

---

---

# Machine Learning

# Pytorch Tutorial 2

Documentation and Common Errors

---

TA: 石旻翰

2023.02.20

[mlta-2023-spring@googlegroups.com](mailto:mlta-2023-spring@googlegroups.com)

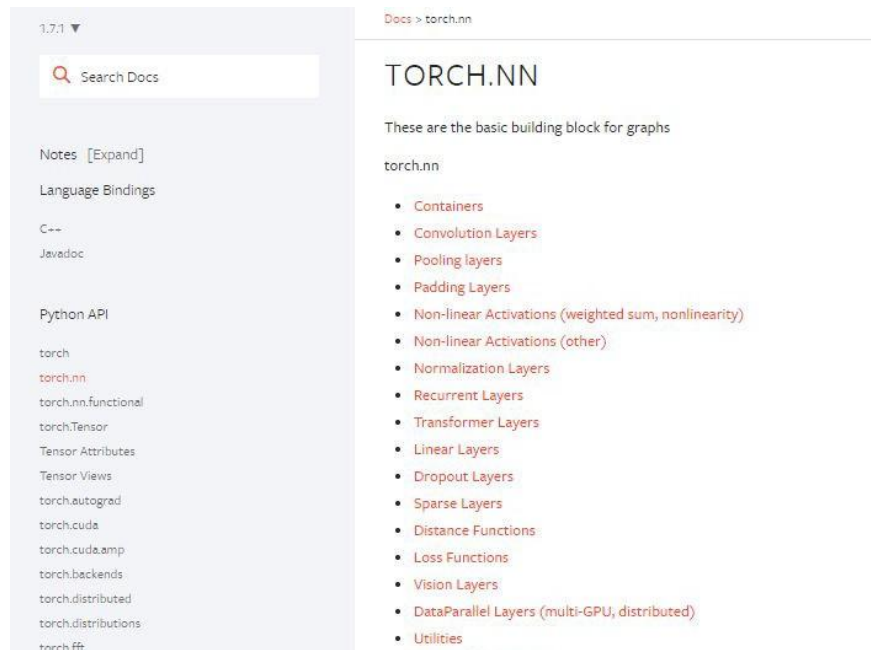
---

---

# PyTorch Documentation

<https://pytorch.org/docs/stable/>

- torch.nn -> Neural Network
- torch.optim -> Optimization Algorithms
- torch.utils.data -> Dataset, Dataloader



The screenshot displays the PyTorch documentation interface for version 1.7.1. On the left, a sidebar lists various documentation sections under the heading 'Python API', with 'torch.nn' highlighted. The main content area shows the 'TORCH.NN' section, which is described as 'These are the basic building block for graphs'. Below this, a list of sub-sections is provided, including Containers, Convolution Layers, Pooling layers, Padding Layers, Non-linear Activations (weighted sum, nonlinearity), Non-linear Activations (other), Normalization Layers, Recurrent Layers, Transformer Layers, Linear Layers, Dropout Layers, Sparse Layers, Distance Functions, Loss Functions, Vision Layers, DataParallel Layers (multi-GPU, distributed), and Utilities.

1.7.1 ▾

Search Docs

Notes [Expand]

Language Bindings

C++

Javadoc

Python API

torch

**torch.nn**

torch.nn.functional

torch.Tensor

Tensor Attributes

Tensor Views

torch.autograd

torch.cuda

torch.cuda.amp

torch.backends

torch.distributed

torch.distributions

torch.fft

Docs > torch.nn

## TORCH.NN

These are the basic building block for graphs

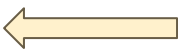
torch.nn

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities

# PyTorch Documentation Example

## TORCH.MAX

Function inputs and outputs



```
torch.max(input) → Tensor
```

Returns the maximum value of all elements in the `input` tensor.

### • WARNING

This function produces deterministic (sub)gradients unlike `max(dim=0)`

Data type and explanation of each input



Parameters

**input** (*Tensor*) – the input tensor.

# PyTorch Documentation Example

- Some functions behave differently with different inputs
- Parameters : You don't need to specify the name of the argument (Positional Arguments)
- Keyword Arguments : You have to specify the name of the argument

*They are separated by \**

```
torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)
```

Returns a namedtuple (values, indices) where values is the maximum value of each row of the input tensor in the given dimension dim. And indices is the index location of each maximum value found (argmax).

If keepdim is True, the output tensors are of the same size as input except in the dimension dim where they are of size 1. Otherwise, dim is squeezed (see torch.squeeze()), resulting in the output tensors having 1 fewer dimension than input.

#### • NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

#### Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has dim retained or not. Default: False.

#### Keyword Arguments

**out** (*tuple, optional*) – the result tuple of two output tensors (max, max\_indices)

# PyTorch Documentation Example

- Some functions behave differently with different inputs
- Arguments with default value : Some arguments have a default value (keepdim=False), so passing a value of this argument is optional

```
torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)
```

Returns a namedtuple (values, indices) where values is the maximum value of each row of the input tensor in the given dimension dim. And indices is the index location of each maximum value found (argmax).

If keepdim is True, the output tensors are of the same size as input except in the dimension dim where they are of size 1. Otherwise, dim is squeezed (see torch.squeeze()), resulting in the output tensors having 1 fewer dimension than input.

#### • NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

#### Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has dim retained or not. Default: False.

#### Keyword Arguments

**out** (*tuple, optional*) – the result tuple of two output tensors (max, max\_indices)

# PyTorch Documentation Example

## Three Kinds of torch.max

1. `torch.max(input) → Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None) → Tensor`

`input : Tensor, dim : int, keepdim : bool`  
`other : Tensor`

# PyTorch Documentation Example

1. `torch.max(input)` → **Tensor**

Find the maximum value of a tensor, and return that value.

```
input
```

```
[[1  2  3]
```

```
 [5  6  4]]
```

## PyTorch Documentation Example

```
2. torch.max(input, dim, keepdim=False, *,  
out=None) → (Tensor, LongTensor)
```

Find the maximum value of a tensor along a dimension, and return that value, along with the index corresponding to that value.

input

|     |   |     |
|-----|---|-----|
| [[1 | 2 | 7]  |
| [5  | 6 | 4]] |



# PyTorch Documentation Example

3. `torch.max(input, other) → Tensor`

Perform element-wise comparison between two tensors of the same size, and select the maximum of the two to construct a tensor with the same size.

input

|                  |                |                  |                  |                |                  |
|------------------|----------------|------------------|------------------|----------------|------------------|
| <code>[[1</code> | <code>2</code> | <code>3]</code>  | <code>[[2</code> | <code>4</code> | <code>6]</code>  |
| <code>[5</code>  | <code>6</code> | <code>4]]</code> | <code>[1</code>  | <code>3</code> | <code>5]]</code> |

# Common Errors - torch.max (Colab)

## Three Kinds of torch.max

1. `torch.max(input)`  
→ Tensor
2. `torch.max(input, dim, keepdim=False, *, out=None)` →  
(Tensor, LongTensor)
3. `torch.max(input, other, *, out=None)` → Tensor

`input` : Tensor

`dim` : int

`keepdim` : bool

`other` : Tensor

## Colab code

```
x = torch.randn(4,5)
```

```
y = torch.randn(4,5)
```

```
m, idx = torch.max(x,0,False,p)→x
```

**\*out is a keyword argument**

```
m, idx = torch.max(x,True)→x
```

**\*did not specify dim**

# Common Errors – Tensor on Different Device to Model

```
model = torch.nn.Linear(5,1).to("cuda:0")
```

```
x = torch.randn(5).to("cpu")
```

```
y = model(x)
```

Tensor for \* is on CPU, but expected them to be on GPU

=> send the tensor to GPU

```
x = torch.randn(5).to("cuda:0")
```

```
y = model(x)
```

```
print(y.shape)
```

# Common Errors - Mismatched Dimensions

```
x = torch.randn(4, 5)
```

```
y = torch.randn(5, 4)
```

```
z = x + y
```

The size of tensor a (5) must match the size of tensor b (4) at non-singleton dimension 1

=> the shape of a tensor is incorrect, use **transpose**, **squeeze**, **unsqueeze** to align the dimensions

```
y = y.transpose(0, 1)
```

```
z = x + y
```

```
print(z.shape)
```

# Common Errors - Cuda Out of Memory

```
import torch
import torchvision.models as models
resnet18 = models.resnet18().to( "cuda:0" ) # Neural Networks for Image
data = torch.randn( 512, 3, 244, 244) # Create fake data (512
out = resnet18(data.to( "cuda:0" )) # Use Data as Input and Feed to
print(out.shape)
```

**CUDA out of memory. Tried to allocate 350.00 MiB (GPU 0; 14.76 GiB total capacity; 11.94 GiB already allocated; 123.75 MiB free; 13.71 GiB reserved in total by PyTorch)**

=> The batch size of data is too large to fit in the GPU. Reduce the batch size.

# Common Errors - Mismatched Tensor Type

```
import torch.nn as nn
L = nn.CrossEntropyLoss()
outs = torch.randn(5, 5)
labels = torch.Tensor([1, 2, 3, 4, 0])
lossval = L(outs, labels) # Calculate CrossEntropyLoss between outs and labels
```

**expected scalar type Long but found Float**

=> labels must be long tensors, cast it to type “Long” to fix this issue

```
labels = labels.long()
lossval = L(outs, labels)
print(lossval)
```

**Any Question?**