TF & KERAS. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: TF_KERAS_Image_04_001

ШАРИ KERAS Part 2

ШАРИ для CNN

Організація моделей машинного навчання у TensorFlow відбувається за допомогою визначення графову обчислень. Граф обчислення - це спосіб опису та організації обчислень у TensorFlow. Граф є спрямований ациклічний граф, у якому вузли представляють операції, а ребра - дані.

Кожна модель машинного навчання TensorFlow може бути представлена у вигляді графа обчислень. Граф визначає порядок виконання операцій та представляє модель у вигляді набору пов'язаних операцій, які можуть бути виконані на різних пристроях, таких як центральний процесор (CPU), графічний процесор (GPU) або спеціалізований пристрій Tensor Processing Unit (TPU).

Для створення моделі машинного навчання в TensorFlow необхідно визначити граф обчислень, який включає **шари, функції активації та оптимізатори**.
Крім того, необхідно вказати функцію втрат, яка визначає, наскільки добре модель справляється з поставленим завданням, та метрики якості, які використовуються для оцінки ефективності моделі.

Шари є основними елементами, необхідними під час створення нейронних мереж. Послідовні шари відповідають за архітектуру моделі глибокого навчання. Кожен шар виконує обчислення на основі даних, отриманих з попереднього шару. Потім інформація передається далі. Зрештою, останній шар видає потрібний результат.

Посилання:

KERAS

KERAS layers API

TensorFlow надає широкий набір типів шарів, які можна використовувати для створення нейромереж. Для створення CNN у TensorFlow доступні:

1. Згортковиі шари Conv1D, Conv2D - використовуються для обробки зображень, аудіо- та відео-сигналів, а також інших типів даних, де важлива локальна структура. Цей шар має ядро згортки, яке переміщається за вхідним зображенням або сигналом, і застосовує лінійну операцію до кожного фрагмента.
2. Шари субдискретизації MaxPooling1D та MaxPooling2D – використовуються для зменшення розмірності тензора шляхом вибору максимальних значень у вікні.
3. Шари субдискретизації AeragePooling1D та AveragePooling2D
використовуються для зменшення розмірності тензора шляхом вибору середніх значень у вікні.

Типово для визначення або створення шару Keras потрібна наступна інформація:

- Форма введення: для розуміння структури вхідної інформації
- Кількість: для визначення кількості вузлів/нейронів у шарі

- Ініціалізатор: для визначення ваги кожного входу, що важливо для виконання обчислень
- Активатори: для перетворення вхідних даних у нелінійний формат, щоб кожен нейрон міг навчатися ефективніше
- Обмежувачі: для накладання обмежень на ваги під час оптимізації
- Регулятори: для застосування штрафів до параметрів під час оптимізації

Визначення версії TF

```
import tensorflow as tf # імпорт tensofflow
import numpy as np # імпорт numpy
import pprint as pprint # імпорт пакету посиленого друку
print(tf.__version__) # версія TF
```

```
2.14.0
```

ШАР Cov2D

```
'''
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
'''
```

Шар `Conv2D` в TensorFlow використовується для створення згорткових нейронних мереж (наприклад, просторова згортка над зображеннями).

Шар `Conv2D` є згортковим шаром нейронної мережі, який застосовує фільтри до вхідних даних з певним розміром ядра згортки. Він має кілька параметрів, включаючи кількість фільтрів, розмір ядра згортки, функцію активації та розмірність вхідних даних. Залежно від конкретного завдання та архітектури нейронної мережі ці параметри можуть змінюватися.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding**: one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding with zeros evenly to the left/right or up/down of the input. When padding="same" and strides=1, the output has the same size as the input.
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch_size, height, width, channels) while channels_first corresponds to inputs with shape (batch_size, channels, height, width). If left unspecified, it uses the image_data_format value found in your Keras config file at ~/.keras/keras.json (if exists) else 'channels_last'. Note that the channels_first format is currently not supported by TensorFlow on CPU. Defaults to 'channels_last'.
- **dilation_rate**: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- **groups**: A positive integer specifying the number of groups in which the input is split along the channel axis. Each group is convolved separately with filters / groups filters. The output is the concatenation of all the groups results along the channel axis. Input channels and filters must both be divisible by groups.
- **activation**: Activation function to use. If you don't specify anything, no activation is applied (see keras.activations).
- **use_bias**: Boolean, whether the layer uses a bias vector.
  kernel_initializer: Initializer for the kernel weights matrix (see keras.initializers). Defaults to 'glorot_uniform'.
  bias_initializer: Initializer for the bias vector (see keras.initializers). Defaults to 'zeros'.
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see keras.regularizers).
- **bias_regularizer**: Regularizer function applied to the bias vector (see keras.regularizers).
  activity_regularizer: Regularizer function applied to the output of the layer (its "activation") (see keras.regularizers).
- **kernel_constraint**: Constraint function applied to the kernel matrix (see keras.constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see keras.constraints).

Використовуючи цей шар як перший шар у моделі, надають аргумент ключового слова input_shape (кортеж цілих чисел або None, не включає вісь зразка), наприклад. input_shape=(128, 128, 3) для зображень 128x128 RGB у data_format="channels_last". Можна використовувати None, якщо розмір має змінний розмір.

```
# The inputs are 28x28 RGB images with `channels_last` and the batch
# size is 4.
input_shape = (4, 28, 28, 3)
x = tf.random.normal(input_shape)
y = tf.keras.layers.Conv2D(2, 3, activation='relu', input_shape=input_shape[1:])
(x)
print(y.shape)
```

```
(4, 26, 26, 2)
```

Шар MaxPooling 2D

Шар MaxPooling (Максимальний пулінг) у пакеті TensorFlow використовується для зменшення розмірності даних шляхом вибору найбільшого значення у кожному вікні (зазвичай квадратному) даних. Цей шар часто використовується в згорткових нейронних мережах для зменшення розмірності даних після згорткових шарів.

```
'''
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs
)
'''
```

Шар двовимірного пуллінгу з використанням максимуму.

Зменшує дискретизацію вхідного сигналу вздовж його просторових розмірів (висота та ширина), беручи максимальне значення у вікні введення (розміру, визначеному параметром **pool_size**) для кожного каналу вхідного сигналу. Вікно зсувається кроками **strdes** вздовж кожного виміру.

Отриманий результат, якщо використовується **valid** параметр доповнення, має просторову форму (кількість рядків або стовпців): **output_shape = math.floor((input_shape - pool_size) / strides) + 1** (коли **input_shape >= pool_size**)

Результуюча вихідна форма при використанні **same** («того самого») параметра паддінгу: **output_shape = math.floor((input_shape - 1) / strides) + 1**

MaxPooling Приклад 1

```
x = tf.constant([[1., 2., 3., 4.],
                 [5., 6., 7., 8.],
                 [9., 10., 11., 12.]])
x
```

```
<tf.Tensor: shape=(3, 4), dtype=float32, numpy=
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]], dtype=float32)>
```

```
x = tf.reshape(x, [1, 3, 4, 1])
x
```

```
<tf.Tensor: shape=(1, 3, 4, 1), dtype=float32, numpy=
array([[[[ 1.],
         [ 2.],
         [ 3.],
         [ 4.]],

        [[ 5.],
         [ 6.],
         [ 7.],
         [ 8.]],

        [[ 9.],
         [10.],
         [11.],
         [12.]]]], dtype=float32)>
```

```
max_pool_2d = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
             strides=(2, 2), padding='valid')
max_pool_2d(x)
```

```
<tf.Tensor: shape=(1, 1, 2, 1), dtype=float32, numpy=
array([[[[6.],
         [8.]]]], dtype=float32)>
```

MaxPooling Приклад 2

```
# Створення шару MaxPooling2D
pool_layer = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))

# Визначення моделі
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),
    pool_layer,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

У прикладі створено шар MaxPooling2D з розміром пула (2, 2), який буде використовуватися для зменшення розмірності даних після згорткового шару. Потім створена модель, яка починається з шару згортки Conv2D, що використовує 32 фільтри з розміром ядра (3, 3) і функцією активації ReLU, а потім застосовано шар MaxPooling2D для зменшення розмірності даних. Потім отримані дані витягуються в одновимірний вектор за допомогою шару Flatten і модель завершується повним шаром Dense і функцією активації softmax для багатокласової класифікації.

Параметр pool_size визначає розмір вікна, що використовується для максимального пулінгу. У цьому прикладі використано вікно розміру (2, 2), що означає, що кожні 2x2 пікселя вхідних даних будуть зведені до одного пікселя вихідних даних.

Шар MaxPooling також може приймати низку інших параметрів, таких як:

- strides: кроки переміщення вікна
- padding: тип заповнення (valid або same)
- data_format: порядок розмірності даних (channels_last або channels_first)

Наприклад, ми можемо встановити параметр strides=2, щоб зменшити розмірність даних удвічі.

```
# Створення шару MaxPooling2D із stride = 2
pool_layer = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2)
```

У цьому випадку використовується вікно розміру (2, 2) для максимального пулінгу, але воно переміщається по вхідних даним з кроком 2, що призведе до зменшення розмірності даних вдвічі.

ПРИКЛАД СТВОРЕННЯ ПРОСТОЇ CNN МОДЕЛІ

```
# Створення моделі нейронної мережі
model = tf.keras.Sequential()

# Ддодавання згортокового шару Conv2D с 32 фільтрами та функцийй активації ReLU
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
28, 1)))

# Додавання шару MaxPooling2D для зменшення разміру вихідних даних
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
```

```
# Додавання шару Flatten для перетворення вихідних даних в одномірний масив (вектор)
model.add(tf.keras.layers.Flatten())

# Додавання шару Dense для классифікації зображень
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

У цьому прикладі створюється модель нейронної мережі для класифікації зображень MNIST. Перший шар - це згортковий шар Conv2D. Він має кілька параметрів:

1. Filters - кількість фільтрів, що застосовуються до вхідних даних. У цьому прикладі використовується 32 фільтри.
2. kernel_size – розмір ядра згортки. Використовуємо ядро розміром 3x3.
3. activation – функція активації, яка застосовується до виходу кожного нейрона у шарі. У прикладі використана ReLU.
4. input_shape – розмірність вхідних даних. У прикладі передаються вхідні дані розміром 28x28 та один канал (чорно-біле зображення).

Далі додається шар MaxPooling2D, який зменшує розмірність вихідних даних шляхом вибору максимального значення з певного вікна. Використовується вікно розміром 2x2.

Далі додається шар Flatten, який перетворює вихідні дані на одновимірний масив. Це необхідно для наступної класифікації зображень.

Зрештою, додається шар Dense для класифікації зображень на 10 класів за допомогою функції активації softmax.