**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**
**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

# Отчет о программном проекте

на тему: Нейросети с нуля

**Выполнил:**

Студент группы БПМИ204 — Подпись — Артур Маратович Гимранов (И.О.Фамилия)

19.05.2022
Дата

**Принял:**
Руководитель проекта — Дмитрий Витальевич Трушин
Имя, Отчество, Фамилия

доцент, к.ф.-м.н.
Должность, ученое звание

ФКН НИУ ВШЭ
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 22 мая 2022 — 9 — Трушин
Оценка (по 10-ти бальной шкале) — Подпись

**Москва 2022**

# Содержание

**Аннотация**

В рамках этого проекта предполагается изучить теорию нейросетей и их обучения и написать проект в котором будут реализованы все необходимые компоненты для работы и обучения нейросети.

# 1 Введение

Репозиторий на github.

Нейронная сеть – это набор нейронов и связей между ними. Структура нейронной сети пришла в программирование прямиком из биологии, где она встречается в виде устройства нервной системы живых существ.

В качестве примера можно рассмотреть зрачковый рефлекс. В нашем глазу есть сенсоры, которые улавливают количество света попадающего через зрачок. Отсюда импульс на сужение зрачка пойдет к нервным окончаниям. Таким образом, нейронная сеть, получив сигнал от фоторецепторов, обрабатывает данную информацию и принимает решение какие мышцы задействовать для ответной реакции, то есть сузить или расширить зрачок.

Нейрон лучшего всего представлять как функцию с множеством входов и одним выходом. Его задача взять числа с входов, преобразовать их каким-то образом и передать дальше.

Между нейронами есть связи – это каналы которые имеют вес и передают выход одного нейрона на вход другому с определенным весом.

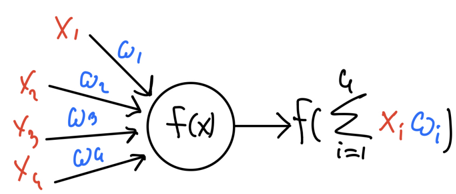


Рис. 1: Устройство нейрона

Чтобы придать дополнительную структуру, нейроны укладывают в слои, где внутри одного слоя нейроны не связаны, но связаны со следующим и предыдущим.

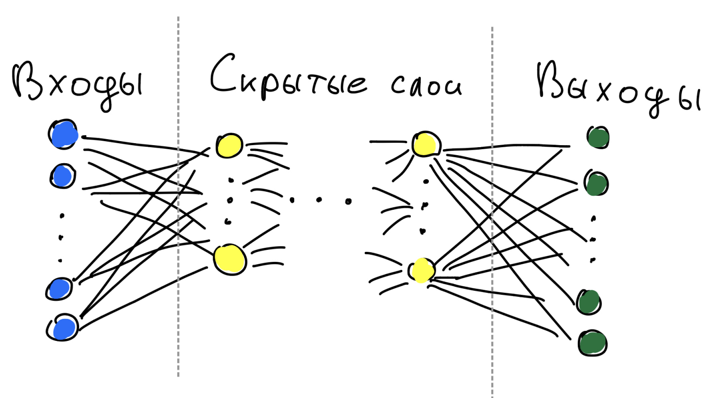


Рис. 2: Структура нейросети

Таким образом наша сеть представляет собой направленный ациклический граф. Где данные идут в одном направлении – от входов к выходам. Это сеть прямого распространения, а если быть точным многослойный перцептрон.

Теперь надо обучить сеть на специальной, размеченной выборке. Назначим все веса случайным образом и посмотрим что нам выдаст наша сетка. Зная результат работы нейросети и правильные ответы для наших входов пройдемся в обратном порядке по всем нейронам и поменяем параметры, чтобы уменьшить ошибку, часть нейронов будем учитывать с меньшим весом, а другую часть с большим. Через какое-то количество таких итераций, наша сетка научится выдавать правильные предсказания по обучающей выборке, потом только стоит протестировать ее на другой выборке и оценить точность.

Перенесем это все на язык математики и программирования. Посмотрим на взаимодействия двух соседних слоев. Пусть в текущем слое $m$ нейронов его выходы будут $y_i$, а в предыдущем $n$ нейронов, выходы $x_i$. Тогда можем посчитать $y_i = f\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)$, где $w_{ij}$ связь между $j$-ым нейроном предыдущего слоя и $i$-ым нейроном текущего слоя, $b_i$ – сдвиг, добавим его для большей гибкости, а $f$ – функция активации нейрона. Можем переписать это все на матричном языке, как $y = f(Ax + b)$, $A$ – содержит веса, $b$ – сдвиги, а $f$ мы применяем поэлементно. Представим наш слой в виде блока, который хранит параметры $\theta = (A, b)$ и $f$.

$$\mathbb{R}^n \xrightarrow{x} \underset{f(x,\theta)}{\boxed{\theta}} \xrightarrow{y} \mathbb{R}^m$$

Теперь о задаче, которую мы решаем. Пусть у нас есть $k$ признаков, которые уложены в вектор $x = [x^{(1)}, \dots, x^{(k)}]^t$, по которым мы хотим предсказать результат $y$ у которого тоже есть свои $l$ признаков $y = [y^{(1)}, \dots, y^{(l)}]^t$, аналогично запишем их в вектор. Тогда хотим выразить $y$ через $x$ в виде функции $F(x) = y$. Понятно, что для $n$ векторов и образов эта задача решается очень легко, нам же хочется обучить модель по начальной выборке, чтобы ошибка на векторах не из обучающей выборке была минимальна. Вот тут и вступает в дело наша нейросеть, именно в таком виде будем искать эту функцию $F$.

Для примера рассмотрим сетку из двух слоев. Пусть у нас есть два блока с параметрами $\theta_1, \theta_2$ и у нас выполняется такая цепочка функций

$$\mathbb{R}^n \xrightarrow{x_i} \underset{f(x,\theta_1)}{\boxed{\theta_1}} \xrightarrow{w_i} \underset{g(x,\theta_2)}{\boxed{\theta_2}} \xrightarrow{z_i} \mathbb{R}^m$$

Тогда мы хотим подобрать такие параметры $\theta_1, \theta_2$, чтобы минимизировать ошибку на обучающей выборке $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^m$.

$$\begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} \to \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix}$$

Добавим функцию ошибки $\phi(\theta_1, \theta_2) = \sum_{i=1}^{p} \|g(f(x_i, \theta_1), \theta_2) - y_i\|^2 \to \underline{\min}$, чтобы понимать что мы хотим уменьшать. Теперь мы хотим посчитать градиент по параметрам $\theta_1, \theta_2$ и градиентным спуском минимизирвать ошибку.

Таким образом, в проекте я реализую следующие компоненты: узел нейросети, который состоит из линейного отображения и нелинейной части, объекты отвечающие функциям штрафа. Будет реализован механизм вычисления градиента для узла и проталкивания градиента в узлы предыдущего слоя. На основе этого механизма будут написаны методы обучения нейросети. Передо мной стоят следующие задачи:

1. изучить теорию нейросетей и градиентоного спуска
2. кратко изложить в отчете теорию
3. имплементировать необходимые классы и структуры
4. изложить в отчете архитектуру и дизайн имплементации
5. написать сопроводительную документацию.

# 2 Важный пример

Рассмотрим задачу. Допустим мы хотим найти зависимость цены квартиры от некоторого набора параметров: жилой площади, расстояния до метро, расстояния до центра. Представьте, что мы измерили все эти параметры для $n$ квартир и получили наборы значений. Цену сложим в $y_i$, а параметры в вектора $x_i = [x_i^{(1)}, x_i^{(2)}, x_i^{(3)}]^t$. И мы хотим подобрать функцию $f$, чтобы $f(x_i) = y_i$. Конечно нет строгой зависимости подходящей любой квартире, но мы можем подобрать функцию в некотором виде, для которой отклонение в наших точках было бы наименьшим:

$$\sum_{i=0}^{n} |f(x_i) - y_i| \to \min$$

# 3 Функциональные требования

Наша программа будет получать на вход обучающую выборку подбирать по ней параметры и вычислять предполагаемое значение в точке.

1. `Net`. Инцилизирует ресурсы, слои нейронки и блок функции ошибок. Главный класс проекта, нужен для обучения и предсказания значения в точке.

2. `ComputeBlock`. Те самые слои нейронки или наши "блоки". Содержит функцию и ее параметры. Умеет вычислять функцию в точке, считать градиент по параметрам и проталкивать его в следующий блок.

3. `LosFunction`. Функция ошибок, нужна для расчета отклонения и вычисления градиента

4. `ActivationFunction`. Класс родитель для функций активации, имеет виртуальные методы для вычисления функции и ее производной в точке. Ниже перечислены ее наследники, в них должны быть имплементированы фукнции для вычисления и взятия производной.

   (a) `Sigmoid`

   (b) `Relu`

   (c) `Softmax`

# 4 Нефункциональные требования

- C++20 [1]
- Google C++ Style Guide [2]
- ClangFormat linter [3]
- Библиотека Eigen [4] для работы с матрицами
- Система поддержки версий: git [5] с github [6]

# 5 Нейросети

## 5.1 Теория

Сеть

$$\mathbb{R}^n \to \underset{f_1(x,\theta_1)}{\boxed{\theta_1}} \to \cdots \to \underset{f_i(x,\theta_i)}{\boxed{\theta_i}} \to \cdots \to \underset{f_k(x,\theta_k)}{\boxed{\theta_k}} \to \mathscr{L} \to \mathbb{R}$$

Где $f_i(x) = \phi(A_i x + b_i)$, $\phi(x)$ – функция активации, $(A_i, b_i) = \theta_i$ – параметры блока, $\mathscr{L}$ – функция потерь. Пусть $F_\Theta(x)$, $\Theta = (\theta_1, \ldots, \theta_k)$ функция предсказания, то есть $F_\Theta$ передает выход $i$ блока на вход $i+1$, пока не дойдет до последнего блока и не посчитает предсказываемый результат. Более формально, будем рекуррентно строить функции $F_{\Theta,i}(x) = f_i(F_{\Theta,i-1}(x), \theta_i)$, $F_{\Theta,1}(x) = f_1(x, \theta_1)$, обозначим $F_\Theta(x) = F_{\Theta,n}(x)$. Тогда функция которую мы хотим минимизировать $\psi(\Theta) = \frac{1}{n}\sum\limits_{i=1}^{n} \mathscr{L}(F_\Theta(x_i), y_i)$.

Цель заключается в том, чтобы пройти вперед, посчитать предсказание, отдать его в функцию потерь, и посчитать градиент по параметрам обратным проходом.

Нам надо уметь считать $\frac{\partial \psi}{\partial \theta_j}$.

$$\frac{\partial \psi}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{\partial \mathscr{L}(F_\Theta(x_i), y_i)}{\partial F_\Theta(x_i)} \frac{\partial F_\Theta(x_i)}{\partial \theta_j} \right)$$

$\frac{\partial \mathscr{L}(z,y)}{\partial z}$ – считается в зависимости от функции потерь.

$$\frac{\partial F_\Theta(x)}{\partial\theta_i} = \prod_{j=0}^{n-i+1}\left(\frac{\partial f_{n-j}(F_{\Theta,n-j-1}(x)),\theta_{n-j})}{\partial F_{\Theta,n-j-1}(x)}\right)\cdot\frac{\partial f_i(F_{\Theta,i-1}(x),\theta_i)}{\partial\theta_i}$$

Последнее выражение мы будем считать поэтапно, идя в обратную сторону по нашим блокам, сначала посчитаем для последнего, там это просто превращается в $\frac{\partial F_{\Theta,n}}{\partial\theta_n} = \frac{\partial f_n(F_{\Theta,n-1}(x),\theta_n)}{\partial\theta_n}$, дальше будет нарастать произведение $\prod_{j=0}^{n-i+1}\left(\frac{\partial f_{n-j}(F_{\Theta,n-j-1}(x)),\theta_{n-j})}{\partial F_{\Theta,n-j-1}(x)}\right)$, множители которого мы будем передавать в следущий блок из текущего. Например для перехода от последнего блока к предпоследнему, $\frac{\partial F_{\Theta,n}}{\partial\theta_{n-1}} = \frac{f_n(F_{\Theta,n-1}(x),\theta_n)}{\partial F_{\Theta,n-1}(x)}\cdot\frac{\partial f_{n-1}(F_{\Theta,n-2}(x),\theta_{n-1})}{\partial\theta_{n-1}}$, тут первое слагаемое пришло из последнего блока, а второе слагаемое относится к предпоследнему блоку. Продемонстрирую на картинке, для наглядности.

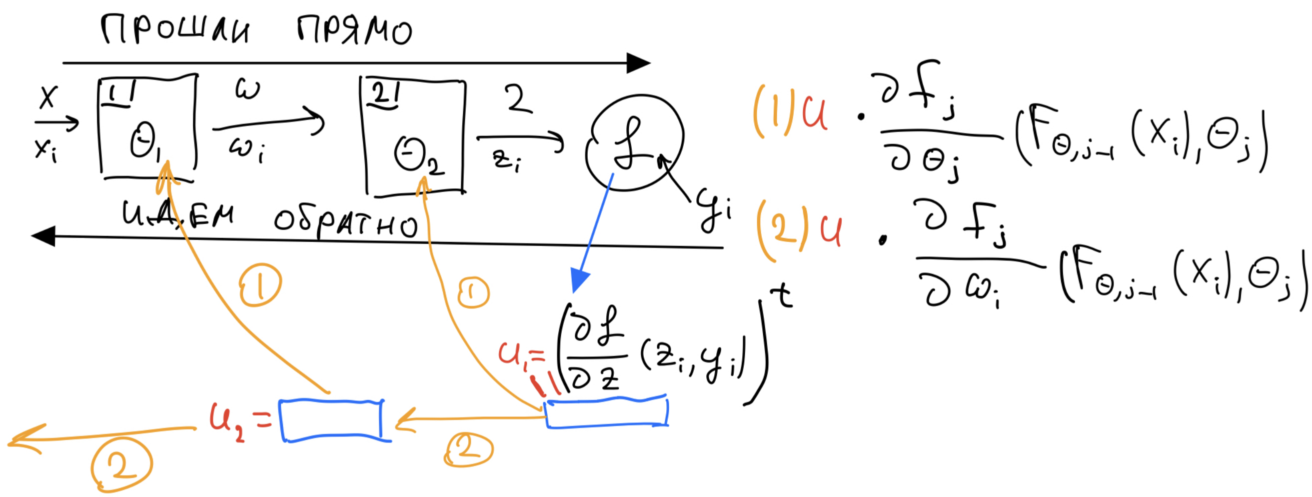


Рис. 3: Структура нейросети

Тут мы сразу считаем градиент по параметрам с учетом функции потерь. Считаем $u_1 = \left(\frac{\partial\mathscr{L}(z_i,y_i)}{\partial z}\right)^t$ и прокидываем дальше. Теперь считаем градиент по $\theta_2$ как $u_1\cdot\frac{\partial f_2(w_i,\theta_2)}{\partial\theta_2}$ и $u_2 = u_1\cdot\frac{\partial f_2(w_i,\theta_2)}{\partial w_i}$, прокидываем $u_2$ дальше и аналогично считаем градиенты по параметрам следующих блоков.

Осталось научиться считать градиенты

$$u^t\frac{\partial f(x,\theta)}{\partial x}, u^t\frac{\partial f(x,\theta)}{\partial\theta} = \left(u^t\frac{\partial f(x,\theta)}{\partial A}, u^t\frac{\partial f(x,\theta)}{\partial b}\right)$$

$$f(x,A,b) = \phi(A_ix+b_i)$$

$$u^td(\phi(Ax+b)) = u^t\phi'(Ax+b)d(Ax+b) = u^t\phi'(Ax+b)db = \langle\left(u^t\phi'(Ax+b)\right)^t, db\rangle \Rightarrow u^t\frac{\partial f(x,\theta)}{\partial b} = \phi'(Ax+b)^tu$$

$$u^td(\phi(Ax+b)) = u^t\phi'(Ax+b)d(Ax+b) = u^t\phi'(Ax+b)(dA)x = \operatorname{tr}\left(u^t\phi'(Ax+b)(dA)x\right) = \operatorname{tr}\left(xu^t\phi'(Ax+b)(dA)\right) =$$

$$= \langle\left(xu^t\phi'(Ax+b)\right)^t, dA\rangle_F \Rightarrow u^t\frac{\partial f(x,\theta)}{\partial A} = \phi'(Ax+b)^tux^t$$

$$u^td(\phi(Ax+b)) = u^t\phi'(Ax+b)d(Ax+b) = u^t\phi'Adx = \langle(u^t\phi'(Ax+b)A)^t, dx\rangle \Rightarrow u^t\frac{\partial f(x,\theta)}{\partial x} = A^t\phi'(Ax+b)u$$

За функцию потерь я взял MSE, то есть $\mathscr{L}(z,y) = \|z-y\|_2^2 = (z-y)^t(z-y)$

$$d\mathscr{L}(z,y) = 2(z-y)^tdx, \frac{\partial\mathscr{L}(z,y)}{\partial z} = 2(z-y)$$

И рассмотрел три функции активации.

**sigmoid**

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

$$\sigma'(x) = \frac{e^{-x}}{(e^{-x}+1)^2}$$

К вектору мы применяем ее поэлементно, тогда $\frac{\partial \sigma(x)}{\partial x}, x \in \mathbb{R}^n$ будет диагональная матрица, у которой на диагонали стоят $\sigma'(x_i)$, а все остальное по нулям

**relu**

Так как я столкнулся с проблемой "умирающего релу было принято решение не занулять $x$ в отрицательной части

$$\mathrm{relu}(x) = \begin{cases} x, & x > 0 \\ 0.01x, & x \le 0 \end{cases}$$

$$\mathrm{relu}'(x) = \begin{cases} 1, & x > 0 \\ 0.01, & x \le 0 \end{cases}$$

Аналогично sigmoid'е, $\frac{\partial\, \mathrm{relu}(x)}{\partial x}, x \in \mathbb{R}^n$ будет диагональная матрциа, у которой на диагонали стоят $\mathrm{relu}'(x_i)$, а все остальное по нулям

**softmax**

$$\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum\limits_{j=1}^{n} e^{x_j}}, x \in \mathbb{R}^n$$

Для

$$\frac{\partial\, \mathrm{softmax}(x)}{\partial x} = \begin{bmatrix} \frac{\partial s_1}{\partial x_1} & \frac{\partial s_1}{\partial x_2} & \cdots & \frac{\partial s_1}{\partial x_n} \\ \frac{\partial s_2}{\partial x_1} & \frac{\partial s_2}{\partial x_2} & \cdots & \frac{\partial s_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_n}{\partial x_1} & \frac{\partial s_n}{\partial x_2} & \cdots & \frac{\partial s_n}{\partial x_n} \end{bmatrix}, s_i = \mathrm{softmax}(x)_i$$

Теперь, так как производная это аддитивная функция, мы можем отдельно посчитать $\frac{\partial \mathscr{L}(F_\Theta(x_i), y_i)}{\partial \theta_j}$, для каждого $x_i$, усреднить и получить $\frac{\partial \psi}{\partial \theta_i}$ и теперь с правильно подобранным шагом градиентного спуска (learning rate), идти против градиента, тем самым уменьшая нашу ошибку

$$\theta_i' = \theta_i - lr \cdot \frac{\partial \psi}{\partial \theta_i}$$

### 5.2 Net

Главный класс сети. Содержит в себе слои и функцию потерь. Распределяет все данные между слоями. Выполняет обучение сети с последующим предсказанием результата. Пройдемся по ключевым функциям.

- В конструкторе мы задаем размеры слоев, функцию активации для каждого слоя, количество итераций обучения, размер батча для стохастического градиентного спуска и шаг градиентного спуска
- `train` – начинает процесс обучения по выборке. На вход принимает две матрицы $x$ – по столбикам содержит параметры, $y$ – по столбикам содержит выходы, для данных параметров
- `predict_1d` – по параметрам выдает предсказание
- `predict_2d` – принимает на вход матрицу у которой по столбцам вектора параметров, возвращает матрицу в которой по столбцам лежат соотвествующие предсказания
- `push_forward` – делает прямой проход, считает все выходы и итоговое предсказание

- back_propagate – запускает обратное распространение, считает нужные градиенты по параметрам и входам, проталкивает их дальше

- update_parameters – обновляет параметры после обратного распространения, принимает на вход шаг градиентного спуска

### 5.3 ComputeBlock

Слой нейронной сети. Содержат параметры $\theta = (A, b)$ и функцию активации $\phi$. Нужен для вычисления функции $f(x) = \phi(Ax + b)$, и градиентов $\frac{\partial f}{\partial A}, \frac{\partial f}{\partial b}, \frac{\partial f}{\partial x}$. Также обновляет параметры по заданному шагу спуска и размеру батча

- В конструкторе получает функцию активации, размеры матрицы и инцилизирует $A$ и $b$ случайными числами из равномерного распределения на $[-1, 1]$

- evaluate_1d – по переданному вектору вычисляет функцию $f(x) = \phi(Ax + b)$ и возвращает результат

- evaluate_2d – аналогично версии 1d, только принимает матрицу и возвращает матрицу, в которой по столбцам лежат результаты

- push_forward – продолжает прямой подход в текущем блоке

- back_propagate – продолжает обратное распространение в текущем блоке

- update_parameters – обновляет свои параметру после обратного распространения

- grad_A – вычисляет $\frac{\partial f}{\partial A}$

- grad_b – вычисляет $\frac{\partial f}{\partial b}$

- grad_x – вычисляет $\frac{\partial f}{\partial x}$

### 5.4 Функции активации

Реализованы $\text{softmax}, \text{relu}, \text{sigmoid}$ у всех трех функций имплементированы функции evaluate и derivative, для вычисления функции и производной в точке

### 5.5 Функция потерь

В качестве функции потерь была взята MSE

- evaluate_1d – вычисляет функцию ошибки по двум векторам $z, y$

- evaluate_2d – вычисляет функцию ошибки по двум матрицам $z, y$ как среднее функций ошибок для соответствующих столбцов

- grad_z – вычисляет $\frac{\partial \mathscr{L}(z,y)}{\partial z}$

## 6 Итоги

Была изучена теория нейронных сетей и градиентного спуска, полностью реализован функционал нейросети.

### Тест

Проект был протестирован на датасете mnist [7], и показал на нем точность более 90%.

**Входные данные**

- Сеть обучалась на выборке из 8500 размеченных изображениях рукописных цифр
- Входной слой состоял из 784 нейронов, каждый из которых отвечал за пиксель в картинке $28 \times 28$, с функцией активации relu
- Первый скрытый слой состоял из 16 нейронов, с функцией активации relu
- Второй скрытый слой состоял из 16 нейронов, с функцией активации softmax
- Выходной слой состоял из 10 нейронов
- Количество эпох (итераций обучения) = 3000
- Learning rate (шаг градиентного спуска) = 0.6
- Размер батча 128
- Функция потерь MSE

Из начальной выборки на каждой итерации обучения случайным образом выбиралось 128 векторов на которых мы считали ошибку и обучали. В первых двух слоях были выбраны функции активации relu, так как градиент sigmoidы быстро затухал и сеть переставала обучаться. Второй скрытый слой был с функцией активации softmax, она хорошо подходит для принятия решения "голосованием". на выходе был вектор из 10 координат и нейроны "голосовали" за координаты, индекс координаты с наибольшим весом и являлся ответом. Процесс обучения на CPU занял 12 минут.

После обучения выборка была протестирована на 1500 других изображений рукописных цифр, и правильно определила цифру на 1351 изображении, таким образом точность составила чуть больше 90%.

## Список литературы


[1] URL: https://en.cppreference.com/w/cpp/20.

[2] URL: https://google.github.io/styleguide/cppguide.html.

[3] URL: https://clang.llvm.org/docs/ClangFormat.html.

[4] URL: https://eigen.tuxfamily.org/index.php?title=Main_Page.

[5] URL: https://git-scm.com.

[6] URL: https://github.com.

[7] URL: http://yann.lecun.com/exdb/mnist/.