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 БПМИ228 —  — И.А.Бобошко
Подпись — И.О.Фамилия

26.04.2024
Дата

Принял:
Руководитель проекта — Никита Сергеевич Лукьяненко
Имя, Отчество, Фамилия

старший преподаватель
Должность, ученое звание

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

Дата проверки _____ 2024 — Оценка (по 10-ти бальной шкале) — Подпись

Москва 2024

Содержание

Аннотация

Цель данного проекта заключается в изучении основ теории нейросетей и их обучения, а также реализации своей нейросети и всех необходимых для ее работы и обучения компонент.

Ссылка на репозиторий

https://github.com/boboxa2010/NeuralNetworksFromScratch

1 Введение

Нейросеть представляет из себя математическую модель, состоящую из нескольких соединенных слоев, которая каким-то образом обучается по входным данным, чтобы выдать наиболее точный ответ на поставленную задачу. Цель данного проекта заключается в изучении теории нейросетей и разработке библиотеки на языке c++ для работы с нейросетями, а также обучении на базе библиотеки своей нейросети для распознавания рукописных цифр из датасета MNIST.

Были поставлены следующие задачи:

1. Изучить основы теории нейросетей и метод градиентного спуска, изложить это в отчете
2. Имплементировать необходимые функции и классы для работы с одним полносвязным слоем нейросети
3. Реализовать классы для работы с разными функциями потерь и активации
4. Создать общий интерфейс нейросети с возможностью настройки параметров, обучения.

5. Провести тесты (в частности обучить модель распознования картинок из датасета MNIST) и написать сопроводительную документацию для пользователей.

На момент написания отчета было проделано: изучена и описана в отчете теория нейросетей и градиентного спуска, полностью реализованы все основные компоненты библиотеки: классы для нейросети (в частности метод для ее обучения), полносвязный слоя, функции активации и потерь, также были написаны функции и классы для удобного чтения и обработки датасета MNIST, написана сопроводительная документация к библиотеке, а также реализованы unit-тесты для проверки корректности работы библиотеки.

2 Обзор литературы

В процессе изучения необходимой теории и написания кода я ознакомился со следующими материалами:

- Большую часть базовой теории, а именно формальную постановку задачи классификации, а также устройство функций потерь и метод градиентного спуска, я изучил по конспектам курса по машинному обучению от Евгения Соколова [11]
- Устройство полносвязнной нейронной сети, алгоритм Backpropagation, а также функции активации были изучены по материалам лекции по глубинному обучению от Ильдуса Садртдинова [10]
- Дифференцирование по матричным аргументам, используется для вычисления градиента, было изучено по материалам Дмитрия Трушина [12]

3 Описание функциональных и нефункциональных требований к программному проекту

3.1 Функциональные требования

Итоговый проект должен представлять из себя библиотеку для работы с нейросетью. Библиотека должна содержать следующие классы и функции:

1. ActivationFunction - интерфейс функции активации, а также классы, реализующие конкретные функции активации: Sigmoid, ReLu, SoftMax
2. Класс Layer, реализующий полносвязный слой нейросети
3. LossFunction - интерфейс функции потерь, а также классы, реализующие конкретные функции потерь: MSE, CrossEntropy
4. Функция LoadData, которая преобразовывает датасет MNIST в удобный для работы формат
5. Класс LearningRate, предоставляющий настройку гиперпараметра для обновления весов во время обучения
6. Класс Network, который реализует полносвязную нейросеть, а также включает в себя методы по нее обучению

3.2 Нефункциональные требования

В данном проекте исключения используются при работе с бинарными файлами, поэтому поставлен блок обработки ошибок, чтобы предотвратить экстренное завершение программы. Ошибки программиста обрабатываются с помощью assert-ов.

- Язык программирования c++17 [1]
- Компилятор gcc (версия 11.4.0) [2]
- Библиотека Eigen (версия 3.4.0) [7]
- Библиотека EigenRand (версия 0.5.0) [8]

- Система поддержки версии git (версия 2.34.1) [3]
- Система сборки cmake (версия 3.27) [4]
- clang-format(версия 14.0.0) [5], clang-tidy(версия 14.0.0) [6]
- MNIST dataset [9]

4 Теория нейросетей

4.1 Определения и обозначения

Определение 1. *Градиентом (Матрицей Якоби)* отображения $f : \mathbb{R}^m \to \mathbb{R}^n$ называется матрица из частных производных:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_m} \end{pmatrix}$$

Замечание 2. Градиент является направлением наискорейшего роста функции, а антиградиент (т.е. $-\nabla f$) — направлением наискорейшего убывания. Это ключевое свойство градиента, обосновывающее его использование в методах оптимизации [11].

Определение 3. *Функцией активации* называется всякая нелинейная, монотонная и желательно непрерывно-дифференцируемая $\sigma : \mathbb{R}^n \to \mathbb{R}^n$

Замечание 4. Функции активации бывают координатными, то есть $\sigma : \mathbb{R} \to \mathbb{R}$ и применяется к каждой координате вектора: $v \in \mathbb{R}^m \to \sigma(v) = (\sigma(v_1), \dots, \sigma(v_m))$. В таком случае $\nabla\sigma(v) = diag(\frac{\partial\sigma v_1}{\partial v_1}, \dots, \frac{\partial\sigma v_n}{\partial v_n})$

Определение 5. Функция, измеряющая ошибку одного предсказания или расстояние между заданными элементами, называется *функцией потерь* $\rho : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}_+$. С помощью этой функции мы будем измерять, насколько сильно отличается ответ и предсказанное значение.

Определение 6. Пусть $\mathbb{F}(x)$ - функция, задающая приближение, $\rho(x, y)$ - функция потерь, (X, y) - данные. *Функцией риска или функционалом ошибки* называется $\mathbb{E}(\rho(\mathbb{F}(X), y))$

4.2 Полносвязная нейросеть

Определение 7. *Полносвязным слоем* называют композицию отображений:

$$F_\theta : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R}^m$$

$$x \to Ax + b \to \sigma(Ax + b)$$

то есть сначала к аргументу применяется линейное отображение, а потом функция активации. Данное отображение определяется параметром $\theta = (A, b)$ - линейная часть, где $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, а также функцией активации $\sigma : \mathbb{R}^m \to \mathbb{R}^m$.

Определение 8. *Полносвязной нейросетью* называют композицию конечного числа полносвязных слоев:

$$\mathbb{F} : \mathbb{R}^{n_1} \xrightarrow{F_{\theta_1}} \mathbb{R}^{n_2} \to \dots \to \mathbb{R}^{n_k}$$

Замечание 9. Существует фундаментальный результат [10] из функционального анализа подтверждающий, что с помощью нейросети с хотя бы двумя слоями можно приблизить некоторые функции на компакте.

4.3 Постановка задачи

Пусть $\mathbb{F} : \mathbb{R}^n \to \mathbb{R}^m$ - неизвестное отображение, которые задано на конечном наборе значений, то есть пары $(x_i, \mathbb{F}(x_i))$. Введем в рассмотрение семейство функций $\mathbb{F}_\theta : \mathbb{R}^n \to \mathbb{R}^m$ - это отображение, параметризованное θ (нейросеть). Общая идея построения аппроксимации $\mathbb{F}$ заключается в подборе параметра θ, чтобы функции $\mathbb{F}$ и $\mathbb{F}_\theta$ были близки относительно функции риска на заданном множестве значений. То есть наша глобальная задача заключается в следующем:

$$\mathbb{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \rho(\mathbb{F}(x_i), \mathbb{F}_\theta(x_i)) \to min$$

Замечание 10. В дальнейшем нам понадобиться решить более узкую задачу с помощью нейросети - классификация.

$$\mathbb{F} : \mathbb{R}^n \to \mathbb{C} \text{ , где } \mathbb{C} \text{ - конечное множество классов, например, цифры.}$$

4.4 Процесс обучения

4.4.1 Метод градиентного спуска

Как мы выяснили ранее задача обучения модели, то есть подбора параметров нейросети, сводится к поиску минимума функции риска. Из *Замечания 2* появляется следующая идея для поиска минимума:

1. Выбрать начальный набор параметров модели - θ_0
2. Посчитать градиент в текущей точке и сдвинуться в противоположную сторону:

   $\theta_k = \theta_{k-1} - \eta_k \nabla\mathbb{L}(\theta_{k-1})$, где η_k - длина шага, которая нужна для контроля скорости движения.
3. Повторять шаг 2 пока не попадем в минимум функции.

Замечание 11. Существуют различные варианты инициализации начальных параметров, в данном проекте данный набор генерируется из $\mathcal{N}(0, 1)$, то есть стандартного нормального распределения.

Есть смысл останавливать данный итерационный процесс, например, в следующих ситуациях [11]:

- при близости градиента к нулю ($\|\nabla\mathbb{L}(\theta_k)\| \to 0$)
- при слишком малом изменении параметров модели на последней итерации ($\|\theta_k - \theta_{k-1}\| \to 0$).

Замечание 12. На практике возникают проблемы с вычислением $\nabla\mathbb{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla\rho_i(\theta)$ при большом N, то есть (объеме обучающей выборки), поэтому используется метод стохастического градиентного спуска [11], который вместо подсчета $\nabla\mathbb{L}(\theta)$ вычисляет градиент одного случайного слагаемого:

$$\nabla\mathbb{L}(\theta) \approx \nabla\rho_\alpha(\theta), \text{ где } \alpha \text{ - случайно выбранный индекс}$$

Также существует *Mini-batch градиентный спуск* - это вариация стохастического градиентного спуска, где вместо одного случайного слагаемого считается градиент небольшого числа слагаемых.

$$\nabla\mathbb{L}(\theta) \approx \frac{1}{c}\sum_{i=1}^{c} \nabla\rho_i(\theta), \text{ где } c \text{ — размер batch-a}$$

4.4.2 Алгоритм BackPropagation

Для градиентного спуска нам необходимо уметь считать $\nabla\mathbb{L}(\theta)$. Заметим, что если мы научимся считать градиент для одного слагаемого $\nabla\rho(\theta)$, то несложно получить и $\nabla\mathbb{L}(\theta)$ (считаем для каждого и усредняем).

Пусть у нас есть $\mathbb{F}_\theta : \mathbb{R}^n \to \mathbb{R}^m$ - нейросеть, состоящая из k слоев, $\theta = (\theta_1, \ldots, \theta_k)$, а также пара (x, y) - входное значение и ожидаемый ответ.

Замечание 13.

$$\frac{\partial \rho(F_\theta(x), y)}{\partial \theta_i} = \frac{\partial \rho(z, y)}{\partial z}(F_k(x), y)\frac{\partial F_k(x)}{\partial x}(F_{k-1}(x))\frac{\partial F_{k-1}(x)}{\partial x}(F_{k-2}(x))\dots\frac{\partial F_{i+1}(x)}{\partial x}(F_i(x))\frac{\partial F_i(x)}{\partial \theta_i}(F_{i-1}(x))$$

, где F_i - композиция первых i слоев. Отсюда видно, что лучше считать производные в обратном порядке, так как вычисления, проделанные при подсчете $i+1$ можно использовать для i. Из этой идеи и появляется эффективный способ вычисления градиента.

Алгоритм BackPropagation можно разбить на следующие шаги:

1. ForwardPass - вычисляем и запоминаем входы in_i и выходы out_i каждого слоя:
$$in_i = out_{i-1} = F_{i-1}(\dots(F_1(x))\dots)$$
$$out_i = \sigma(A_i in_i + b_i)$$
2. Вычисляем $u_{k+1} = \frac{\partial \rho(z,y)}{\partial z}(out_k, y)$ - вектор-строка.
3. BackwardPass - Для каждого $i = k, k-1, \dots, 1$ вычисляем и сохраняем:
$$\nabla A_i = \sigma_i'(A_i in_i + b_i)u_{i+1}^\top in_i^\top$$
$$\nabla b_i = \sigma_i'(A_i in_i + b_i)u_{i+1}^\top$$
и проталкиваем назад вектор: $u_i = u_{i+1}\ \sigma_i'(A_i in_i + b_i)\ A_i$
4. Итог: Получили градиент для всех $\theta_i = (A_i, b_i)$

Выведем формулы для $\nabla A, \nabla b, u_k$. Пусть имеется $F_\theta(x) = \sigma(Ax + b)$.

$$\nabla \rho(\theta) = \nabla \rho(F(x), y) = \frac{\partial \rho(z, y)}{\partial z}\frac{\partial F_\theta(x)}{\partial \theta}$$

Обозначим $\frac{\partial \rho(z,y)}{\partial z} = u, G(A, b, x) = F_\theta(x)$. Тогда:

$$dG(A, b, x) = \sigma'(Ax + b)[(dA)x + Adx + db] = \sigma'(Ax + b)(dA)x + \sigma'(Ax + b)Adx + \sigma'(Ax + b)db$$

$$\nabla \rho(\theta) = u(\sigma'(Ax + b)(dA)x + \sigma'(Ax + b)Adx + \sigma'(Ax + b)db) = tr(u(\sigma'(Ax + b)(dA)x) + \langle(u\sigma'(Ax + b)A)^\top, dx\rangle +$$
$$+\langle(u\sigma'(Ax + b))^\top, db\rangle$$

Тогда получаем необходимые формулы:

$$\nabla A = \sigma'(Ax + b)u^\top x^\top$$
$$\nabla b = \sigma'(Ax + b)u^\top$$
$$u_k = u_{k+1}\sigma'(Ax + b)A$$

Замечание 14. Стоит отметить, что у данного подхода есть три преимущества, которые обуславливают его применение на практике:

1. Избавляет нас от «символьного» вычисления производных. Достаточно лишь уметь считать производную функции потерь (2 шаг), а также σ (3 шаг)
2. Вычислительная сложность одной итерации Backward Pass (то есть вычисление градиента для одного слоя) $\approx$ два матричных умножения, а Forward Pass $\approx$ одно матричное умножение. То есть данный алгоритм позволяет вычислить все градиенты так же быстро, как и значения функций.
3. Все вычисления обобщаются на случай (то есть можно будет представить все вычисления через матрично-векторные операции), когда мы хотим посчитать градиент сразу в нескольких точках, например, если используем Mini-Batch градиентный спуск.

Таким образом, процесс обучения нейросети можно разбить на следующие шаги:

1. Инициализировать случайным образом θ_0
2. Посчитать $\nabla\mathbb{L}(\theta)$ (тут все зависит от вида градиентного спуска)
3. Сделать шаг градиентного спуска:
$$\theta_k = \theta_{k-1} - \eta_k\nabla\mathbb{L}(\theta_{k-1})$$ (зачастую η_k подбирается эмпирически)
4. Повторять шаги 2-3 пока это имеет смысл

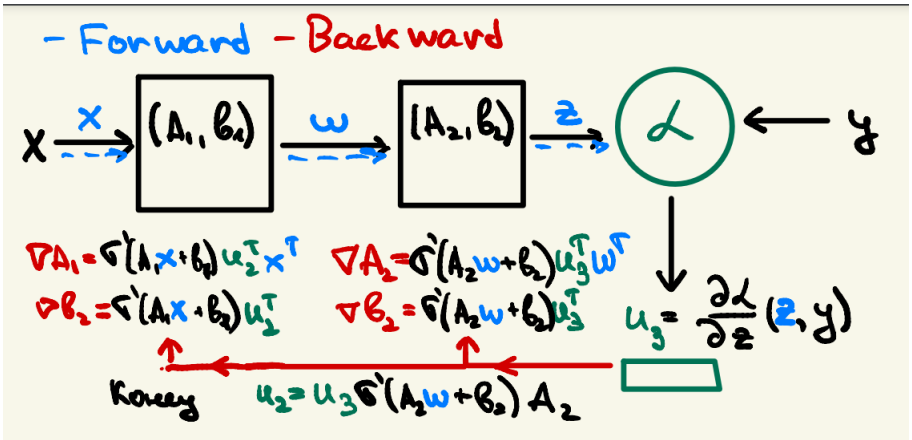
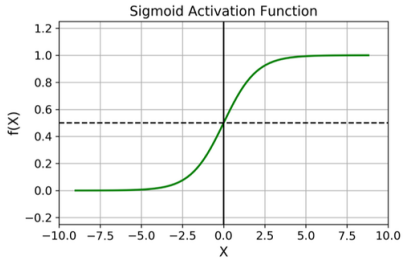


Рис. 1: Иллюстрация работы алгоритма для двух слоев

4.5 Функции Активации

4.5.1 Sigmoid

$$Sigmoid : \mathbb{R}^n \to \mathbb{R}^n, x_i \to \frac{1}{1 + \exp -x_i}$$

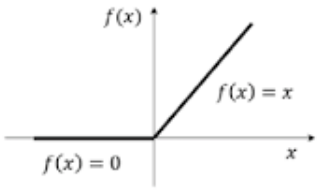


4.5.2 ReLu

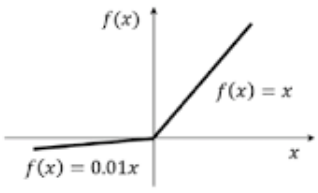
$$ReLu : \mathbb{R}^n \to \mathbb{R}^n, x_i \to max(0, x_i)$$

Замечание 15. Функция недифференцируема в нуле, данная проблема может быть решена с помощью определения производной в нуле, например, 1 или 0. Также существует LeakyRelu, у которой нет проблем в нуле:

$$LeakyReLu : \mathbb{R}^n \to \mathbb{R}^n, x_i \to max(\epsilon x_i, x_i), \epsilon > 0$$



ReLU activation function



LeakyReLU activation function

4.5.3 SoftMax

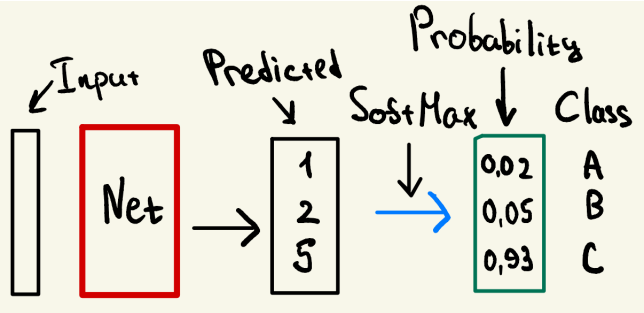
$$SoftMax : \mathbb{R}^n \to (0,1)^n, x_i \to \frac{\exp x_i}{\sum_{k=1}^{n} \exp x_k}$$

Для удобства введем обозначение:

$$w_i = \frac{\exp x_i}{\sum_{k=1}^{n} \exp x_k}$$

Пусть решается задача классификации, то есть мы хотим, чтобы нейросеть выдавала нам по объекту его класс. $x \in \mathbb{R}^n = (x_1, \dots x_n)$, где $x_i \in \mathbb{R}$ - уверенность нейросети в том, что объект принадлежит классу i. Тогда $SoftMax(x)_i$ - это вероятность принадлежности объекта классу i. Это действительно вероятность, потому что: $\sum_{i=1}^{n} w_i = 1$ и $0 < w_i < 1$

Данная функция используется как последняя функция активации при решении задачи классификации для того, чтобы отобразить выходы нейросети в вероятности принадлежности классу.



Для обучение нейросети нам необходимо уметь считать $\nabla SoftMax$. Так как это многомерное отображение, то градиентом будет матрица (то есть Якобиан). Воспользуется следующим трюком:

$$\frac{\partial ln(w_i)}{x_j} = \frac{1}{w_i}\frac{\partial w_i}{x_j}$$

Тогда искомая частная производная выражается:

$$\frac{\partial w_i}{x_j} = w_i \frac{\partial ln(w_i)}{x_j}$$

$$\frac{\partial ln(w_i)}{x_j} = \frac{\partial x_i}{x_j} - \frac{\partial ln(\sum_{k=1}^{n} \exp x_k)}{x_j} = \delta_{ij} - w_j$$

Тогда:

$$\nabla SoftMax_{ij} = w_i(\delta_{ij} - w_j)$$

Для более эффективного подсчета градиента можно векторизовать вычисления:
Положим: $w = (w_1, \dots, w_n)$

$$\nabla SoftMax = \begin{pmatrix} \frac{\partial w_1}{\partial x_1} & \frac{\partial w_1}{\partial x_2} & \cdots & \frac{\partial w_1}{\partial x_n} \\ \frac{\partial w_2}{\partial x_1} & \frac{\partial w_2}{\partial x_2} & \cdots & \frac{\partial w_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial w_n}{\partial x_1} & \frac{\partial w_n}{\partial x_2} & \cdots & \frac{\partial w_n}{\partial x_n} \end{pmatrix} = \begin{pmatrix} w_1(1-w_1) & -w_1w_2 & \cdots & -w_1w_n \\ -w_1w_2 & w_2(1-w_2) & \cdots & -w_2w_n \\ \vdots & \vdots & \ddots & \vdots \\ -w_1w_n & -w_2w_n & \cdots & w_n(1-w_n) \end{pmatrix} =$$

$$= \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n \end{pmatrix} - \begin{pmatrix} w_1w_1 & w_1w_2 & \cdots & w_1w_n \\ w_1w_2 & w_2w_2 & \cdots & w_2w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_1w_n & w_2w_n & \cdots & w_nw_n \end{pmatrix} = \text{diag}(w) - ww^\top$$

Численная устойчивость SoftMax. Данная версия $SoftMax$ неустойчивая (быстро переполняется) из-за быстрого роста экспоненты. Например, $exp(1000) = inf$

```
Vector BadSoftMax(const Vector &v) {
    Vector exp = v.array().exp();
    return exp.array() / exp.sum();
}

int main() {
    std::cout << BadSoftMax({1, 2, 3, 1000}) <<'\n';
}
// 0    0    0 -nan
```

$$w_i = \frac{\exp x_i}{\sum_{k=1}^{n} \exp x_k} = \frac{const}{const} \frac{\exp x_i}{\sum_{k=1}^{n} \exp x_k} = \frac{\exp\left(x_i + ln(const)\right)}{\sum_{k=1}^{n} \exp\left(x_i + ln(const)\right)}$$

Тогда если взять в качестве $ln(const) = -max(x)$, то $exp(x_i - max(x)) \in (0, 1)$. Пусть $k = argmax(x)$, тогда $exp(x_k - max(k)) = 1$. Получили, что исчезает вероятность поделить на ноль, так как знаменатель хотя бы 1.

```
Vector StableSoftMax(const Vector &v) {
    Vector exp = (v.array() - v.maxCoeff()).exp();
    return exp.array() / exp.sum();
}

int main() {
    std::cout << StableSoftMax({1, 2, 3, 1000}) << '\n';
}
// 5.556e-309 5.556e-309 5.556e-309           1
```

4.6 Функции Потерь

4.6.1 MSE

$$\rho : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$$

$(x, y) \to \|x - y\|_2^2 = \langle x - y, x - y \rangle$, где $\langle x, y \rangle = x^\top y$ - стандартное скалярное произведение в $\mathbb{R}^n$.

Пусть у нас есть некоторые данные: $(y_1, \ldots, y_n), (\hat{y_1}, \ldots, \hat{y_n})$, где $y_i \in \mathbb{R}^m, \hat{y}_i \in \mathbb{R}^m$. В нашем случае это выходы нейросети и правильные ответы, тогда:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \rho(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} \|y_i - \hat{y}_i\|_2^2$$

Для обучения нам нужно уметь считать частную производную:

$$\frac{\partial \rho(z, y)}{\partial z} = \frac{\partial(\langle z - y, z - y \rangle)}{\partial z} = \frac{\partial(\langle z, z \rangle)}{\partial z} - 2\frac{\partial(\langle z, y \rangle)}{\partial z} + \frac{\partial(\langle y, y \rangle)}{\partial z} = 2z - 2y = 2(z - y)$$

4.6.2 CrossEntropy

Пусть решается задача классификации и мы получили два вероятностных распределения: $p = (p_1, \ldots, p_c), p_k$ - уверенность алгоритма (нейросети) в том, что ответ - это класс k, $q = (q_1, \ldots, q_c)$ - истинное распределение (правильный ответ на задачу), где c - количество классов. Тогда, чтобы измерить их близость используют CrossEntropy.

$$CE(p, q) = -\sum_{k=1}^{c} q_k log(p_k) = -\langle q, log(p) \rangle$$

$$\frac{\partial CE(z, y)}{\partial z} = -\frac{\partial \langle q, log(p) \rangle}{\partial z} = -\sum_{k=1}^{c} \frac{q_k}{log(p_k)} = -\frac{q}{p}$$

Замечание 16. Как получить данные вероятностные распределения в контексте задачи классификации? Положим $q_k = \mathbb{I}(y = k)$, где y - правильный ответ(это называется OneHotEncoding), в качестве p можно взять $SoftMax(z)$, где $z = \mathbb{F}_\theta(x)$ Подставим в CrossEntropy:

$$-\sum_{k=1}^{c} q_k log(SoftMax(z_k)) = -\sum_{k=1}^{c} q_k (z_k - log(\sum_{i=1}^{c} \exp(z_i)))$$

Посчитаем производную по z:

$$\frac{\partial CE(z, q)}{\partial z_a} = -\sum_{k=1}^{c} q_k \frac{\partial}{\partial z_a}(z_k - log(\sum_{i=1}^{c} \exp(z_i))) = -q_k - (\sum q_k) softmax(z_a) = softmax(z_a) - q_k$$

$$\frac{\partial CE(z, q)}{\partial z} = softmax(z) - q$$

5 Руководство по использованию библиотеки

В данном разделе описано устройство библиотеки и как ей пользоваться. Библиотека находится в **namespace nn**, то есть чтобы воспользоваться чем-нибудь из библиотеки необходимо написать **nn::**. В проекте используется небольшая обертка над библиотекой **Eigen**(она используется для матрично-векторных операций).

```
using Index = Eigen::Index;
using Scalar = double;
using Vector = Eigen::VectorXd;
using RowVector = Eigen::RowVectorXd;
using Matrix = Eigen::MatrixXd;
```

5.1 Activation Function

Класс ActivationFunction представляет из себя стирающий тип, в котором заложены два метода:

- **Evaluate(v)** - возвращает $\sigma(v)$, также есть версия для матриц, в таком случае вычисляется значение функции для каждого столбца
- **GetDifferential(v)** - возвращает матрицу Якоби функции σ в точке v

Замечание 17. Что такое стирающий тип и почему это используется ? Стирание типов - это намеренная потеря части информации о классе для достижения общего интерфейса, то есть оставляем только ключевую информации о функционале класса. Данная техника позволяет добиться полиморфизма, а также более дружелюбна к пользователю и безопасна, так как не использует указателей. Также стоит отметить, что написанный класс ActivationFunction позволяет пользователю использовать свои функции активации. Пример как это работает:

```
class Foo {
public:
    nn::Vector Evaluate(const nn::Vector &v) const {
        return v;
    }

    nn::Matrix GetDifferential(const nn::Vector &v) const {
        return v * v.transpose();
    }
};

int main() {
    nn::ActivationFunction f = Foo();
    std::cout << f->Evaluate({1, 2, 3}) << '\n';
}
```

Также библиотека представляет 5 конкретных функций активации: **Sigmoid**, **ReLu**, **LeakyReLu**, **Linear**, **SoftMax**

5.2 Loss Function

Класс **Loss Function** - стирающий тип, предоставляющий интерфейс функции потерь, а именно следующие методы:

- **Evaluate(x, y)** - возвращает $\rho(x, y)$, то есть расстояние между x, y. Поддерживает работу с матрицами, считается расстояние для каждой пары столбцов.
- **GetGradient(x, y)** - возвращает вектор-строку $\frac{\partial\rho(x,y)}{\partial x}$ в заданной точке. В случае матриц: возвращается матрица, составленная из вектор-строк.

Аналогично **Activation Function** позволяет пользователю использовать свои собственные функции потерь. Библиотека предоставляет следующие функции потерь: **MSE**, **CrossEntropy**

5.3 Layer

Класс **Layer** - реализация полносвязного слоя нейросети. Предоставляет следующие методы:

- **Конструктор** - принимает размер входа, размер выхода и **Activation Function**. Параметры слоя (A, b) - генерируются случайно из $\mathcal{N}(0, 1)$
- **Evaluate(v)** - возвращает $\sigma(Av+b)$, есть версия для матриц: вычисляется для каждого столбца матрицы
- Список методов, которые нужны для обучения:
  - **Update** - обновляет параметры слоя
  - **BackPropagation** - шаг алгоритма BackPropagation для одного слоя
  - **ZeroGrad** - зануляет $\nabla A, \nabla b$
  - Методы для подсчета $\nabla A, \nabla b, u_k$ для алгоритма BackPropagation

5.4 LearningRate

LearningRate - это стирающий тип, предоставляющий возможность посчитать гиперпараметр во время градиентного спуска. Предоставляет следующий метод:

- Get - возвращает текущий коэфициент для градиентного спуска

5.5 Network

Главный класс библиотеки, который реализует полносвязную нейросеть и предоставляет метод для ее обучения. Содержит следующие методы и классы:

- **Конструктор** - принимает размер выхода и выхода, а также **Activation Function** для каждого слоя, создает нейросеть.
- **Train** - принимает тренировочный датасет Data (две матрицы X, y), **Loss Function**, **LearningRate**, размер batch-a, количество эпох (итераций обучения). В этом методе происходит процесс обучения модели.
- **Predict(v)** - возвращает $\mathbb{F}_\theta(v)$ - значение нейросети на v.
- Список методов, которые нужны для обучения:
  - **ForwardPass(v)** - выполняет 1 шаг (проход вперед) алгоритма BackPropagation
  - **BackwardPass(forward, u)** - принимает результат прохода вперед - forward, а также градиент функции потерь - u. Выполняет проход назад.
  - **ZeroGrad** - зануляет градиент каждого слоя (вызывает **Layer::ZeroGrad** для каждого слоя)
  - **Step** - обновляет параметры нейросети, вызывает **Layer::Update** для каждого слоя
- **Save** - сохраняет параметры модели в файл

5.6 LoadData

Данная функция находится в **namespace nn::mnist** и предоставляет удобный способ считать данные из датасета MNIST. Принимает путь до файла с картинками и ответами, возвращает структуру Data = (Matrix X, Matrix y). В случае если пользователь попросил прочитать некорректный файл или не датасет MNIST, то функция отправит соответствующее сообщение об ошибке, поэтому необходимо вызывать ее try catch() блоке.

6 Тестирование

В основном в библиотеке используются **assert**-ы для проверки корректности программы изнутри. Реализованы unit-тесты, которые проверяют корректность работы ключевых компоненты библиотеки: Activation Function, Loss Function, Layer, Network и для конкретных функций активации и потерь. Для этих классов были написаны тесты, проверяющие:

- Поведение при некорректных данных
- Корректность алгоритма

В качестве примера приведу тест для проверки на корректность вычисления $ReLu$ в заданной точке.

```
void TestReLuEvaluate() {
    nn::ReLu f;
    nn::Vector v{3};
    v << 1, 2, 3;
    assert(f.Evaluate(v) == v);
    assert(f.Evaluate(-v) == nn::Vector::Zero(3));
    std::cout << "TestReLuEvaluate Passed" << '\n';
}
```

Для проверки производительности и корректности также будет рассмотрен пример: Обучение нейросети для распознавания цифр из MNIST и будет проделано следующее тестирования:

- Сравнение разных вариаций градиентного спуска на данном примере
- В случае Mini-Batch градиентного спуска проверить скорость и качество работы в зависимости от размера batch-a
- Сравнить реализованные функции активации по скорости и качеству работы при обучении

Список литературы

[1] URL: https://en.cppreference.com/w/cpp/17.

[2] URL: https://gcc.gnu.org/onlinedocs/.

[3] URL: https://git-scm.com/doc.

[4] URL: https://cmake.org/cmake/help/latest/release/3.27.html.

[5] URL: https://releases.llvm.org/14.0.0/tools/clang/docs/ClangFormat.html.

[6] URL: https://releases.llvm.org/14.0.0/tools/clang/tools/extra/docs/clang-tidy/index.html.

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

[8] Eigenrand. URL: https://bab2min.github.io/eigenrand/v0.5.0/en/index.html.

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

[10] I. Sadrtdinov. Deep learning 1. 2023. URL: https://github.com/isadrtdinov/intro-to-dl-hse/tree/2023-2024/lecture-notes.

[11] E. Sokolov. Machine learning 1. 2023. URL: https://github.com/esokolov/ml-course-hse/tree/master/2023-fall/lecture-notes.

[12] D. Trushin. Matrix derivatives. URL: https://disk.yandex.ru/i/YynctjURriDSaQ.