Structure

BNNSNDArrayDescriptor

A structure that describes the shape, stride, data type, and, optionally, the memory location of an n-dimensional array.

iOS | iPadOS | Mac Catalyst | macOS | tvOS | visionOS | watchOS

```
struct BNNSNDArrayDescriptor
```

Overview

You use a `BNNSNDArrayDescriptor` structure as the primary mechanism to pass the description of data to and from BNNS functions. The description may include a pointer to the memory location.

For example, use the following code when you're passing immutable weights to a convolution layer:

```
let weights: [Float] = [ ... ]

weights.withUnsafeBufferPointer { weightsPtr in

    let weightsDescriptor = BNNSNDArrayDescriptor(flags: BNNSNDArrayFlags(0),
                                                  layout: BNNSDataLayoutConvolutionW
                                                  size: (3, 3, 1, 1, 0, 0, 0, 0),
                                                  stride: (0, 0, 0, 0, 0, 0, 0, 0),
                                                  data: UnsafeMutableRawPointer(muta
                                                  data_type: .float,
                                                  table_data: nil,
                                                  table_data_type: .float,
                                                  data_scale: 1,
```

```
                                                data_bias: 0)

    // Create and apply convolution layer.

}
```

Setting a stride value of `0` indicates that BNNS calculates stride, without padding, for that axis. For example, the stride for both of the following n-dimensional array descriptors is the same:

```
let inputDescriptor = BNNSNDArrayDescriptor(flags: flags,
                                            layout: BNNSDataLayoutRowMajorMatrix,
                                            size: (3, 3, 0, 0, 0, 0, 0, 0),
                                            stride: (0, 0, 0, 0, 0, 0, 0, 0),
                                            data: nil,
                                            data_type: .float,
                                            table_data: nil,
                                            table_data_type: .float,
                                            data_scale: 1,
                                            data_bias: 0)

let inputDescriptor = BNNSNDArrayDescriptor(flags: flags,
                                            layout: BNNSDataLayoutRowMajorMatrix,
                                            size: (3, 3, 0, 0, 0, 0, 0, 0),
                                            stride: (1, 3, 0, 0, 0, 0, 0, 0),
                                            data: nil,
                                            data_type: .float,
                                            table_data: nil,
                                            table_data_type: .float,
                                            data_scale: 1,
                                            data_bias: 0)
```

You don't need to specify the data when, for example, you're passing that data directly to `BNNSFilterApplyBatch(_:_:_:_:_:)`. The following code creates `BNNSNDArrayDescriptor` structures for the input and output of a convolution operation. The data property of both descriptors is nil, and the input and output data are passed directly to `BNNSFilterApplyBatch(_:_:_:_:_:)`:

```
let input: [Float] = [ ... ]
var output: [Float] = [ ... ]

let inDescriptor = BNNSNDArrayDescriptor(flags: BNNSNDArrayFlags(0),
                                         layout: BNNSDataLayoutImageCHW,
```

```
                                        size: (6, 6, 1, 0, 0, 0, 0, 0),
                                        stride: (0, 0, 0, 0, 0, 0, 0, 0),
                                        data: nil,
                                        data_type: .float,
                                        table_data: nil,
                                        table_data_type: .float,
                                        data_scale: 1,
                                        data_bias: 0)

let outDescriptor = BNNSNDArrayDescriptor(flags: BNNSNDArrayFlags(0),
                                         layout: BNNSDataLayoutImageCHW,
                                         size: (4, 4, 1, 0, 0, 0, 0, 0),
                                         stride: (0, 0, 0, 0, 0, 0, 0, 0),
                                         data: nil,
                                         data_type: .float,
                                         table_data: nil,
                                         table_data_type: .float,
                                         data_scale: 1,
                                         data_bias: 0)

var parameters = BNNSLayerParametersConvolution(i_desc: inDescriptor,
                                                w_desc: weightsDescriptor,
                                                o_desc: outDescriptor,
                                                bias: biasDescriptor,
                                                activation: .identity,
                                                x_stride: 1, y_stride: 1,
                                                x_dilation_stride: 0, y_dilation_str
                                                x_padding: 0, y_padding: 0,
                                                groups: 1,
                                                pad: (0, 0, 0, 0))

// `convolutionLayer` is a `BNNSFilter` created by `BNNSFilterCreateLayerConvolutior

let error = BNNSFilterApplyBatch(convolutionLayer, 1,
                                 input, inStride,
                                 &output, outStride)
```

Topics

Creating an Array Descriptor

`init(flags: BNNSNDArrayFlags, layout: BNNSDataLayout, size: (Int, Int, Int, Int, Int, Int, Int, Int), stride: (Int, Int, Int, Int, Int, Int, Int, Int), data: UnsafeMutableRawPointer?, data_type: BNNSDataType, table_data: UnsafeMutableRawPointer?, table_data_type: BNNSDataType, data_scale: Float, data_bias: Float)`

Returns a new n-dimensional array descriptor with the specified parameters.

`init?(data: UnsafeMutableRawBufferPointer, scalarType: any BNNSScalar.Type, shape: BNNS.Shape)`

Returns a new n-dimensional array descriptor that references the same data as the specified raw pointer.

`init?<T>(data: UnsafeMutableBufferPointer<T>, shape: BNNS.Shape)`

Returns a new n-dimensional array descriptor that references the same data as the specified pointer.

`init(dataType: BNNSDataType, shape: BNNS.Shape)`

Returns a new n-dimensional array descriptor from the specified data type and shape.

`init()`

Returns a new n-dimensional array descriptor.

Specifying the Behavior of an N-Dimensional Array.

`struct BNNSNDArrayFlags`

Options that control the behavior of an n-dimensional array.

Accessing the Properties of an Array Descriptor

`var flags: BNNSNDArrayFlags`

Flags that control some behaviors of the n-dimensional array.

`var layout: BNNSDataLayout`

The dimension of the n-dimensional array.

`var size: (Int, Int, Int, Int, Int, Int, Int, Int)`

The number of values in each dimension.

`var stride: (Int, Int, Int, Int, Int, Int, Int, Int)`

The increment, in values, between consecutive elements in each dimension.

`var data: UnsafeMutableRawPointer?`

A pointer that is optional and points to the underlying data.

`var data_type: BNNSDataType`

The data type of the n-dimensional array.

`var table_data: UnsafeMutableRawPointer?`

The lookup table for indexed data types.

`var table_data_type: BNNSDataType`

The data type of the lookup table.

`var data_scale: Float`

The scale you use to convert integer and unsigned integer data to floating point.

`var data_bias: Float`

The bias you use to convert integer and unsigned integer data to floating point.

`var shape: BNNS.Shape`

The shape of the n-dimensional array.

Allocating and Deallocating Memory

`static func allocate<C>(initializingFrom: C, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor`

Returns a new n-dimensional array descriptor that's initialized with a copy of the elements in the specified collection.

`static func allocate<Scalar>(randomUniformUsing: BNNS.RandomGenerator, range: ClosedRange<Scalar>, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor?`

Returns a new array descriptor that's initialized with random integer values from the continuous uniform distribution.

`static func allocate<Scalar>(randomUniformUsing: BNNS.RandomGenerator, range: ClosedRange<Scalar>, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor?`

Returns a new array descriptor that's initialized with random floating-point values from the continuous uniform distribution.

`static func allocate<Scalar>(randomIn: ClosedRange<Scalar>, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor`

Returns a new n-dimensional array descriptor that's initialized with random values within the specified range.

```
static func allocate<Scalar>(randomIn: ClosedRange<Scalar>, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor
```

Returns a new n-dimensional array descriptor that's initialized with random values within the specified range.

```
static func allocate<Scalar, Generator>(randomIn: ClosedRange<Scalar>, using: inout Generator, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor
```

Returns a new array descriptor that's initialized with random values within the specified range, using the given generator as a source for randomness.

```
static func allocate<Scalar, Generator>(randomIn: ClosedRange<Scalar>, using: inout Generator, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor
```

Returns a new array descriptor that's initialized with random values within the specified range, using the given generator as a source for randomness.

```
static func allocate<T>(repeating: T, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor
```

Returns a new n-dimensional array descriptor that's initialized with a single, repeated scalar value.

```
static func allocateUninitialized(scalarType: any BNNSScalar.Type, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor
```

Returns a new n-dimensional array descriptor that's allocated with uninitialized memory.

```
func deallocate()
```

Deallocates the memory block previously allocated to this n-dimensional array descriptor.

Generating an Array from an Array Descriptor's Data

```
func makeArray<T>(of: T.Type, batchSize: Int) -> [T]?
```

Returns a new array that contains a copy of the n-dimensional array descriptor's data.

Initializers

```
init?(data: UnsafeMutableRawBufferPointer, scalarType: any BNNSScalar.Type, shape: BNNS.Shape, batchSize: Int)
```

```
init?<T>(data: UnsafeMutableBufferPointer<T>, shape: BNNS.Shape, batchSize: Int)
```

Instance Properties

```
var dataSize: Int
```

Type Methods

```
static func allocate<Scalar>(randomNormalUsing: BNNS.RandomGenerator, mean: Scalar, standardDeviation: Scalar, shape: BNNS.Shape, batchSize: Int) -> BNNSNDArrayDescriptor?
```

Relationships

Conforms To

`BitwiseCopyable`

See Also

N-dimensional array descriptor essentials

~~`struct BNNSLayerData`~~

A structure containing common layer parameters.

Deprecated

`enum Shape`

Constants that describe the size and data layout of an n-dimensional array descriptor.

`struct BNNSDataLayout`

Constants that describe the data type of an n-dimensional array.

`struct BNNSDataType`

BNNS Data Types.

```
func BNNSDataLayoutGetRank(BNNSDataLayout) -> Int
```