BeyondML

BeyondML Labs

Jan 03, 2024

DOCUMENTATION:

BeyondML is a Python package which enables creating sparse multitask artificial neural networks (MANNs) compatible with TensorFlow and PyTorch. This package contains custom layers and utilities to facilitate the training and optimization of models using the Reduction of Sub-Network Neuroplasticity (RSN2) training procedure developed by AI Squared, Inc.

`View this Documentation in PDF Format`

CHAPTER

ONE

INSTALLATION

This package is available through Pypi and can be installed by running the following command:

```
pip install beyondml
```

Alternatively, the latest version of the software can be installed directly from GitHub using the following command:

```
pip install git+https://github.com/beyond-ml-labs/beyondml
```

1.1 beyondml

1.1.1 beyondml package

Subpackages

beyondml.pt package

Subpackages

beyondml.pt.layers package

Submodules

beyondml.pt.layers.Conv2D module

class beyondml.pt.layers.Conv2D.**Conv2D**(*kernel*, *bias*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: `Module`

Convolutional 2D layer initialized directly with weights, rather than with hyperparameters

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.Conv3D module

`class beyondml.pt.layers.Conv3D.Conv3D(kernel, bias, padding='same', strides=1, device=None, dtype=None)`

Bases: `Module`

Convolutional 3D layer initialized directly with weights, rather than with hyperparameters

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.Dense module

`class beyondml.pt.layers.Dense.Dense(weight, bias, device=None, dtype=None)`

Bases: `Module`

Fully-connected layer initialized directly with weights, rather than hyperparameters

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.FilterLayer module

`class beyondml.pt.layers.FilterLayer.FilterLayer(is_on=True, device=None, dtype=None)`

Bases: `Module`

Layer which filters input data, either returning values or all zeros depending on state

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property is_on

turn_off()
Turn off the layer

turn_on()
Turn on the layer

beyondml.pt.layers.MaskedConv2D module

class beyondml.pt.layers.MaskedConv2D.**MaskedConv2D**(*in_channels*, *out_channels*, *kernel_size=3*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Masked 2D Convolutional layer

forward(*inputs*)
Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property in_channels

property kernel_size

property out_channels

prune(*percentile*)
Prune the layer by updating the layer's mask

Parameters
percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MaskedConv3D module

class beyondml.pt.layers.MaskedConv3D.**MaskedConv3D**(*in_channels*, *out_channels*, *kernel_size=3*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Masked 3D Convolutional layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property in_channels

property kernel_size

property out_channels

prune(*percentile*)

Prune the layer by updating the layer's masks

Parameters
percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MaskedDense module

class beyondml.pt.layers.MaskedDense.**MaskedDense**(*in_features*, *out_features*, *device=None*, *dtype=None*)

Bases: Module

Masked fully-connected layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

prune(*percentile*)

Prune the layer by updating the layer's mask

Parameters

percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MaskedMultiHeadAttention module

class beyondml.pt.layers.MaskedMultiHeadAttention.**MaskedMultiHeadAttention**(*embed_dim*, *num_heads*, *dropout=0*, *batch_first=False*, *device=None*, *dtype=None*)

Bases: Module

Masked Multi-Headed Attention Layer

forward(*query*, *key*, *value*, *key_padding_mask=None*, *need_weights=True*, *attn_mask=None*, *average_attn_weights=True*)

Call the layer on input data

Parameters

- **query** (*torch Tensor*) – Query tensor
- **key** (*torch Tensor*) – Key tensor
- **value** (*torch Tensor*) – Value tensor
- **key_padding_mask** (*None or torch Tensor (default None)*) – If specified, a mask indicating which elements in key to ignore
- **need_weights** (*Bool (default True)*) – If specified, returns attn_output_weights as well as attn_outputs
- **attn_mask** (*None or torch Tensor (default None)*) – If specified, a 2D or 3D mask preventing attention
- **average_attn_weights** (*Bool (default True)*) – If True, indicates that returned attn_weights should be averaged across heads

prune(*percentile*)

Prune the layer by updating the layer's mask

Parameters

percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be made inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MaskedTransformerDecoderLayer module

class beyondml.pt.layers.MaskedTransformerDecoderLayer.**MaskedTransformerDecoderLayer**(*d_model: int, nhead: int, dim_feedforward: int = 2048, dropout: float = 0.1, activation: str | ~typing.Callable[[~torch.Tensor], ~torch.Tensor] = <function relu>, layer_norm_eps: float = 1e-05, batch_first: bool = False, norm_first: bool = False, device=None, dtype=None*)

Bases: `Module`

TransformerDecoderLayer is made up of self-attn, multi-head-attn and feedforward network. This standard decoder layer is based on the paper "Attention Is All You Need". :param d_model: the number of expected features in the input (required). :param nhead: the number of heads in the multiheadattention models (required). :param dim_feedforward: the dimension of the feedforward network model (default=2048). :param dropout: the dropout value (default=0.1). :param activation: the activation function of the intermediate layer, can be a string

("relu" or "gelu") or a unary callable. Default: relu

Parameters

- **layer_norm_eps** – the eps value in layer normalization components (default=1e-5).
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False` (seq, batch, feature).
- **norm_first** – if `True`, layer norm is done prior to self attention, multihead attention and feedforward operations, respectively. Otherwise it's done after. Default: `False` (after).

forward(*tgt: Tensor*, *memory: Tensor*)

Pass the inputs (and mask) through the decoder layer. :param tgt: the sequence to the decoder layer. :param memory: the sequence from the last layer of the encoder.

Shape:

see the docs in Pytorch Transformer class.

prune(*percentile*)

beyondml.pt.layers.MaskedTransformerEncoderLayer module

class beyondml.pt.layers.MaskedTransformerEncoderLayer.**MaskedTransformerEncoderLayer**(*d_model: int, nhead: int, dim_feedforward: int = 2048, dropout: float = 0.1, activation: str | ~typing.Callable[[~torch.Tensor], ~torch.Tensor] = <function relu>, layer_norm_eps: float = 1e-05, batch_first: bool = False, norm_first: bool = False, device=None, dtype=None*)

Bases: Module

TransformerEncoderLayer is made up of self-attn and feedforward network. :param d_model: the number of expected features in the input (required). :param nhead: the number of heads in the multiheadattention models (required). :param dim_feedforward: the dimension of the feedforward network model (default=2048). :param dropout: the dropout value (default=0.1). :param activation: the activation function of the intermediate layer, can be a string

("relu" or "gelu") or a unary callable. Default: relu

Parameters

- **layer_norm_eps** – the eps value in layer normalization components (default=1e-5).
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature). Default: False (seq, batch, feature).
- **norm_first** – if True, layer norm is done prior to attention and feedforward operations, respectivaly. Otherwise it's done after. Default: False (after).

forward(*src: Tensor*)

Pass the input through the encoder layer. :param src: the sequence to the encoder layer (required).

prune(*percentile*)

beyondml.pt.layers.MultiConv2D module

class beyondml.pt.layers.MultiConv2D.**MultiConv2D**(*kernel*, *bias*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Multi- 2D Convolutional layer initialized with weights rather than hyperparameters

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.MultiConv3D module

class beyondml.pt.layers.MultiConv3D.**MultiConv3D**(*kernel*, *bias*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Multitask 3D Convolutional layer initialized with weights rather than with hyperparameters

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.MultiDense module

class beyondml.pt.layers.MultiDense.**MultiDense**(*weight*, *bias*, *device=None*, *dtype=None*)

Bases: Module

Multi-Fully-Connected layer initialized with weights rather than hyperparameters

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.MultiMaskedConv2D module

class beyondml.pt.layers.MultiMaskedConv2D.**MultiMaskedConv2D**(*in_channels*, *out_channels*, *num_tasks*, *kernel_size=3*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Multi 2D Convolutional layer which supports masking and pruning

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property in_channels

property kernel_size

property out_channels

prune(*percentile*)

Prune the layer by updating the layer's mask

Parameters
percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MultiMaskedConv3D module

class beyondml.pt.layers.MultiMaskedConv3D.**MultiMaskedConv3D**(*in_channels*, *out_channels*, *num_tasks*, *kernel_size=3*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Masked Multitask 3D Convolutional layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property in_channels

property kernel_size

property out_channels

prune(*percentile*)

Prune the layer by updating the layer's masks

Parameters
percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MultiMaskedDense module

class beyondml.pt.layers.MultiMaskedDense.**MultiMaskedDense**(*in_features*, *out_features*, *num_tasks*, *device=None*, *dtype=None*)

Bases: Module

Multi-Fully-Connected layer which supports masking and pruning

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

prune(*percentile*)

Prune the layer by updating the layer's mask

Parameters
percentile (*int*) – Integer between 0 and 99 which represents the proportion of weights to be inactive

Notes

Acts on the layer in place

beyondml.pt.layers.MultiMaxPool2D module

class beyondml.pt.layers.MultiMaxPool2D.**MultiMaxPool2D**(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*)

Bases: Module

Multitask implementation of 2-dimensional Max Pooling layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.MultiMaxPool3D module

class beyondml.pt.layers.MultiMaxPool3D.**MultiMaxPool3D**(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*)

Bases: Module

Multitask implementation of 2-dimensional Max Pooling layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.MultitaskNormalization module

class beyondml.pt.layers.MultitaskNormalization.**MultitaskNormalization**(*device=None*, *dtype=None*)

Bases: Module

Layer which normalizes a set of inputs to sum to 1

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor or list of Tensors*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor or list of Tensors

beyondml.pt.layers.SelectorLayer module

class beyondml.pt.layers.SelectorLayer.**SelectorLayer**(*sel_index*)

Bases: Module

Layer which selects an individual input based on index and only returns that one

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

property sel_index

beyondml.pt.layers.SparseConv2D module

class beyondml.pt.layers.SparseConv2D.**SparseConv2D**(*kernel*, *bias*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: Module

Sparse implementation of a 2D Convolutional layer, expected to be converted from a trained, pruned layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.SparseConv3D module

`class beyondml.pt.layers.SparseConv3D.SparseConv3D(kernel, bias, padding='same', strides=1, device=None, dtype=None)`

Bases: `Module`

Sparse 3D Convolutional layer, expected to be converted from a trained, pruned layer

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.SparseDense module

`class beyondml.pt.layers.SparseDense.SparseDense(weight, bias, device=None, dtype=None)`

Bases: `Module`

Sparse implementation of a fully-connected layer

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.SparseMultiConv2D module

`class beyondml.pt.layers.SparseMultiConv2D.SparseMultiConv2D(kernel, bias, padding='same', strides=1, device=None, dtype=None)`

Bases: `Module`

Sparse implementation of a Multi 2D Convolutional layer

`forward(inputs)`

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.SparseMultiConv3D module

class beyondml.pt.layers.SparseMultiConv3D.**SparseMultiConv3D**(*kernel*, *bias*, *padding='same'*, *strides=1*, *device=None*, *dtype=None*)

Bases: `Module`

Sparse implementation of a Multitask 3D Convolutional layer, expected to be converted from a trained, pruned layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

beyondml.pt.layers.SparseMultiDense module

class beyondml.pt.layers.SparseMultiDense.**SparseMultiDense**(*weight*, *bias*, *device=None*, *dtype=None*)

Bases: `Module`

Sparse implementation of the Multi-Fully-Connected layer

forward(*inputs*)

Call the layer on input data

Parameters
inputs (*torch.Tensor*) – Inputs to call the layer's logic on

Returns
results – The results of the layer's logic

Return type
torch.Tensor

Module contents

Layers compatible with PyTorch models

beyondml.pt.utils package

Submodules

beyondml.pt.utils.utils module

beyondml.pt.utils.utils.**prune_model**(*model*, *percentile*)

Prune a compatible model

Parameters

- **model** (*PyTorch model*) – A model that has been developed to have a *.layers* property containing layers to be pruned
- **percentile** (*int*) – An integer between 0 and 99 which corresponds to how much to prune the model

Returns

pruned_model – The pruned model

Return type

PyTorch model

Notes

- The model input **must** have a *.layers* property to be able to function. Only layers within the *.layers* property which are recognized as prunable are pruned, via their own *.prune()* method
- Also acts on the model in place, but returns the model for ease of use

Module contents

Some additional utilities for building MANN models in PyTorch.

Module contents

PyTorch compatibility for building MANN models

The *beyondml.pt* subpackage contains layers and utilities for creating and pruning models using [PyTorch](https://pytorch.org). The package contains two subpackages, the *beyondml.pt.layers* package, and the *beyondml.pt.utils* package.

Within the *layers* package, there is current functionality for the the following layers: - *beyondml.pt.layers.Conv2D* - *beyondml.pt.layers.Dense* - *beyondml.pt.layers.FilterLayer* - *beyondml.pt.layers.MaskedConv2D* - *beyondml.pt.layers.MaskedDense* - *beyondml.pt.layers.MultiConv2D* - *beyondml.pt.layers.MultiDense* - *beyondml.pt.layers.MultiMaskedConv2D* - *beyondml.pt.layers.MultiMaskedDense* - *beyondml.pt.layers.SelectorLayer* - *beyondml.pt.layers.SparseConv2D* - *beyondml.pt.layers.SparseDense* - *beyondml.pt.layers.SparseMultiConv2D* - *beyondml.pt.layers.SparseMultiDense*

Within the *beyondml.pt.utils* package, there is currently only one function, the *prune_model* function. Because of the openness of developing with PyTorch in comparison to TensorFlow, there is far less functionality that can be supplied directly via BeyondML. Instead, for converting models from training to inference, the user is left to devise the best way to do so by building his or her own classes.

Best Practices for Pruning In order to use the *utils.prune_model* function, the model itself must have a *.layers* property. This property is used to determine which layers can be pruned. **Only layers which support pruning and which are included in the `.layers` property are pruned,** meaning the user can determine which exact layers in the model he or she wants pruned. Alternatively, the user can create their own pruning function or method on the class itself and prune that way, utilizing each of the *.prune()* methods of the layers provided.

beyondml.tflow package

Subpackages

beyondml.tflow.layers package

Submodules

beyondml.tflow.layers.FilterLayer module

class beyondml.tflow.layers.FilterLayer.**FilterLayer**(**args*, ***kwargs*)

Bases: Layer

Layer which filters inputs based on status of *on* or *off*

Example:

```
>>> # Create a model with just a FilterLayer
>>> input_layer = tf.keras.layers.Input(10)
>>> filter_layer = mann.layers.FilterLayer()(input_layer)
>>> model = tf.keras.models.Model(input_layer, filter_layer)
>>> model.compile()
>>> # Call the model with the layer turned on
>>> data = np.arange(10).reshape((1, 10))
>>> model.predict(data)
array([[0., 1., 2., 3., 4., 5., 6., 7., 8., 9.]], dtype=float32)
>>> # Turn off the FilterLayer and call it again
>>> model.layers[-1].turn_off()
>>> # Model must be recompiled after turning the layer on or off
>>> model.compile()
>>> model.predict(data)
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config(config)`

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config()`

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

`turn_off()`

Turn the layer *off* so inputs are destroyed and all-zero tensors are output

`turn_on()`

Turn the layer *on* so inputs are returned unchanged as outputs

beyondml.tflow.layers.MaskedConv2D module

`class beyondml.tflow.layers.MaskedConv2D.MaskedConv2D(*args, **kwargs)`

Bases: `Layer`

Masked 2-dimensional convolutional layer. For full documentation of the convolutional architecture, see the TensorFlow Keras Convolutional2D layer documentation.

This layer implements masking consistent with the BeyondML API to support developing sparse models.

`build(input_shape)`

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

`call(inputs)`

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config(config)`

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config()`

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

`property kernel_size`

`set_masks(new_masks)`

Set the masks for the layer

Parameters
new_masks (*list of arrays or array-likes*) – The new masks to set for the layer

beyondml.tflow.layers.MaskedConv3D module

`class beyondml.tflow.layers.MaskedConv3D.MaskedConv3D(*args, **kwargs)`

Bases: `Layer`

Masked 3-dimensional convolutional layer. For full documentation of the convolutional architecture, see the TensorFlow Keras Convolutional3D layer documentation.

This layer implements masking consistent with the BeyondML API to support developing sparse models

`build(input_shape)`

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

`call(inputs)`

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

property kernel_size

set_masks(*new_masks*)

Set the masks for the layer

Parameters
new_masks (*list of arrays or array-likes*) – The new masks to set for the layer

beyondml.tflow.layers.MaskedDense module

class beyondml.tflow.layers.MaskedDense.**MaskedDense**(**args*, ***kwargs*)

Bases: Layer

Masked fully connected layer. For full documentation of the fully-connected architecture, see the TensorFlow Keras Dense layer documentation.

This layer implements masking consistent with the BeyondML API to support developing sparse models.

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config(config)`

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config()`

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

`set_masks(new_masks)`

Set the masks for the layer

Parameters
new_masks (*list of arrays or array-likes*) – The new masks to set for the layer

beyondml.tflow.layers.MultiConv2D module

`class beyondml.tflow.layers.MultiConv2D.MultiConv2D(*args, **kwargs)`

Bases: `Layer`

Multitask 2-dimensional convolutional layer

This layer implements multiple stacks of convolutional weights to account for different ways individual neurons activate for various tasks. It is expected that to train using the RSN2 algorithm that MultiMaskedConv2D layers be used during training and then those layers be converted to this layer type.

`build(input_shape)`

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

`call(inputs)`

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config(config)`

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config()`

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

`property kernel_size`

beyondml.tflow.layers.MultiConv3D module

`class beyondml.tflow.layers.MultiConv3D.MultiConv3D(*args, **kwargs)`

Bases: `Layer`

Multitask 3-dimensional convolutional layer

This layer implements multiple stacks of convolutional weights to account for different ways individual neurons activate for various tasks. It is expected that to train using the RSN2 algorithm that MultiMaskedConv3D layers be used during training and then those layers be converted to this layer type.

`build(input_shape)`

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

`call(inputs)`

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

property kernel_size

beyondml.tflow.layers.MultiDense module

class beyondml.tflow.layers.MultiDense.**MultiDense**(**args*, ***kwargs*)

Bases: Layer

Multitask fully connected layer

This layer implements multiple stacks of fully connected weights to account for different ways neurons can activate for various tasks. It is expected that to train using the RSN2 algorithm that MultiMaskedDense layers be used during training and then those layers be converted to this layer type.

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config()`

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.MultiMaskedConv2D module

`class beyondml.tflow.layers.MultiMaskedConv2D.MultiMaskedConv2D(*args, **kwargs)`

Bases: `Layer`

Masked multitask 2-dimensional convolutional layer. This layer implements multiple stacks of the convolutional architecture and implements masking consistent with the BeyondML API to support developing sparse multitask models.

`build(input_shape)`

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

`call(inputs)`

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config(config)`

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

property kernel_size

set_masks(*new_masks*)

beyondml.tflow.layers.MultiMaskedConv3D module

class beyondml.tflow.layers.MultiMaskedConv3D.**MultiMaskedConv3D**(**args*, ***kwargs*)

Bases: Layer

Masked multitask 3-dimensional convoluational layer. This layer implements multiple stacks of the convolutional architecture and implements masking consistent with the BeyondML API to support developing sparse multitask models.

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

property kernel_size

set_masks(*new_masks*)

beyondml.tflow.layers.MultiMaskedDense module

class beyondml.tflow.layers.MultiMaskedDense.**MultiMaskedDense**(**args*, ***kwargs*)

Bases: Layer

Masked multitask fully connected layer. This layer implements multiple stacks of the fully-connected architecture and implements masking with the BeyondML API to support developing sparse multitask models.

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

`set_masks`(*new_masks*)

Set the masks for the layer

Parameters
new_masks (*list of arrays or array-likes*) – The new masks to set for the layer

beyondml.tflow.layers.MultiMaxPool2D module

`class` beyondml.tflow.layers.MultiMaxPool2D.**MultiMaxPool2D**(**args*, ***kwargs*)

Bases: `Layer`

Multitask Max Pooling Layer. This layer implements the Max Pooling algorithm across multiple inputs for developing multitask models

`call`(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config`(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config`()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.MultiMaxPool3D module

class beyondml.tflow.layers.MultiMaxPool3D.**MultiMaxPool3D**(**args*, ***kwargs*)

Bases: Layer

Multitask 3D Max Pooling Layer. This layer implements the Max Pooling algorithm across multiple inputs for developing multitask models

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.MultitaskNormalization module

class beyondml.tflow.layers.MultitaskNormalization.**MultitaskNormalization**(**args*, ***kwargs*)

Bases: Layer

Multitask layer which normalizes all inputs to sum to 1

build(*input_shape*)

Creates the variables of the layer (for subclass implementers).

This is a method that implementers of subclasses of *Layer* or *Model* can override if they need a state-creation step in-between layer instantiation and layer call. It is invoked automatically before the first execution of *call()*.

This is typically used to create the weights of *Layer* subclasses (at the discretion of the subclass implementer).

Parameters
input_shape – Instance of *TensorShape*, or list of instances of *TensorShape* if the layer expects a list of inputs (one instance per input).

`call`(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

`classmethod from_config`(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

`get_config`()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SelectorLayer module

`class` beyondml.tflow.layers.SelectorLayer.`SelectorLayer`(**args*, ***kwargs*)

Bases: `Layer`

Layer which selects individual inputs

Example:

```
>>> # Create a model with two inputs and one SelectorLayer
>>> input_1 = tf.keras.layers.Input(10)
>>> input_2 = tf.keras.layers.Input(10)
>>> selector = mann.layers.SelectorLayer(1)([input_1, input_2]) # 1 here indicates␣
```

(continues on next page)

(continued from previous page)

```
↪to select the second input and return it
>>> model = tf.keras.models.Model([input_1, input_2], selector)
>>> model.compile()
>>> # Call the model
>>> data1 = np.arange(10).reshape((1, 10))
>>> data2 = 2*np.arange(10).reshape((1, 10))
>>> model.predict([data1, data2])
array([[ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18.]], dtype=float32)
```

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

property sel_index

beyondml.tflow.layers.SparseConv2D module

class beyondml.tflow.layers.SparseConv2D.**SparseConv2D**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the Convolutional layer. If used in a model, must be saved and loaded via pickle

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SparseConv3D module

class beyondml.tflow.layers.SparseConv3D.**SparseConv3D**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the Convolutional layer. If used in a model, must be saved and loaded via pickle

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SparseDense module

class beyondml.tflow.layers.SparseDense.**SparseDense**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the Dense layer. If used in a model, must be saved and loaded via pickle

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SparseMultiConv2D module

class beyondml.tflow.layers.SparseMultiConv2D.**SparseMultiConv2D**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the MultiConv layer. If used in a model, must be saved and loaded via pickle

build(*input_shapes*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SparseMultiConv3D module

class beyondml.tflow.layers.SparseMultiConv3D.**SparseMultiConv3D**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the MultiConv layer. If used in a model, must be saved and loaded via pickle

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SparseMultiDense module

class beyondml.tflow.layers.SparseMultiDense.**SparseMultiDense**(**args*, ***kwargs*)

Bases: Layer

Sparse implementation of the MultiDense layer. If used in a model, must be saved and loaded via pickle

build(*input_shape*)

Build the layer in preparation to be trained or called. Should not be called directly, but rather is called when the layer is added to a model

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

classmethod from_layer(*layer*)

Create a layer from an instance of another layer

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

beyondml.tflow.layers.SumLayer module

class beyondml.tflow.layers.SumLayer.**SumLayer**(**args*, ***kwargs*)

Bases: Layer

Layer which adds all inputs together. All inputs must have compatible shapes

Example:

```
>>> # Create a model with just a SumLayer and two inputs
>>> input_1 = tf.keras.layers.Input(10)
>>> input_2 = tf.keras.layers.Input(10)
>>> sum_layer = mann.layers.SumLayer()([input_1, input_2])
>>> model = tf.keras.models.Model([input_1, input_2], sum_layer)
>>> model.compile()
>>> # Call the model
>>> data = np.arange(10).reshape((1, 10))
>>> model.predict([data, data])
array([[ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18.]], dtype=float32)
```

call(*inputs*)

This is where the layer's logic lives and is called upon inputs

Parameters
inputs (*TensorFlow Tensor or Tensor-like*) – The inputs to the layer

Returns
outputs – The outputs of the layer's logic

Return type
TensorFlow Tensor

classmethod from_config(*config*)

Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).

Parameters
config – A Python dictionary, typically the output of get_config.

Returns
A layer instance.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Note that *get_config()* does not guarantee to return a fresh copy of dict every time it is called. The callers should make a copy of the returned dict if they want to modify it.

Returns
Python dictionary.

Module contents

Custom layers to use when building MANN models

beyondml.tflow.utils package

Submodules

beyondml.tflow.utils.transformer module

beyondml.tflow.utils.transformer.**build_token_position_embedding_block**(*sequence_length*, *vocab_size*, *embed_dim*)

Builds a token and position embedding block

Parameters

- **sequence_length** (*int*) – The length of each sequence
- **vocab_size** (*int*) – The size of the vocabulary used
- **embed_dim** (*int*) – The desired embedding dimension

Returns

embedding_block – The embedding block, which can be used alone or as a layer in another model

Return type

TensorFlow keras Functional model

beyondml.tflow.utils.transformer.**build_transformer_block**(*input_shape*, *embed_dim*, *num_heads*, *neurons*, *dropout_rate=0.1*)

Build a Transformer Block

Parameters

- **input_shape** (*int or tuple of int*) – The input shape for the model to use
- **embed_dim** (*int*) – The dimension of the embedding
- **num_heads** (*int*) – The number of attention heads to use
- **neurons** (*int*) – The number of hidden neurons to use in the hidden layer
- **dropout_rate** (*float (default 0.1)*) – Rate at which dropout is applied
- **value_dim** (*int or None (default None)*) – The dimension to use for the *value* matrix, if provided

Returns

transformer_block – The transformer block, which can then be used alone or as a layer in another model

Return type

TensorFlow keras Functional model

beyondml.tflow.utils.utils module

class beyondml.tflow.utils.utils.**ActiveSparsification**(*performance_cutoff*, *performance_measure='auto'*, *starting_sparsification=None*, *max_sparsification=99*, *sparsification_rate=1*, *sparsification_patience=10*, *stopping_delta=0.01*, *stopping_patience=5*, *restore_best_weights=True*, *verbose=1*)

Bases: `Callback`

Keras-compatible callback object which enables active sparsification, allowing for increased sparsification as models train.

on_epoch_end(*epoch*, *logs=None*)

Called at the end of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **epoch** – Integer, index of epoch.
- **logs** – Dict, metric results for this training epoch, and for the validation epoch if validation is performed. Validation result keys are prefixed with *val_*. For training epoch, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_train_begin(*logs=None*)

Called at the beginning of training.

Subclasses should override for any actions to run.

Parameters

logs – Dict. Currently no data is passed to this argument for this method but that may change in the future.

beyondml.tflow.utils.utils.**add_layer_masks**(*model*, *additional_custom_objects=None*)

Convert a trained model from one that does not have masking weights to one that does have masking weights

Parameters

- **model** (*TensorFlow Keras model*) – The model to be converted
- **additional_custom_objects** (*dict or None (default None)*) – Additional custom layers to use

Returns

new_model – The converted model

Return type

TensorFlow Keras model

beyondml.tflow.utils.utils.**get_custom_objects**()

Return a dictionary of custom objects (layers) to use when loading models trained using this package

beyondml.tflow.utils.utils.**get_task_masking_gradients**(*model*, *task_num*)

Get the gradients of masking weights within a model

Parameters

model (*TensorFlow Keras model*) – The model to retrieve the gradients of

Notes

- **This function should only be run *before* the model has been trained**
  or used to predict. There is an unknown bug related to TensorFlow which is leading to incorrect results after initial training
- **When running this function, randomized input and output data is sent**
  through the model to retrieve gradients respective to each task. If the model is compiled using *sparse_categorical_crossentropy' loss, this will break this function's functionality. As a result, please use `categorical_crossentropy* (or even better, *mse*) before running this function. After retrieving gradients, the model can be recompiled with whatever parameters are desired.

Returns
gradients – The gradients of the masking weights of the model

Return type
list of TensorFlow tensors

beyondml.tflow.utils.utils.**mask_model**(*model*, *percentile*, *method='gradients'*, *exclusive=True*, *x=None*, *y=None*)

Mask the multitask model for training respective using the gradients for the tasks at hand

Parameters

- **model** (*keras model with MANN masking layers*) – The model to be masked
- **percentile** (*int*) – Percentile to use in masking. Any weights less than the *percentile* value will be made zero
- **method** (*str (default 'gradients')*) – One of either 'gradients' or 'magnitude' - the method for how to identify weights to mask If method is 'gradients', utilizes the gradients with respect to the passed x and y variables to identify the subnetwork to activate for each task If method is 'magnitude', uses the magnitude of the weights to identify the subnetwork to activate for each task
- **exclusive** (*bool (default True)*) – Whether to restrict previously-used weight indices for each task. If *True*, this identifies disjoint subsets of weights within the layer which perform the tasks requested.
- **x** (*list of np.ndarray or array-like*) – The training data input values, ignored if "method" is 'magnitude'
- **y** (*list of np.ndarray or array-like*) – The training data output values, ignored if "method" is 'magnitude'

beyondml.tflow.utils.utils.**mask_task_weights**(*model*, *task_masking_gradients*, *percentile*, *respect_previous_tasks=True*)

Parameters

- **model** (*TensorFlow Keras model*) – The model to be masked
- **task_masking_gradients** (*list of TensorFlow tensors*) – The gradients for the specific task requested
- **percentile** (*int*) – The percentile to mask/prune
- **respect_previous_tasks** (*bool (default True)*) – Whether to respect the weights used for previous tasks and not use them for subsequent tasks

Returns
masked_model – The masked model

Return type
TensorFlow Keras model

beyondml.tflow.utils.utils.**quantize_model**(*model*, *dtype='float16'*, *additional_custom_objects=None*)

Apply model quantization

Parameters

- **model** (*TensorFlow Keras Model*) – The model to quantize
- **dtype** (*str or TensorFlow datatype (default 'float16')*) – The datatype to quantize to
- **additional_custom_objects** (*None or dict (default None)*) – Additional custom objects to use to instantiate the model

Returns
new_model – The quantized model

Return type
TensorFlow Keras Model

beyondml.tflow.utils.utils.**replace_config**(*config*)

Replace the model config to remove masking layers

beyondml.tflow.utils.utils.**replace_weights**(*new_model*, *old_model*)

Replace the weights of a newly created model with the weights (sans masks) of an old model

beyondml.tflow.utils.utils.**train_model**(*model*, *train_x*, *train_y*, *loss*, *metrics*, *optimizer*, *cutoff*, *batch_size=32*, *epochs=100*, *starting_sparsification=0*, *max_sparsification=99*, *sparsification_rate=5*, *sparsification_patience=10*, *stopping_patience=5*)

beyondml.tflow.utils.utils.**train_model_iteratively**(*model*, *task_gradients*, *train_x*, *train_y*, *validation_split*, *delta*, *batch_size*, *losses*, *optimizer='adam'*, *metrics=None*, *starting_pruning=0*, *pruning_rate=10*, *patience=5*, *max_epochs=100*)

Train a model iteratively on each task, first obtaining baseline performance on each task and then iteratively training and pruning each task as far back as possible while maintaining acceptable performance on each task

Parameters

- **model** (*TensorFlow Keras model*) – The model to be trained
- **task_gradients** (*list of TensorFlow tensors*) – Gradients for each task, output from the *get_task_masking_gradients* function
- **train_x** (*list of numpy arrays, TensorFlow Datasets, or other*) – data types models can train with The input data to use to train on
- **train_y** (*list of numpy arrays, TensorFlow Datasets, or other*) – data types model can train with The output data to use to train on
- **validation_split** (*float, or list of float*) – The proportion of data to use for validation
- **delta** (*float*) – The tolerance between validation losses to be considered "acceptable" performance to continue

- **batch_size** (*int*) – The batch size to train with
- **losses** (*str, list, or Keras loss function*) – The loss or losses to use when training
- **optimizer** (*str, list, or Keras optimizer*) – The optimizer to use when training (default 'adam')
- **starting_pruning** (*int or list of int (default 0)*) – The starting pruning rate to use for each task
- **pruning_rate** (*int or list of int (default [10, 5, 2, 1])*) – The pruning rate to use
- **patience** (*int (default 5)*) – The patience for number of epochs to wait for performance to improve sufficiently
- **max_epochs** (*int or list of int (default 100)*) – The maximum number of epochs to use for training each task

Module contents

Some utilities to use when building, loading, and training MANN models

Module contents

TensorFlow compatibility for building MANN models.

The *beyondml.tflow* package contains two subpackages, *beyondml.tflow.layers* and *beyondml.tflow.utils*, which contain the functionality to create and train MANN layers within TensorFlow. For individuals who are familiar with the former name of this package, *mann*, backwards compatibility can be achieved (assuming only TensorFlow support is needed), by replacing the following line of code:

```
>>> import mann
```

with the following line:

```
>>> import beyondml.tflow as mann
```

in all existing scripts.

Within the *layers* package, there is current functionality for the the following layers: - *beyondml.tflow.layers.FilterLayer* - *beyondml.tflow.layers.MaskedConv2D* - *beyondml.tflow.layers.MaskedDense* - *beyondml.tflow.layers.MultiConv2D* - *beyondml.tflow.layers.MultiDense* - *beyondml.tflow.layers.MultiMaskedConv2D* - *beyondml.tflow.layers.MultiMaskedDense* - *beyondml.tflow.layers.MultiMaxPool2D* - *beyondml.tflow.layers.SelectorLayer* - *beyondml.tflow.layers.SumLayer* - *beyondml.tflow.layers.SparseDense* - *beyondml.tflow.layers.SparseConv* - *beyondml.tflow.layers.SparseMultiDense* - *beyondml.tflow.layers.SparseMultiConv*

Note that with any of the sparse layers (such as the `SparseDense` layer), any model which utilizes these layers will not be loadable using the traditional `load_model` functions available in TensorFlow. Instead, the model should be saved using either joblib or pickle.

Within the *utils* package, there are the current functions and classes: - *ActiveSparsification* - *build_transformer_block* - *build_token_position_embedding_block* - *get_custom_objects* - *mask_model* - *remove_layer_masks* - *add_layer_masks* - *quantize_model* - *get_task_masking_gradients* - *mask_task_weights* - *train_model_iteratively*

Module contents

BeyondML (formerly MANN) is a Python package which enables creating sparse multitask artificial neural networks (MANNs) compatible with [TensorFlow](https://tensorflow.org) and [PyTorch](https://pytorch.org). This package contains custom layers and utilities to facilitate the training and optimization of models using the Reduction of Sub-Network Neuroplasticity (RSN2) training procedure developed by [AI Squared, Inc](https://squared.ai).

Installation

This package is available through [PyPi](https://pypi.org) and can be installed via the following command:

```bash pip install beyondml ```

Capabilities

There are two major subpackages within the BeyondML package, the *beyondml.tflow* and the *beyondml.pt* packages. The *beyondml.tflow* package contains functionality for building multitask models using TensorFlow, and the *beyondml.pt* package contains functionality for building multitask models using PyTorch.
```

CHAPTER

TWO

CHANGELOG

- **Version 0.1.0**
  - Refactored existing MANN repository to rename to BeyondML
- **Version 0.1.1**
  - **Added the *SparseDense*, *SparseConv*, *SparseMultiDense*, and *SparseMultiConv* layers to** *beyondml.tflow.layers*, giving users the functionality to utilize sparse tensors during inference
- **Version 0.1.2**
  - Added the *MaskedMultiHeadAttention*, *MaskedTransformerEncoderLayer*, and *MaskedTransformerDecoderLayer* layers to *beyondml.pt.layers* to add pruning to the transformer architecture
  - Added *MaskedConv3D*, *MultiMaskedConv3D*, *MultiConv3D*, *MultiMaxPool3D*, *SparseConv3D*, and *SparseMultiConv3D* layers to *beyondml.tflow.layers*
  - Added *MaskedConv3D*, *MultiMaskedConv3D*, *MultiConv3D*, *MultiMaxPool3D*, *SparseConv3D*, *SparseMultiConv3D*, and *MultiMaxPool2D* layers to *beyondml.pt.layers*
- **Version 0.1.3**
  - Added *beyondml.pt* compatibility with more native PyTorch functionality for using models on different devices and datatypes
  - Added *train_model* function to *beyondml.tflow.utils*
  - Added *MultitaskNormalization* layer to *beyondml.tflow.layers* and *beyondml.pt.layers*
- **Version 0.1.4**
  - Updated documentation to use Sphinx
- **Version 0.1.5**
  - Updated requirements to use newer version of TensorFlow
  - Fixed errors with changes to types of *input_shape* in TensorFlow Keras layers
  - Fixed errors resulting from model/configuration changes with TensorFlow
- **Version 0.1.6**
  - Fixed issues with converting between masked and unmasked models in TensorFlow
- **Version 0.1.7**
  - Updated Pytorch implementation of Transformer-based architectures

PYTHON MODULE INDEX

b

INDEX

C

D

F

O

P

Q

R

S

T