torch.nn

These are the asic uil ing locks for graphs:

torch.nn

`Buffer`	A kin of Tensor that shoul not e consi ere a mo el parameter.
`Parameter`	A kin of Tensor that is to e consi ere a mo ule parameter.
`UninitializedParameter`	A parameter that is not initialize .
`UninitializedBuffer`	A uffer that is not initialize .

Containers

`Module`	Base class for all neural network mo ules.
`Sequential`	A sequential container.
`ModuleList`	Hol s su mo ules in a list.
`ModuleDict`	Hol s su mo ules in a ictionary.
`ParameterList`	Hol s parameters in a list.
`ParameterDict`	Hol s parameters in a ictionary.

Glo al Hooks For Mo ule

`register_module_forward_pre_hook`	Register a forwar pre-hook common to all mo ules.
`register_module_forward_hook`	Register a glo al forwar hook for all the mo ules.
`register_module_backward_hook`	Register a ackwar hook common to all the mo ules.
`register_module_full_backward_pre_hook`	Register a ackwar pre-hook common to all the mo ules.
`register_module_full_backward_hook`	Register a ackwar hook common to all the mo ules.
`register_module_buffer_registration_hook`	Register a uffer registration hook common to all mo ules.
`register_module_module_registration_hook`	Register a mo ule registration hook common to all mo ules.
`register_module_parameter_registration_hook`	Register a parameter registration hook common to all mo ules.

Convolution Layers

`nn.Conv1d`	Applies a 1D convolution over an input signal compose of several input planes.
`nn.Conv2d`	Applies a 2D convolution over an input signal compose of several input planes.
`nn.Conv3d`	Applies a 3D convolution over an input signal compose of several input planes.
`nn.ConvTranspose1d`	Applies a 1D transpose convolution operator over an input image compose of several input planes.
`nn.ConvTranspose2d`	Applies a 2D transpose convolution operator over an input image compose of several input planes.
`nn.ConvTranspose3d`	Applies a 3D transpose convolution operator over an input image compose of several input planes.
`nn.LazyConv1d`	A `torch.nn.Conv1d` mo ule with lazy initialization of the `in_channels` argument.
`nn.LazyConv2d`	A `torch.nn.Conv2d` mo ule with lazy initialization of the `in_channels` argument.
`nn.LazyConv3d`	A `torch.nn.Conv3d` mo ule with lazy initialization of the `in_channels` argument.
`nn.LazyConvTranspose1d`	A `torch.nn.ConvTranspose1d` mo ule with lazy initialization of the `in_channels` argument.
`nn.LazyConvTranspose2d`	A `torch.nn.ConvTranspose2d` mo ule with lazy initialization of the `in_channels` argument.
`nn.LazyConvTranspose3d`	A `torch.nn.ConvTranspose3d` mo ule with lazy initialization of the `in_channels` argument.
`nn.Unfold`	Extracts sli ing local locks from a atche input tensor.
`nn.Fold`	Com ines an array of sli ing local locks into a large containing tensor.

Pooling layers

`nn.MaxPool1d`	Applies a 1D max pooling over an input signal compose of several input planes.
`nn.MaxPool2d`	Applies a 2D max pooling over an input signal compose of several input planes.
`nn.MaxPool3d`	Applies a 3D max pooling over an input signal compose of several input planes.
`nn.MaxUnpool1d`	Computes a partial inverse of `MaxPool1d`.
`nn.MaxUnpool2d`	Computes a partial inverse of `MaxPool2d`.
`nn.MaxUnpool3d`	Computes a partial inverse of `MaxPool3d`.
`nn.AvgPool1d`	Applies a 1D average pooling over an input signal compose of several input planes.
`nn.AvgPool2d`	Applies a 2D average pooling over an input signal compose of several input planes.
`nn.AvgPool3d`	Applies a 3D average pooling over an input signal compose of several input planes.
`nn.FractionalMaxPool2d`	Applies a 2D fractional max pooling over an input signal compose of several input planes.
`nn.FractionalMaxPool3d`	Applies a 3D fractional max pooling over an input signal compose of several input planes.
`nn.LPPool1d`	Applies a 1D power-average pooling over an input signal compose of several input planes.
`nn.LPPool2d`	Applies a 2D power-average pooling over an input signal compose of several input planes.
`nn.LPPool3d`	Applies a 3D power-average pooling over an input signal compose of several input planes.
`nn.AdaptiveMaxPool1d`	Applies a 1D a aptive max pooling over an input signal compose

`nn.AdaptiveMaxPool3d`	Applies a 3D a aptive max pooling over an input signal compose of several input planes.
`nn.AdaptiveAvgPool1d`	Applies a 1D a aptive average pooling over an input signal compose of several input planes.
`nn.AdaptiveAvgPool2d`	Applies a 2D a aptive average pooling over an input signal compose of several input planes.
`nn.AdaptiveAvgPool3d`	Applies a 3D a aptive average pooling over an input signal compose of several input planes.

Pa ing Layers

`nn.ReflectionPad1d`	Pa s the input tensor using the reflection of the input oun ary.
`nn.ReflectionPad2d`	Pa s the input tensor using the reflection of the input oun ary.
`nn.ReflectionPad3d`	Pa s the input tensor using the reflection of the input oun ary.
`nn.ReplicationPad1d`	Pa s the input tensor using replication of the input oun ary.
`nn.ReplicationPad2d`	Pa s the input tensor using replication of the input oun ary.
`nn.ReplicationPad3d`	Pa s the input tensor using replication of the input oun ary.
`nn.ZeroPad1d`	Pa s the input tensor oun aries with zero.
`nn.ZeroPad2d`	Pa s the input tensor oun aries with zero.
`nn.ZeroPad3d`	Pa s the input tensor oun aries with zero.
`nn.ConstantPad1d`	Pa s the input tensor oun aries with a constant value.
`nn.ConstantPad2d`	Pa s the input tensor oun aries with a constant value.
`nn.ConstantPad3d`	Pa s the input tensor oun aries with a constant value.
`nn.CircularPad1d`	Pa s the input tensor using circular pa ing of the input oun ary.
`nn.CircularPad2d`	Pa s the input tensor using circular pa ing of the input oun ary.
`nn.CircularPad3d`	Pa s the input tensor using circular pa ing of the input oun ary.

Non-linear Activations (weighte sum, nonlinearity)

`nn.ELU`	Applies the Exponential Linear nit (EL) function, element-wise.
`nn.Hardshrink`	Applies the Har Shrinkage (Har shrink) function element-wise.
`nn.Hardsigmoid`	Applies the Har sigmoi function element-wise.
`nn.Hardtanh`	Applies the Har Tanh function element-wise.
`nn.Hardswish`	Applies the Har swish function, element-wise.
`nn.LeakyReLU`	Applies the LeakyReL function element-wise.
`nn.LogSigmoid`	Applies the Logsigmoi function element-wise.
`nn.MultiheadAttention`	Allows the mo el to ointly atten to information from ifferent representation su spaces.
`nn.PReLU`	Applies the element-wise PReL function.
`nn.ReLU`	Applies the rectifie linear unit function element-wise.
`nn.ReLU6`	Applies the ReL 6 function element-wise.
`nn.RReLU`	Applies the ran omize leaky rectifie linear unit function, element-wise.
`nn.SELU`	Applies the SEL function element-wise.
`nn.CELU`	Applies the CEL function element-wise.

`nn.Sigmoid`	Applies the Sigmoi function element-wise.
`nn.SiLU`	Applies the Sigmoi Linear nit (SiL) function, element-wise.
`nn.Mish`	Applies the Mish function, element-wise.
`nn.Softplus`	Applies the Softplus function element-wise.
`nn.Softshrink`	Applies the soft shrinkage function element-wise.
`nn.Softsign`	Applies the element-wise Softsign function.
`nn.Tanh`	Applies the Hyper olic Tangent (Tanh) function element-wise.
`nn.Tanhshrink`	Applies the element-wise Tanhshrink function.
`nn.Threshold`	Threshol s each element of the input Tensor.
`nn.GLU`	Applies the gate linear unit function.

Non-linear Activations (other)

`nn.Softmin`	Applies the Softmin function to an n- imensional input Tensor.
`nn.Softmax`	Applies the Softmax function to an n- imensional input Tensor.
`nn.Softmax2d`	Applies SoftMax over features to each spatial location.
`nn.LogSoftmax`	Applies the $\log(\mathrm{Softmax}(x))$ function to an n- imensional input Tensor.
`nn.AdaptiveLogSoftmaxWithLoss`	Efficient softmax approximation.

Normalization Layers

`nn.BatchNorm1d`	Applies Batch Normalization over a 2D or 3D input.
`nn.BatchNorm2d`	Applies Batch Normalization over a 4D input.
`nn.BatchNorm3d`	Applies Batch Normalization over a 5D input.
`nn.LazyBatchNorm1d`	A `torch.nn.BatchNorm1d` mo ule with lazy initialization.
`nn.LazyBatchNorm2d`	A `torch.nn.BatchNorm2d` mo ule with lazy initialization.
`nn.LazyBatchNorm3d`	A `torch.nn.BatchNorm3d` mo ule with lazy initialization.
`nn.GroupNorm`	Applies Group Normalization over a mini- atch of inputs.
`nn.SyncBatchNorm`	Applies Batch Normalization over a N-Dimensional input.
`nn.InstanceNorm1d`	Applies Instance Normalization.
`nn.InstanceNorm2d`	Applies Instance Normalization.
`nn.InstanceNorm3d`	Applies Instance Normalization.
`nn.LazyInstanceNorm1d`	A `torch.nn.InstanceNorm1d` mo ule with lazy initialization of the `num_features` argument.
`nn.LazyInstanceNorm2d`	A `torch.nn.InstanceNorm2d` mo ule with lazy initialization of the `num_features` argument.
`nn.LazyInstanceNorm3d`	A `torch.nn.InstanceNorm3d` mo ule with lazy initialization of the `num_features` argument.
`nn.LayerNorm`	Applies Layer Normalization over a mini- atch of inputs.
`nn.LocalResponseNorm`	Applies local response normalization over an input signal.
`nn.RMSNorm`	Applies Root Mean Square Layer Normalization over a mini- atch of inputs.

Recurrent Layers

	linearity to an input sequence.
`nn.LSTM`	Apply a multi-layer long short-term memory (LSTM) RNN to an input sequence.
`nn.GRU`	Apply a multi-layer gate recurrent unit (GR) RNN to an input sequence.
`nn.RNNCell`	An Elman RNN cell with tanh or ReL non-linearity.
`nn.LSTMCell`	A long short-term memory (LSTM) cell.
`nn.GRUCell`	A gate recurrent unit (GR) cell.

Transformer Layers

`nn.Transformer`	A transformer mo el.
`nn.TransformerEncoder`	TransformerEnco er is a stack of N enco er layers.
`nn.TransformerDecoder`	TransformerDeco er is a stack of N eco er layers.
`nn.TransformerEncoderLayer`	TransformerEnco erLayer is ma e up of self-attn an fee forwar network.
`nn.TransformerDecoderLayer`	TransformerDeco erLayer is ma e up of self-attn, multi-hea -attn an fee forwar network.

Linear Layers

`nn.Identity`	A placehol er i entity operator that is argument-insensitive.
`nn.Linear`	Applies an affine linear transformation to the incoming ata: $y = xA^T + b$.
`nn.Bilinear`	Applies a ilinear transformation to the incoming ata: $y = x_1^T A x_2 + b$.
`nn.LazyLinear`	A `torch.nn.Linear` mo ule where *in_features* is inferre .

Dro out Layers

`nn.Dropout`	During training, ran omly zeroes some of the elements of the input tensor with pro a ility `p`.
`nn.Dropout1d`	Ran omly zero out entire channels.
`nn.Dropout2d`	Ran omly zero out entire channels.
`nn.Dropout3d`	Ran omly zero out entire channels.
`nn.AlphaDropout`	Applies Alpha Dropout over the input.
`nn.FeatureAlphaDropout`	Ran omly masks out entire channels.

S arse Layers

`nn.Embedding`	A simple lookup ta le that stores em e ings of a fixe ictionary an size.
`nn.EmbeddingBag`	Compute sums or means of ′ ags′ of em e ings, without instantiating the interme iate em e ings.

Distance Functions

`nn.CosineSimilarity`	Returns cosine similarity etween x_1 an x_2, compute along *dim*.
`nn.PairwiseDistance`	Computes the pairwise istance etween input vectors, or etween columns of input matrices.

Loss Functions

`nn.L1Loss`	Creates a criterion that measures the mean a solute error (MAE) etween each element in the input x an target y.

`nn.CrossEntropyLoss`	This criterion computes the cross entropy loss etween input logits an target.
`nn.CTCLoss`	The Connectionist Temporal Classification loss.
`nn.NLLLoss`	The negative log likelihoo loss.
`nn.PoissonNLLLoss`	Negative log likelihoo loss with Poisson istri ution of target.
`nn.GaussianNLLLoss`	Gaussian negative log likelihoo loss.
`nn.KLDivLoss`	The Kull ack-Lei ler ivergence loss.
`nn.BCELoss`	Creates a criterion that measures the Binary Cross Entropy etween the target an the input pro a ilities:
`nn.BCEWithLogitsLoss`	This loss com ines a *Sigmoid* layer an the *BCELoss* in one single class.
`nn.MarginRankingLoss`	Creates a criterion that measures the loss given inputs $x1$, $x2$, two 1D mini- atch or 0D *Tensors*, an a la el 1D mini- atch or 0D *Tensor* y (containing 1 or -1).
`nn.HingeEmbeddingLoss`	Measures the loss given an input tensor x an a la els tensor y (containing 1 or -1).
`nn.MultiLabelMarginLoss`	Creates a criterion that optimizes a multi-class multi-classification hinge loss (margin- ase loss) etween input x (a 2D mini- atch *Tensor*) an output y (which is a 2D *Tensor* of target class in ices).
`nn.HuberLoss`	Creates a criterion that uses a square term if the a solute element-wise error falls elow elta an a elta-scale L1 term otherwise.
`nn.SmoothL1Loss`	Creates a criterion that uses a square term if the a solute element-wise error falls elow eta an an L1 term otherwise.
`nn.SoftMarginLoss`	Creates a criterion that optimizes a two-class classification logistic loss etween input tensor x an target tensor y (containing 1 or -1).
`nn.MultiLabelSoftMarginLoss`	Creates a criterion that optimizes a multi-la el one-versus-all loss ase on max-entropy, etween input x an target y of size (N, C).
`nn.CosineEmbeddingLoss`	Creates a criterion that measures the loss given input tensors x_1, x_2 an a *Tensor* la el y with values 1 or -1.
`nn.MultiMarginLoss`	Creates a criterion that optimizes a multi-class classification hinge loss (margin- ase loss) etween input x (a 2D mini- atch *Tensor*) an output y (which is a 1D tensor of target class in ices, $0 \leq y \leq \text{x.size}(1) - 1$):
`nn.TripletMarginLoss`	Creates a criterion that measures the triplet loss given an input tensors $x1$, $x2$, $x3$ an a margin with a value greater than 0.
`nn.TripletMarginWithDistanceLoss`	Creates a criterion that measures the triplet loss given input tensors a, p, an n (representing anchor, positive, an negative examples, respectively), an a nonnegative, real-value function (" istance function") use to compute the relationship etween the anchor an positive example ("positive istance") an the anchor an negative example ("negative istance").

Vision Layers

`nn.PixelShuffle`	Rearrange elements in a tensor accor ing to an upscaling factor.
`nn.PixelUnshuffle`	Reverse the PixelShuffle operation.
`nn.Upsample`	psamples a given multi-channel 1D (temporal), 2D (spatial) or 3D (volumetric) ata.
`nn.UpsamplingNearest2d`	Applies a 2D nearest neigh or upsampling to an input signal compose of several input channels.
`nn.UpsamplingBilinear2d`	Applies a 2D ilinear upsampling to an input signal compose of several input channels.

Shuffle Layers

`nn.ChannelShuffle`	Divi es an rearranges the channels in a tensor.

DataParallel Layers (multi-GP , istri ute)

`nn.parallel.DistributedDataParallel`	Implement istri ute ata parallelism ase on `torch.distributed` at mo ule level.

tilities

From the `torch.nn.utils` mo ule:

tility functions to clip parameter gra ients.

`clip_grad_norm_`	Clip the gra ient norm of an itera le of parameters.
`clip_grad_norm`	Clip the gra ient norm of an itera le of parameters.
`clip_grad_value_`	Clip the gra ients of an itera le of parameters at specifie value.
`get_total_norm`	Compute the norm of an itera le of tensors.
`clip_grads_with_norm_`	Scale the gra ients of an itera le of parameters given a pre-calculate total norm an esire max norm.

tility functions to flatten an unflatten Mo ule parameters to an from a single vector.

`parameters_to_vector`	Flatten an itera le of parameters into a single vector.
`vector_to_parameters`	Copy slices of a vector into an itera le of parameters.

tility functions to fuse Mo ules with BatchNorm mo ules.

`fuse_conv_bn_eval`	Fuse a convolutional mo ule an a BatchNorm mo ule into a single, new convolutional mo ule.
`fuse_conv_bn_weights`	Fuse convolutional mo ule parameters an BatchNorm mo ule parameters into new convolutional mo ule parameters.
`fuse_linear_bn_eval`	Fuse a linear mo ule an a BatchNorm mo ule into a single, new linear mo ule.
`fuse_linear_bn_weights`	Fuse linear mo ule parameters an BatchNorm mo ule parameters into new linear mo ule parameters.

tility functions to convert Mo ule parameter memory formats.

`convert_conv2d_weight_memory_format`	Convert `memory_format` of `nn.Conv2d.weight` to `memory_format`.
`convert_conv3d_weight_memory_format`	Convert `memory_format` of `nn.Conv3d.weight` to `memory_format` The conversion recursively applies to neste `nn.Module`, inclu ing `module`.

tility functions to apply an remove weight normalization from Mo ule parameters.

`weight_norm`	Apply weight normalization to a parameter in the given mo ule.
`remove_weight_norm`	Remove the weight normalization reparameterization from a mo ule.
`spectral_norm`	Apply spectral normalization to a parameter in the given mo ule.
`remove_spectral_norm`	Remove the spectral normalization reparameterization from a mo ule.

tility functions for initializing Mo ule parameters.

`skip_init`	Given a mo ule class o ect an args / kwargs, instantiate the mo ule without initializing parameters / uffers.

tility classes an functions for pruning Mo ule parameters.

`prune.BasePruningMethod`	A stract ase class for creation of new pruning techniques.
`prune.PruningContainer`	Container hol ing a sequence of pruning metho s for iterative pruning.
`prune.Identity`	tility pruning metho that oes not prune any units ut generates the pruning parametrization with a mask of ones.

`prune.RandomStructured`	Prune entire (currently unpruned) channels in a tensor at random.
`prune.LnStructured`	Prune entire (currently unpruned) channels in a tensor based on their L`n`-norm.
`prune.CustomFromMask`	
`prune.identity`	Apply pruning reparametrization without pruning any units.
`prune.random_unstructured`	Prune tensor by removing random (currently unpruned) units.
`prune.l1_unstructured`	Prune tensor by removing units with the lowest L1-norm.
`prune.random_structured`	Prune tensor by removing random channels along the specified dimension.
`prune.ln_structured`	Prune tensor by removing channels with the lowest L`n`-norm along the specified dimension.
`prune.global_unstructured`	Globally prunes tensors corresponding to all parameters in `parameters` by applying the specified `pruning_method`.
`prune.custom_from_mask`	Prune tensor corresponding to parameter called `name` in `module` by applying the pre-computed mask in `mask`.
`prune.remove`	Remove the pruning reparameterization from a module and the pruning method from the forward hook.
`prune.is_pruned`	Check if a module is pruned by looking for pruning pre-hooks.

Parametrizations implemented using the new parametrization functionality in `torch.nn.utils.parameterize.register_parametrization()`.

`parametrizations.orthogonal`	Apply an orthogonal or unitary parametrization to a matrix or a batch of matrices.
`parametrizations.weight_norm`	Apply weight normalization to a parameter in the given module.
`parametrizations.spectral_norm`	Apply spectral normalization to a parameter in the given module.

Utility functions to parametrize Tensors on existing Modules. Note that these functions can be used to parametrize a given Parameter or Buffer given a specific function that maps from an input space to the parametrized space. They are not parameterizations that would transform an object into a parameter. See the Parametrizations tutorial for more information on how to implement your own parametrizations.

`parametrize.register_parametrization`	Register a parametrization to a tensor in a module.
`parametrize.remove_parametrizations`	Remove the parametrizations on a tensor in a module.
`parametrize.cached`	Context manager that enables the caching system within parametrizations registered with `register_parametrization()`.
`parametrize.is_parametrized`	Determine if a module has a parametrization.
`parametrize.ParametrizationList`	A sequential container that holds and manages the original parameters or buffers of a parametrized `torch.nn.Module`.

Utility functions to call a given Module in a stateless manner.

`stateless.functional_call`	Perform a functional call on the module by replacing the module parameters and buffers with the provided ones.

Utility functions in other modules

`nn.utils.rnn.PackedSequence`	Holds the data and list of `batch_sizes` of a packed sequence.
`nn.utils.rnn.pack_padded_sequence`	Packs a Tensor containing padded sequences of variable length.
`nn.utils.rnn.pad_packed_sequence`	Pads a packed batch of variable length sequences.
`nn.utils.rnn.pad_sequence`	Pads a list of variable length Tensors with `padding_value`.
`nn.utils.rnn.pack_sequence`	Packs a list of variable length Tensors.
`nn.utils.rnn.unpack_sequence`	Unpack PackedSequence into a list of variable length Tensors.
`nn.utils.rnn.unpad_sequence`	Unpad padded Tensor into a list of variable length Tensors.

`nn.Flatten`	Flattens a contiguous range of ims into a tensor.
`nn.Unflatten`	nflattens a tensor im expan ing it to a esire shape.

Quantize Functions

Quantization refers to techniques for performing computations an storing tensors at lower itwi ths than floating point precision. PyTorch supports oth per tensor an per channel asymmetric linear quantization. To learn more how to use quantize functions in PyTorch, please refer to the Quantization ocumentation.

Lazy Mo ules Initialization

`nn.modules.lazy.LazyModuleMixin`	A mixin for mo ules that lazily initialize parameters, also known as "lazy mo ules".

Aliases

The following are aliases to their counterparts in `torch.nn`:

`nn.modules.normalization.RMSNorm`	Applies Root Mean Square Layer Normalization over a mini- atch of inputs.

‹ Previous

Next ›