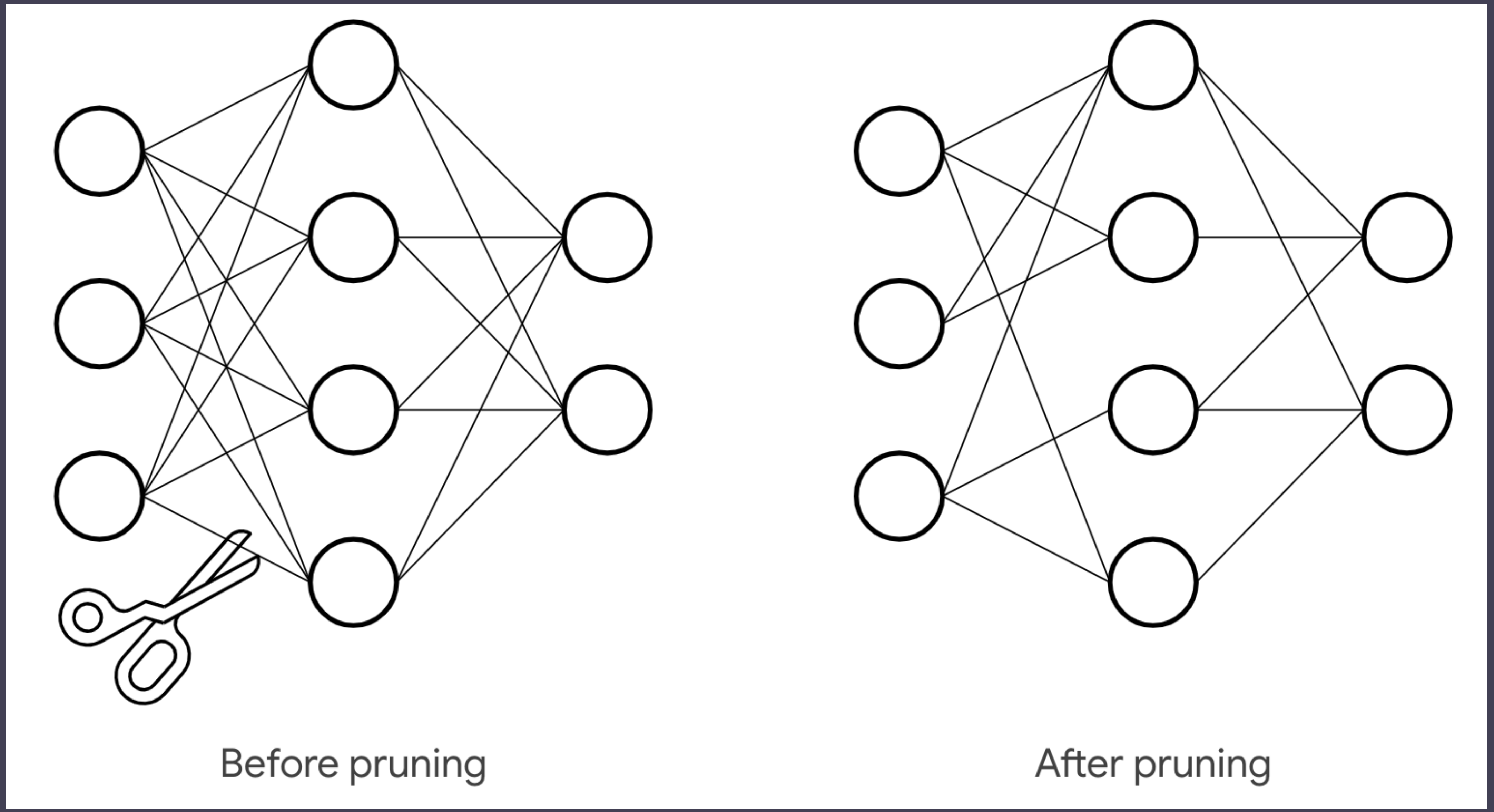
Streamlining Tensor and Network Pruning in PyTorch

Workshop on Practical ML for Developing Countries, ICLR 2020

Michela Paganini (Facebook AI Research), Jessica Forde (Brown University)

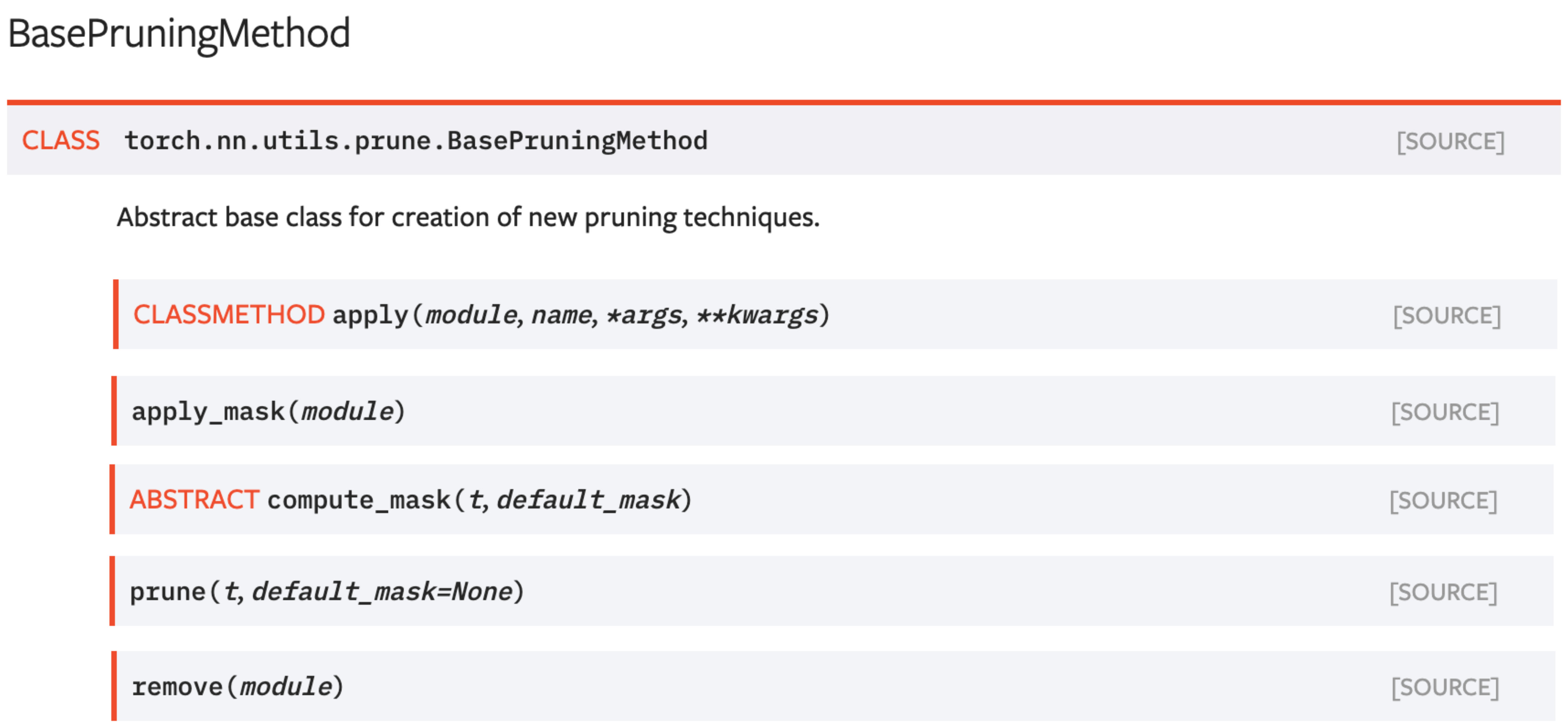
What is pruning?

- Pruning methods selectively set weights of a neural network to zero, sparsifying the model
- Pruned models can maintain the accuracy of the original model and gain computational efficiency for on-device use
- Methods remove weights based on different heuristics, such as their absolute value
- Weights can also be removed in a structured way, zero-ing out an entire channel, or in a unstructured manner
- In PyTorch, pruning is performed through the application of a mask onto the parameter



torch.nn.utils.prune

Different tensor pruning techniques enabled under a unified framework



BasePruningMethod

CLASS torch.nn.utils.prune.BasePruningMethod [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD apply(*module, name, *args, **kwargs*) [SOURCE]

apply_mask(*module*) [SOURCE]

ABSTRACT compute_mask(*t, default_mask*) [SOURCE]

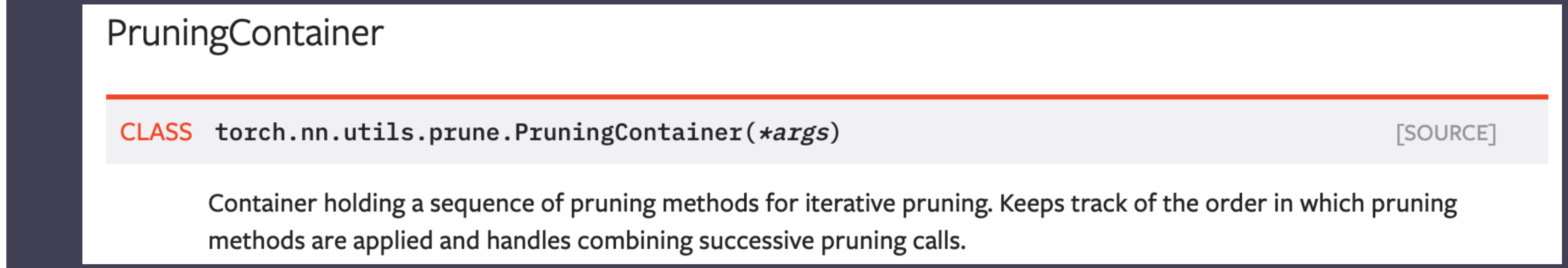
prune(*t, default_mask=None*) [SOURCE]

remove(*module*) [SOURCE]

New pruning technique?

Just subclass `BasePruningMethod` and implement `compute_mask`!

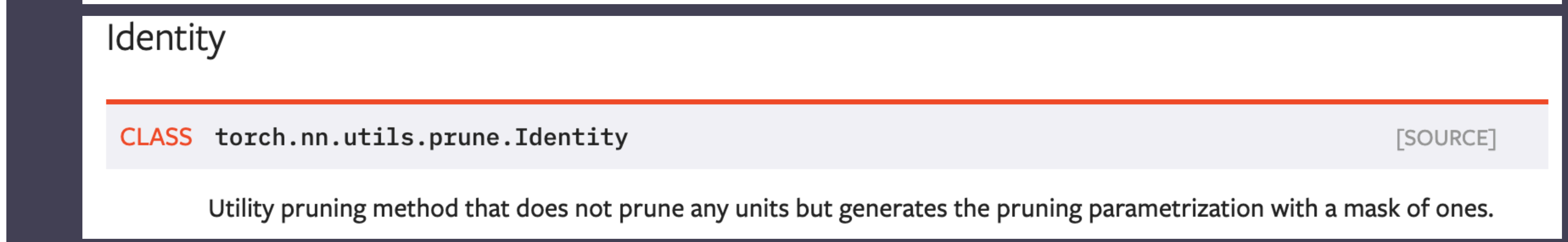
FACEBOOK AI



PruningContainer

CLASS torch.nn.utils.prune.PruningContainer(**args*) [SOURCE]

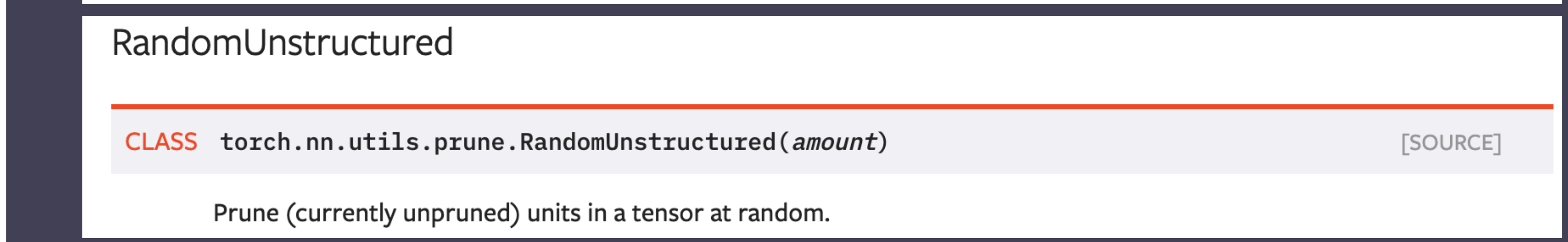
Container holding a sequence of pruning methods for iterative pruning. Keeps track of the order in which pruning methods are applied and handles combining successive pruning calls.



Identity

CLASS torch.nn.utils.prune.Identity [SOURCE]

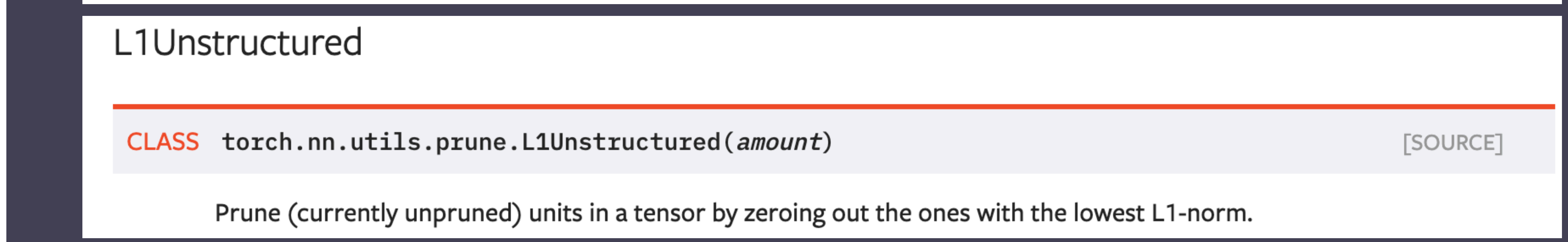
Utility pruning method that does not prune any units but generates the pruning parametrization with a mask of ones.



RandomUnstructured

CLASS torch.nn.utils.prune.RandomUnstructured(*amount*) [SOURCE]

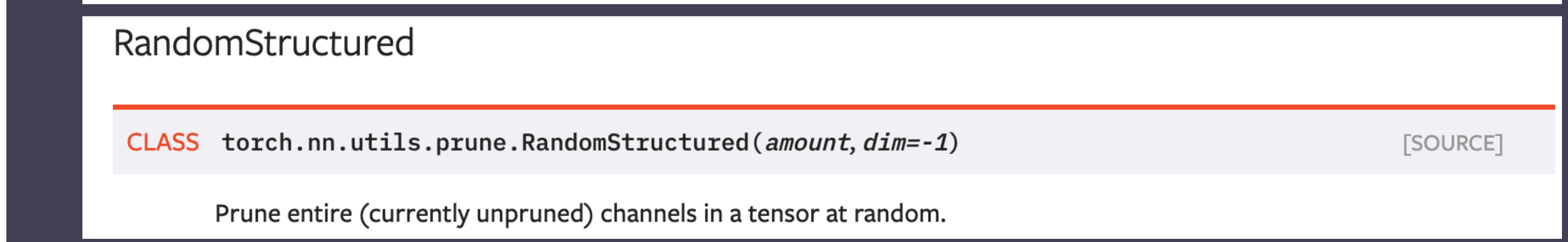
Prune (currently unpruned) units in a tensor at random.



L1Unstructured

CLASS torch.nn.utils.prune.L1Unstructured(*amount*) [SOURCE]

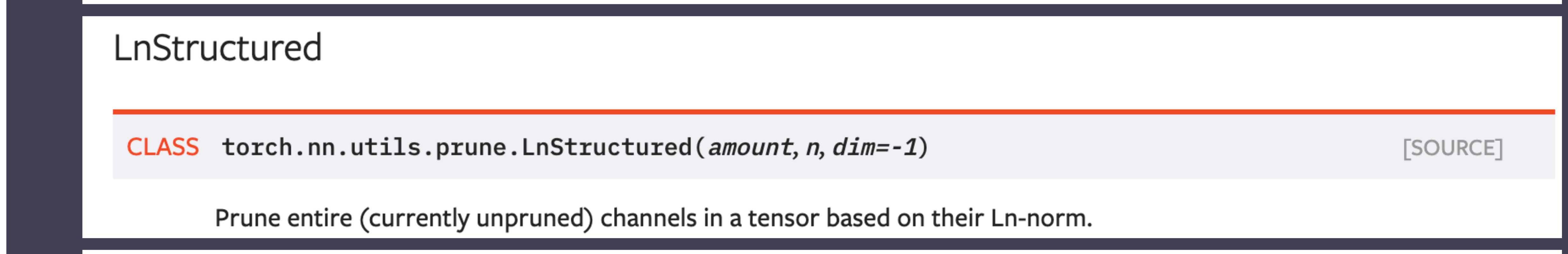
Prune (currently unpruned) units in a tensor by zeroing out the ones with the lowest L1-norm.



RandomStructured

CLASS torch.nn.utils.prune.RandomStructured(*amount, dim=-1*) [SOURCE]

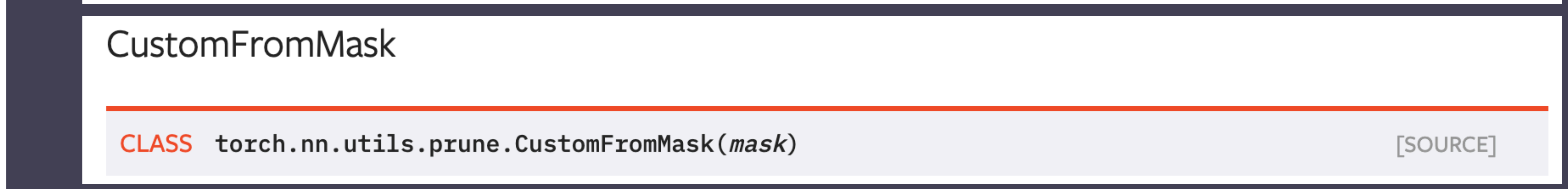
Prune entire (currently unpruned) channels in a tensor at random.



LnStructured

CLASS torch.nn.utils.prune.LnStructured(*amount, n, dim=-1*) [SOURCE]

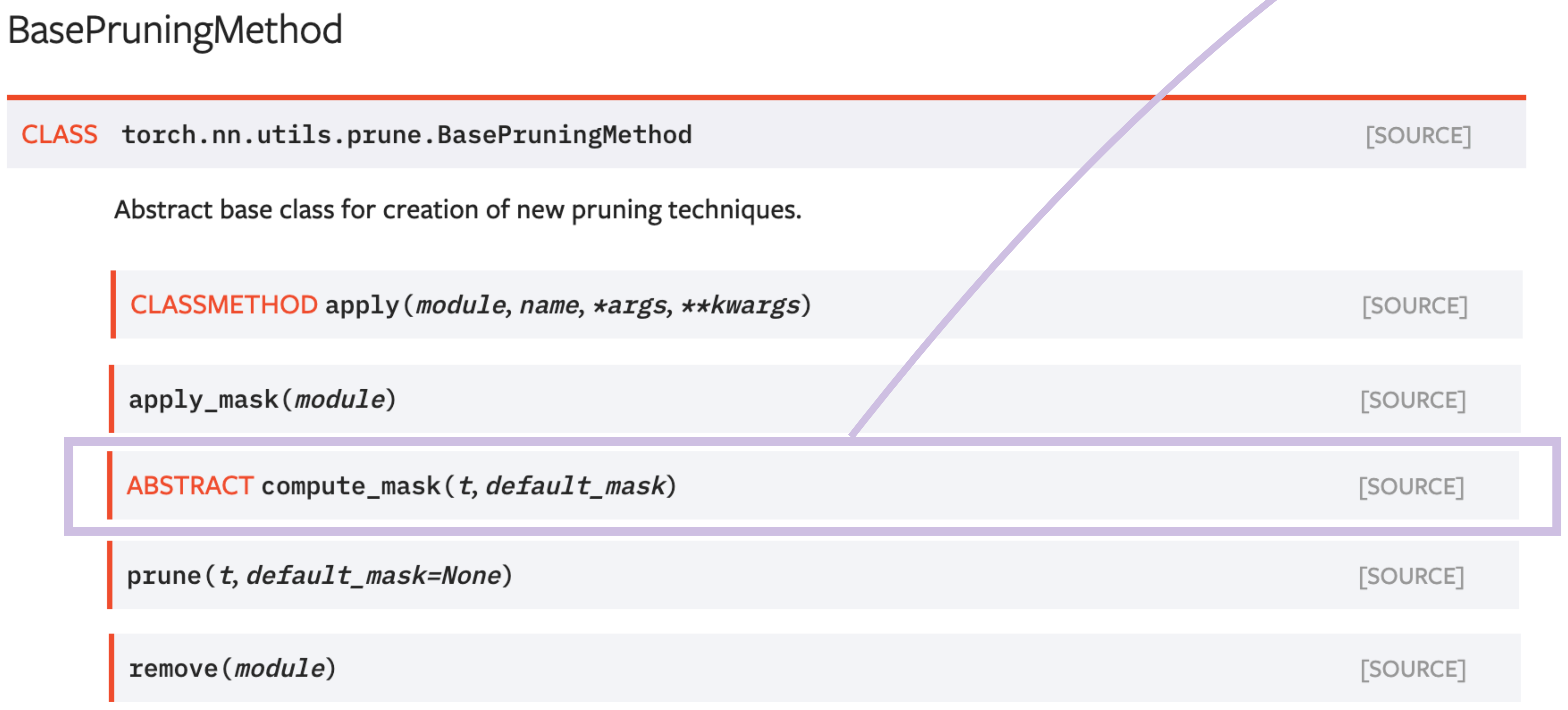
Prune entire (currently unpruned) channels in a tensor based on their Ln-norm.



CustomFromMask

CLASS torch.nn.utils.prune.CustomFromMask(*mask*) [SOURCE]

torch.nn.utils.prune



BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]

`apply_mask(module)` [SOURCE]

ABSTRACT `compute_mask(t, default_mask)` [SOURCE]

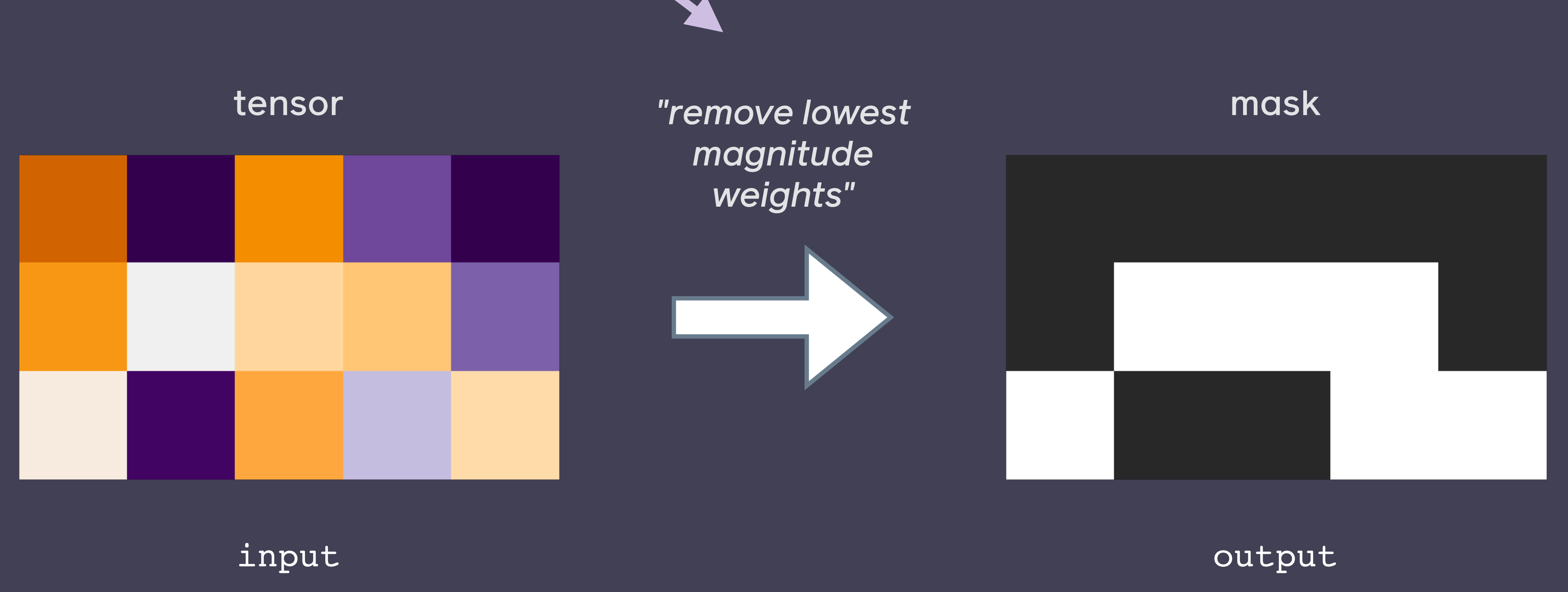
`prune(t, default_mask=None)` [SOURCE]

`remove(module)` [SOURCE]

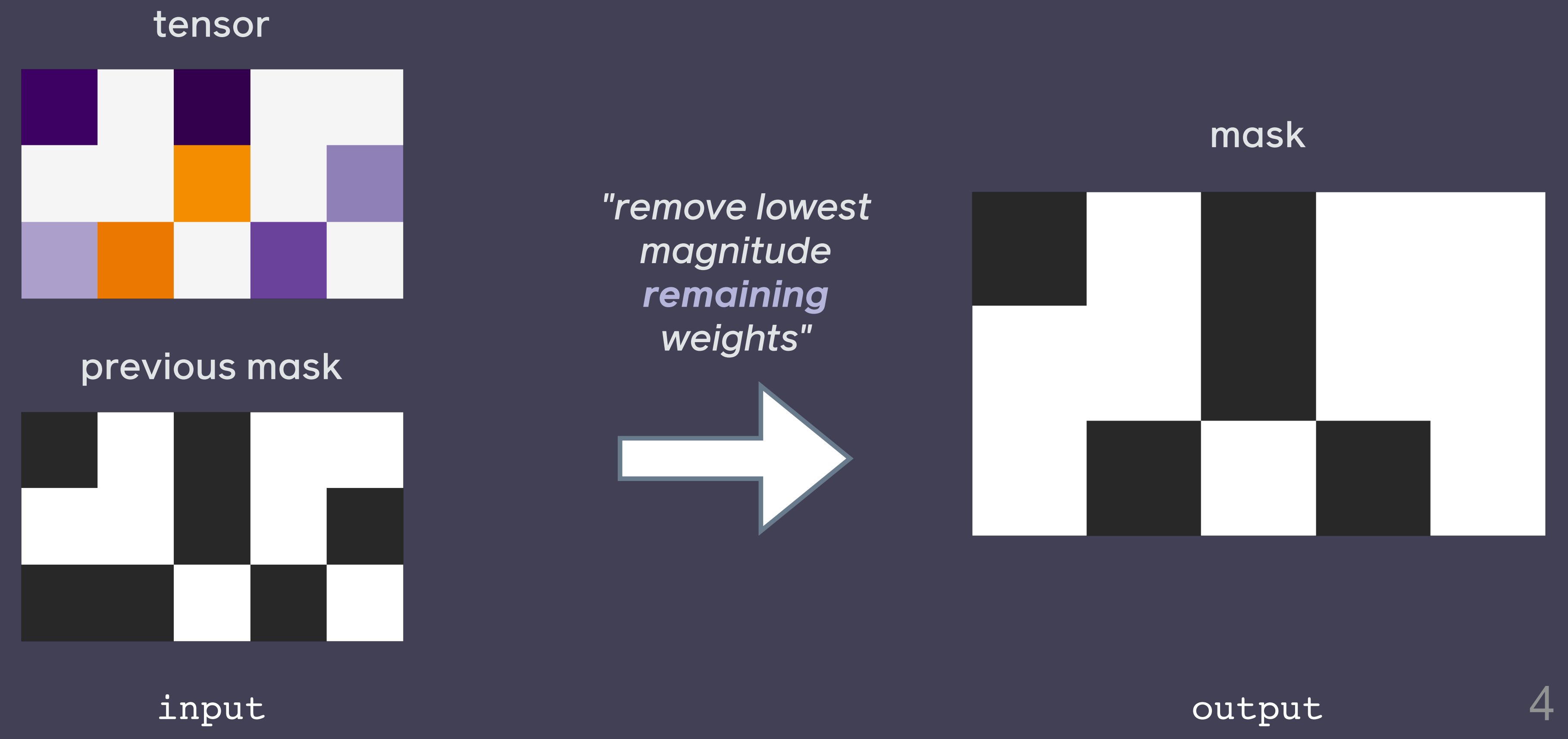
defines the interface → concrete subclasses must implement the logic

For example, in `prune.L1Unstructured`:

implements the logic that defines which portions of the tensors will be zeroed out while accounting for previously pruned entries



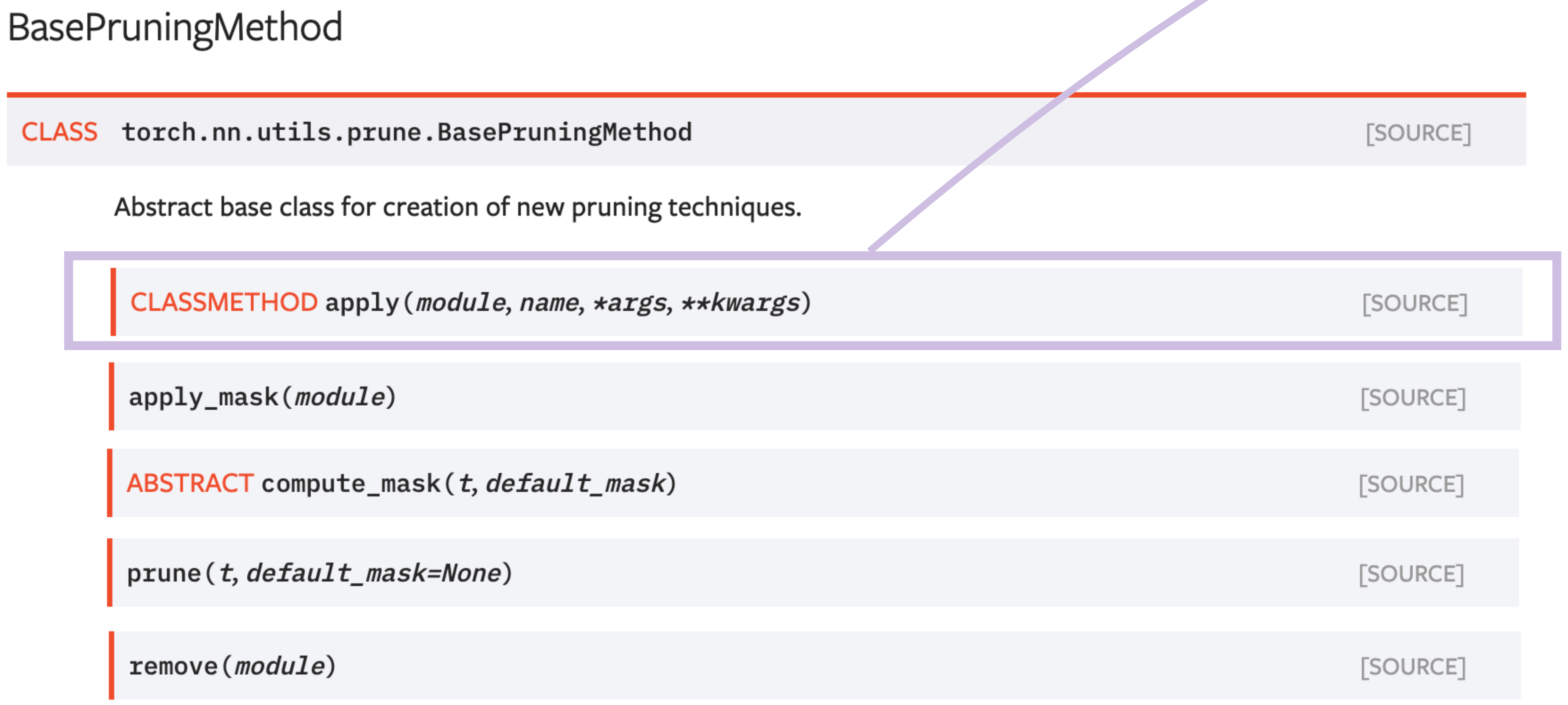
(through a `prune.PruningContainer`) it handles the case in which the tensor had previously been pruned by computing the valid entries in the tensor that can still be pruned and then applying the new pruning technique exclusively on those entries



PruningContainer

PruningContainer()
[
SomePruningMethod()
,
AnotherPruningMethod()
,
FinalPruningMethod()
]
compute_mask(t)
compute_mask(t[slice])
compute_mask(t[slice][slice])
masks[0]
masks[1]
masks[2]
{cumulative_mask}
{cumulative_mask}
{cumulative_mask}

torch.nn.utils.prune



BasePruningMethod

CLASS torch.nn.utils.prune.BasePruningMethod [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD apply(*module, name, *args, **kwargs*) [SOURCE]

apply_mask(*module*) [SOURCE]

ABSTRACT compute_mask(*t, default_mask*) [SOURCE]

prune(*t, default_mask=None*) [SOURCE]

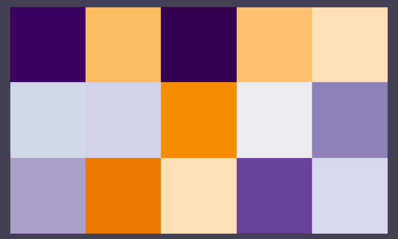
remove(*module*) [SOURCE]

Reparametrizes of the pruned tensor in terms of the original tensor and the pruning mask, and adds a forward pre-hook to enable pruning on the fly.

1. if there is any previous mask applied to this parameter
   I. fetch the previous mask
   II. combine successive pruning calls into a `prune.PruningContainer`
2. move the unpruned parameter to "`<param_name>_orig`"
3. compute new mask via `compute_mask`
4. add mask as a buffer named "`<param_name>_mask`"
5. attach the pruned version of the tensor as an attribute
6. register the pruning technique as a forward pre-hook

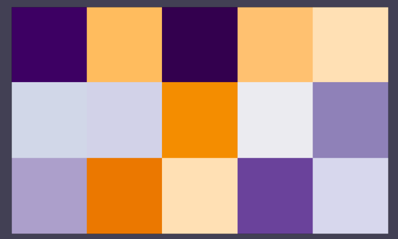
Before pruning

"`weight`" is an unpruned parameter

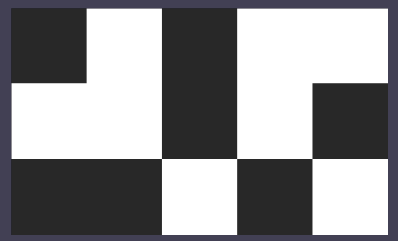


During `apply`

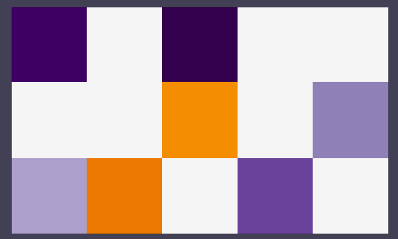
the unpruned parameter is moved to "`weight_orig`"



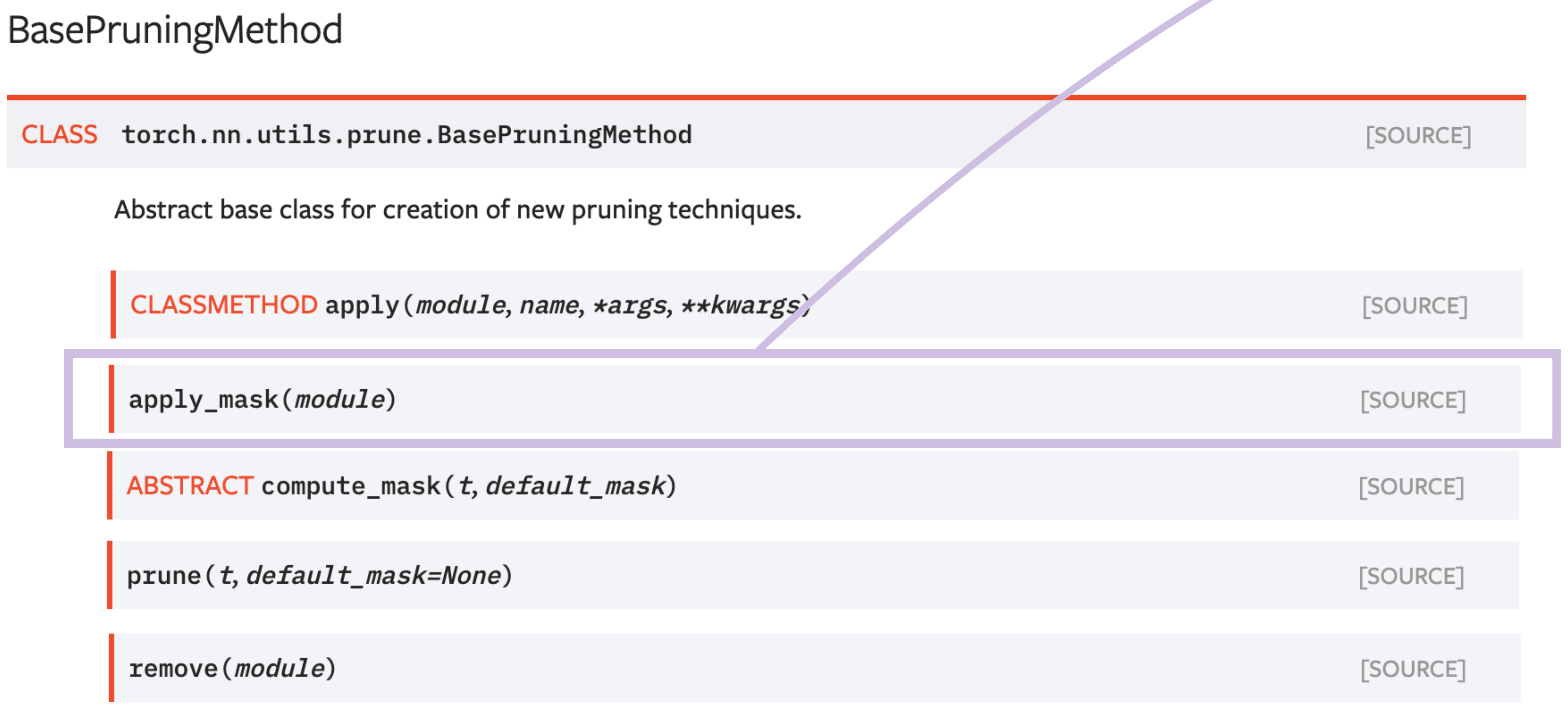
the mask is saved to a buffer called "`weight_mask`"



the pruned tensor is stored as an attribute called "`weight`"



torch.nn.utils.prune



BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]

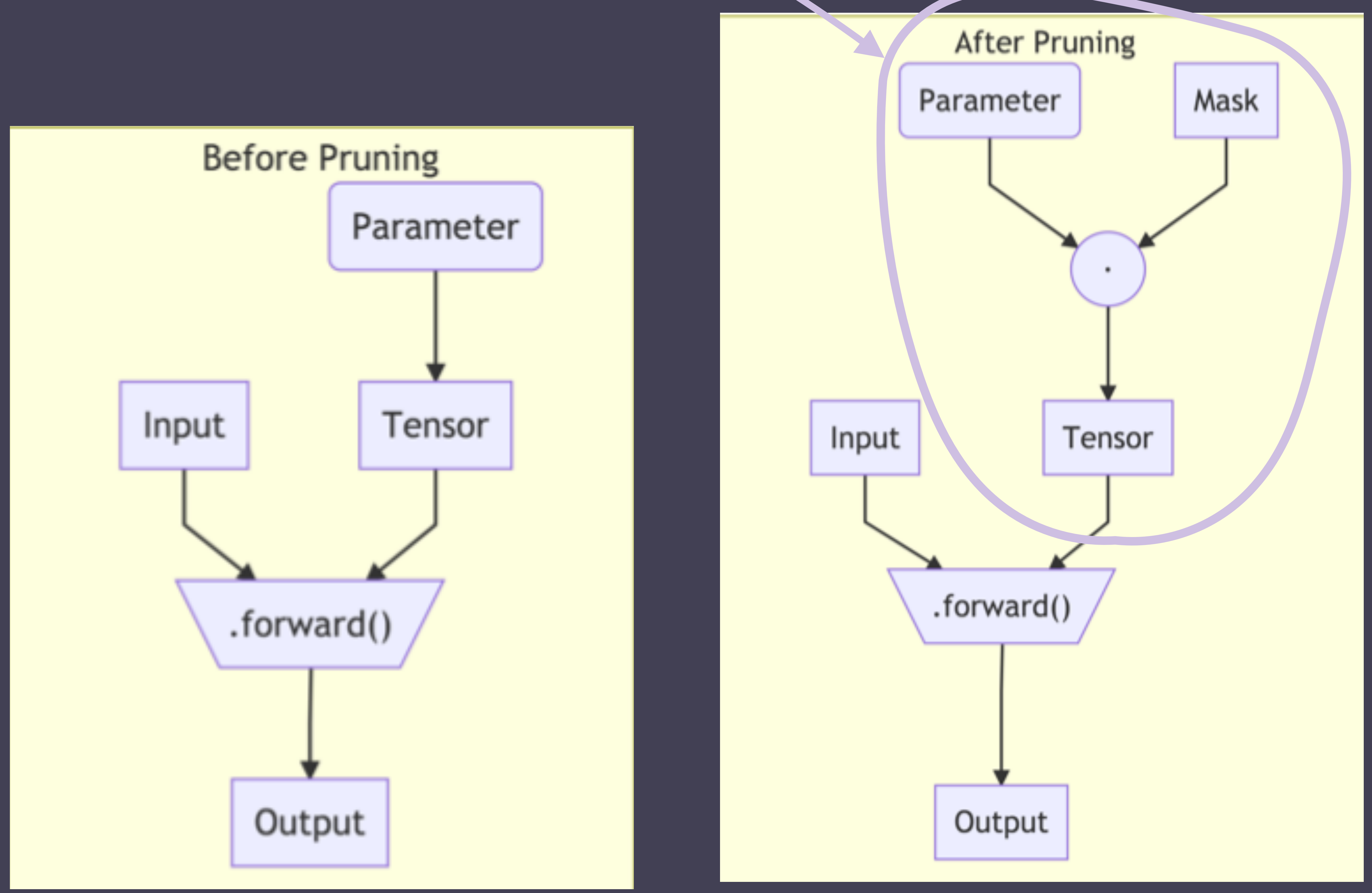
`apply_mask(module)` [SOURCE]

ABSTRACT `compute_mask(t, default_mask)` [SOURCE]

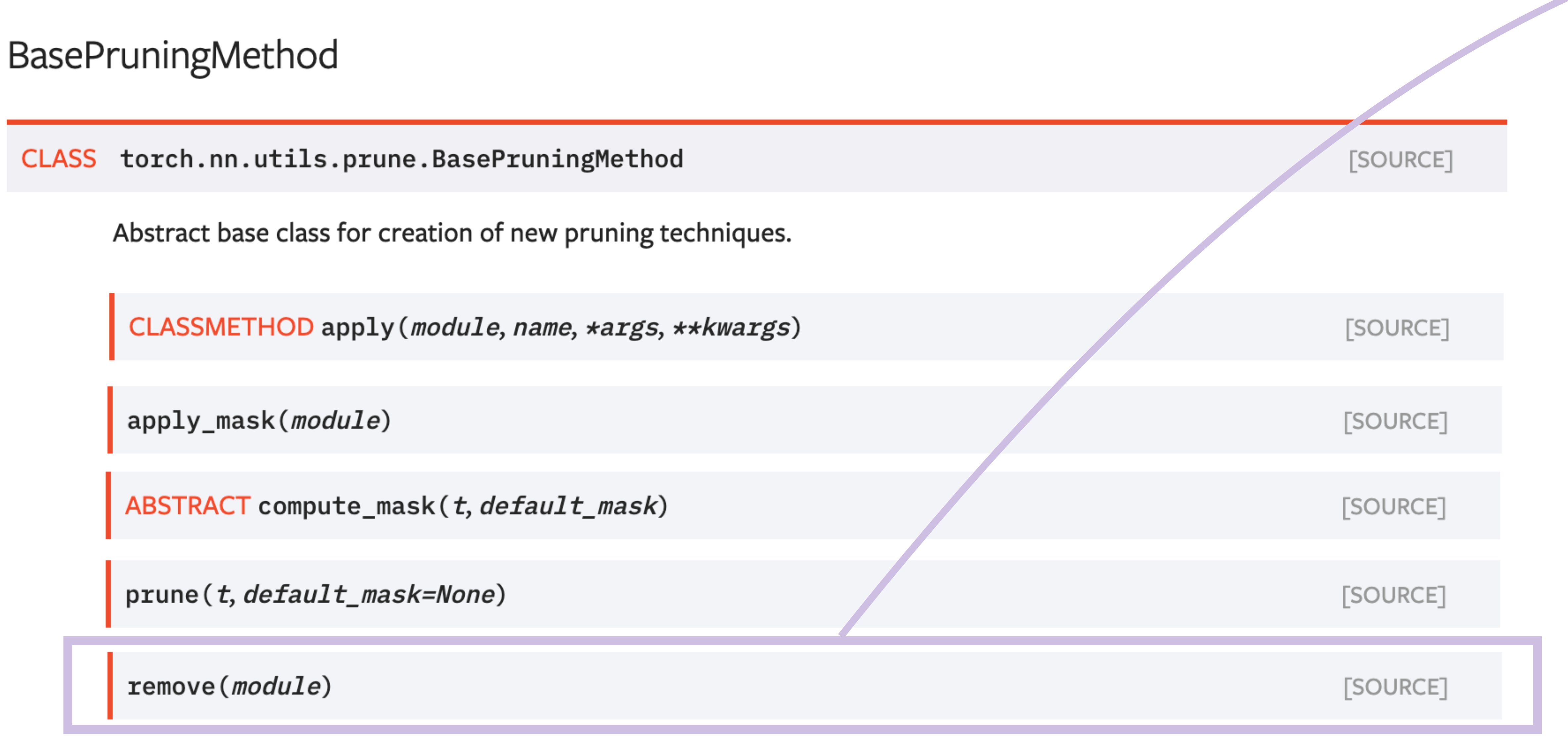
`prune(t, default_mask=None)` [SOURCE]

`remove(module)` [SOURCE]

Fetches the mask and the original, unpruned tensor to compute the pruned tensor during the forward pass → op is accounted for in the backward pass, too



torch.nn.utils.prune



BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]

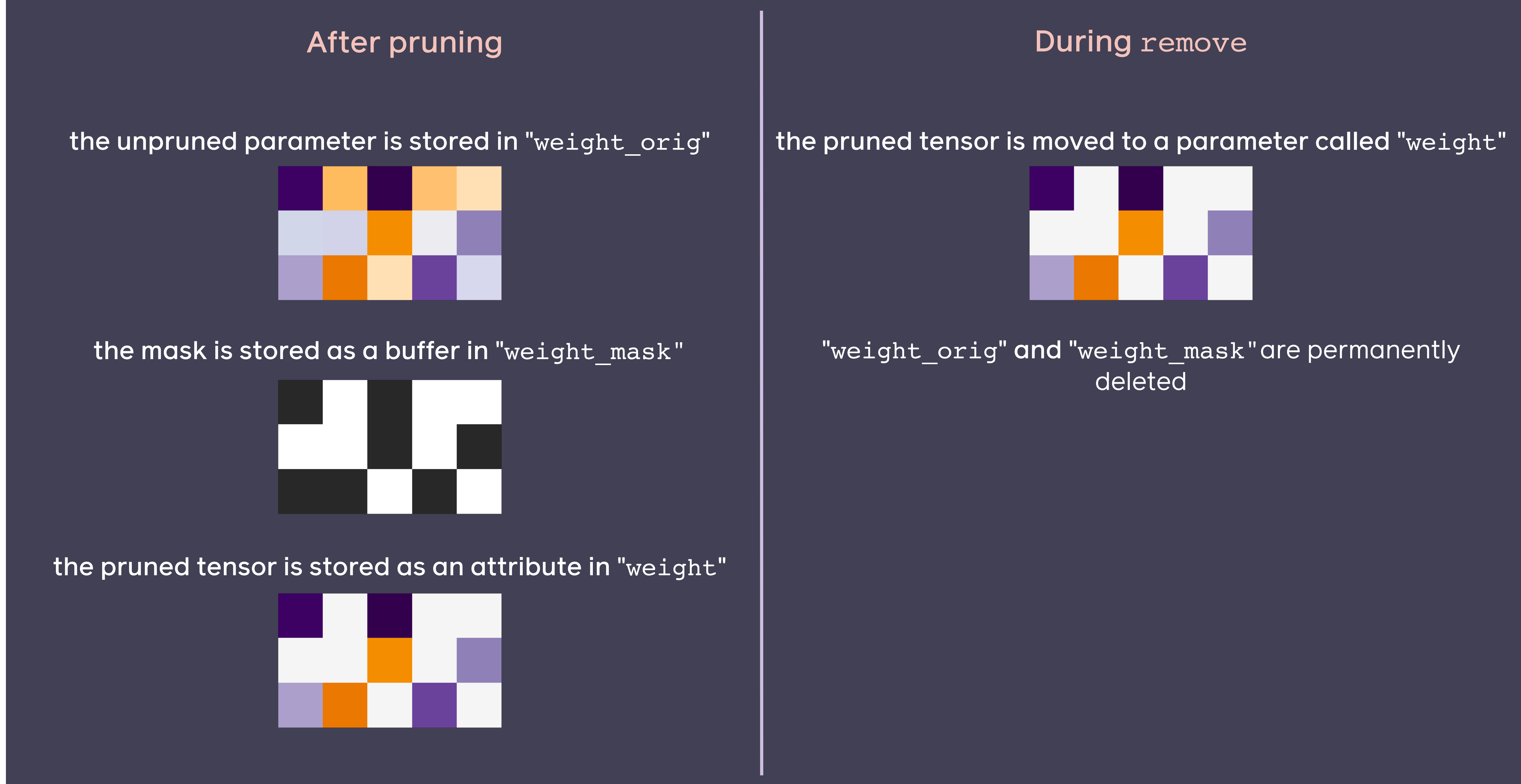
`apply_mask(module)` [SOURCE]

ABSTRACT `compute_mask(t, default_mask)` [SOURCE]

`prune(t, default_mask=None)` [SOURCE]

`remove(module)` [SOURCE]

Makes the pruning reparametrization permanent
!= undoing pruning



After pruning

the unpruned parameter is stored in "`weight_orig`"

the mask is stored as a buffer in "`weight_mask`"

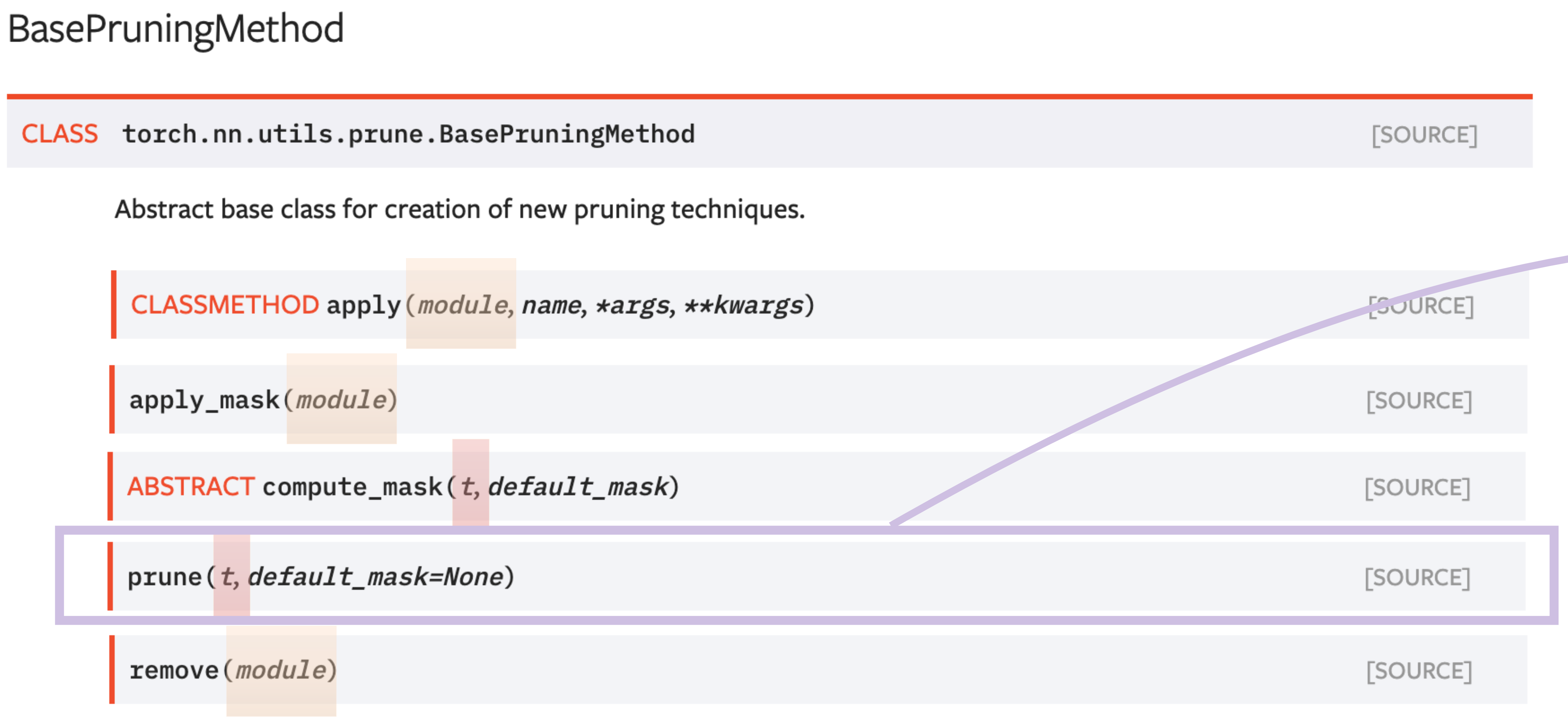
the pruned tensor is stored as an attribute in "`weight`"

During `remove`

the pruned tensor is moved to a parameter called "`weight`"

"`weight_orig`" and "`weight_mask`" are permanently deleted

torch.nn.utils.prune



BasePruningMethod

CLASS torch.nn.utils.prune.BasePruningMethod [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD apply(*module*, *name*, **args*, ***kwargs*) [SOURCE]

apply_mask(*module*) [SOURCE]

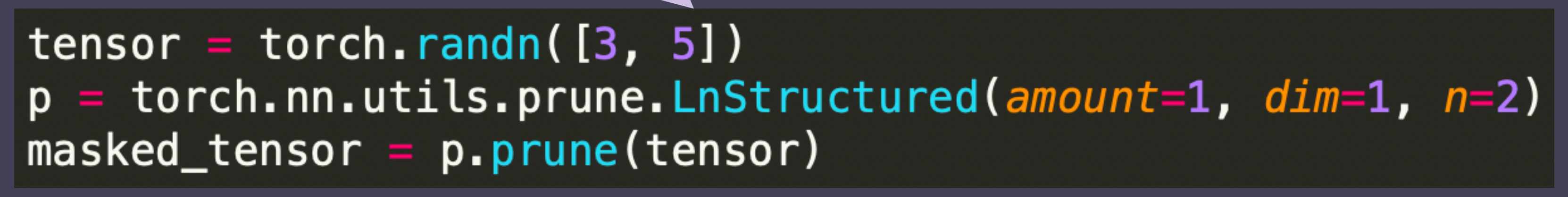
ABSTRACT compute_mask(*t*, *default_mask*) [SOURCE]

prune(*t*, *default_mask=None*) [SOURCE]

remove(*module*) [SOURCE]

torch.nn.utils.prune is designed to act on a torch.nn.Module

provides an interface for acting directly on a tensor



```
tensor = torch.randn([3, 5])
p = torch.nn.utils.prune.LnStructured(amount=1, dim=1, n=2)
masked_tensor = p.prune(tensor)
```

FACEBOOK AI

torch.nn.utils.prune

Easy to use

```
model = LeNet()  # unpruned model

# L_2 structured pruning will remove 50% of channels across axis 0
prune.ln_structured(
    module=model.conv1,
    name="weight",
    amount=0.5,
    n=2,
    dim=0
)
```

Iterative pruning made easy

`prune.PruningContainer` handles the combination of successive masks for you

```
for _ in range(10):
    # Remove 2 connections per iteration
    prune.l1_unstructured(module=model.fc1, name="bias", amount=2)
```

Global pruning made easy

```
parameters_to_prune = (
    (model.conv1, "weight"),
    (model.conv2, "weight"),
    (model.fc1, "weight"),
)

prune.global_unstructured(
    parameters_to_prune,
    pruning_method=prune.L1Unstructured,
    amount=0.2,
)
```

Easy to extend

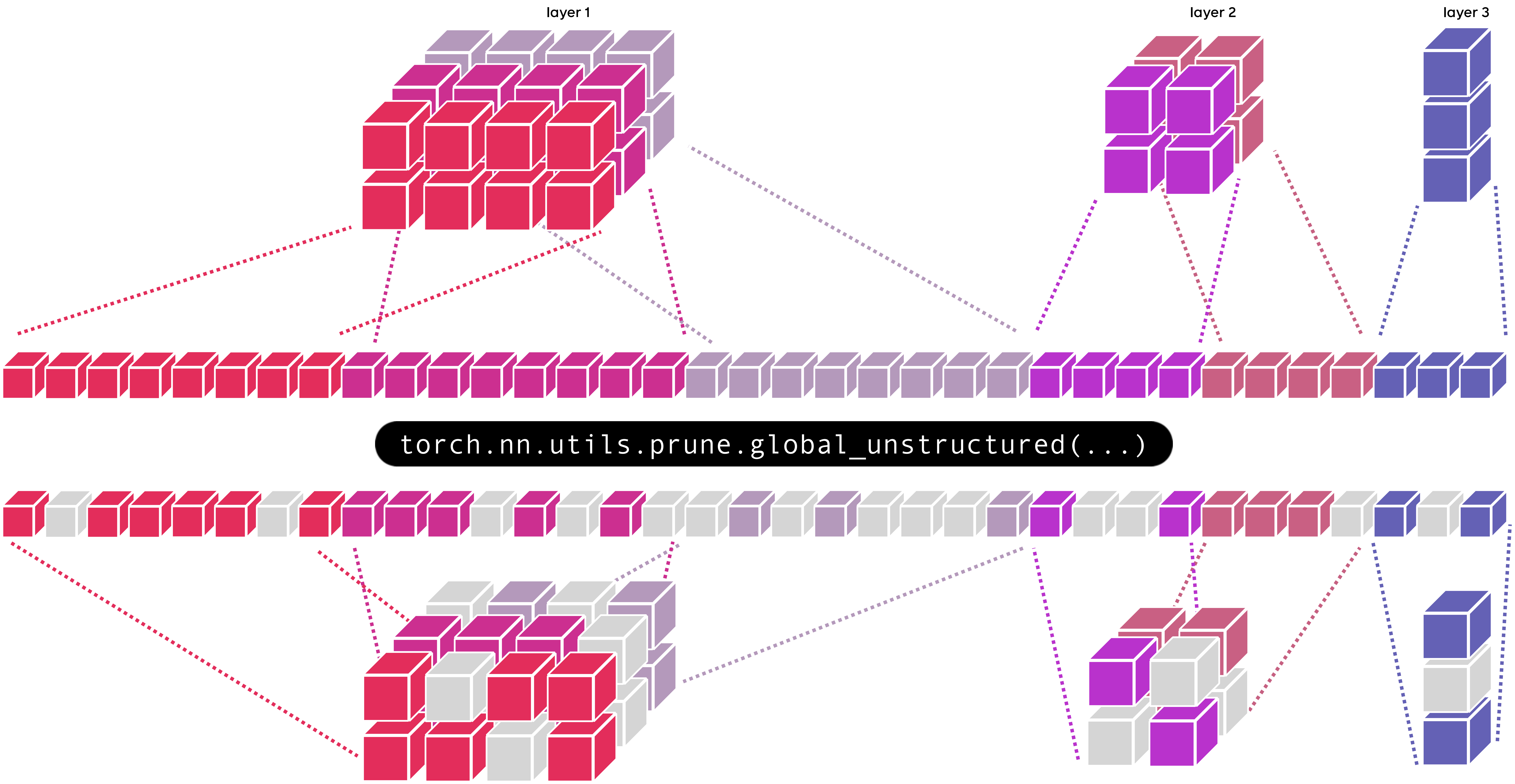
```
class FooBarPruningMethod(prune.BasePruningMethod):
    """Prune every other entry in a tensor
    """
    PRUNING_TYPE = 'unstructured'

    def compute_mask(self, t, default_mask):
        mask = default_mask.clone()
        mask.view(-1)[::2] = 0
        return mask
```

supports 3 PRUNING_TYPEs: `'global'`, `'structured'`, and `'unstructured'` (to determine how to combine masks if pruning is applied iteratively)

instructions on how to compute the mask for the given tensor according to the logic of your pruning technique

```
def foobar_unstructured(module, name):
    FooBarPruningMethod.apply(module, name)
    return module
```

GlobalPruning



```
torch.nn.utils.prune.global_unstructured(...)
```

Questions?

Contact: michela@fb.com, jessica_forde@brown.edu

Tutorials