Сравнение нейронных сетей

27 октября 2023 г.

В этом описании представлены две разные нейронные сети, каждая из которых разработана с разным уровнем сложности и с разными инструментами.

Первая нейронная сеть создана вручную, лишь с использованием материалов книги по глубокому обучению и библиотеки NumPy. Она разработана без использования какого-либо готового фреймворка, и каждый аспект сети был реализован вручную. Эта нейронная сеть представляет собой небольшую исследовательскую работу всех деталей и связей, для более глубокого понимания архитектуры нейронных сетей.

Вторая нейронная сеть создана с использованием популярного фреймворка Keras. Keras предоставляет удобные инструменты для создания и обучения нейронных сетей, упрощая процесс разработки. С помощью Keras можно быстро создавать сложные модели и экспериментировать с различными архитектурами.

Нейрнонная сеть №1

Загрузка данных и первичная обработка изображений

В этой нейронной сети используется первая тысяча изображений и меток из набора данных MNIST, которые загружается в де матрицы NumPy с именами images и labels.

Каждое изображение, нужно привести к необходимому размеру, для подачи во входной слой. Так как изображения имеют размеры 28×28 пикселов необходимо вытянуть их в вектор размерностью $1 \times 28 \cdot 28 = 1 \times 784$.

После этого, значения полученного вектора необходимо нормализовать. Изображения MNIST являются монохромными и представляют собой оттенки серого (градации серого). Каждый пиксел имеет значение от 0 до 255, где 0 обозначает черный цвет, а 255 - белый.

Нормализация путем деления на 255 выполняется с целью масштабирования значений пикселов в диапазоне от 0 до 1. Это удобно для обучения нейронных сетей, потому что нейроны имеют активационные функции, которые работают лучше, когда входные данные находятся в небольшом диапазоне. Так же, это помогает в улучшении сходимости модели.

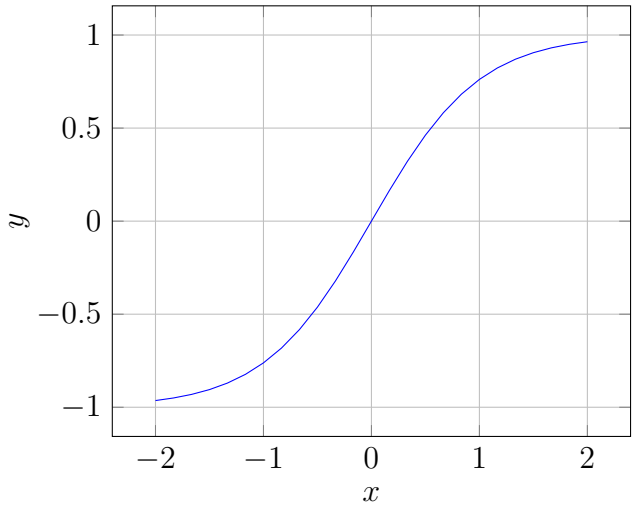
Затем идет создание векторов меток размером 1×10, которые затем, в циклах, инициализируются единицами на местах правильного значения из загруженных меток набора MNIST.

Функции активации

tanh(x) - гиперболический тангенс:

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

- Эта функция активации используется обычно для скрытых слоев нейронных сетей
- Гиперболический тангенс преобразует входные значения в диапазоне от -1 до 1.
- Функция имеет форму S-образной кривой и позволяет нейронам выражать как положительные, так и отрицательные значения.



Производная $tanh2deriv(output)$

- Эта функция вычисляет производную функции гиперболического тангенса.
- Производная $tanh2deriv$ используется при обучении нейронных сетей, чтобы определить, как изменения во входных данных влияют на изменения в выходе нейронов.

softmax(x)

$$softmax(x) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

- Эта функция активации обычно используется на выходном слое нейронной сети для многоклассовой классификации.
- $softmax$ преобразует входные значения в вероятностное распределение, где каждый элемент выходного вектора представляет вероятность принадлежности к определенному классу.

- $softmax$ полезен, когда необходимо определить вероятности для нескольких классов.

Эти функции активации важны для передачи информации и вычисления градиентов при обучении нейронных сетей. Каждая из них имеет свои особенности и применяется в соответствии с задачей и структурой сети.

Задание параметров

Далее идет задание параметров, таких как:

- размер пакета изображений
- коэффициент обучения (определяет, насколько быстрым или медленным будет обучение)
- число итераций обучения
- размер скрытого слоя.

После инициализируются случайными числами матрицы весов.

Обучение и проверка

Начало цикла обучения, который повторяется iterations раз. Внутри цикла выполняются следующие шаги:

a. Обнуление ошибки ($error$) и счетчика правильных ответов ($correct\_cnt$).

b. Разбиение обучающего набора изображений на пакеты размером im_batch_size.

c. Прямое распространение: вычисление активаций скрытого слоя ($layer_1$) с использованием гиперболического тангенса ($tanh$), применение "dropout"(случайного исключения нейронов) к $layer_1$ и вычисление выходного слоя ($layer_2$) с применением функции softmax.

d. Вычисление ошибки (сумма квадратов разницы между ожидаемыми метками labels и выходом ($layer_2$) и обновление счетчика правильных ответов.

e. Обратное распространение: вычисление градиентов для коррекции весовых коэффициентов.

f. Обновление весовых коэффициентов с использованием градиентного спуска.

g. Проверка правильности классификации на тестовом наборе данных и сохранение количества правильных ответов в $test\_correct\_cnt$.

h. Вывод информации о текущей итерации, включая тестовую и обучающую точность.

После завершения обучения итерации выводится информация о точности на тестовом и обучающем наборах данных.

Нейронная сеть №2

Пример нейронной сети, созданной с использованием библиотеки TensorFlow/Keras.

Загрузка и предварительная обработка данных:
Импортируются необходимые библиотеки, включая NumPy для работы с данными и Matplotlib для визуализации. Загружаются данные MNIST, включая обучающий и тестовый наборы, которые содержат изображения рукописных цифр и соответствующие метки.

Предобработка данных:
Изображения в наборах данных нормализуются путем деления на 255, чтобы значения пикселей оказались в диапазоне от 0 до 1. Метки (классы) преобразуются в one-hot кодировку с использованием to_categorical из Keras.

Создание модели нейронной сети:
Создается последовательная модель с использованием Keras. Модель состоит из трех слоев:

- Входной слой с формой (28, 28, 1), где 28x28 - размер изображений, 1 - количество каналов (черно-белые изображения).
- Скрытый полносвязный слой (Dense) с 128 нейронами и активацией ReLU.
- Выходной полносвязный слой с 10 нейронами (по одному для каждой цифры) и активацией softmax.

Вывод информации о модели:
Выводится сводка модели с использованием model.summary(), отображающая архитектуру и количество параметров.

Компиляция модели:
Оптимизатор "adam":
Оптимизатор - это алгоритм, который оптимизирует весовые коэффициенты нейронной сети в процессе обучения. "Adam"(Adaptive Moment Estimation) является одним из популярных методов оптимизации и эффективно работает при обучении нейронных сетей. Он адаптируется к изменяющимся скоростям обучения и моменту градиента для ускорения сходимости.

Функция потерь "categorical_ crossentropy":
Функция потерь (или функция стоимости) используется для измерения разницы между прогнозами модели и фактическими метками данных. В данном случае используется функция потерь "categorical_crossentropy"(кросс-энтропия), которая является стандартным выбором для задачи многоклассовой классификации. Она измеряет расхождение между вероятностными прогнозами и истинными метками классов.

Метрика "accuracy":
Метрика определяет, как будет оцениваться производительность модели в процессе обучения и во время оценки. В данном случае используется метрика "accuracy"(точность), которая измеряет долю правильно классифицированных примеров от общего числа примеров. Точность является часто используемой метрикой для задач классификации и позволяет оценить, насколько хорошо модель предсказывает классы. Таким образом, компиляция

модели задает ключевые настройки для обучения, определяя, каким образом производится оптимизация весовых коэффициентов, каким образом измеряется потеря и каким образом оценивается производительность модели. Эти параметры важны для успешного обучения нейронной сети и получения точных результатов.

Обучение модели:
Указаны параметры, такие как :

Размер пакета (batch_ size):
Этот параметр определяет количество образцов, которые обрабатываются одновременно в каждой итерации обучения. Вместо того, чтобы обновлять веса модели после каждого образца, модель обновляется после каждого пакета образцов. Размер пакета влияет на скорость обучения и использование ресурсов. Значение по умолчанию обычно равно 32.

Количество эпох (epochs):
Эпоха представляет собой один проход по всем обучающим данным. Количество эпох определяет, сколько раз модель будет видеть и обучаться на всем наборе данных. Увеличение числа эпох может улучшить производительность модели, но может также привести к переобучению. Вам нужно экспериментировать, чтобы найти оптимальное количество эпох для вашей задачи.

Коэффициент валидации (validation_ split): Этот параметр определяет, какая часть обучающих данных будет использоваться в качестве валидационного набора данных. Обычно его значение составляет небольшую долю от общего объема обучающих данных (например, 0.2 означает, что
20% обучающих данных будет использовано для валидации). Валидационный набор данных используется для оценки производительности модели во время обучения и проверки, помогая обнаруживать переобучение.

Оценка модели: Модель оценивается на тестовом наборе данных с использованием model.evaluate(). Выводится информация о точности классификации на тестовом наборе. Этот отчет представляет собой обзор нейронной сети для задачи распознавания цифр MNIST, включая предварительную обработку данных, создание и обучение модели, а также оценку ее производительности на тестовых данных

Сравнение

Сложность разработки:

Нейронная сеть, написанная вручную, требует более продолжительного времени и усилий для разработки. Вам придется самостоятельно создавать архитектуру сети, реализовывать методы оптимизации, функции активации и обработку данных. В то время как Keras предоставляет высокоуровневые абстракции и готовые слои, что делает процесс разработки более быстрым и простым.

Производительность:

Нейронная сеть, реализованная вручную, может дать вам полный контроль над каждым аспектом модели и позволить вам оптимизировать ее для конкретной задачи. Однако это также означает, что вам нужно уделить больше внимания деталям и гарантировать, что ваша реализация оптимизирована. В то время как Keras предоставляет удобство и быстроту разработки, но может ограничивать степень оптимизации.

В пердставленных вариантах нейросетей, победу с огромным отрывом забирает реализация с использованием фреймворка Keras.

Сравнивать точнось эих двух моделей по конечному результату тестовых замеров было бы некорректно, так как эти сети выполнили разное количество итераций, на разном количестве изображений.

Точность модели реализованной без использования фрейморка достигла 87%, а ее обучение на 1000 изображений занимало 10 минут, в то время как с использованием Keras обучение происходило на 60 000 изображений и заняло всего минуту, достигнув точности в 97%.

Итог

Использование фреймворка Keras предоставляет ряд преимуществ, таких как увеличение скорости обучения, повышение точности и упрощение разработки нейронных сетей. Однако следует помнить, что выбор между ручной реализацией и использованием фреймворка зависит от конкретной задачи, ресурсов и уровня опыта разработчика.

Литература

Грокаем глубокое обучение Книга, Траск Эндрю

Глубокое обучение на Python Книга, Шолле Франсуа

https://ru.wikipedia.org/wiki/