Нейросети с нуля

Выполнил: Скрибченко Арсений Владиславович (БПМИ 218)
Руководитель: Трушин Дмитрий Витальевич, к.ф.-м.н., доцент, ДБДИП ФКН НИУ ВШЭ

2023

Цель и задачи

Цель - изучение устройства полносвязных нейросетей с их дальнейшей реализацией в виде библиотеки на языке C++.

Задачи:

- изучение структуры и принципа обучения полносвязных нейросетей, изучение разных видов градиентного спуска
- разработка архитектуры библиотеки, позволяющей создать и обучить полносвязную нейросеть с указанием размеров слоёв, функций активаций, функции ошибки
- программная реализация

Теоретические сведения

Пусть у нас есть неизвестное отображение G. Мы знаем значение отображения на векторах $x_1, \ldots, x_k$ и эти значения равны $y_1, \ldots, y_k$.

Наша задача - приблизить отображение G. Для этого будем использовать полносвязные нейросети.

Нейросеть представляет собой некоторое дифференцируемое отображение $F : \mathbb{R}^n \to \mathbb{R}^m$, параметризованное набором параметров θ. Обучение нейросети – это процесс изменения этих параметров с учётом некоторой поставленной задачи.

Теоретические сведения

Нейросеть строится из слоёв. Каждый слой содержит в себе следующую информацию: матрицу A, столбец b и функцию активации σ.

Функция активации $\sigma\colon \mathbb{R} \to \mathbb{R}$ – некоторое нелинейное отображение, применяемое к вектору покоординатно.

Каждый слой принимает на вход вектор x, а затем передаёт его следующему слою выполнив над ним следующие операции:

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

Теоретические сведения

Будем называть вектор $w_i = F(x_i)$ предсказанием нейросети.

Функция штрафа – это отображение $\phi : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$, характеризующее отклонение предсказания w_i от соответствующего истинного значения y_i.

Рассмотрим функцию:

$$L(\theta) = \frac{1}{k} \sum_{i=1}^{k} \phi(w_i, y_i)$$

Суть обучения нейросети заключается в минимизации функции $L(\theta)$.

Теоретические сведения

Антиградиент показывает направление наскорейшего убывания функции.

$$A_i' = A_i - \lambda \frac{\partial L}{\partial A_i}(\theta)$$

$$b_i' = b_i - \lambda \frac{\partial L}{\partial b_i}(\theta)$$

A_i и b_i - параметры i-го слоя до обновления, A_i' и b_i' - параметры i-го слоя после обновления, $\frac{\partial L}{\partial A_i}(\theta)$ и $\frac{\partial L}{\partial b_i}(\theta)$ - направления вдоль градиента для параметров A_i и b_i, λ - скорость обучения.

Градиенты вычисляются с помощью метода обратного распространения ошибки (backward propagation).

Теоретические сведения

x_l - вектор из входных данных, $y_l = G(x_l)$, m - количество слоёв

1. Вычисляем предсказание $w_l = F(x_l)$
2. $U = \frac{\partial \phi}{\partial w}(w_l, y_l)$ - информация об ошибке
3. Вычисление градиентов для последнего слоя:
   $\nabla A_m^l = \frac{\partial \phi}{\partial A_m}(\theta) = \sigma'(A_m x + b_m) U x^T$; $\nabla b_m^l = \frac{\partial \phi}{\partial b_m}(\theta) = \sigma'(A_m x + b_m) U$;
4. Передача информации об ошибке предыдущему слою:
   $U' = U^T \sigma'(A_m x + b_m) A_m$;

Шаги 3-4 повторяются вплоть до первого слоя. Повторяя вышеописанную процедуру для всех векторов $x1 \dots xk$, получаем набор градиентов для i-го слоя $\nabla A_i^1 \dots \nabla A_i^k$ и $\nabla b_i^1 \dots \nabla b_i^k$

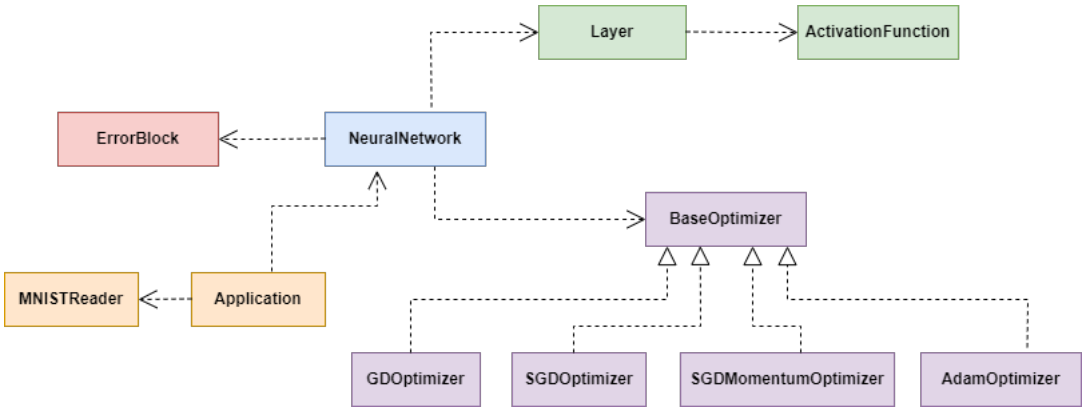
$$\frac{\partial L}{\partial A_i}(\theta) = \frac{1}{k} \sum_{j=1}^{k} \nabla A_i^j$$

$$\frac{\partial L}{\partial b_i}(\theta) = \frac{1}{k} \sum_{j=1}^{k} \nabla b_i^j$$

Теоретические сведения

- можно обновлять параметры сразу после рассмотрения одного вектора x_j: $A'_i = A_i - \lambda \frac{\partial \phi}{\partial A_i}(\theta)$; $b'_i = b_i - \lambda \frac{\partial \phi}{\partial b_i}(\theta)$
- можно обновлять параметры после рассмотрения какой-то части входных данных
- можно учитывать градиенты из предыдщих итераций с некоторым "затуханием"
- Adam

Диаграмма классов



Описание архитектуры

Все вычисления с объектами линейной алгебры производились с помощью библиотеки Eigen.

Класс ErrorBlock - блок функции ошибки, используется для вычисления градиента функции ошибки. Возможно указать:

- ErrorType::MSE - Mean Squared Error
- ErrorType::MAE - Mean Absolute Error

Класс ActivationFunction - класс для функции активации, используется внутри слоя.

- FunctionType::Sigmoid - сигмоида: $\sigma(x) = \frac{1}{1+e^{-x}}$
- FunctionType::Relu - ReLU: $\sigma(x) = max(0, x)$
- FunctionType::LeakyRelu - LeakyReLU: $\sigma(x) = max(0.01x, x)$

Описание архитектуры

Класс Layer - слой нейросети.

- хранит параметры слоя (матрица и вектор)
- функция PushForward - применение отображения слоя к входному вектору
- функция PushBackward - вычисление информации об ошибке для предыдущего слоя и вычисление градиентов

Созданная нейросеть внутри себя хранит вектор объектов класса Layer.

- обучение нейросети означает изменение параметров слоёв из данного вектора
- предсказание нейросети означает проталкивание вперёд входного вектора через эти слои

Описание архитектуры

Для реализации разных способов обновления параметров используются классы:

- GDOptimizer - полный градиентный спуск
- SGDOptimizer - стохастический градиентный спуск
- SGDMomentumOptimizer - стохастический градиентный спуск с инерцией
- AdamOptimizer - алгоритм Adam

Все они наследуют класс абстрактный класс BaseOptimizer с виртуальным методом Train. Наследники переопределяют метод в соответствии с реализуемым алгоритмом.

Схема обучения нейросети

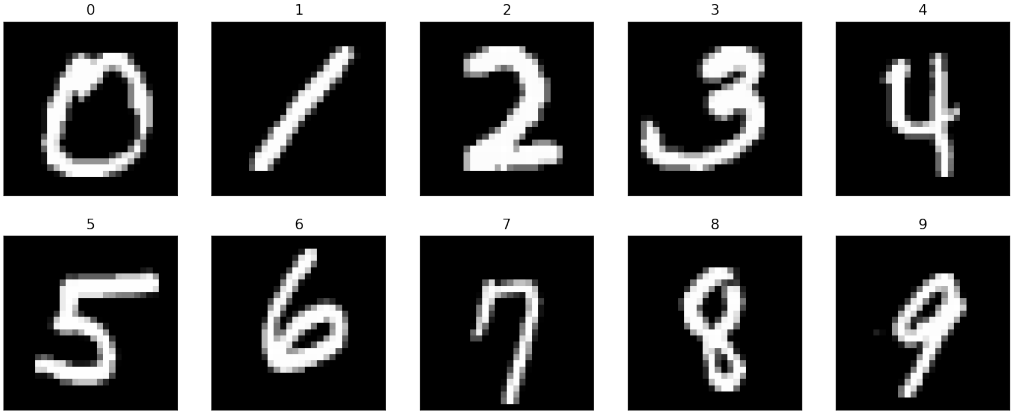
Описание архитектуры

Класс NeuralNetwork представляет непосредственно саму нейросеть. Данный класс:

- собирает воедино вышеупомянутые компоненты
- позволяет указать архитектуру нейросети
- предоставляет интерфейс для обучения (метод Train)
- предоставляет интерфейс для предсказания (функция Predict)

Пример использования

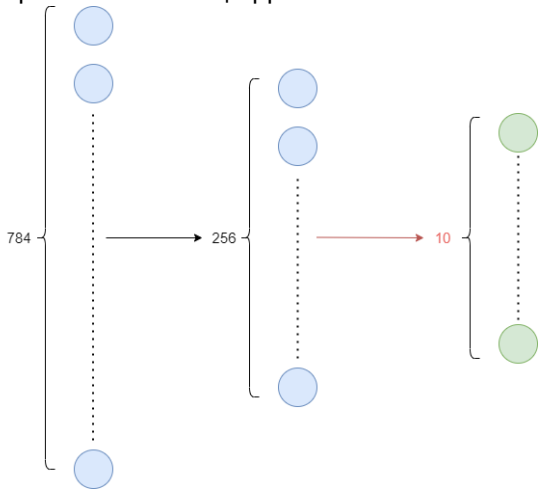
Будем рассматривать пример обучения нейросети на базе данных рукописных цифр MNIST.



Каждая картинка - набор из 28x28 пикселей от 0 до 255.

Пример использования

На вход нейросеть принимает вектор из 784 пикселей. На выходе хотим получать вектор v размера 10, такой что $v[i]$ соответствует "вероятности" того, что на картинке именно цифра i.



Пример использования

```
auto train_input = NormalizeMNISTImages(mnist_train_images_);
auto train_output = OneHotEncodeMNISTLabels(mnist_train_labels_);
NeuralNetwork network({784, 256, 10},
                      {
                          FunctionType::Relu,
                          FunctionType::Sigmoid,
                      },
                      AdamOptimizer(0.0001, 0.9, 0.999, 10e-8, 4),
                      ErrorType::MSE);
network.Train(train_input, train_output, 4000);
auto test_images_normalized = NormalizeMNISTImages(mnist_test_images_);
CalculateMNISTAccuracy(test_images_normalized,mnist_test_labels_, network);
```

В результате получена точность 93,13%.