с искусственными нейронными сетями с прямой связью (*Feed-forward Neural Network*, FNN). Нейронные сети представляют собой композицию линейных и нелинейных преобразований. Элементарным элементом нейронной сети является нейрон. В случае полносвязной нейронной сети значение, возвращаемое нейроном, выражается следующим образом:

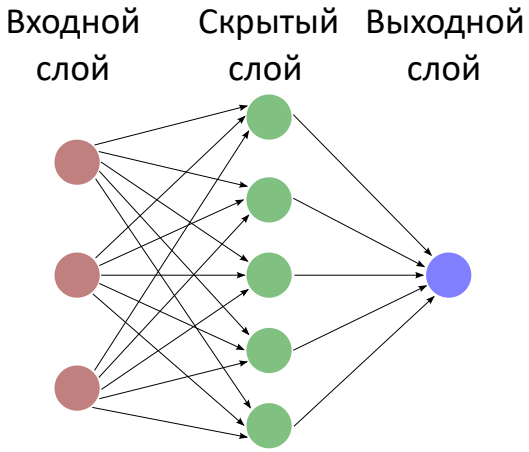

Рисунок 1.
Однослойная нейронная сеть

$$y_j = \sigma_j\left(b_j + \sum_i^n w_{ji}x_i\right)$$

x_i — значения на выходе из нейронов предыдущего слоя,
w_{ji} — весовые коэффициенты, b_j — сдвиги, σ_j — функция активации.

Функция активации должна быть нелинейной и непрерывно дифференцируемой функцией. В таком случае нейронная сеть является универсальным аппроксиматором [11][12], что обуславливает их применимость в широком спектре задач.

Многослойная (глубокая) нейронная сеть выражается суперпозицией подобных преобразований:

$$f(\boldsymbol{x}) = \varphi_L(\ldots\varphi_2(\varphi_1(\boldsymbol{x})))$$

Нейронные сети предоставляют значительную гибкость при выборе своей структуры. Для достижения хорошего результата необходимо подобрать оптимальную конфигурацию сети (модель), что является довольно сложной задачей.

После построения модели сети её необходимо обучить, т. е. подобрать значения её параметров θ (весов и сдвигов), при которых модель правильно сопоставляет входные и выходные данные. Точность модели должна быть выражена численно, и для этого используется целевая функция (функция ошибки, loss-функция). Целевая функция должна быть дифференцируемой функцией от выходных параметров модели. Её необходимо правильно подобрать в соответствии с решаемой задачей. Базовым методом обучения нейронных сетей является градиентный спуск. В данном методе вычисляются производные от целевой функции по параметрам модели θ, которые в дальнейшем используются для корректировки θ. Поскольку производные указывают направление роста функции, то для уменьшения целевой функции значения θ сдвигают в противоположную сторону:

$$\theta_i = \theta_{i-1} - lr \cdot \nabla_\theta \text{loss}(\theta_{i-1})$$

Здесь lr (learning rate) — коэффициент, называемый скоростью обучения.

В нейронных сетях производные по параметрам в большинстве случаев вычисляются аналитически, а современные фреймворки для машинного обучения предоставляют средства для их эффективного расчёта (автоматическое дифференцирование). За обучение модели отвечает оптимизирующий алгоритм. Большинство оптимизирующих алгоритмов так или иначе используют в своей работе производные от целевой функции.

1.2. Решение уравнений с помощью нейронных сетей

Постановка задачи

Искусственные нейронные сети можно применить для решения дифференциальных уравнений в частных производных вида [13]:

$$f\left(\boldsymbol{x}; u; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_n \partial x_n}\right) = 0, \quad \boldsymbol{x} \in D, \quad D \subset \mathbb{R}^n$$

с граничными условиями:

$$b(\boldsymbol{x}, u)\big|_\Gamma = 0$$

которые могут быть условиями Неймана, Дирихле или периодическими. Нейронные сети, применяемые для решения задач такого рода, обозначают термином «Physics-Informed Neural Networks» (PINN) [7].

Ключевой особенностью использования нейронной сети в качестве аппроксимирующей функции для $u(\boldsymbol{x})$ является возможность аналитического вычисления производных по $\boldsymbol{x}$.

Целевая функция

В качестве меры ошибки и целевой функции можно использовать взвешенную сумму L^2 норм отклонений (невязок) по уравнению и по границе:

$\text{loss} = \omega_f \text{loss}_f + \omega_b \text{loss}_b$

$$\text{loss}_f = \frac{1}{|N_f|} \sum_{\boldsymbol{x} \in N_f} \left\| f\left(\boldsymbol{x}; \hat{u}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_n}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_n \partial x_n} \right) \right\|^2$$

$$\text{loss}_b = \frac{1}{|N_b|} \sum_{\boldsymbol{x} \in N_b} \left\| b(\boldsymbol{x}, \hat{u}) \right\|^2$$

где ω_f, ω_b — веса; $N_f = \left\{ x_1, \ldots, x_{|N_f|} \right\}$, $N_b = \left\{ x_1, \ldots, x_{|N_b|} \right\}$ — множества точек коллокации (тренировочная выборка) по уравнению и по границе соответственно; $|N_f|$, $|N_b|$ — количества точек; $\hat{u}$ — решение нейронной сети.

В случае простейших граничных условий вида $u|_\Gamma = g(\boldsymbol{x})$, $D = [a, b]^n$ их возможно форсировать, домножив выходное значение нейронной сети на гладкую функцию, удовлетворяющую этим условиям:

$$\widetilde{u} = g(\boldsymbol{x}) + u \cdot \prod_i^n (x_i - a)(b - x_i)$$

Тогда пропадает необходимость в точках коллокации на границе и в целевой функции остаётся только слагаемое, отвечающее за невязку по уравнению:

$$\text{loss} = \text{loss}_f$$

Анализ ошибки

Ограниченный размер нейронной сети обуславливает ограниченную точность достигаемого решения. Обозначим через F множество всех функций, которые могут быть выражены выбранной архитектурой сети. Крайне маловероятно, что точное решение уравнения u принадлежит данному множеству. Обозначим u_F наилучшее приближение u функцией из F: $u_F = \arg\min_{f \in F} \|f - u\|$. Другой вклад в ошибку решения вносит конечность набора точек коллокации N. Решение, при котором на заданном наборе достигается минимальное значение целевой функции, обозначим u_N: $u_N = \arg\min_{f \in F} \|\text{loss}(f)\|$. Наконец, при оптимизации модели почти никогда не удаётся достигнуть глобального минимума целевой функции, и мы получаем решение $\widetilde{u}_N \neq u_N$.

Таким образом, полная ошибка состоит из трёх частей:

$$\varepsilon = \|\widetilde{u}_N - u\| \leq \|\widetilde{u}_N - u_N\| + \|u_N - u_F\| + \|u_F - u\|$$

— ошибки оптимизации, ошибки обобщения и ошибки аппроксимации.

1.3. Гиперпараметры

Гиперпараметры можно разделить на две группы: относящиеся к архитектуре модели и относящиеся к обучению. Типичными архитектурными параметрами сети являются количество слоёв L, ширина слоя N и функция активации σ. Эти гиперпараметры влияют на способность модели к обучению, т. е. её максимально возможную точность и способность к генерализации. Выбор гиперпараметров, связанных с обучением модели, влияет по большей мере на скорость её обучения. Типичными такими параметрами являются скорость обучения, размер батча и оптимизатор. В настоящее время наиболее распространённым оптимизатором для обучения глубокой нейронной сети является стохастический градиентный спуск (SGD) с импульсом [14] и его вариации: AdaGrad [15], RMSProp [16] и Adam [17]. При этом у оптимизатора могут быть свои параметры (например, импульс), которые в некоторых случаях оказывают критическое влияние на обучение модели.

Скорость обучения

Скорость обучения (lr) — положительная скалярная величина, определяющая величину шага градиентного спуска. Часто этот параметр приходится варьировать в процессе обучения для достижения лучшей точности модели. Как правило, обучение начинают с большого значения lr и уменьшают его по ходу обучения. Это можно делать вручную, либо используя планировщик, автоматически изменяющий скорость обучения в процессе обучения. В таком случае lr может быть функцией итерации обучения или значения функции потерь (loss). Возможные реализации планировщика:

- *Линейный*

  Может быть выражен функцией:

  $$lr = lr_0 + (lr_1 - lr_0) \cdot \min\left(\frac{t}{T},\ 1\right)$$

  lr_0 — начальное значение скорости обучения;
  lr_1 — конечное значение скорости обучения;
  t — номер итерации;
  T — количество итераций, в ходе которых lr меняется.

- *Экспоненциальный*

  Может быть выражен функцией:

  $$lr = lr_0 \cdot \gamma^t$$

  lr_0 — начальное значение скорости обучения;
  γ — мультипликативный коэффициент;
  t — номер итерации.

- *Пошаговый*

  Может быть выражен функцией:

  $$lr = lr_0 \cdot \gamma^{\text{floor}\left(\frac{t}{T}\right)}$$

  lr_0 — начальное значение скорости обучения;
  γ — мультипликативный коэффициент;
  t — номер итерации;
  T — период изменения lr.

- *Циклический*

  Скорость обучения варьируется циклически между границами, которые могут изменятся во времени:

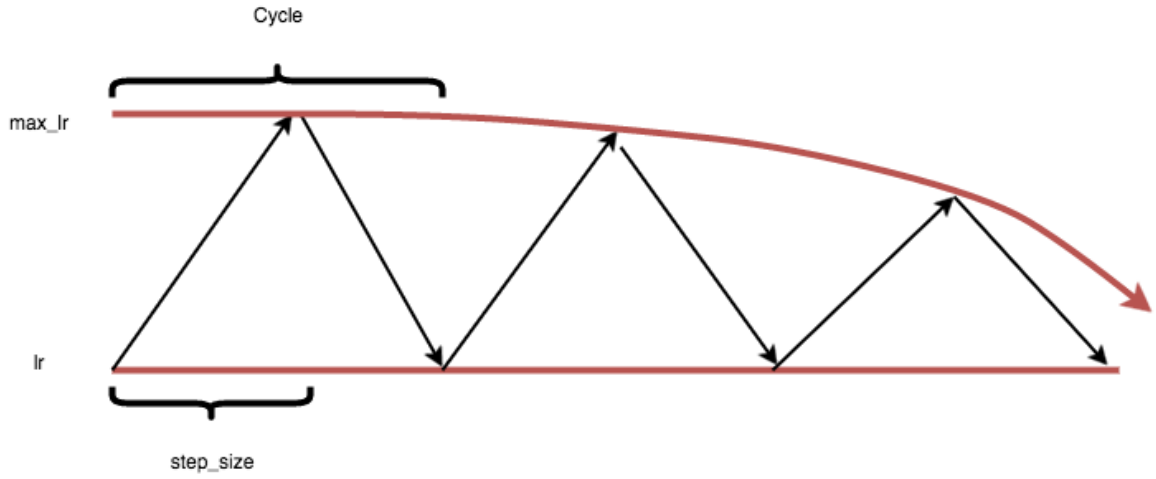

Рисунок 2. *Динамика скорости обучения при циклическом планировщике* [18]

- *Уменьшение скорости обучения при выходе на плато*

  lr стартует с некоторого значения lr_0 и уменьшается в γ раз при отсутствии улучшения целевой функции в течение T итераций.

Среди перечисленных выше обозначений lr_0, lr_1, T, γ являются гиперпараметрами. Выбор оптимального значения lr или параметров его планировщика является непростой задачей. Малое значение lr приводит к медленной сходимости, а большое значение lr препятствует сходимости. Кроме того, оптимальные параметры lr-планировщика зависят от выбранного оптимизатора [19], поэтому их следует оптимизировать друг относительно друга.

Размер батча

Использование батчей при градиентном спуске решает две проблемы [20]. По сравнению со стандартным стохастическим градиентным спуском, когда спуск производится с целевой функции, вычисленной по одной точке данных, разбиение данных на батчи ускоряет процесс обучения, особенно в случае больших датасетов. Помимо этого, использование батчей уменьшает шум и повышает вероятность сходимости [21]. Максимальный размер батча ограничен доступной памятью устройства, на котором ведётся обучение модели.

Между оптимальными значениями размера батча и скорости обучения наблюдается связь: чем больше размер батча, тем больше соответствующее оптимальное значение скорости обучения [22].

Оптимизирующий алгоритм

Базовым методом оптимизации нейронной сети является стохастический градиентный спуск (SGD). Данному оптимизатору свойственны осцилляции около локального минимума, поэтому часто используется его модифицированная версия: стохастический градиентный спуск с импульсом (SGD with momentum) [23]. В этом случае параметры модели θ (веса и сдвиги) обновляются следующим образом:

$$d\theta_i = \nabla_\theta \text{loss}(\theta_{i-1})$$
$$v_i = \beta v_{i-1} + (1-\beta) d\theta_i$$
$$\theta_i = \theta_{i-1} - lr \cdot v_i$$

Таким образом, величина импульса β тоже является гиперпараметром. Добавление импульса уменьшает осцилляции и способствует изменению параметров модели в аккумулированном за предыдущие итерации направлении.

Root mean square prop (RMSprop) — один из распространённых оптимизаторов [24]. RMSprop ускоряет градиентный спуск по аналогии с Adagrad и Adadelta, но демонстрирует лучшую эффективность, когда сходимость замедляется. По сравнению со стандартным градиентным спуском RMSprop уменьшает осцилляции целевой функции.

Adaptive momentum estimation (Adam) тоже быстро достигает хороших результатов на большинстве нейросетевых архитектур. Являясь комбинацией SGD и RMSprop, Adam корректирует смещения импульсов, что позволяет ему превзойти RMSprop по эффективности на поздних стадиях обучения. В прикладных задачах, Adam рекомендуется использовать по умолчанию при обучении глубоких нейронных сетей [24]. Хотя у этого алгоритма больше гиперпараметров, чем у других оптимизаторов, он хорошо работает и при стандартных значениях.

Как правило, оптимизаторы следует конфигурировать в связке с lr-планировщиком.

Количество скрытых слоёв

Глубокие нейронные сети с большим количеством скрытых слоёв обычно обладают бóльшим потенциалом к обучению и достижению лучшей точности. Добавление слоёв является типичным методом повышения точности модели [25].

Ширина скрытого слоя

С одной стороны, очень малое количество нейронов в скрытом слое приводит к «слабой» модели, неспособной аппроксимировать зависимость между входными и выходными данными. С другой стороны, увеличивает временны́е затраты на обучение и может привести к переобучению модели.

Регуляризация

По мере увеличения количества слоёв и нейронов возникает проблема переобучения, когда результат модели улучшается на тренировочном наборе данных, при этом уменьшаясь на тестовом наборе. Для решения этой проблемы применяют методы регуляризации, когда вместе с минимизацией целевой функции стараются минимизировать абсолютные значения параметров модели θ. Член регуляризации w^* добавляется к целевой функции с некоторым коэффициентом λ:

$$\text{loss}^* = \text{loss} + \lambda w^*$$

Типичные варианты регуляризации:

L1-регуляризация:

$$w^* = \sum_i |\theta_i|$$

L2-регуляризация:

$$w^* = \sum_i \theta_i^2$$

Слишком большие значения λ могут привести к чрезмерному упрощению сети, а слишком малые λ могут не решить проблему переобучения. Поэтому данный коэффициент является гиперпараметром, подлежащим оптимизации.

Функция активации

Функции активации необходимы в нейронных сетях, потому что они вносят нелинейность на выходе нейронов. Без функций активации нейронная сеть оказывается моделью линейной регрессии, не способной выразить сложную зависимость между данными. Функции активации должны быть дифференцируемы, чтобы можно было рассчитать производные и провести градиентный спуск по всем параметрам сети. Примеры самых распространённых функций активации: сигмоида, гиперболический тангенс, ReLU и её вариации. Более узконаправленные функции активации включают в себя синус и арктангенс, которые могут оказаться полезны в рассматриваемой задаче.

1.4. Алгоритмы поиска

Автоматическая оптимизация гиперпараметров состоит из двух основных компонентов:

- **алгоритм поиска** — выбирает точку из области гиперпараметров для рассмотрения;
- **планировщик обучения** — заранее останавливает процесс обучения сети в случае плохой сходимости относительно предыдущих результатов.

Поиск по сетке

Самым базовым методом поиска является поиск по сетке. Он производит исчерпывающий поиск по заданному набору точек в области гиперпараметров. Пользователь сам выбирает рассматриваемые точки, поэтому в таком случае следует обладать некоторым пониманием влияния гиперпараметров на результат. Такой поиск применим только для небольшого набора гиперпараметров. При увеличении числа гиперпараметров количество точек растёт экспоненциально («проклятие размерности»). Поиск по сетке может производиться параллельно, потому что выбор всех рассматриваемых точек производится заранее.

Случайный поиск

Улучшение по сравнению с поиском по сетке даёт случайный поиск. В соответствии с этим методом точки выбираются случайно из заданных распределений. Процесс поиска продолжается до тех пор, пока не будет получена необходимая точность, либо до истечения выделенных ресурсов. Являясь методом Монте-Карло, он обеспечивает более быструю сходимость по сравнению с поиском по сетке, особенно при больших размерностях области поиска. Случайный поиск может производиться параллельно, потому что выбор каждой последующей точки производится независимо от предыдущей.

Байесовские методы оптимизации

Байесовская оптимизация — последовательный итерационный алгоритм, основанный на двух компонентах: *вероятностной суррогатной модели* и *функции сбора*. На каждой итерации суррогатная модель подгоняется ко всем накопленным наблюдениям целевой функции. Начальный набор точек выбирается случайно. На основе этой модели функция сбора выбирает следующую точку для рассмотрения, пытаясь обеспечить оптимальный баланс между «эксплуатацией» найденного минимума и рассмотрением точек в неисследованных областях.

Суррогатная модель используется для выражения гипотетического вида целевой функции. Существует множество вариантов таких моделей [26], но традиционно применяются гауссовы процессы [27]. Другой популярной моделью является древовидный оцениватель Парзена (*Tree Parzen Estimator*, TPE) [28]. В отличие от гауссовых процессов, работающих с непрерывными пространствами поиска, TPE может эффективно обрабатывать категориальные и условные пространства, и поэтому в задачах машинного обучения TPE может оказаться эффективнее.

Наиболее распространённой функцией сбора является ожидаемое улучшение (*expected improvement*, EI) [29]:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = \mathbb{E}[\max(f_{\min} - y,\ 0)]$$

$\mathbb{I}(\boldsymbol{\lambda})$ — функция улучшения;
λ — конфигурация в пространстве гиперпараметров;
y — моделирующая функция;
$f_{\min}$ — наилучшее значение целевой функции в данный момент.

Ожидаемое улучшение можно вычислить аналитически, если моделирующая функция строится по нормальному распределению.

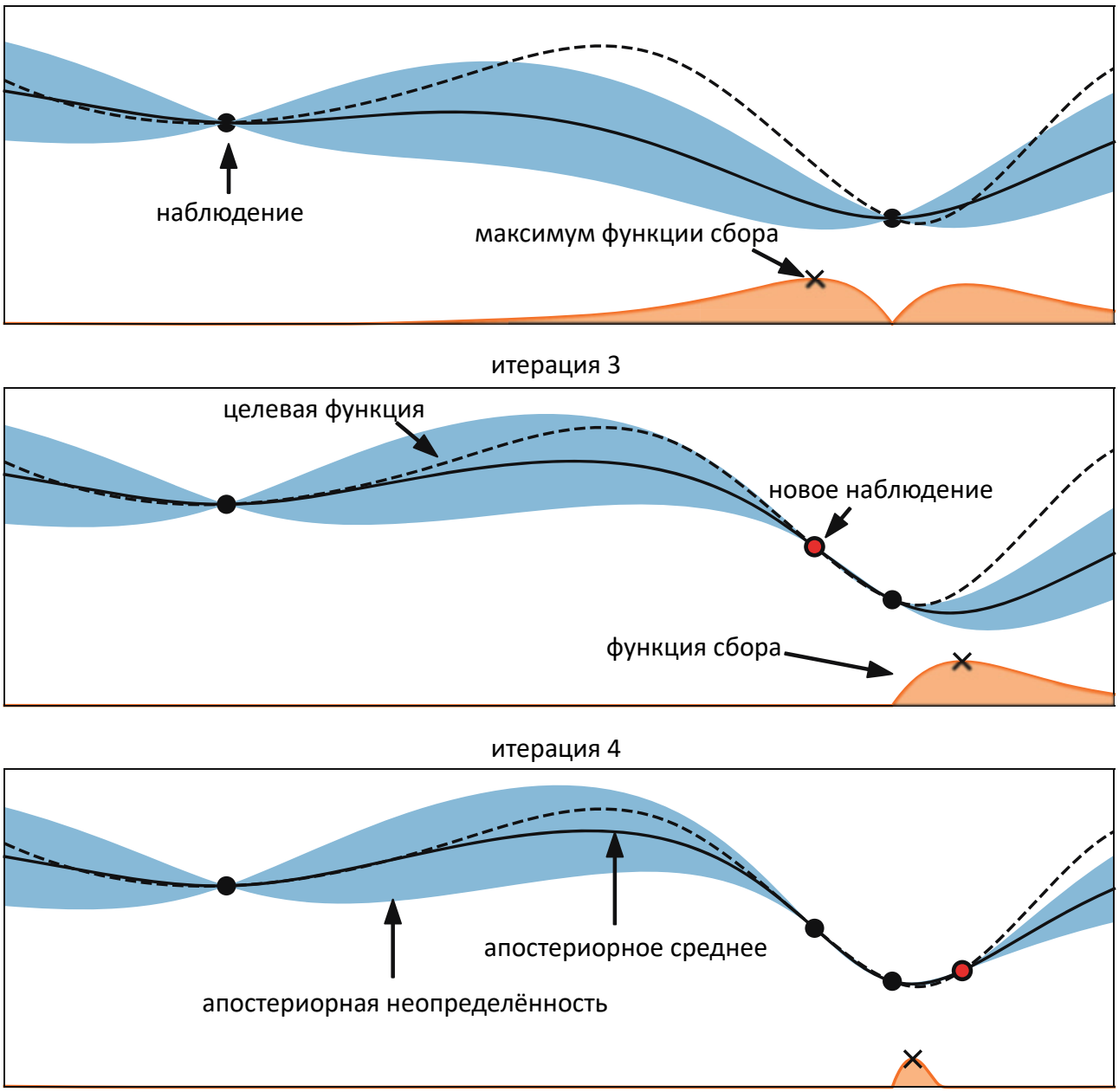

Рисунок 3. *Иллюстрация байесовской оптимизации одномерной функции* [30]

Байесовская оптимизация является ресурсоёмким алгоритмом, поэтому её целесообразно применять только для оптимизации достаточно ресурсоёмких функций. Задача оптимизации гиперпараметров в машинном обучении как раз удовлетворяет этому условию. Другим недостатком метода является его последовательность, обусловленная перестроением суррогатной модели после каждой итераций. По этой причине алгоритм не подлежит распараллеливанию, которое, тем не менее, возможно реализовать, строя временные суррогатные модели на основе оценок значений целевых функций, что, однако, приводит к понижению эффективности метода.

1.5. Планировщики обучения

Обучение нейронной сети — ресурсоёмкий процесс. Скорость обучения модели обусловлена выбором её гиперпараметров, и в случае неоптимальной конфигурации целесообразно прерывать обучение и переходить к другой конфигурации. При этом важно не допустить отказа от потенциально хорошей конфигурации. Для автоматизации этой процедуры применяют планировщики обучения, или алгоритмы ранней остановки. Эти алгоритмы сравнивают динамику обучения при различных конфигурациях между собой и принимают решение о временной либо окончательной остановке обучения при очередной конфигурации.

Медианная остановка

Медианная остановка является простейшим методом ранней остановки. Обучение очередной конфигурации прерывается, если значение её целевой функции начиная с некоторого момента времени оказывается хуже медианного значения предыдущих конфигураций в этот же момент времени.

Successive Halving

Алгоритм последовательного деления (*Successive Halving*, SHA) работает следующим образом. Задаётся максимально допустимый бюджет ресурсов B на конфигурацию (ограничение по времени, количество итераций и т. п.) и количество рассматриваемых конфигураций n. На все конфигурации выделяется доля от бюджета, по истечении которой обучение половины конфигураций останавливается. Через некоторое время прекращается обучение половины от оставшихся конфигураций. Процесс «деления» продолжается несколько раз, и в итоге только наилучшие конфигурации обучаются в течение полного времени.

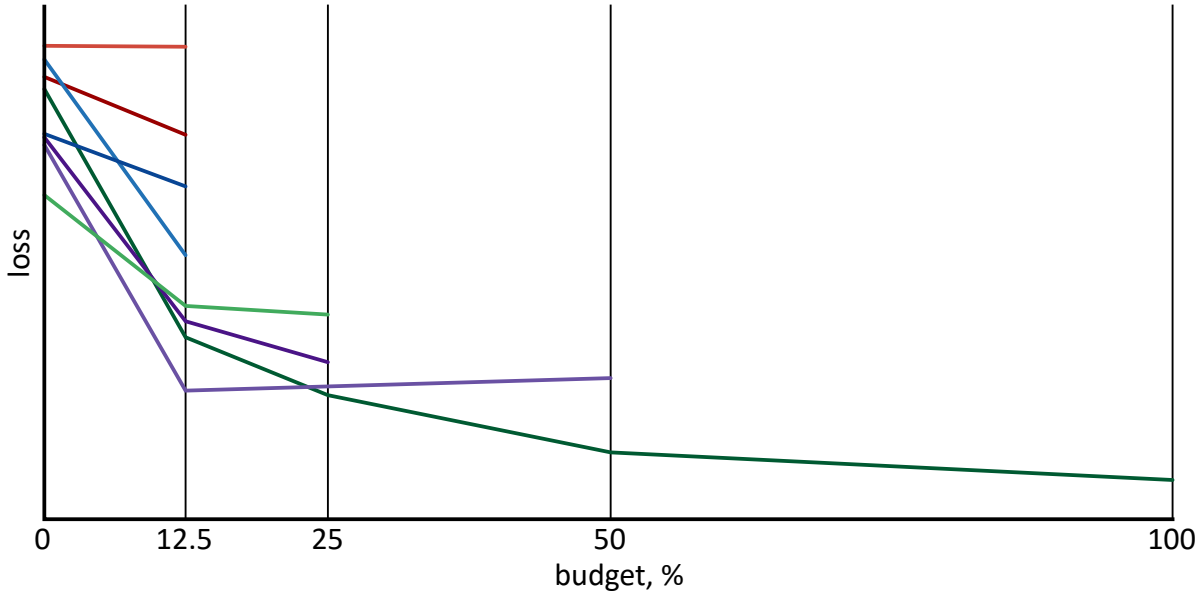

Рисунок 4. *Иллюстрация метода последовательного деления*

HyperBand

Чем больше заданное число конфигураций n, тем меньше ресурсов выделяется на каждую конфигурацию на первом этапе. Поэтому при слишком больших значениях n возможна поспешная остановка обучения, а слишком малое значение n означает недостаточное количество рассматриваемых опций. HyperBand [31] является расширением SHA, призванным решить проблему выбора бюджета B и числа n. HyperBand помещает SHA внутрь цикла, который варьирует значения B и n. Эта мера повышает эффективность алгоритма.

Asynchronous Successive Halving

Асинхронное последовательное деление (*Asynchronous Successive Halving*, ASHA) [32] улучшает производительность SHA в параллельном режиме. В последовательном режиме скорость ASHA не отличается от SHA.

BOHB

Алгоритмы поиска и планировщики обучения могут работать в комбинации независимо друг от друга. BOHB [33] был разработан как попытка объединить байесовский поиск и HyperBand в единый эффективный алгоритм.

2. Описание задачи

В работе рассматривается возможность использования нейронной сети для решения дифференциального уравнения с частными производными. В качестве решаемого уравнения было выбрано уравнение Гельмгольца:

$$-\Delta u(\boldsymbol{x}) - u(\boldsymbol{x}) = (4\pi^2 n - 1) \cdot \prod_i^n \sin(2\pi x_i)$$

$$u|_\Gamma = 0$$

где n — размерность пространства. В качестве области определения выбираем единичный куб:

$$D = \{\boldsymbol{x}\colon\ 0 \leqslant x_i \leqslant 1,\ i = 1, ..., n\}$$

Правая часть уравнения выбрана таким образом, чтобы решение рассчитывалось аналитически:

$$u(\boldsymbol{x}) = \prod_i^n \sin(2\pi x_i)$$

Это решение будет использоваться лишь для проверки сходимости сети, но не будет использоваться при обучении.

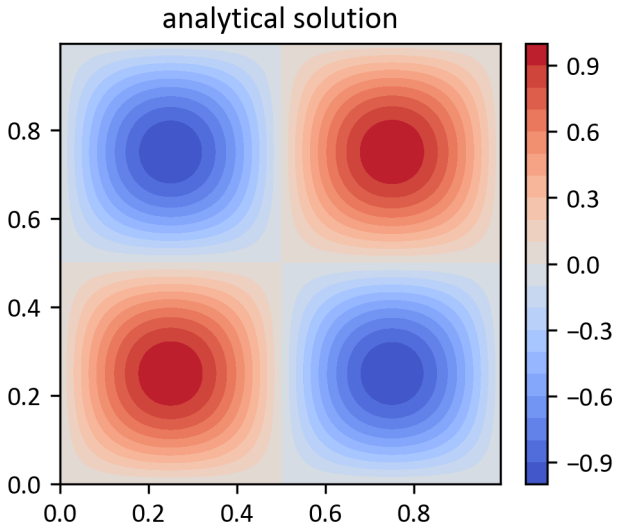

График 1. *Аналитическое решение в двумерном пространстве*

Таким образом, нужно получить сеть с n входами и 1 выходом, наиболее точно решающую заданное уравнение. В качестве меры ошибки используем корень из среднеквадратичного отклонения предсказанных значений функции от аналитического решения (RMSE_u).

3. Методика

3.1. Программные пакеты

Построение и тренировку модели производим с помощью ML-фреймворка PyTorch. В нём реализованы методы, упрощающие построение модели, большинство популярных оптимизирующих алгоритмов и специальные технические средства, такие как автоматическое дифференцирование, векторизация и т. п.

Для автоматической оптимизации гиперпараметров используем фреймворк Ray Tune. Ray Tune предоставляет единый интерфейс для работы с множеством популярных HPO-пакетов:

- алгоритмы поиска: поиск по сетке, случайный поиск, Ax, BayesOptSearch, BOHB, BlendSearch, CFO, Dragonfly, HEBO, HyperOpt, Optuna, SigOpt, Scikit-Optimize, ZOOpt;
- планировщики: ASHA, HyperBand, Median Stopping Rule, Population Based Training (PBT), Population Based Bandits (PB2), BOHB.

В настоящей работе из алгоритмов поиска рассматриваются случайный поиск и HyperOpt, основанный на дереве парзеновских оценок (TPE); из планировщиков рассматривается ASHA. Также рассматривается комбинированный алгоритм BOHB.

3.2. Функционал ошибки

В решаемой задаче возможно жёсткое форсирование граничного условия (см. пп. **3.3**), поэтому функция ошибки будет состоять только из невязки по уравнению:

$$\text{loss} = \text{MSE}_f = \left\langle \left(\Delta u(\boldsymbol{x}) + u(\boldsymbol{x}) + (4\pi^2 n - 1) \cdot \prod_i^n \sin(2\pi x_i) \right)^2 \right\rangle$$

Расчёт лапласиана от выходного значения производится с помощью метода `torch.autograd.grad`, векторизованного посредством метода `torch.func.vmap`. Возможно несколько способов реализации лапласиана, но комбинация этих методов оказалась наиболее эффективной.

Регуляризацию параметров в функцию ошибки мы не добавляем, и в рамках работы её как гиперпараметр не рассматриваем.

3.3. Структура сети

Для решения задачи строим полносвязную нейронную сеть с прямой связью, составленную из линейных слоёв одинаковой ширины, разделённых одинаковыми функциями активации:

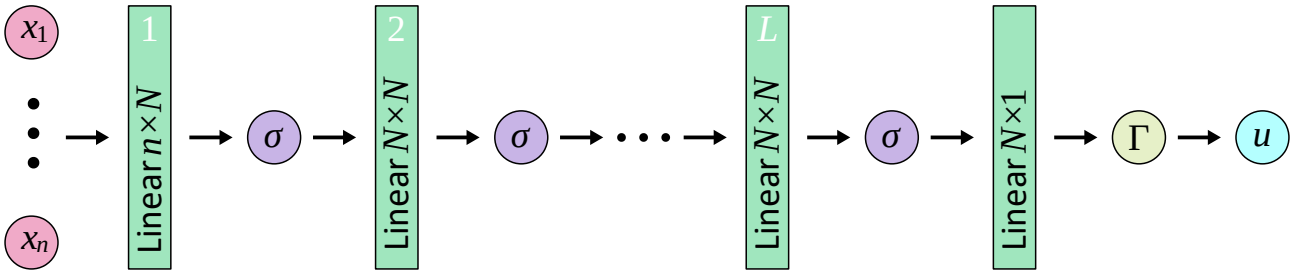

Рисунок 5. *Схема используемой нейронной сети*

Количество слоёв L, ширина слоя N, и функция активации σ подлежат подбору. На выходе сети производится форсирование граничного условия:

$$\widetilde{u} = u \cdot \prod_{i}^{n} (x_i - 0)(1 - x_i)$$

3.4. Генерация точек

Сгенерируем два набора точек коллокации: обучающий набор и тестовый набор. Количество точек в них всегда выбираем одинаковым. Наборы выберем по методу Монте-Карло, т. е. случайно сгенерируем равномерно распределённые по заданной области точки. Количество точек и способ генерации являются дополнительными гиперпараметрами в нашей задаче.

У метода выбора точек из равномерного распределения есть недостаток: равномерное распределение не гарантируется, поэтому возможно образование участков с недостаточным количеством обучающих точек. В связи с этим рассмотрим также *квазислучайную* генерацию обучающей выборки. В качестве псевдослучайного генератора используем скремблированную последовательность Соболя [34][35]. Точки из этой последовательности гарантированно имеют равномерное распределение по области, причём генерируются в таком порядке, что каждый батч оказывается равномерно заполненным. Гипотетически, это должно обеспечить более плавную сходимость модели, однако разница при достаточно больших размерах батча может быть незаметна.

PyTorch содержит готовую реализацию генератора последовательностей Соболя, которую мы будем использовать.

В дальнейшем способ генерации точек выборкой из равномерного распределения будем кратко называть *случайным*, а способ генерации выборкой из последовательности Соболя — *квазислучайным*.

Тестовую выборку будем всегда выбирать из классического равномерного распределения.

Перед генерацией точек генераторы всегда инициализируются одним и тем же значением затравки, поэтому в случаях, где количество точек и способ генерации совпадают, наборы состоят из одинаковых точек.

Все указанные в результатах значения невязки по уравнению (MSE_f) и невязки по решению (RMSE_u) рассчитаны по тестовому набору точек.

3.5. Оптимизация гиперпараметров

Работа состоит из трёх частей.

1. Сначала проводим качественное исследование влияния гиперпараметров на сходимость сети в случаях 2-мерной и 5-мерной задач. Задаём сетку конфигураций и проводим случайный поиск по 100 точкам с одинаковым временем обучения на каждую точку. Стартуя с полученной оптимальной конфигурации, разбиваем гиперпараметры на группы и варьируем их по сетке. Проанализировав результаты, выбираем оптимальные значения и переходим к другим гиперпараметрам.
2. Сравним методы автоматического подбора гиперпараметров. Рассмотрим случайный поиск, байесовский поиск из пакета HyperOpt и планировщик ASHA в разных комбинациях друг с другом, а также комбинированный алгоритм BOHB. На каждый эксперимент выделим одинаковое ограничение по времени на одну конфигурацию и одинаковое полное время.
3. Определив лучшую комбинацию алгоритмов HPO, запустим поиск по гиперпараметрам с этими алгоритмами для 2-, 5- и 8-мерной задачи. После этого для лучших конфигураций запустим обучение и попытаемся получить предельное значение целевой функции.

Далее перечислены все гиперпараметры, которые мы будем оптимизировать, и соответствующие обозначения, используемые в таблицах.

Гиперпараметры, связанные с процессом обучения:

- $|N_f|$ — количество точек в обучающей выборке;
- RNG — способ генерации точек в обучающей выборке;
- $|N_{\text{batch}}|$ — размер батча;
- lr — начальная скорость обучения;
- lr scheduler — планировщик скорости обучения. Рассмотрим 4 варианта планировщика:
  - None — без планировщика, т. е. постоянное значение lr;

- `ExponentialLR-0.95 (ELR-0.95)` — экспоненциальный планировщик. lr изменяется в 0.95 раз после каждой итерации;
- `ReduceLROnPlateau-0.1-10 (RLRoP-0.1-10)` — lr умножается на 0.1, если целевая функция не улучшается на протяжении 11 итераций;
- `ReduceLROnPlateau-0.5-2 (RLRoP-0.5-2)` — lr умножается на 0.5, если целевая функция не улучшается на протяжении 3 итераций.

Гиперпараметры, связанные с архитектурой сети:

- N — ширина скрытого слоя (количество нейронов в скрытом слое);
- L — количество скрытых слоёв;
- σ — функция активации. Рассмотрим 5 функций активации:
  - `ELU` — *exponential linear unit*, реализована в PyTorch и выражается следующим образом:

$$\mathrm{ELU}(x) = \begin{cases} x, & x > 0 \\ e^x - 1, & x \le 0 \end{cases}$$

  - `sigmoid` — сигмоида, реализована в PyTorch и выражается следующим образом:

$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

  - `tanh` — гиперболический тангенс, реализован в PyTorch и выражается следующим образом:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

  - `sin` — синус:

$$\sigma(x) = \sin(x)$$

  - `atan` — нормированный арктангенс, выраженный следующим образом:

$$\sigma(x) = \frac{\tan^{-1}(x)}{\pi/2}$$

Другие обозначения:

- i — номер последней итерации обучения (эпохи). Может быть дробным числом, если итерация не успела завершиться за отведённое время;
- t — ограничение по времени на одну конфигурацию. Обучение сети прекращается посреди итерации, если время выйдет до того, как через модель пройдут все обучающие точки;

- T — полное время, выделенное алгоритму HPO;
- $|N_{\text{cfg}}|$ — количество рассмотренных конфигураций.

Область поиска в Ray Tune может задаваться разными методами. В настоящей работе используются четыре типа распределения величины:

- категориальный тип — равновероятный выбор из перечисленных значений. Далее в результатах для величин такого типа через запятую перечисляются возможные значения;
- `randint(a, b)` — выбор целого числа из равномерного распределения между `a` (включительно) и `b` (не включительно);
- `qlograndint(a, b, q)` — выбор целого числа из логарифмического распределения между `a` (включительно) и `b` (включительно) с округлением до `q`;
- `qloguniform(a, b, q)` — выбор вещественного числа из логарифмического распределения между `a` (включительно) и `b` (включительно) с округлением до `q`.

Разумеется, только случайный поиск выбирает конфигурации в точном соответствии с заданными распределениями. Байесовские методы модифицируют заданные распределения суррогатными моделями и выбирают точки посредством функции сбора.

Варьируя гиперпараметры, сеть может быть оптимизирована по двум критериям:

- по скорости сходимости;
- по предельной точности.

При этом один набор параметров может обеспечить более быструю сходимость решения (более быстрое убывание функции ошибки), а другой набор параметров позволяет получить более точное решение (меньшее абсолютное значение функции ошибки), но медленную сходимость. Соответственно, набор параметров следует выбирать исходя из поставленной цели. В данной работе делается акцент на скорости сходимости.

3.6. Вычислительные ресурсы

Обучение нейронных сетей проводилось на узлах вычислительного центра научного парка СПбГУ. Обучение при ручной оптимизации гиперпараметров (пп. **4.1** и **4.2**) производилось на видеокарте NVIDIA Tesla P100, 16 GB. Обучение при автоматической оптимизации гиперпараметров и длительное обучение на оптимальной конфигурации (пп. **4.3** и **4.4**) производилось на видеокарте NVIDIA RTX A6000, 48 GB. RTX A6000 превосходит Tesla P100 по скорости обучения моделей примерно в 1.8 раз.

4. Результаты

Все указанные далее значения невязки по уравнению (MSE_f) и невязки по решению (RMSE_u) рассчитаны по тестовому набору точек.

4.1. Ручная оптимизация гиперпараметров на примере 2-мерной задачи

Случайный поиск по всем параметрам

Таблица 1. *Сетка поиска и ограничение по времени*

N	64, 128, 256, 512, 1024, 2048
L	1, 2, 3, 4
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^2, 10^3, 10^4
RNG	случайно, квазислучайно
$\lvert N_{\text{batch}} \rvert$	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768
lr	10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}
lr scheduler	`None, ELR-0.95, RLRoP-0.1-10, RLRoP-0.5-2`
t	30 сек
$\lvert N_{\text{cfg}} \rvert$	100

Таблица 2. *Лучшие конфигурации модели*

№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	RNG	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
52	51.56	30.02	4.36E−03	4.51E−04	128	2	`sin`	10^4	квазисл.	128	10^{-3}	`None`
37	144.92	30.02	7.40E−03	8.52E−05	256	4	`sin`	10^4	квазисл.	512	10^{-2}	`ExponentialLR-0.95`
92	62.28	30.01	8.57E−03	3.87E−04	64	1	`tanh`	10^4	квазисл.	128	10^{-1}	`ReduceLROnPlateau-0.5-2`

Скорость обучения и размер батча

Таблица 3. *Значения фиксированных параметров*

t, сек	N	L	σ	$\lvert N_f \rvert$	RNG
30	128	2	`sin`	10^4	квазислучайно

Таблица 4. *Невязки по уравнению и по решению на последней итерации при разных значениях скорости обучения и размера батча*

MSE_f								
lr	*lr* scheduler	$\lvert N_{\text{batch}} \rvert$						
		64	128	256	512	1024	2048	4096
10^{-4}	None	1.06E+00	5.85E−01	7.23E−01	5.21E−01	5.52E−01	7.01E−01	1.15E+00
10^{-3}	None	5.47E−03	1.25E−02	1.66E−02	3.40E−03	2.97E−03	1.07E−02	3.36E−03
10^{-2}	None	1.09E−01	9.00E−01	3.21E−01	7.01E−02	5.41E−02	3.74E−01	1.76E+00
10^{-1}	None	7.65E+08	1.97E+02	2.19E+03	2.44E+03	8.97E+02	3.81E−01	3.02E+03
10^{-2}	ExponentialLR-0.95	1.14E−02	8.03E−04	2.00E−03	3.20E−03	1.97E−02	7.09E−02	3.04E−01
10^{-2}	ReduceLROnPlateau-0.1-10	2.04E−01	7.41E−04	1.16E−03	**6.28E−04**	9.79E−04	1.98E−03	1.01E−03
10^{-2}	ReduceLROnPlateau-0.5-2	2.72E−03	9.68E−04	2.07E−03	7.02E−04	7.37E−04	1.56E−03	1.91E−03
RMSE_u								
lr	*lr* scheduler	$\lvert N_{\text{batch}} \rvert$						
		64	128	256	512	1024	2048	4096
10^{-4}	None	4.53E−03	3.53E−03	4.15E−03	3.72E−03	4.10E−03	5.11E−03	7.71E−03
10^{-3}	None	8.93E−04	1.61E−03	2.15E−03	6.05E−04	4.93E−04	1.53E−03	2.80E−04
10^{-2}	None	4.65E−03	1.58E−02	1.09E−02	7.33E−03	9.80E−03	1.09E−02	2.47E−02
10^{-1}	None	5.06E−01	8.84E−02	4.98E−01	4.98E−01	3.41E−01	1.90E−03	4.99E−01
10^{-2}	ExponentialLR-0.95	1.72E−03	2.13E−04	1.01E−04	1.10E−04	3.39E−04	7.46E−04	3.18E−03
10^{-2}	ReduceLROnPlateau-0.1-10	6.09E−03	1.03E−04	1.38E−04	7.26E−05	5.96E−05	8.51E−05	5.95E−05
10^{-2}	ReduceLROnPlateau-0.5-2	4.39E−04	8.96E−05	1.37E−04	**5.29E−05**	5.52E−05	7.42E−05	8.46E−05

Получаем, что в данном случае оптимальное значение скорости обучения без использования планировщика — 10^{-3}. Применение планировщиков скорости обучения во всех случаях улучшает результат по сравнению с фиксированной скоростью обучения.

Размер батча в меньшей степени влияет на результат. Тем не менее, при $\lvert N_{\text{batch}} \rvert = 512$ были получены наилучшие значения MSE_f и RMSE_u.

Невязка по уравнению и невязка по решению не связаны взаимно-однозначным соответствием, поэтому наименьшее полученное значение MSE_f не соответствует наименьшему RMSE_u. Тем не менее, сходимость одной величины предсказывает сходимость другой величины, что свидетельствует о правильном выборе целевой функции.

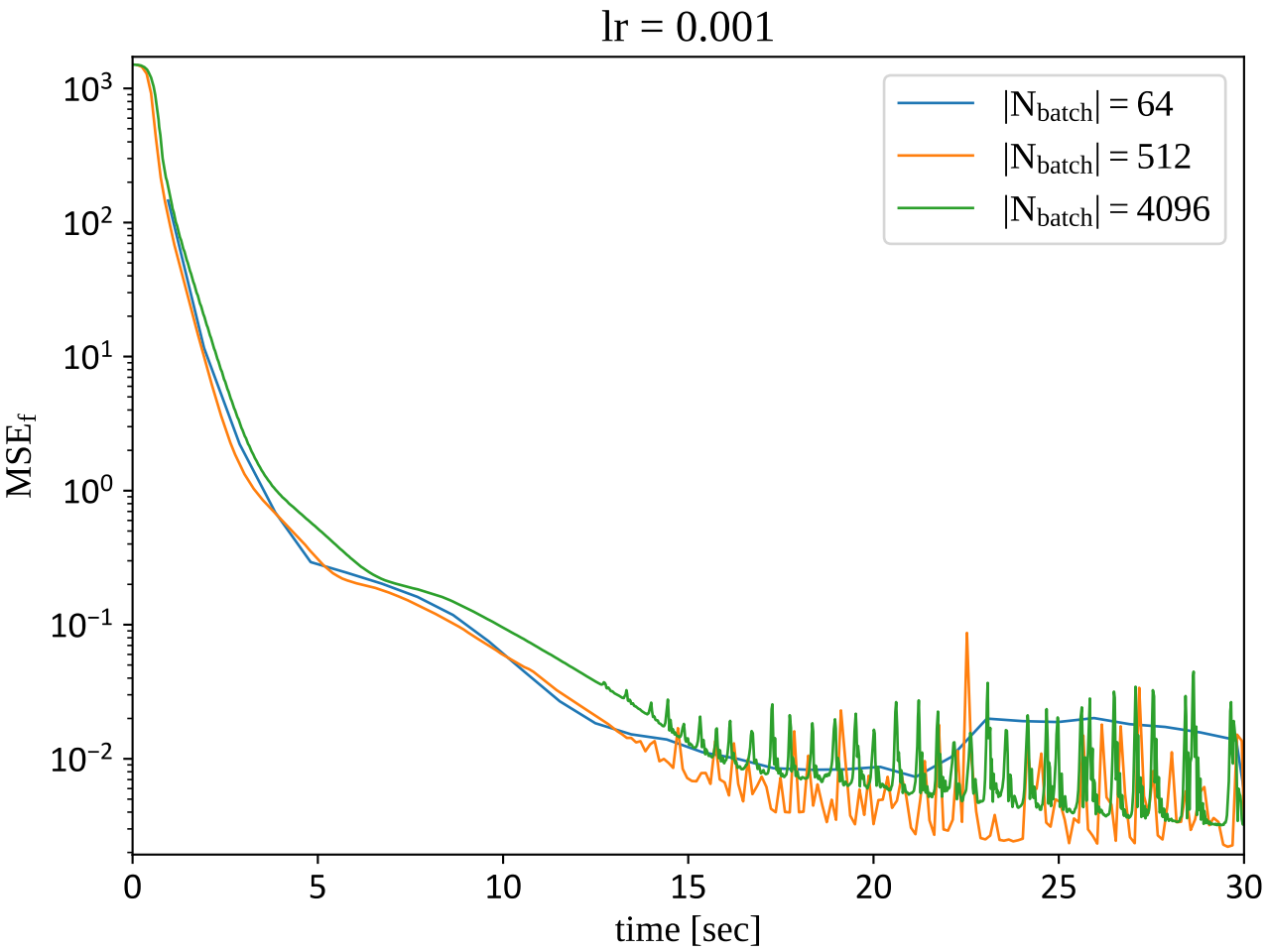

График 2.

Зависимость невязки по уравнению от времени при разном размере батча

Операция прогона точек через сеть (расчёт целевой функции и обратное распространение) векторизуется. Поэтому при увеличении размера батча количество итераций сети, т. е. прогонов через неё всей обучающей выборки, растёт пропорционально до тех пор, пока позволяют вычислительные ресурсы. Несмотря на это, увеличение размера батча не всегда ускоряет обучение относительно времени, как видно из графика. При $|N_{batch}| = 512$ было выполнено 238 итераций обучения, а при $|N_{batch}| = 4096$ за то же время выполнено 1337 итераций. Размер батча также способен влиять на абсолютную точность модели.

При относительно больших размерах батча график со временем приобретает периодическую структуру.

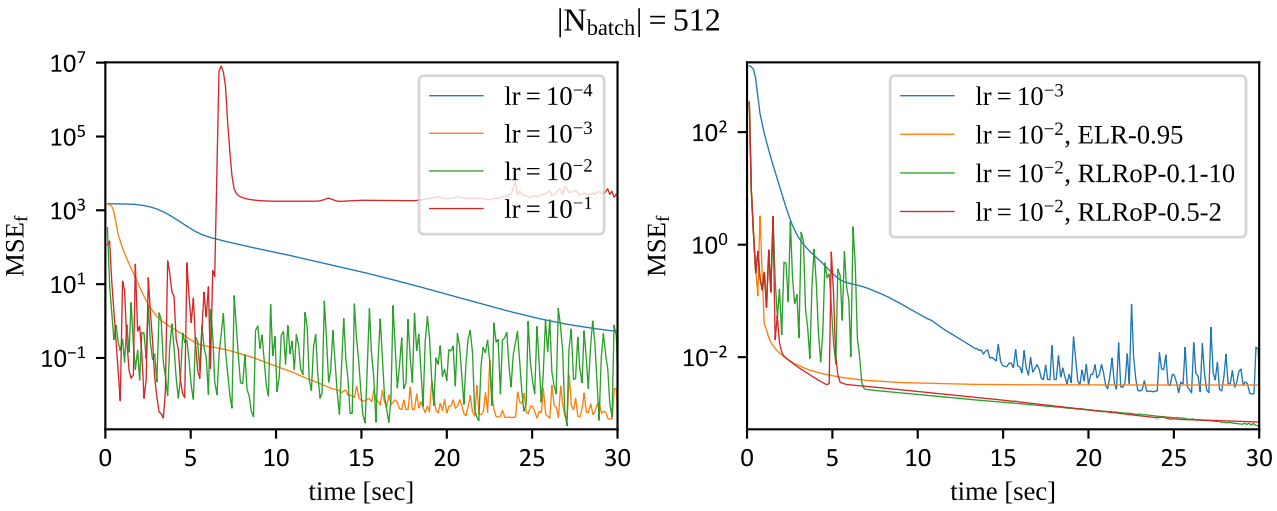

График 3. *Зависимость невязки по уравнению от времени при разной скорости обучения*

При малых значениях lr сходимость медленная, но плавная. При больших значениях lr целевая функция быстро уменьшается, но начинает осциллировать. В некоторых случаях, как здесь при $lr = 10^{-1}$, осцилляции приводят к очень плохому состоянию сети, после чего обучение застревает.

Применение планировщика скорости обучения позволяет объединить преимущества больших и малых lr. Начиная с большого lr, обучение происходит быстро, а затем lr постепенно понижается по заданному правилу. Понижение lr при выходе целевой функции на плато превзошло по точности экспоненциальное понижение lr. При этом на графиках заметны моменты понижения lr, которые проявляются прекращением осцилляций и резким понижением целевой функции. Подбор планировщика и его параметров тоже имеет важное значение, поскольку неправильно заданный планировщик может достигнуть минимального lr слишком рано, что демонстрируется на графике на примере экспоненциального планировщика. Для решения этой проблемы можно использовать циклический планировщик, однако в рамках работы он не рассматривается.

В дальнейшем зафиксируем $|N_{\text{batch}}| = 512$, $lr = 10^{-2}$ и lr scheduler = `RLRoP-0.1-10`.

Количество точек и способ генерации точек

Таблица 5. *Значения фиксированных параметров*

t, сек	N	L	σ	$\lvert N_{\text{batch}}\rvert$	lr	lr scheduler
30	128	2	`sin`	512	10^{-2}	`ReduceLROnPlateau-0.1-10`

Таблица 6. *Невязки по уравнению и решению на последней итерации при разном количестве и способах генерации точек коллокации*

$\lvert N_f\rvert$	MSE_f		RMSE_u	
	RNG		RNG	
	случайно	квазислучайно	случайно	квазислучайно
10^2	1.68E−01	9.96E−03	1.32E−03	2.46E−04
10^3	1.51E−03	2.20E−03	7.97E−05	9.33E−05
10^4	1.40E−03	**6.28E−04**	9.07E−05	**7.26E-05**

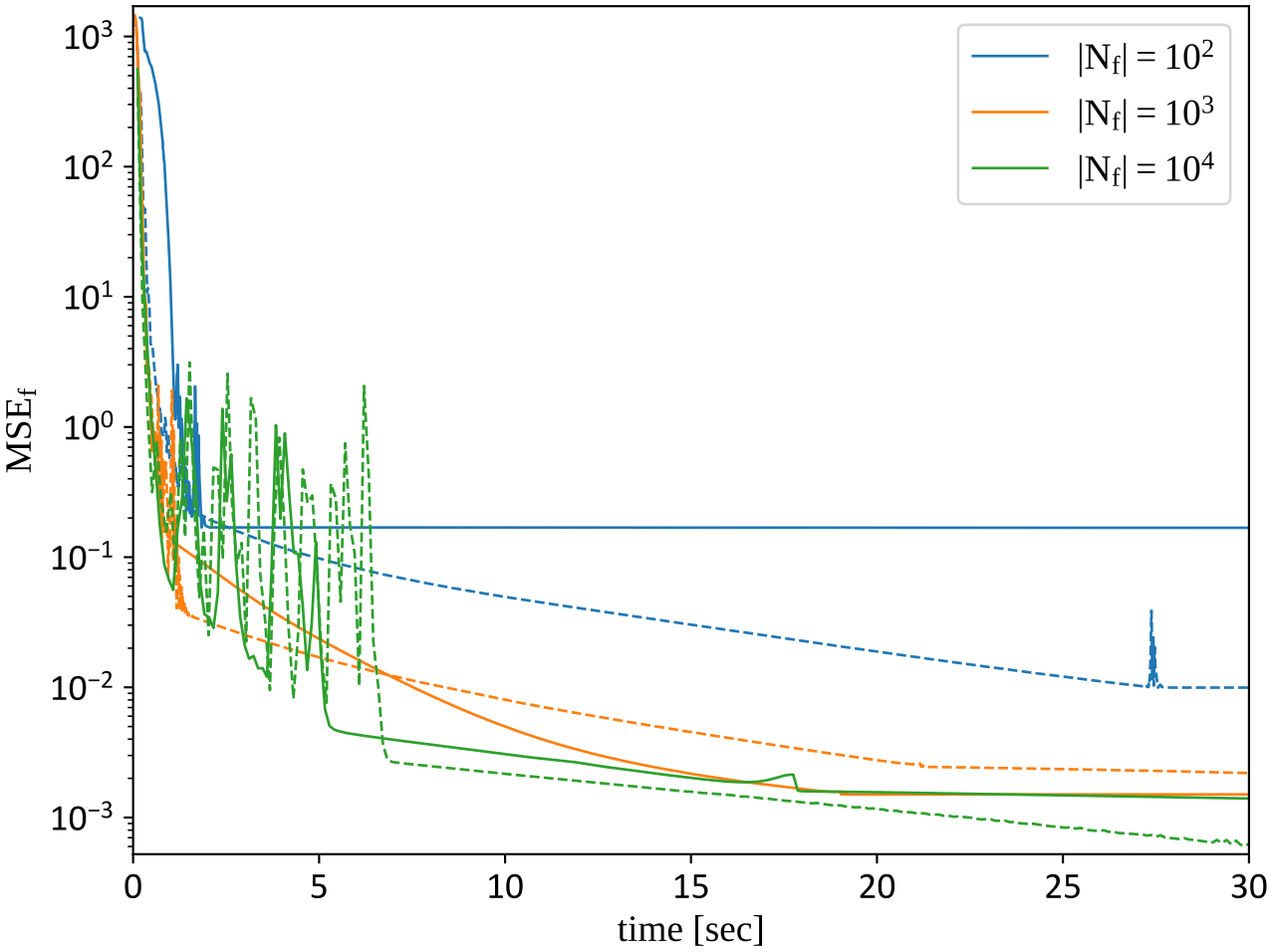

График 4.

Зависимость невязки по уравнению от времени при разном количестве точек коллокации. Сплошными линиями изображены зависимости для случайного метода генерации точек, пунктирными — для квазислучайного

Увеличение количества точек улучшает точность модели. При достаточно большом количестве точек случайный метод генерации не уступает квазислучайному. Преимущество квазислучайного метода генерации проявляется при малом количестве точек (10^2). Поэтому в случае большой размерности задачи, когда для равномерного заполнения области определения требуется недостижимое количество точек, квазислучайный метод может оказаться более эффективным.

В дальнейшем зафиксируем $|N_f| = 10^4$ и квазислучайный метод генерации точек.

Ширина скрытого слоя, количество скрытых слоёв и функция активации

Таблица 7. *Значения фиксированных параметров*

t, сек	$\lvert N_f \rvert$	RNG	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
30	10^4	квазислучайно	512	10^{-2}	`ReduceLROnPlateau-0.1-10`

Таблица 8. *Невязки по уравнению и решению на последней итерации при разных ширине слоя, количестве скрытых слоёв и функции активации*

σ	N	MSE_f				RMSE_u			
		L				L			
		1	2	3	4	1	2	3	4
ELU	64	4.55E+01	7.48E+02	1.52E+03	9.61E+02	2.31E−02	4.64E−01	5.42E−01	4.66E−01
	128	4.28E+01	1.46E+03	9.42E+02	7.81E+01	2.56E−02	4.91E−01	3.75E−01	4.11E−02
	256	1.32E+01	2.07E+02	1.07E+01	5.90E+02	1.01E−02	1.34E−01	5.47E−03	3.28E−01
	512	8.04E+00	1.22E+03	2.92E+02	1.78E+02	6.71E−03	4.45E−01	2.03E−01	9.87E−02
	1024	1.43E+00	8.29E+01	7.70E+02	7.25E+02	1.67E−03	4.34E−02	3.36E−01	3.29E−01
	2048	5.61E−01	1.85E+02	1.53E+02	3.04E+02	1.25E−03	8.20E−02	6.71E−02	1.72E−01
sigmoid	64	4.45E−01	4.95E−01	7.67E−02	1.15E−01	7.26E−04	3.35E−03	5.44E−04	1.12E−03
	128	4.30E−02	1.40E−01	7.90E−02	2.01E−02	9.45E−04	3.41E−04	2.32E−04	2.42E−04
	256	6.35E−03	1.57E−01	7.31E−03	5.44E−02	1.42E−03	3.18E−04	1.30E−04	6.23E−04
	512	1.83E−02	1.05E−01	4.07E−01	2.18E−02	2.67E−03	4.05E−04	9.72E−04	3.97E−04
	1024	1.47E−02	2.76E−01	8.41E−01	7.59E+00	1.55E−04	4.20E−04	3.18E−03	2.34E−02
	2048	2.58E−02	1.31E+03	1.51E+03	1.51E+03	2.74E−04	4.64E−01	4.98E−01	4.98E−01
tanh	64	9.79E−02	2.18E−01	1.77E−02	2.46E−02	1.13E−03	3.75E−04	3.54E−04	2.32E−04
	128	1.33E−02	2.36E−01	6.73E−03	6.46E−03	9.93E−04	8.18E−04	1.06E−04	2.23E−04
	256	1.45E−03	5.54E−02	1.27E−02	4.61E+02	4.19E−05	1.18E−03	1.56E−03	2.57E−01
	512	2.69E−03	3.39E−03	4.01E−02	8.78E+02	4.51E−04	**3.99E−05**	1.34E−04	3.71E−01
	1024	8.32E−03	3.15E−03	1.02E+03	2.48E+02	1.46E−04	1.66E−04	4.03E−01	1.79E−01
	2048	9.26E−03	1.51E+03	1.51E+03	1.48E+03	1.76E−04	4.98E−01	4.98E−01	4.94E−01
sin	64	2.78E−03	2.47E−03	6.70E−03	1.14E−02	7.24E−04	1.28E−04	2.43E−04	1.38E−04
	128	3.71E−03	**6.28E−04**	3.41E−03	2.65E−01	9.64E−04	7.26E−05	2.01E−04	9.15E−04
	256	4.80E−03	4.26E−03	1.18E−01	1.52E−03	1.92E−04	9.95E−04	2.97E−04	9.84E−05
	512	3.95E−03	1.10E−03	3.69E+02	1.13E+03	1.19E−04	9.47E−05	2.18E−01	4.22E−01
	1024	1.54E−03	2.07E−03	9.85E+08	2.50E+14	8.48E−05	4.96E−05	5.03E−01	5.18E−01
	2048	2.75E−03	3.87E−03	6.64E+07	4.42E+14	1.14E−04	8.72E−05	5.00E−01	5.05E−01
atan	64	6.35E−01	1.56E−01	2.59E−02	9.45E−02	2.02E−03	9.97E−03	3.12E−04	3.71E−04
	128	5.62E−02	6.10E−01	1.22E−02	2.61E−02	2.17E−03	8.16E−04	2.89E−04	2.70E−04
	256	1.23E−02	1.10E−01	5.22E−03	8.32E−03	1.04E−03	1.59E−03	3.41E−04	1.32E−04
	512	9.67E−03	3.94E−02	1.26E+01	4.55E−01	1.08E−04	1.45E−04	2.80E−02	4.05E−04
	1024	1.06E−01	9.26E−03	6.46E+00	5.29E+00	1.57E−02	2.16E−04	3.31E−03	5.31E−03
	2048	2.20E−02	2.58E−02	8.47E+02	1.06E+03	2.46E−04	7.06E−04	3.24E−01	4.13E-01

Наилучшие результаты достигнуты с использованием `sin` в качестве функции активации. Хорошей функцией в данной задаче также является `tanh`. Самые плохие результаты получены с использованием `ELU`.

С решением поставленной задачи лучше справляются неглубокие сети с 1–2 скрытыми слоями.

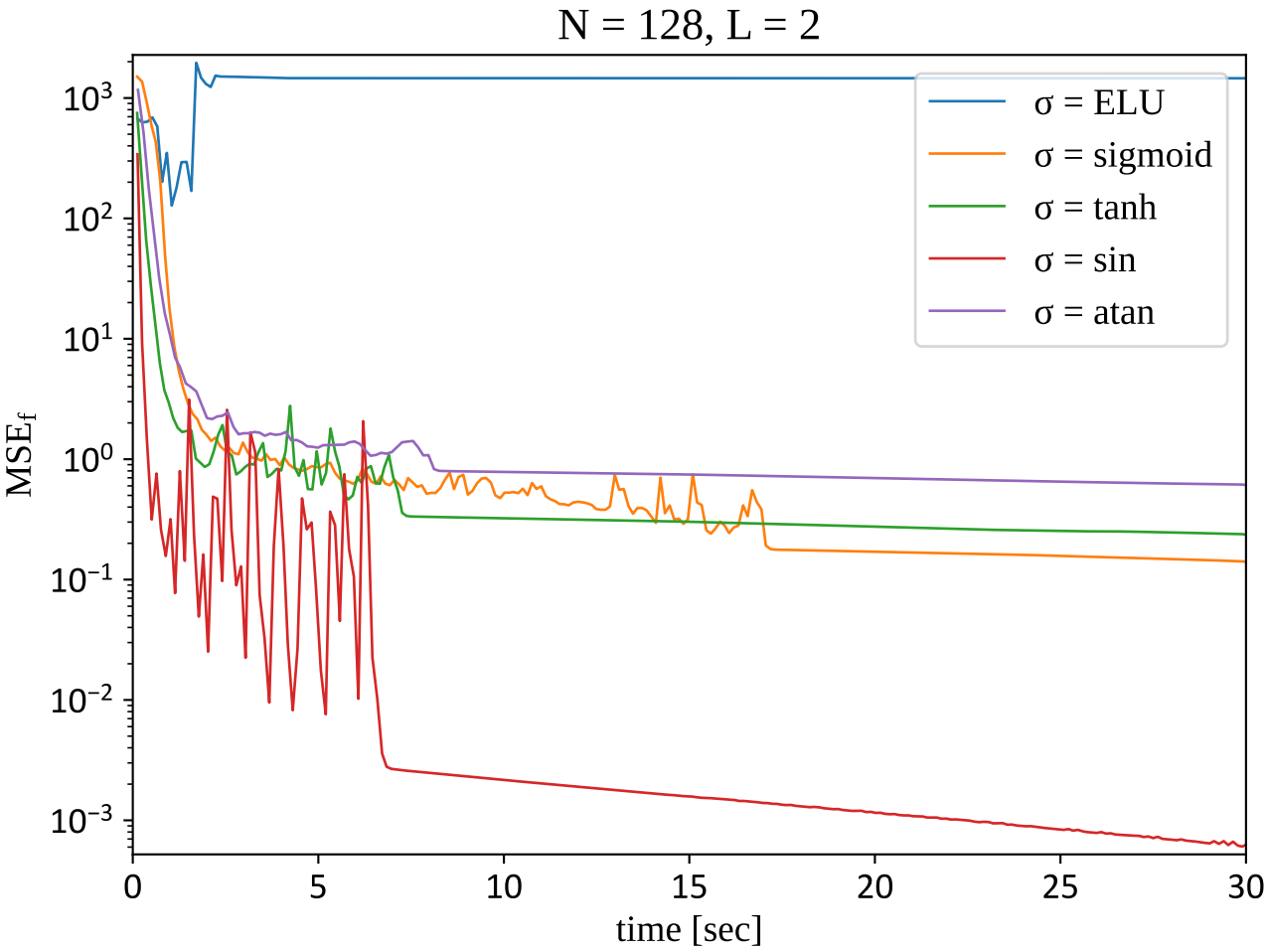

График 5. *Зависимость невязки по уравнению от времени при разных функциях активации*

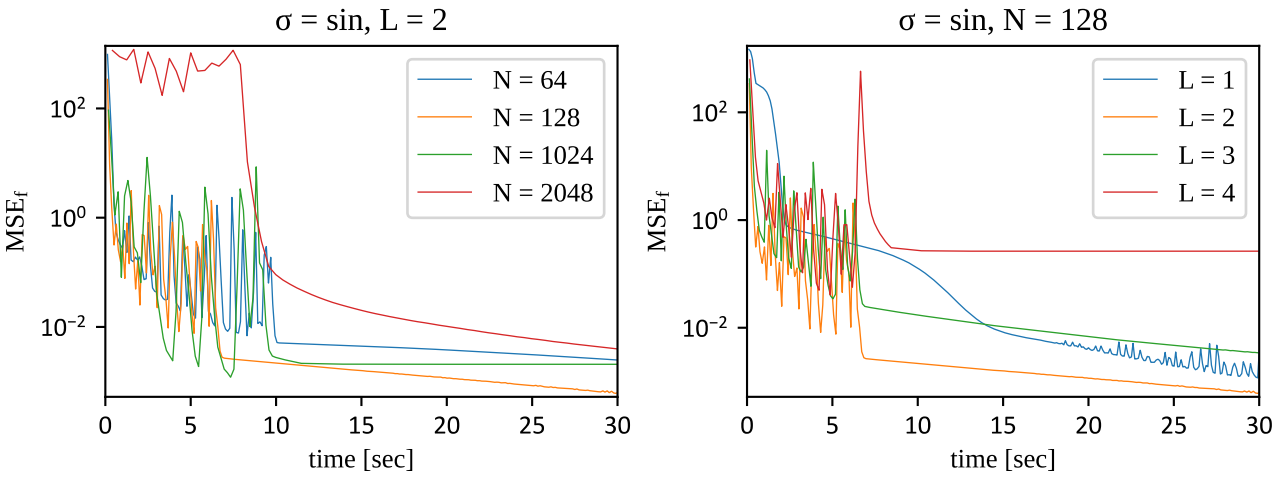

График 6. *Зависимость невязки по уравнению от времени при разных ширине слоя и количестве скрытых слоёв*

При $L = 1$ сходимость очень плавная по сравнению с другими L, и lr не изменялся в ходе обучения. Самый лучший результат достигнут при $L = 2$, и при увеличении L результат только ухудшается.

4.2. Ручная оптимизация гиперпараметров на примере 5-мерной задачи

Случайный поиск по всем параметрам

Таблица 9. *Сетка поиска и ограничение по времени*

N	64, 128, 256, 512, 1024, 2048
L	1, 2, 3, 4
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^4, 10^5, 10^6
RNG	случайно, квазислучайно
$\lvert N_{\text{batch}} \rvert$	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768
lr	10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}
lr scheduler	`None, ELR-0.95, RLRoP-0.1-10, RLRoP-0.5-2`
t	300 сек
$\lvert N_{\text{cfg}} \rvert$	100

Таблица 10. *Лучшие конфигурации модели*

№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	RNG	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
84	125.12	300.07	3.83E−01	1.25E−03	512	1	`sin`	10^5	квазисл.	256	10^{-1}	`ExponentialLR-0.95`
76	22.26	302.97	1.73E+01	1.19E−02	2048	1	`atan`	10^6	случ.	4096	10^{-1}	`None`
59	25.52	300.63	2.64E+01	1.96E−02	512	1	`sigmoid`	10^6	случ.	512	10^{-1}	`ReduceLROnPlateau-0.5-2`

Скорость обучения и размер батча

Таблица 11. *Значения фиксированных параметров*

t, сек	N	L	σ	$\lvert N_f \rvert$	RNG
30	512	1	sin	10^5	квазислучайно

Таблица 12. *Невязки по уравнению и по решению на последней итерации при разных значениях скорости обучения и размера батча*

MSE_f								
lr	lr scheduler	$\lvert N_{\text{batch}} \rvert$						
		64	128	256	512	1024	2048	4096
10^{-4}	None	1.22E+03	1.22E+03	1.22E+03	1.22E+03	1.22E+03	1.22E+03	1.22E+03
10^{-3}	None	1.22E+03	1.22E+03	1.22E+03	1.19E+03	1.16E+03	1.14E+03	1.12E+03
10^{-2}	None	1.22E+03	1.22E+03	1.22E+03	3.96E−01	2.25E−01	1.73E−01	3.86E−01
10^{-1}	None	4.71E+01	5.85E+00	3.75E+01	4.94E−01	1.52E+00	**1.53E−01**	3.96E−01
10^{0}	None	1.15E+07	6.74E+06	1.69E+06	1.77E+05	2.03E+04	8.19E+02	2.39E+01
10^{-1}	ExponentialLR-0.95	5.55E−01	4.51E−01	3.81E−01	3.89E−01	4.16E−01	5.71E+02	1.47E+02
10^{-1}	ReduceLROnPlateau-0.1-10	1.13E+01	3.62E−01	3.98E−01	4.13E−01	4.21E−01	4.21E−01	4.21E−01
10^{-1}	ReduceLROnPlateau-0.5-2	3.80E−01	4.25E−01	3.82E−01	4.20E−01	4.23E−01	4.26E−01	4.25E−01
RMSE_u								
lr	lr scheduler	$\lvert N_{\text{batch}} \rvert$						
		64	128	256	512	1024	2048	4096
10^{-4}	None	1.78E−01	1.78E−01	1.78E−01	1.78E−01	1.78E−01	1.78E−01	1.78E−01
10^{-3}	None	1.78E−01	1.78E−01	1.78E−01	1.76E−01	1.73E−01	1.71E−01	1.70E−01
10^{-2}	None	1.78E−01	1.78E−01	1.78E−01	2.11E−03	1.36E−03	1.33E−03	1.37E−03
10^{-1}	None	3.73E−02	1.16E−02	4.39E−02	1.42E−03	1.03E−02	**9.41E−04**	4.04E−03
10^{0}	None	1.80E−01	1.81E−01	1.81E−01	1.78E−01	1.78E−01	8.62E−02	2.12E−02
10^{-1}	ExponentialLR-0.95	2.63E−03	1.85E−03	1.21E−03	1.21E−03	1.26E−03	1.18E−01	5.27E−02
10^{-1}	ReduceLROnPlateau-0.1-10	2.37E−02	2.56E−03	1.25E−03	1.27E−03	1.27E−03	1.28E−03	1.27E−03
10^{-1}	ReduceLROnPlateau-0.5-2	1.49E−03	3.66E−03	1.71E−03	1.27E−03	1.28E−03	1.30E−03	1.25E−03

Получаем, что в данном случае лучшее значение целевой функции получено при $lr = 10^{-1}$ без планировщика и $|N_{\text{batch}}| = 2048$, что отличается от 2-мерной задачи. Более того, применение планировщиков скорости обучения ухудшило этот результат и результаты при некоторых других размерах батча.

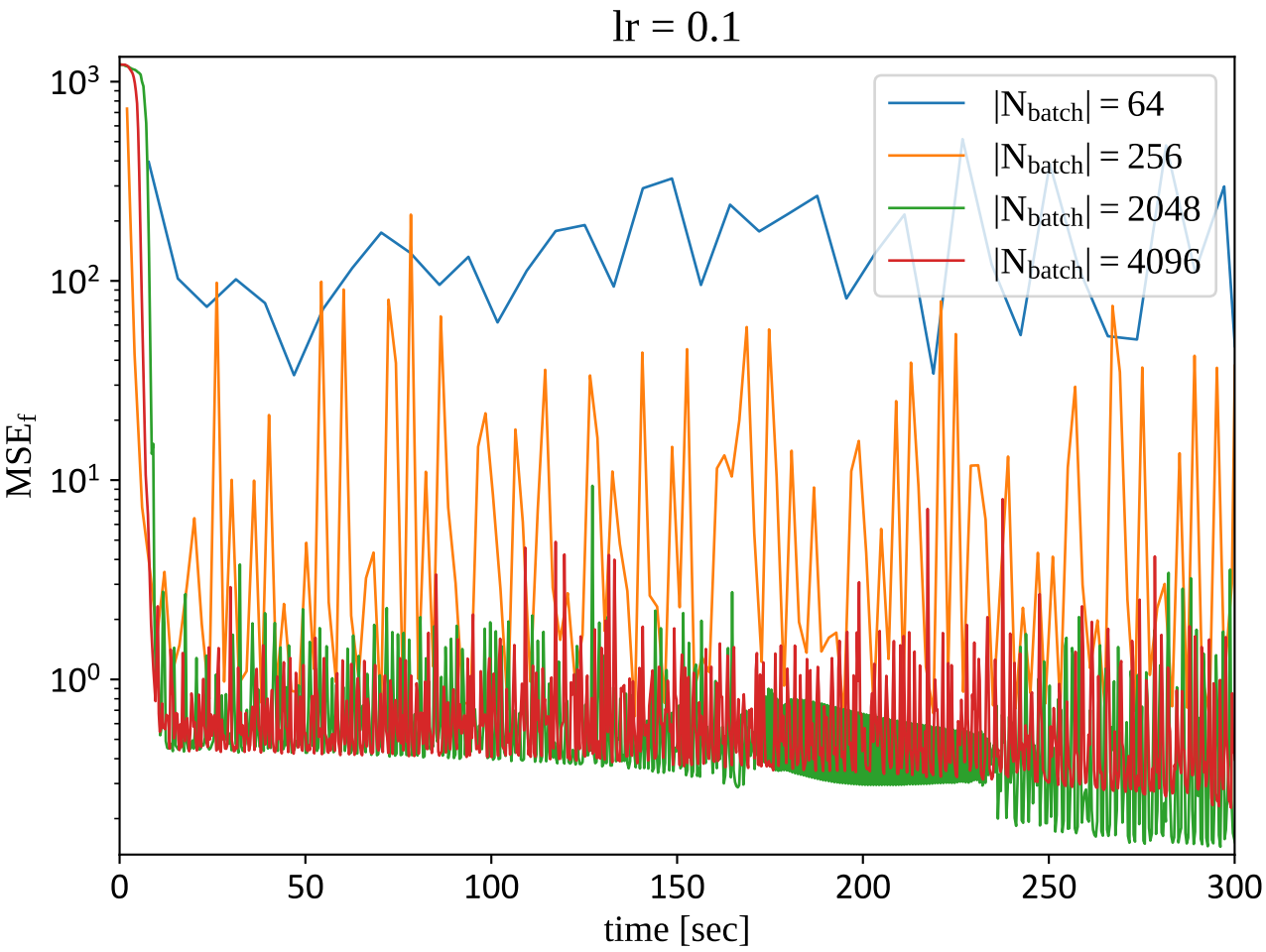

График 7.

Зависимость невязки по уравнению от времени при разном размере батча

Значение целевой функции сильно колеблется во всех случаях, что свидетельствует о слишком высоком значении lr. Хотя при таком значении получен наилучший результат, такой lr не следует использовать из-за плохого качества сходимости.

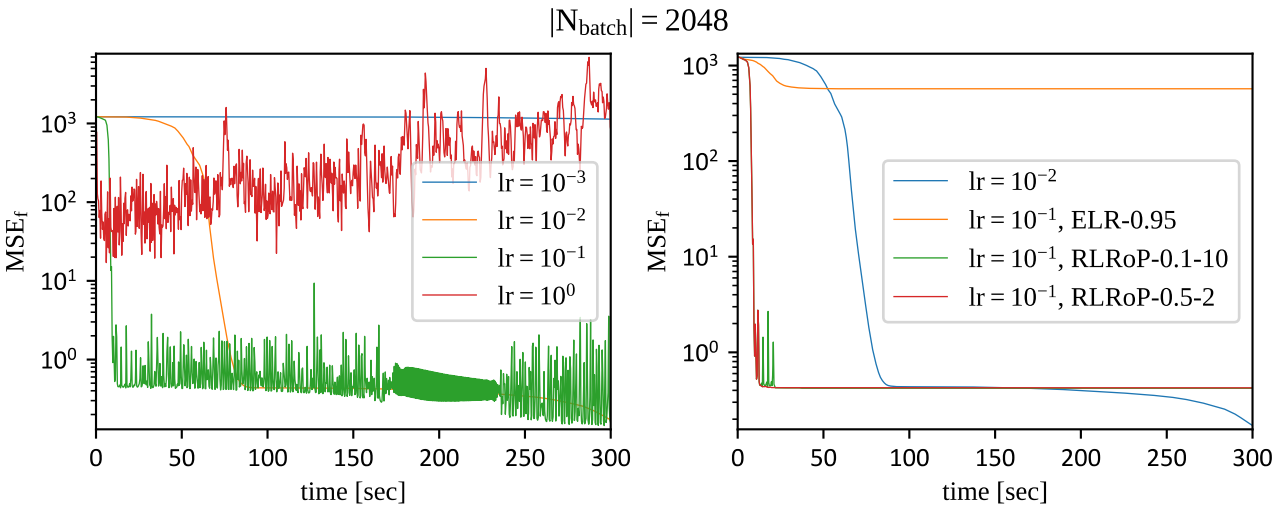

График 8. *Зависимость невязки по уравнению от времени при разной скорости обучения*

При $lr = 10^{-3}$ целевая функция вообще не изменилась за выделенное время. При $lr = 10^{-2}$ целевая функция менялась волнообразно: сначала плато, потом быстрый спуск, затем снова плато, затем снова быстрый спуск. Из-за такой непредсказуемой сходимости сети очень сложно вручную подобрать хороший планировщик обучения. При $lr = 10^{-1}$ целевая функция быстро спадает и начинает осциллировать. В какой-то момент амплитуда осцилляций уменьшается, а потом снова увеличивается. При $lr = 10^{0}$ целевая функция быстро снижается в начале, а потом постепенно движется в противоположную сторону.

Из-за того, что обучение на некоторое время застревает, рассмотренные планировщики только ухудшили абсолютную точность сети.

В дальнейшем зафиксируем $|N_{\text{batch}}| = 2048$, $lr = 10^{-2}$ и lr scheduler = None.

Количество точек и способ генерации точек

Таблица 13. *Значения фиксированных параметров*

t, сек	N	L	σ	$\lvert N_{\text{batch}}\rvert$	lr	lr scheduler
300	512	1	sin	2048	10^{-2}	None

Таблица 14. *Невязки по уравнению и решению на последней итерации при разном количестве и способах генерации точек коллокации*

$\lvert N_f\rvert$	MSE_f		RMSE_u	
	RNG		RNG	
	случайно	квазислучайно	случайно	квазислучайно
10^4	9.65E−02	**8.92E−02**	1.45E−03	1.52E−03
10^5	2.94E−01	1.73E−01	1.76E−03	**1.33E−03**
10^6	1.21E+03	1.21E+03	1.77E−01	1.77E-01

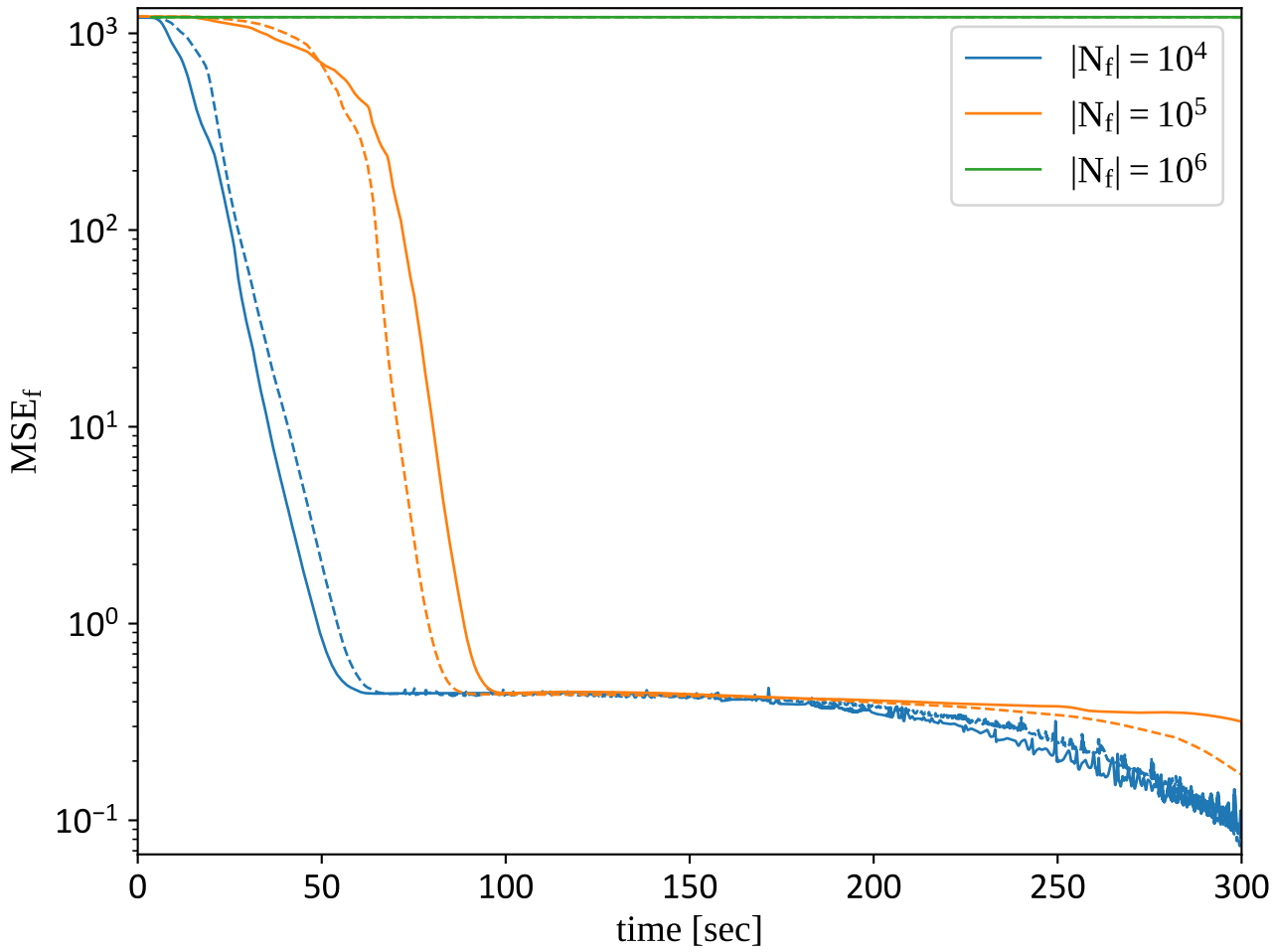

График 9.

Зависимость невязки по уравнению от времени при разном количестве точек коллокации. Сплошными линиями изображены зависимости для случайного метода генерации точек, пунктирными — для квазислучайного

В данном случае увеличение количества точек уже ухудшает сходимость модели. Объяснить данный феномен затруднительно.

Результат при квазислучайном методе генерации точек незначительно улучшился.

В дальнейшем зафиксируем $|N_f| = 10^4$ и квазислучайный метод генерации точек.

Ширина скрытого слоя, количество скрытых слоёв и функция активации

Таблица 15. *Значения фиксированных параметров*

t, сек	$\lvert N_f \rvert$	RNG	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
300	10^4	квазислучайно	2048	10^{-2}	None

Таблица 16. *Невязки по уравнению и решению на последней итерации при разных ширине слоя, количестве скрытых слоёв и функции активации*

σ	N	MSE_f				RMSE_u			
		L				L			
		1	2	3	4	1	2	3	4
ELU	64	1.11E+03	1.20E+03	1.20E+03	1.20E+03	1.67E−01	1.77E−01	1.77E−01	1.77E−01
	128	1.04E+03	1.20E+03	1.20E+03	1.20E+03	1.60E−01	1.77E−01	1.77E−01	1.77E−01
	256	8.60E+02	1.20E+03	1.07E+03	1.20E+03	1.34E−01	1.77E−01	1.70E−01	1.77E−01
	512	6.13E+02	1.19E+03	1.20E+03	1.20E+03	1.01E−01	1.76E−01	1.77E−01	1.77E−01
	1024	6.04E+02	1.12E+03	1.20E+03	1.20E+03	1.04E−01	1.72E−01	1.79E−01	1.77E−01
	2048	2.86E+02	1.20E+03	1.23E+03	1.20E+03	5.72E−02	1.77E−01	1.82E−01	1.77E−01
sigmoid	64	1.11E+03	1.90E+03	1.75E+03	1.20E+03	1.66E−01	1.80E−01	1.80E−01	1.77E−01
	128	1.11E+03	1.90E+03	2.30E+03	1.20E+03	1.61E−01	1.80E−01	1.72E−01	1.77E−01
	256	1.03E+03	1.99E+03	2.06E+03	1.20E+03	1.40E−01	1.75E−01	1.60E−01	1.77E−01
	512	7.34E+02	1.20E+03	1.20E+03	1.20E+03	1.09E−01	1.77E−01	1.77E−01	1.77E−01
	1024	2.99E+02	1.20E+03	1.20E+03	1.20E+03	6.57E−02	1.77E−01	1.77E−01	1.77E−01
	2048	7.86E+01	1.20E+03	1.20E+03	1.20E+03	2.98E−02	1.77E−01	1.77E−01	1.77E−01
tanh	64	9.41E+02	1.61E+03	1.20E+03	1.20E+03	1.52E−01	1.77E−01	1.77E−01	1.77E−01
	128	6.64E+02	2.12E+03	1.20E+03	1.20E+03	1.15E−01	1.76E−01	1.77E−01	1.77E−01
	256	5.68E+02	1.20E+03	1.20E+03	1.20E+03	8.95E−02	1.77E−01	1.77E−01	1.77E−01
	512	1.26E+02	1.20E+03	1.20E+03	1.20E+03	2.85E−02	1.77E−01	1.77E−01	1.77E−01
	1024	5.90E+01	1.20E+03	1.20E+03	1.20E+03	1.72E−02	1.77E−01	1.77E−01	1.77E−01
	2048	5.56E+00	1.20E+03	1.20E+03	1.20E+03	5.38E−03	1.77E−01	1.77E−01	1.77E−01
sin	64	1.64E+01	1.17E+01	3.22E+01	7.10E+02	1.60E−02	7.90E−03	2.06E−02	9.43E−02
	128	3.24E−01	8.83E+00	1.58E+01	3.05E+02	1.93E−03	1.06E−02	1.49E−02	6.75E−02
	256	2.10E−01	3.58E+00	3.70E+01	1.19E+10	1.94E−03	8.13E−03	3.04E−02	1.77E−01
	512	**8.92E−02**	9.13E+00	9.18E+05	3.89E+07	**1.52E−03**	1.49E−02	1.77E−01	1.77E−01
	1024	2.72E−01	4.31E+01	3.89E+06	4.02E+06	1.54E−03	3.65E−02	1.77E−01	1.77E−01
	2048	3.73E−01	8.99E+01	1.41E+07	7.97E+15	1.57E−03	5.14E−02	1.77E−01	1.77E−01
atan	64	9.42E+02	1.42E+03	3.00E+02	1.20E+03	1.47E−01	1.77E−01	8.69E−02	1.77E−01
	128	9.23E+02	1.20E+03	1.20E+03	1.20E+03	1.34E−01	1.77E−01	1.77E−01	1.77E−01
	256	1.01E+03	1.20E+03	1.20E+03	1.20E+03	1.17E−01	1.77E−01	1.77E−01	1.77E−01
	512	6.76E+02	1.20E+03	1.20E+03	1.20E+03	7.30E−02	1.77E−01	1.77E−01	1.77E−01
	1024	3.54E+02	1.20E+03	1.20E+03	1.20E+03	5.71E−02	1.77E−01	1.77E−01	1.77E−01
	2048	3.59E+01	1.20E+03	1.20E+03	1.20E+03	1.37E−02	1.77E−01	1.77E−01	1.77E-01

На этот раз хорошие результаты были получены только при использовании `sin` в качестве функции активации. Случайно вышло так, что подобранные изначально архитектурные параметры оказались наиболее оптимальными.

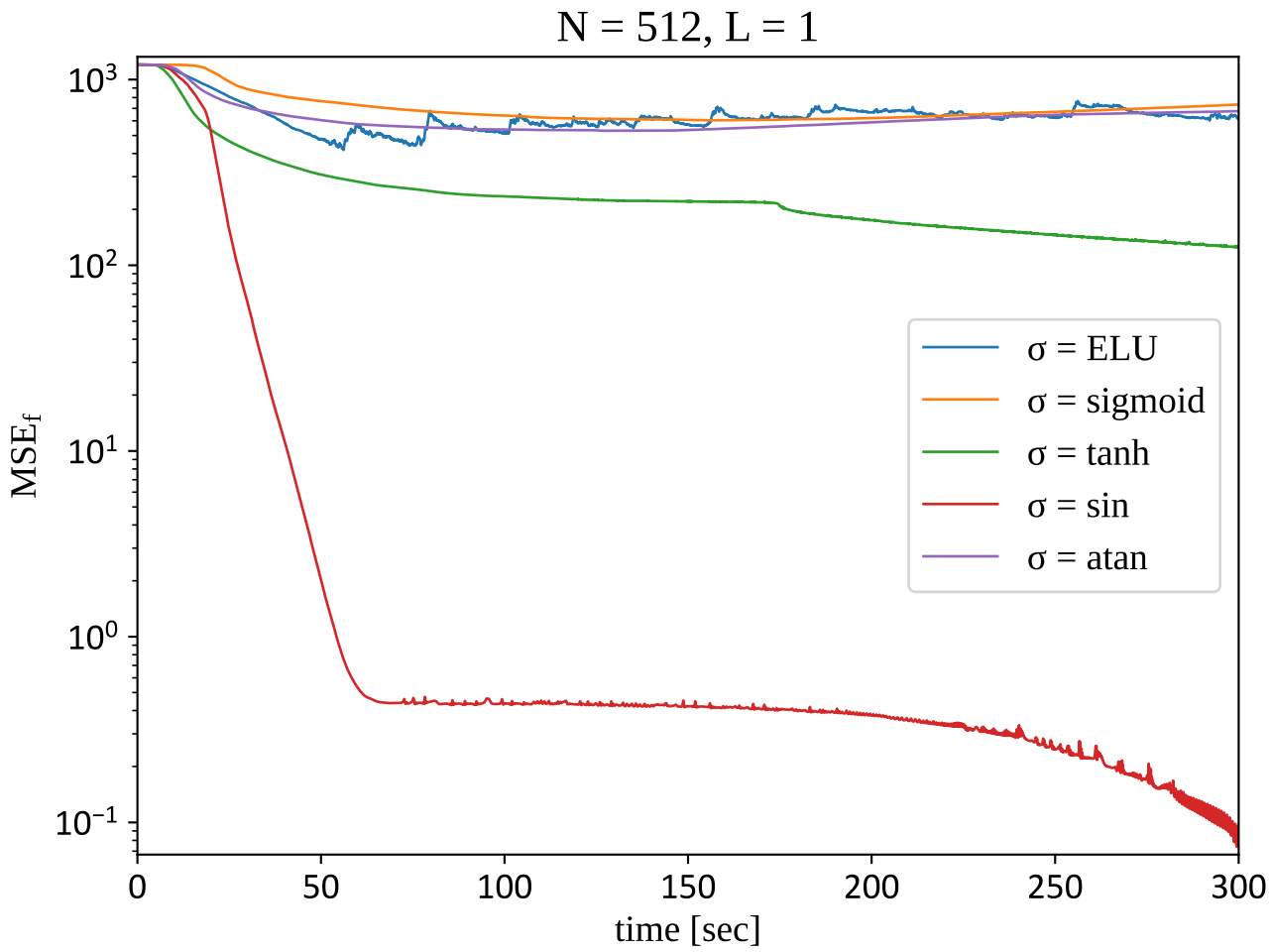

График 10. *Зависимость невязки по уравнению от времени при разных функциях активации*

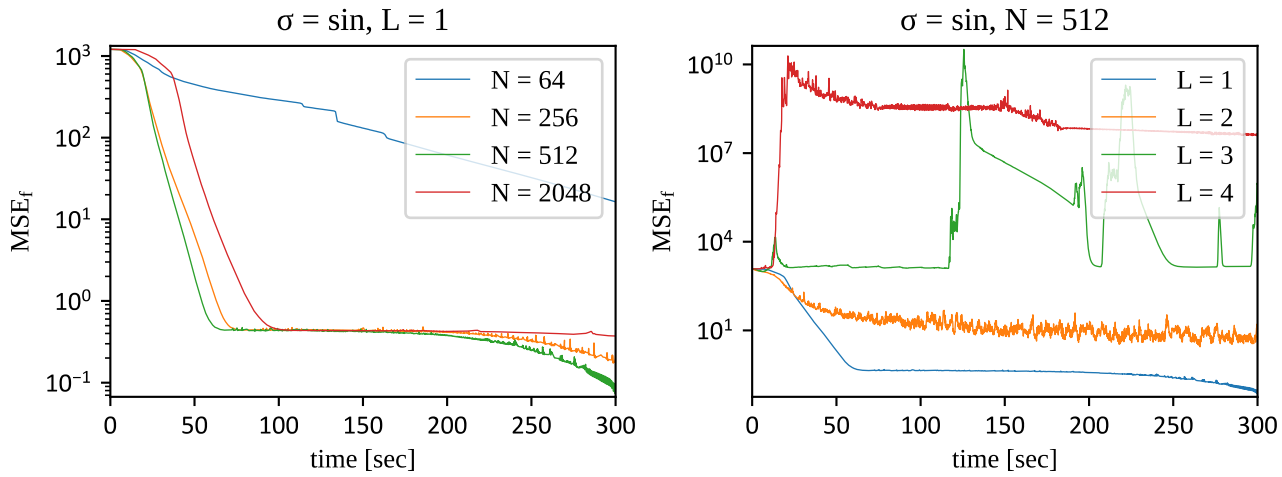

График 11. *Зависимость невязки по уравнению от времени при разных ширине слоя и количестве скрытых слоёв*

При $L = 1$ был получен наилучший результат и плавная сходимость по сравнению с другими L.

4.3. Сравнение алгоритмов оптимизации гиперпараметров на примере 5-мерной задачи

В этом и следующем пункте всегда используем квазислучайный метод генерации точек коллокации, поскольку нет сомнений в его превосходстве над случайным методом.

Таблица 17.

Область поиска и ограничение по времени для всех алгоритмов HPO

N	`qlograndint(64, 2048, 32)`
L	`randint(1, 5)`
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^4, 10^5, 10^6
$\lvert N_{\text{batch}} \rvert$	`qlograndint(64, 32768, 32)`
lr	`qloguniform(1e-4, 1, 1e-4)`
lr scheduler	`None, ExponentialLR-0.95, ReduceLROnPlateau-0.1-10, ReduceLROnPlateau-0.5-2`
t	300 сек
T	8 часов

Далее представлено по 3 лучших конфигурации, полученных разными алгоритмами HPO.

Таблица 18. *Лучшие конфигурации модели, полученные алгоритмами HPO*

$\lvert N_{\text{cfg}} \rvert$	№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
Случайный поиск												
96	5	107	300.1	5.0E−02	6.3E−04	1728	1	sin	10^5	160	3.6E−01	ReduceLROnPlateau-0.5-2
	10	697	300.0	4.3E−01	1.3E−03	96	1	sin	10^5	1056	2.2E−01	ReduceLROnPlateau-0.1-10
	21	6	304.9	3.2E+00	1.3E−02	1088	3	sin	10^6	128	8.0E−04	None
Случайный поиск + ASHAScheduler												
1593	364	550	300.1	1.2E−02	2.6E−04	1216	1	sin	10^5	1760	7.9E−01	ReduceLROnPlateau-0.5-2
	581	1039	300.1	2.8E−02	2.7E−04	704	1	sin	10^5	4608	9.5E−01	ReduceLROnPlateau-0.1-10
	5	106	300.1	4.7E−02	4.4E−04	1728	1	sin	10^5	160	3.6E−01	ReduceLROnPlateau-0.5-2
HyperOptSearch												
96	90	277	300.2	3.3E−01	1.7E−03	768	2	sin	10^5	928	8.1E−03	ReduceLROnPlateau-0.5-2
	76	1130	300.1	4.3E−01	1.3E−03	416	1	sin	10^5	2816	2.1E−01	ReduceLROnPlateau-0.5-2
	81	250	300.2	5.3E−01	2.1E−03	928	2	sin	10^5	1408	2.0E−03	ReduceLROnPlateau-0.5-2
HyperOptSearch + ASHAScheduler												
1053	817	624	300.1	**1.1E−02**	**1.7E−04**	928	1	sin	10^5	2016	1.0E+00	ReduceLROnPlateau-0.5-2
	548	696	300.1	1.1E−02	2.3E−04	736	1	sin	10^5	1824	8.6E−01	ReduceLROnPlateau-0.5-2
	607	690	300.1	1.1E−02	1.7E−04	704	1	sin	10^5	1728	9.9E−01	ReduceLROnPlateau-0.5-2
BOHB												
1108	820	1695	200.0	4.1E−01	1.4E−03	544	2	sin	10^4	736	1.4E−02	ReduceLROnPlateau-0.1-10
	223	326	219.5	4.2E−01	1.2E−03	128	1	sin	10^5	736	4.3E−01	ReduceLROnPlateau-0.1-10
	1059	2561	219.0	4.3E−01	1.2E−03	160	1	sin	10^4	736	7.9E−01	ReduceLROnPlateau-0.1-10

Простейшим методом — случайным поиском без планировщика — удалось получить неплохую конфигурацию, дающую хорошее решение, которое даже превосходит решение, полученное с помощью метода HyperOptSearch без планировщика. Однако последующие две конфигурации дают значения целевой функции на порядок хуже. Поэтому выходит, что нам просто повезло, и только одна конфигурация из 96 добилась такой точности.

Добавление планировщика обучения к алгоритмам поиска многократно увеличивает количество рассматриваемых конфигураций, и благодаря этому удалось получить несколько хороших экземпляров и улучшить абсолютную точность. В этом плане применение планировщика гораздо более выгодно, чем применение умного алгоритма поиска.

Комбинация HyperOptSearch и ASHAScheduler позволила добиться результатов лучше, чем комбинация случайного поиска и ASHAScheduler. При этом в случае байесовского метода суммарно было рассмотрено меньше точек, чем в случае случайного поиска. Эти результаты показывают превосходство байесовского поиска над случайным поиском.

Алгоритм BOHB, совмещающий в себе алгоритм поиска и планировщик, показал себя хуже всех остальных алгоритмов. Планировщик этого алгоритма работает по принципу классического HyperBand. Он начинает с короткого обучения нескольких конфигураций и прерывает их. Затем продолжает обучение наилучших конфигураций. Из-за того, что период сохранения сети на диск был установлен достаточно большим (30 сек, из-за опасений по потерям времени на запись), алгоритму приходилось перезапускать обучение. Из-за этого указанное для него в таблице время обучения ниже, чем у остальных алгоритмов, потому что часть выделенного на конфигурацию времени была потрачена на утерянный прогресс.

В итоге, из рассмотренных алгоритмов наилучший результат получен комбинацией HyperOptSearch + ASHAScheduler.

4.4. Автоматическая оптимизация гиперпараметров

В данном разделе оптимизация во всех случаях проводилась алгоритмами HyperOptSearch + ASHAScheduler.

2-мерная задача

Таблица 19. *Область поиска и ограничение по времени для 2-мерной задачи*

N	`qlograndint(64, 2048, 32)`
L	`randint(1, 5)`
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^2, 10^3, 10^4
$\lvert N_{\text{batch}} \rvert$	`qlograndint(64, 32768, 32)`
lr	`qloguniform(1e-4, 1, 1e-4)`
lr scheduler	`None, ExponentialLR-0.95, ReduceLROnPlateau-0.1-10, ReduceLROnPlateau-0.5-2`
t	30 сек
T	1 час

Таблица 20. *Лучшие конфигурации модели, полученные для 2-мерной задачи*

$\lvert N_{\text{cfg}} \rvert$	№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
665	457	1306	30.0	5.6E−05	1.4E−05	416	2	sin	1000	640	2.8E−02	`ReduceLROnPlateau-0.5-2`
	594	1286	30.0	7.8E−05	1.1E−05	672	2	sin	1000	384	1.5E−02	`ReduceLROnPlateau-0.5-2`
	596	1720	30.0	8.8E−05	1.2E−05	640	2	sin	1000	544	2.0E−02	`ReduceLROnPlateau-0.5-2`

Средствами автоматической оптимизации удалось получить лучшее значение целевой функции, чем при ручной оптимизации.

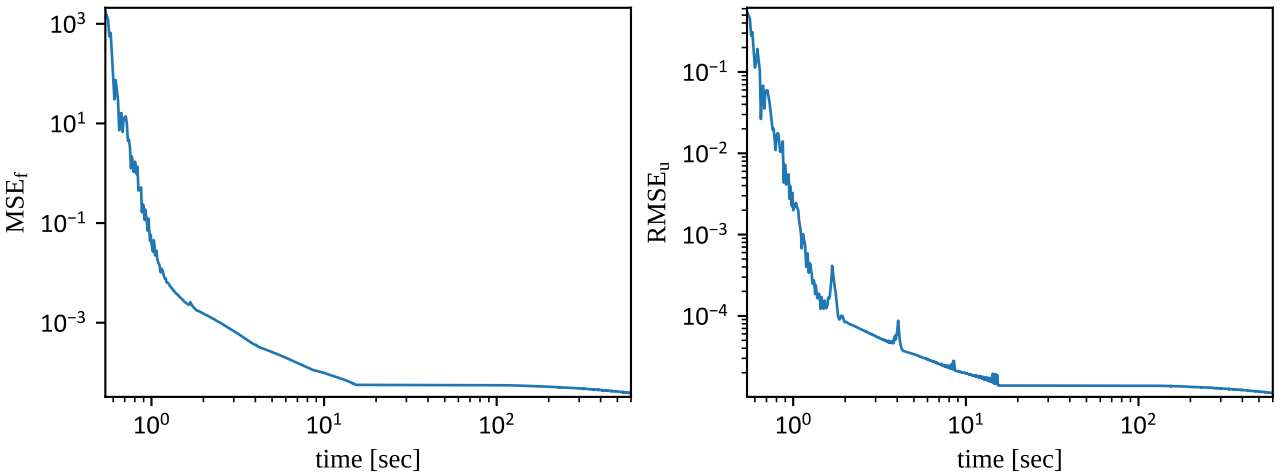

График 12. *Зависимость невязки по уравнению и невязки по решению от времени лучшей конфигурации для 2-мерной задачи*

Скорость обучения модели резко замедлилась. Примерно на 19 секунде lr резко начал уменьшаться, пока не достиг минимального значения (1.3×10^{-8}). В результате этого обучение резко замедлилось. Видимо, оптимизатор наткнулся на стохастический участок и слишком рано сбросил lr. Таким образом, параметры планировщика скорости обучения тоже имеет смысл оптимизировать наравне с другими гиперпараметрами, но в рамках работы было рассмотрено лишь несколько вариантов, чтобы избежать построения условной области поиска.

Конечные значения невязок:

$$\mathrm{MSE}_f = 3.885\mathrm{E}{-0.5} \qquad \mathrm{RMSE}_u = 1.125\mathrm{E}{-05}$$

Таким образом, решение удалось аппроксимировать с высокой точностью.

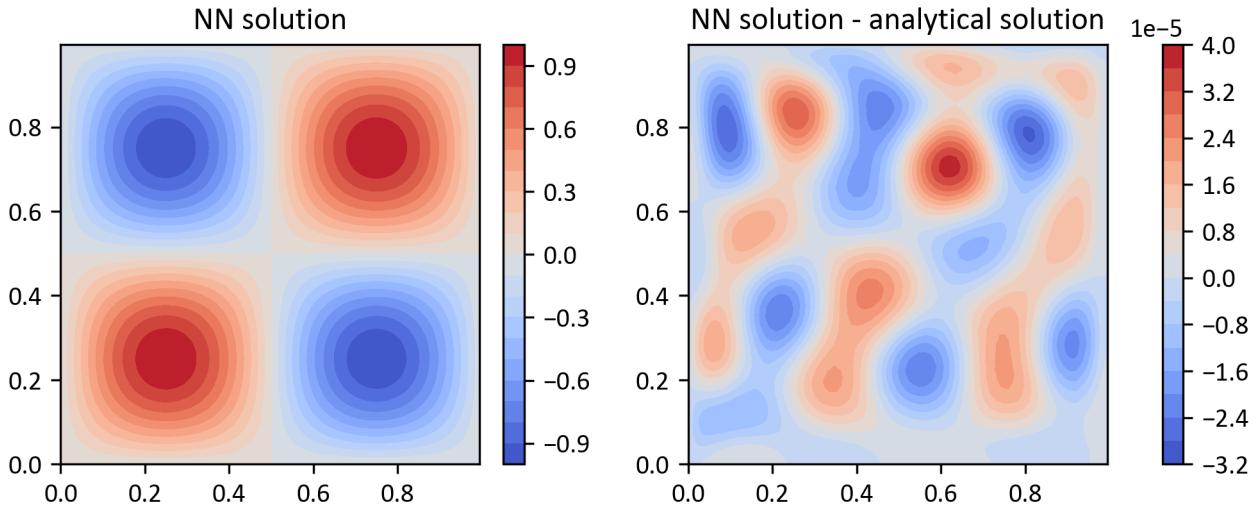

График 13. *Лучшее решение 2-мерной задачи и его отклонение от аналитического решения*

2-мерная задача на области [0; 2]

Таблица 21. *Область поиска и ограничение по времени для 2-мерной задачи на области [0; 2]*

N	`qlograndint(64, 2048, 32)`
L	`randint(1, 5)`
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^2, 10^3, 10^4
$\lvert N_{\text{batch}} \rvert$	`qlograndint(64, 32768, 32)`
lr	`qloguniform(1e-4, 1, 1e-4)`
lr scheduler	`None, ExponentialLR-0.95, ReduceLROnPlateau-0.1-10, ReduceLROnPlateau-0.5-2`
t	30 сек
T	1 час

Таблица 22. *Лучшие конфигурации модели, полученные для 2-мерной задачи на области [0; 2]*

$\lvert N_{\text{cfg}} \rvert$	№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
660	492	196	30.0	1.2E−03	2.4E−04	1504	2	sin	10^5	640	7.1E−03	`ReduceLROnPlateau-0.5-2`
	629	77	30.0	1.3E−03	2.0E−04	1568	2	sin	10^5	160	8.1E−03	`ReduceLROnPlateau-0.5-2`
	404	267	30.0	1.3E−03	1.6E−04	896	2	sin	10^5	608	1.2E−02	`ReduceLROnPlateau-0.5-2`

Средствами автоматической оптимизации удалось получить лучшее значение целевой функции, чем при ручной оптимизации.

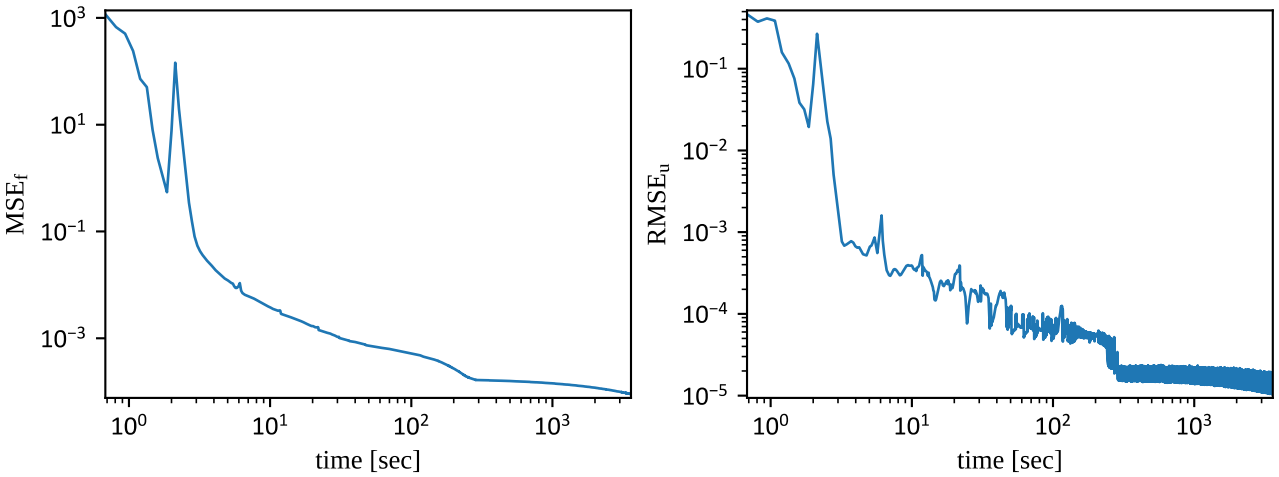

График 14. *Зависимость невязки по уравнению и невязки по решению от времени лучшей конфигурации для 2-мерной задачи на области [0; 2]*

Скорость обучения модели резко замедлилась. Примерно на 19 секунде lr резко начал уменьшаться, пока не достиг минимального значения (1.3×10^{-8}). В результате этого обучение резко замедлилось. Видимо, оптимизатор наткнулся на стохастический участок и слишком рано сбросил lr. Таким образом, параметры планировщика скорости обучения тоже имеет смысл оптимизировать наравне с другими гиперпараметрами, но в рамках работы было рассмотрено лишь несколько вариантов, чтобы избежать построения условной области поиска.

Конечные значения невязок:

$$\mathrm{MSE}_f = 9.017\mathrm{E}{-0.5} \qquad \mathrm{RMSE}_u = 1.153\mathrm{E}{-05}$$

Таким образом, решение удалось аппроксимировать с высокой точностью.

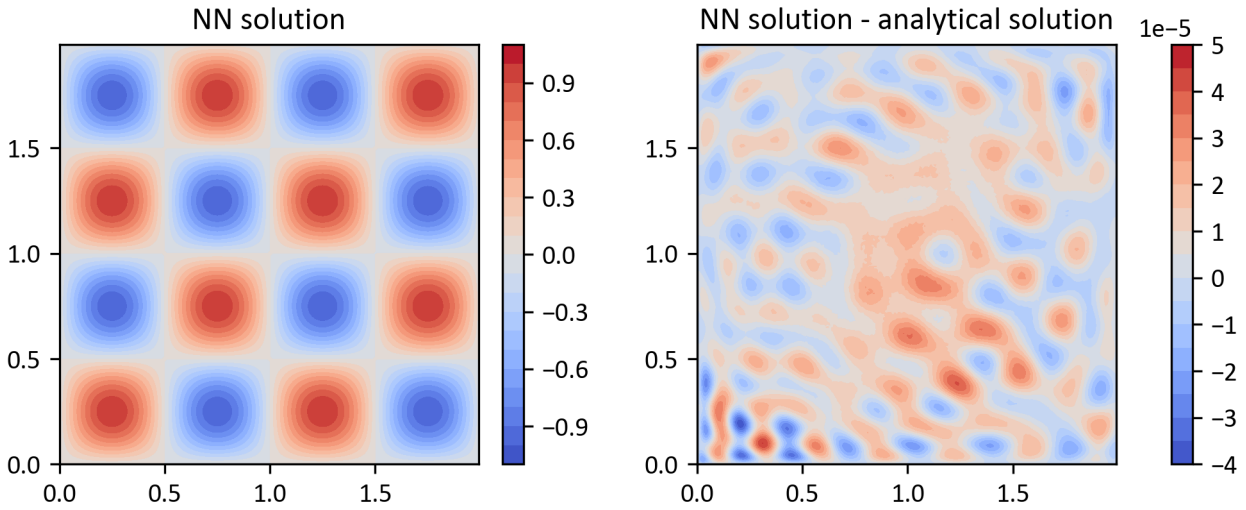

График 15. *Лучшее решение 2-мерной задачи на области [0; 2] и его отклонение от аналитического решения*

5-мерная задача

Таблица 23. *Область поиска и ограничение по времени для 5-мерной задачи*

N	`qlograndint(64, 2048, 32)`
L	`randint(1, 5)`
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^4, 10^5, 10^6
$\lvert N_{\text{batch}} \rvert$	`qlograndint(64, 32768, 32)`
lr	`qloguniform(1e-4, 1, 1e-4)`
lr scheduler	`None, ExponentialLR-0.95, ReduceLROnPlateau-0.1-10, ReduceLROnPlateau-0.5-2`
t	300 сек
T	8 часов

Таблица 24. *Лучшие конфигурации модели, полученные для 5-мерной задачи*

$\lvert N_{\text{cfg}} \rvert$	№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
1053	817	624	300.1	1.1E−02	1.7E−04	928	1	sin	10^5	2016	1.0E+00	ReduceLROnPlateau-0.5-2
	548	696	300.1	1.1E−02	2.3E−04	736	1	sin	10^5	1824	8.6E−01	ReduceLROnPlateau-0.5-2
	607	690	300.1	1.1E−02	1.7E−04	704	1	sin	10^5	1728	9.9E-01	ReduceLROnPlateau-0.5-2

В данном случае оптимальными являются конфигурации лишь с одним скрытым слоем.

Средствами автоматической оптимизации удалось получить лучшее значение целевой функции, чем при ручной оптимизации.

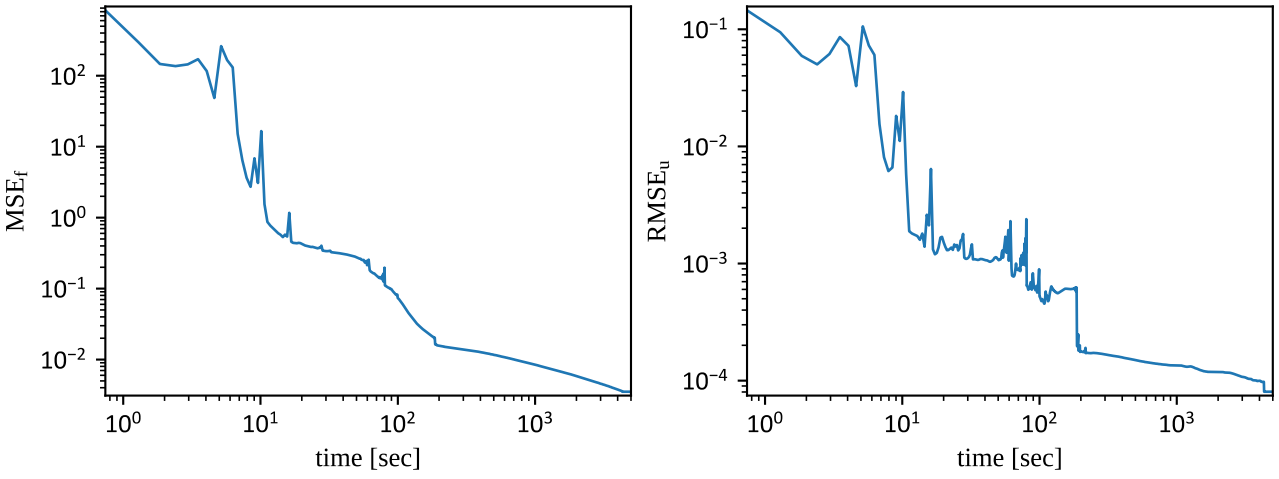

График 16. *Зависимость невязки по уравнению и невязки по решению от времени лучшей конфигурации для 5-мерной задачи*

Конечные значения невязок:

$$\text{MSE}_f = 3.518\text{E}{-03} \qquad \text{RMSE}_u = 8.036\text{E}{-05}$$

Таким образом, решение удалось аппроксимировать с хорошей точностью, но несколько хуже, чем в случае 2-мерной задачи.

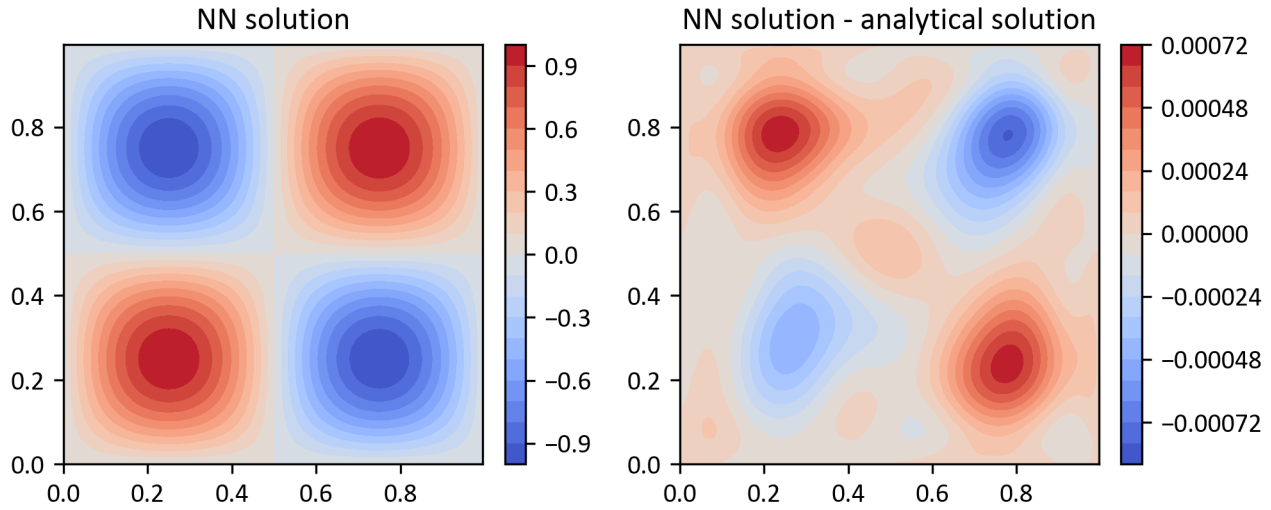

График 17. *Лучшее решение 5-мерной задачи и его отклонение от аналитического решения*

Представленный график изображает решение в плоскости (x_1, x_2) при $x_3{=}x_4{=}x_5{=}1/4$.

8-мерная задача

Таблица 25. *Область поиска и ограничение по времени для 8-мерной задачи*

N	`qlograndint(64, 2048, 32)`
L	`randint(1, 5)`
σ	`ELU, sigmoid, tanh, sin, atan`
$\lvert N_f \rvert$	10^6, 10^7, 10^8
$\lvert N_{\text{batch}} \rvert$	`qlograndint(64, 32768, 32)`
lr	`qloguniform(1e-4, 1, 1e-4)`
lr scheduler	`None, ExponentialLR-0.95, ReduceLROnPlateau-0.1-10, ReduceLROnPlateau-0.5-2`
t	1800 сек
T	48 часов

Таблица 26. *Лучшие конфигурации модели, полученные для 8-мерной задачи*

$\lvert N_{\text{cfg}} \rvert$	№	i	t, сек	MSE_f	RMSE_u	N	L	σ	$\lvert N_f \rvert$	$\lvert N_{\text{batch}} \rvert$	lr	lr scheduler
313	260	60.9	1802.1	3.3E+01	1.5E−02	2016	1	sin	10^6	160	6.8E−02	ExponentialLR-0.95
	262	60.2	1802.2	3.3E+01	1.5E−02	2016	1	sin	10^6	160	7.1E−02	ExponentialLR-0.95
	273	61.9	1803.0	3.9E+01	1.7E−02	1632	1	sin	10^6	160	7.7E−02	ExponentialLR-0.95

Попробуем дообучить лучшую конфигурацию, но всё-таки с использованием планировщика `ReduceLROnPlateau-0.5-2`. При использовании `ExponentialLR-0.95` обучение выходит на плато слишком рано.

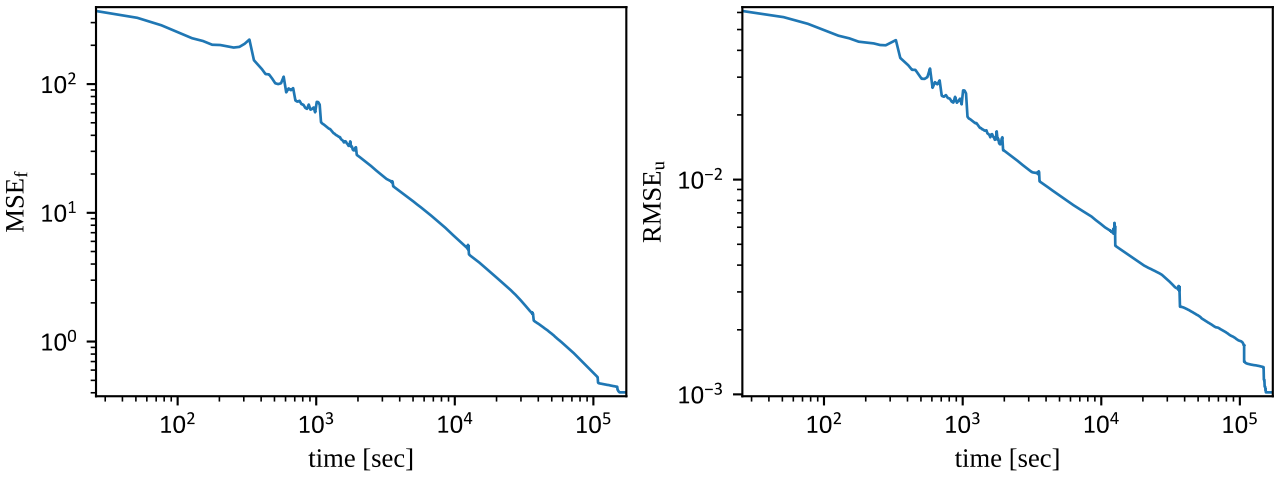

График 18. *Зависимость невязки по уравнению и невязки по решению от времени лучшей конфигурации для 8-мерной задачи*

Чтобы выйти на плато, понадобилось более 42 часов. Таким образом, при повышении размерности задачи колоссально увеличиваются потребности в вычислительных ресурсах.

Конечные значения невязок:

$$\mathrm{MSE}_f = 4.030\mathrm{E}{-01} \qquad \mathrm{RMSE}_u = 1.022\mathrm{E}{-03}$$

Решение удалось аппроксимировать с приемлемой точностью.

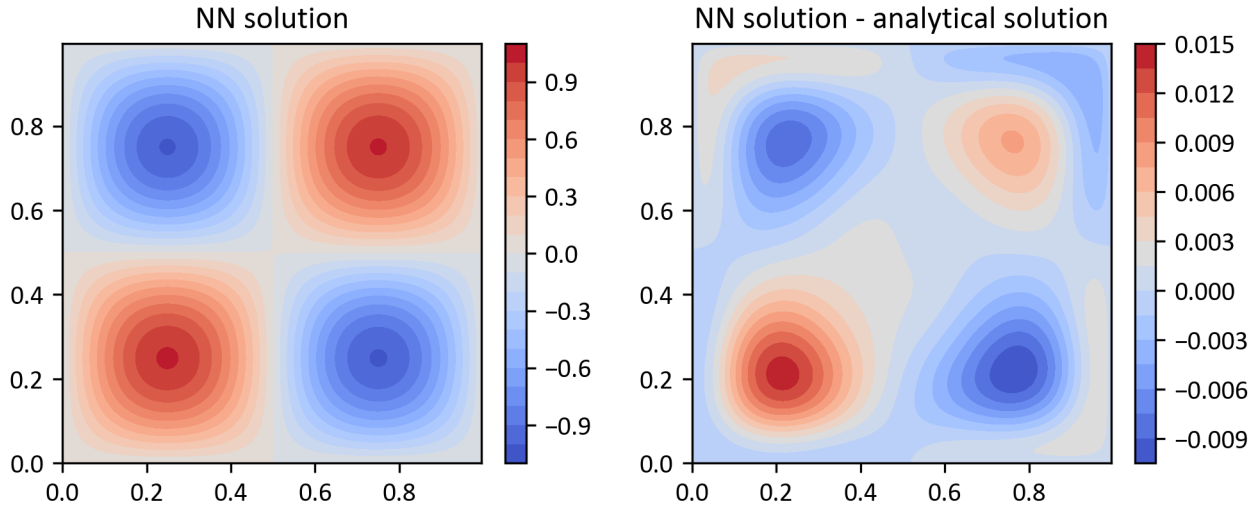

График 19. *Лучшее решение 8-мерной задачи и его отклонение от аналитического решения*

Представленный график изображает решение в плоскости (x_1, x_2) при $x_3=x_4=\ldots=x_8=1/4$. Хотя невязка по уравнению достаточно высокая по сравнению с решениями в другой размерности, графически решение довольно точно соответствует аналитическому.

Выводы

В ходе работы удалось эффективно использовать средства автоматической оптимизации гиперпараметров (HPO) для подбора конфигурации модели, аппроксимирующей уравнение Гельмгольца. Использование алгоритма ранней остановки позволяет рассмотреть гораздо больше конфигураций, и целесообразно в первую очередь применять его, а для большей точности можно подключить и байесовский алгоритм поиска. Из рассмотренных алгоритмов HPO наилучший результат был получен при комбинации HyperOptSearch + ASHAScheduler.

Посредством автоматической оптимизации гиперпараметров было получено более точное решение, чем при ручной оптимизации.

В работе продемонстрирована возможность решения уравнения Гельмгольца посредством нейронных сетей в пространствах больших размерностей (до 8). Однако, при увеличении размерности задачи быстро растут потребности в вычислительных ресурсах.

Литература

[1] Igel, H.: Computational seismology: a practical introduction. Oxford University Press, Oxford, United Kingdom (2017)

[2] Paganini, M., De Oliveira, L., Nachman, B.: Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters. Phys. Rev. Lett. 120, 042003 (2018). https://doi.org/10.1103/PhysRevLett.120.042003

[3] Rasp, S., Pritchard, M.S., Gentine, P.: Deep learning to represent subgrid processes in climate models. Proc. Natl. Acad. Sci. U.S.A. 115, 9684–9689 (2018). https://doi.org/10.1073/pnas.1810286115

[4] Kasim, M.F., Watson-Parris, D., Deaconu, L., Oliver, S., Hatfield, P., Froula, D.H., Gregori, G., Jarvis, M., Khatiwala, S., Korenaga, J., Topp-Mugglestone, J., Viezzer, E., Vinko, S.M.: Building high accuracy emulators for scientific simulations with deep neural architecture search. Mach. Learn.: Sci. Technol. 3, 015013 (2022). https://doi.org/10.1088/2632-2153/ac3ffa

[5] Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine Learning for Fluid Mechanics. Annu. Rev. Fluid Mech. 52, 477–508 (2020). https://doi.org/10.1146/annurev-fluid-010719-060214

[6] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks. 2, 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8

[7] Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 378, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045

[8] Moseley, B., Markham, A., Nissen-Meyer, T.: Solving the wave equation with physics-informed deep learning, http://arxiv.org/abs/2006.11894, (2020)

[9] Yu, T., Zhu, H.: Hyper-Parameter Optimization: A Review of Algorithms and Applications, http://arxiv.org/abs/2003.05689, (2020)

[10] Escapil-Inchauspé, P., Ruz, G.A.: Hyper-parameter tuning of physics-informed neural networks: Application to Helmholtz problems, http://arxiv.org/abs/2205.06704, (2023)

[11] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems. 2, 303–314 (1989). https://doi.org/10.1007/BF02551274

[12] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks. 2, 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8

[13] Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: A Deep Learning Library for Solving Differential Equations. SIAM Rev. 63, 208–228 (2021). https://doi.org/10.1137/19M1274067

[14] Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning. pp. 1139–1147. PMLR (2013)

[15] Duchi, J., Hazan, E., Singer, Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research. 12, 2121–2159 (2011)

[16] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. Coursera, video lectures, 264:1, (2012a)

[17] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization, https://arxiv.org/abs/1412.6980v9

[18] https://github.com/bckenstler/CLR

[19] Lau, S.: Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning, https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

[20] Li, M., Zhang, T., Chen, Y., Smola, A.J.: Efficient mini-batch training for stochastic optimization. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 661–670. ACM, New York New York USA (2014)

[21] Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization, https://www.coursera.org/learn/deep-neural-network

[22] https://miguel.data.sc/2017-11-05-first

[23] Loizou, N., Richtárik, P.: Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods, https://arxiv.org/abs/1712.09677v2

[24] Andrej Karpathy et al. Cs231n convolutional neural networks for visual recognition. Neural networks, 1:1, (2016)

[25] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778. IEEE, Las Vegas, NV, USA (2016)

[26] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In NIPS workshop on Bayesian Optimization in Theory and Practice, volume 10, page 3, 2013.

[27] Rasmussen, C.E.: Gaussian Processes in Machine Learning. In: Bousquet, O., von Luxburg, U., and Rätsch, G. (eds.) Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures. pp. 63–71. Springer, Berlin, Heidelberg (2004)

[28] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for Hyper-Parameter Optimization. Presented at the December 12 (2011)

[29] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization. 13, 455–492 (1998). https://doi.org/10.1023/A:1008306431147

[30] Hutter, F., Kotthoff, L., Vanschoren, J. eds: Automated machine learning: methods, systems, challenges. Springer, Cham, Switzerland (2019)

[31] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization (2018). http://arxiv.org/abs/1603.06560

[32] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., Talwalkar, A.: A System for Massively Parallel Hyperparameter Tuning, https://arxiv.org/abs/1810.05934v5

[33] Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Proceedings of the 35th International Conference on Machine Learning. pp. 1437–1446. PMLR (2018)

[34] I. M. Sobol. The distribution of points in a cube and the accurate evaluation of integrals. Zh. Vychisl. Mat. i Mat. Phys. 7, 784-802, 1967.

[35] Owen, A.B.: Scrambling Sobol' and Niederreiter–Xing Points. Journal of Complexity. 14, 466–489 (1998). https://doi.org/10.1006/jcom.1998.0487