

Операции с тензорами

Все преобразования, выполняемые глубокими нейронными сетями при обучении, можно свести к горстке операций с тензорами, применяемых к тензорам с числовыми данными. Например, тензоры можно складывать, перемножать и т. д.

В одном из примеров со 2 занятия создавалась сеть, где было 2 Dense слоя и в коде это выглядело как «`model.add(layers.Dense(16, activation='relu'))`». Этот слой можно интерпретировать как функцию, которая принимает двумерный тензор и возвращает другой двумерный тензор — новое представление исходного тензора. В данном случае функция имеет следующий вид (где W — это двумерный тензор, a b — вектор, оба значения являются атрибутами слоя):

```
output = relu(dot(W, input) + b)
```

Давайте развернем ее. Здесь у нас имеется три операции с тензорами: скалярное произведение (`dot`) исходного тензора `input` и тензора с именем `W` ; сложение (`+`) получившегося двумерного тензора и вектора `b` ; и, наконец, операция `relu` . `relu(x)` эквивалентна операции `max(x, 0)`.

Поэлементные операции

Операция `relu` и сложение — это поэлементные операции: операции, которые применяются к каждому элементу в тензоре по отдельности. То есть эти операции поддаются массовому распараллеливанию. Для реализации поэлементных операций на Python можно использовать цикл `for`.

```
import numpy as np

def naive_relu(x):
    assert len(x.shape) == 2 #проверка размерности 2
    x = x.copy() #копирования от защиты изменения исходного тензора
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x
```

По аналогии можно реализовать сложение двух тензоров

```
def naive_add(x, y):
    assert len(x.shape) == 2
    assert x.shape == y.shape #проверка, что x и y двумерные тензоры с одинаковой
    формой
    x = x.copy() #копирования для защиты от изменения исходного тензора
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[i, j]
    return x
```

Аналогично примерам можно реализовывать скалярное умножение, деление, вычитание и т.д.

При работе с массивами NumPy можно пользоваться уже готовыми, оптимизированными реализациями этих операций, доступными в виде функций из пакета NumPy, которые сами делегируют основную работу реализациям базовых подпрограмм линейной алгебры (Basic Linear Algebra Subprograms, BLAS). BLAS — это комплект низкоуровневых, параллельных и эффективных

процедур для вычислений с тензорами, которые обычно реализуются на Fortran или C. Используя комплект этих операций реализация функций из примеров выше может быть упрощена

```
import numpy as np

z = x + y #сложение тензоров
z = np.maximum(z, 0.) #функция relu
```

Расширение

Предыдущая реализация `naive_add` поддерживает только сложение двумерных тензоров с идентичными формами. Но в слое `Dense`, представленном выше, мы складывали двумерный тензор с вектором. Рассмотрим, что происходит при сложении, когда формы складываемых тензоров различаются.

Когда это возможно и не вызывает неоднозначности, меньший тензор расширяется так, чтобы его новая форма соответствовала форме большего тензора. Расширение выполняется в два этапа:

1. В меньший тензор добавляются оси (называются осями расширения), чтобы значение его атрибута `ndim` соответствовало значению этого же атрибута большего тензора.
2. Меньший тензор копируется в эти новые оси до полного совпадения с формой большего тензора

Пусть имеются тензоры X с формой $(32, 10)$ и y с формой $(10,)$. Чтобы привести их в соответствие, сначала нужно добавить в тензор y первую пустую ось, чтобы он приобрел форму $(1, 10)$, а затем скопировать вторую ось 32 раза, чтобы в результате получился тензор Y с формой $(32, 10)$, где $Y[i, :] == y$ для i в диапазоне `range(0, 32)`. После этого можно сложить X и Y , которые имеют одинаковую форму.

В фактической реализации новый двумерный тензор, конечно же, не создается, потому что это было бы неэффективно. Операция копирования выполняется чисто виртуально: она происходит на алгоритмическом уровне, а не в памяти. Пример сложения матрицы:

```
def naive_add_matrix_and_vector(x, y):
    assert len(x.shape) == 2.
    assert len(y.shape) == 1.
    assert x.shape[1] == y.shape[0]
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[j]
    return x
```

Прием расширения в общем случае можно применять в поэлементных операциях с двумя тензорами, если один тензор имеет форму $(a, b, \dots, n, n + 1, \dots, m)$, а другой — форму $(n, n + 1, \dots, m)$. В этом случае при расширении будут добавлены оси до $n - 1$.

Скалярное произведение тензоров

Скалярное произведение, также иногда называемое тензорным произведением (не путайте с поэлементным произведением), — наиболее общая и наиболее полезная операция с тензорами. В отличие от поэлементных операций, она объединяет элементы из исходных тензоров.

Поэлементное произведение в Numpy, Keras, Theano и TensorFlow выполняется с помощью оператора `*`. Операция скалярного произведения в TensorFlow имеет иной синтаксис, но в Numpy и Keras используется простой оператор `dot`.

Для того, чтобы понять как работает скалярное произведение над тензорами, рассмотрим реализацию скалярного произведения векторов

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] == y.shape[0]
    z = 0.
    for i in range(x.shape[0]):
        z += x[i] * y[i]
    return z
```

в результате скалярного произведения двух векторов получается скаляр и в операции могут участвовать только векторы с одинаковым количеством элементов. Также есть возможность получить скалярное произведение матрицы x на вектор y , являющееся вектором, элементы которого — скалярные произведения строк x на y

```
def naive_matrix_vector_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]

    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            z[i] += x[i, j] * y[j]
    return z
```

Разумеется, скалярное произведение можно распространить на тензоры с произвольным количеством осей. Наиболее часто на практике применяется скалярное произведение двух матриц. Получить скалярное произведение двух матриц, x и y (`dot(x, y)`), можно, только если `x.shape[1] == y.shape[0]`. В результате получится матрица с формой `(x.shape[0], y.shape[1])`, элементами которой являются скалярные произведения строк x на столбцы y .

В общем случае скалярное произведение тензоров с большим числом измерений выполняется в соответствии с теми же правилами совместимости форм, как описывалось выше для случая двумерных матриц.

Например:

- $(a, b, c, d) \cdot (d,) \rightarrow (a, b, c)$
- $(a, b, c, d) \cdot (d, e) \rightarrow (a, b, c, e)$

Геометрическая интерпретация глубокого обучения

Как только что было рассмотрено, нейронные сети состоят из цепочек операций с тензорами и что все эти операции, по сути, выполняют простые геометрические преобразования исходных данных. Отсюда следует, что нейронную сеть можно интерпретировать как сложное геометрическое преобразование в многомерном пространстве, реализованное в виде последовательности простых шагов.

Иногда полезно представить следующий мысленный образ в трехмерном пространстве. Вообразите два листа цветной бумаги: один лист красного цвета и другой синего. Положите их друг на друга. Теперь скомкайте их в маленький комочек. Этот мятый бумажный комочек — ваши входные данные, а каждый лист бумаги — класс данных в задаче классификации. Суть работы нейронной сети (или любой другой модели машинного обучения) заключается в таком преобразовании комка бумаги,

чтобы разглядеть его и сделать два класса снова ясно различимыми. В глубоком обучении это реализуется как последовательность простых преобразований в трехмерном пространстве.