Title: study of cudnn API

JX-Ma

2024/8/31

INTRODUCE

This week, I mainly read about the cudnnConvolutionForward() algorithm in cudnn, as well as the search for the convolution algorithm cudnnFindConvolutionForwardAlgorithm(), to understand the meanings of the parameters required for these two functions and how to implement them.

cudnnConvolutionForward()

The prototype of this function is as follows:

```
cudnnStatus_t cudnnConvolutionForward(
    cudnnHandle_t                       handle,
    const void                         *alpha,
    const cudnnTensorDescriptor_t       xDesc,
    const void                         *x,
    const cudnnFilterDescriptor_t       wDesc,
    const void                         *w,
    const cudnnConvolutionDescriptor_t  convDesc,
    cudnnConvolutionFwdAlgo_t           algo,
    void                               *workSpace,
    size_t                              workSpaceSizeInBytes,
    const void                         *beta,
    const cudnnTensorDescriptor_t       yDesc,
    void                               *y)
```

- handle: The handle is a pointer to an opaque structure that holds the context of the cuDNN library. The cuDNN library context must be created using cudnnCreate (), and the returned handle must be passed to all subsequent library function calls. The context should be destroyed using cudnnDestroy() at the end.
- alpha beta: These two parameters point to pointers to the scaling factor (in host memory), which is used to mix the calculated result with the previous value in the output layer.

- xDesc: This parameter represents the input tensor, and we need to use cudnnCreate TensorDescriptor() to create such an instance, and then use the function cudnnSetTensor4dDcript() to initialize it
- x : This parameter points to the data pointer of the GPU memory associated with the tensor descriptor xDesc.
- wdec: This parameter represents the convolutional kernel tensor, and we need to use CUDNnCreate Filter Descriptor() to create such an instance, and use CUDNnSetFilter 4dDescriber() to initialize this instance.
- w: Similar to the x function, it is used to point to the GPU memory data pointer associated with the filter descriptor wDesc
- convDesc: To initialize a convolution descriptor, we need to use CUDNnCreate ConvolutionDescriptor() to create it, and then use CUDNnSet Convolution2dDescriptor() to initialize it. This parameter contains some information about the convolution, such as padding on the input tensor, convolution stride, dilation size on the convolution kernel, and convolution mode and accuracy.
- algo :The algorithm used for convolution.
- Workspace:The data pointer to GPU memory points to the workspace required to execute the specified algorithm. If a specific algorithm does not require a workspace, the pointer can be NIL.
- WorkSpaceSizeInBytes: Specify the size of the provided workSpace in bytes.
- yDesc: Used to represent the output tensor, with the same data type as the input tensor
- y: Same function as x

cudnnFindConvolutionForwardAlgorithm()

The prototype of this function is as follows:

```
cudnnStatus_t cudnnFindConvolutionForwardAlgorithm(
    cudnnHandle_t                    handle,
    const cudnnTensorDescriptor_t      xDesc,
    const cudnnFilterDescriptor_t      wDesc,
    const cudnnConvolutionDescriptor_t convDesc,
    const cudnnTensorDescriptor_t      yDesc,
    const int                          requestedAlgoCount,
    int                               *returnedAlgoCount,
    cudnnConvolutionFwdAlgoPerf_t     *perfResults)
```

The other parameters of this function have been introduced above, but here we mainly introduce three parameters:

- requestedAlgoCount: This variable allows us to input the desired number of algorithms to try.We can usually use cudnnGetConvolutionForwardAlgorithmMaxCount() to retrieve the types of algorithms present in CUDNN.
- *returnedAlgoCount: The actual number of algorithms attempted
- *perfResults: This variable stores the results of all attempted algorithms