Memory Fusion Network Documentation

Contents

Module `memory_fusion_network`

Functions

Function `MFN`

```
def MFN(
    input_shapes,
    output_classes,
    mem_size=256
)
```

This is just a function to abstract between the number of modalities to be provided to MFN. MFN can be extended to an arbitrary number of modalities. Here implementation of upto 3 modalities has been done.

Args —-= **`input_shapes`** : list of length atmost 3 containing tuples of (seq_length, feature_size) for upto 3 modalities

`output_classes` the number of output classes
`mem_size` the size of states of MultiviewGatedMemory

Returns —-= an instance of keras Model class with the required architecture defined.

Function `MFN_bimodal`

```
def MFN_bimodal(
    input_shapes,
    output_classes,
    mem_size=256
)
```

Returns the bimodal version of the Memory Fusion model.

Args —-= **`input_shapes`** : list of length 2 containing tuples of (seq_length, feature_size) for 2 modalities

`output_classes` the number of output classes

mem_size the size of states of MultiviewGatedMemory

Returns —-= an instance of keras Model class with the required architecture defined.

Function MFN_trimodal

```
def MFN_trimodal(
    input_shapes,
    output_classes,
    mem_size=256
)
```

Returns the trimodal version of the Memory Fusion model.

Args —-= **input_shapes** : list of length 3 containing tuples of (seq_length, feature_size) for 3 modalities

output_classes the number of output classes
mem_size the size of states of MultiviewGatedMemory

Returns —-= an instance of keras Model class with the required architecture defined.

Function MFN_unimodal

```
def MFN_unimodal(
    input_shapes,
    output_classes,
    mem_size=256
)
```

Returns the unimodal version of the Memory Fusion model.

Args —-= **input_shapes** : list of length 1 containing a tuple of (seq_length, feature_size) for one modality

output_classes the number of output classes
mem_size the size of states of MultiviewGatedMemory

Returns —-= an instance of keras Model class with the required architecture defined.

Function customLSTM

```
def customLSTM(
    dim
)
```

RNN applied over an implemented LSTM cell to output a custom LSTM which can be used as a Keras Layer.

Args —-= **dim** : defines the state size for the LSTM

Returns —-= LSTM layer formed from Mod_LSTMCELL

Classes

Class Mod_LSTMCELL

```
class Mod_LSTMCELL(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
```

```
        kernel_constraint=None,
        recurrent_constraint=None,
        bias_constraint=None,
        dropout=0.0,
        recurrent_dropout=0.0,
        implementation=2,
        **kwargs
    )
```

Cell class for the LSTM layer. Following Arguments are present in the parent class LSTMCell, which have been used in this implementation. Reimplementation is done so that we can extract the complete hidden sequence h,c which cannot be obtained directly using the native LSTM implementation.

Args —-= **`units`** : Positive integer, dimensionality of the output space.

`activation` Activation function to use. Default: hyperbolic tangent (tanh). If you pass None, no activation is applied (ie. "linear" activation: `a(x) = x`).

`recurrent_activation` Activation function to use for the recurrent step. Default: sigmoid (sigmoid). If you pass None, no activation is applied (ie. "linear" activation: `a(x) = x`).

`use_bias` Boolean, (default True), whether the layer uses a bias vector.

`kernel_initializer` Initializer for the kernel weights matrix, used for the linear transformation of the inputs. Default: glorot_uniform. recurrent_initializer: Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state. Default: orthogonal.

`bias_initializer` Initializer for the bias vector. Default: zeros.

`unit_forget_bias` Boolean (default True). If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in Jozefowicz et al.

`kernel_regularizer` Regularizer function applied to the kernel weights matrix. Default: None.

`recurrent_regularizer` Regularizer function applied to the recurrent_kernel weights matrix. Default: None.

`bias_regularizer` Regularizer function applied to the bias vector. Default: None.

`kernel_constraint` Constraint function applied to the kernel weights matrix. Default: None.

`recurrent_constraint` Constraint function applied to the recurrent_kernel weights matrix. Default: None.

`bias_constraint` Constraint function applied to the bias vector. Default: None.

`dropout` Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs. Default: 0.

`recurrent_dropout` Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state. Default: 0.

`implementation` Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 (default) will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications. Default: 2.

Ancestors (in MRO)

- tensorflow.python.keras.layers.recurrent_v2.LSTMCell
- tensorflow.python.keras.layers.recurrent.LSTMCell
- tensorflow.python.keras.layers.recurrent.DropoutRNNCellMixin
- tensorflow.python.keras.engine.base_layer.Layer
- tensorflow.python.module.module.Module
- tensorflow.python.training.tracking.tracking.AutoTrackable
- tensorflow.python.training.tracking.base.Trackable
- tensorflow.python.keras.utils.version_utils.LayerVersionSelector

Methods

Method `call`

```
    def call(
        self,
        inputs,
        states,
```

```
    training=None
)
```

The function that contains the logic for one RNN step calculation.

Args —-= **inputs** : the input tensor, which is a slide from the overall RNN input by the time dimension (usually the second dimension).

states the state tensor from previous step, which has the same shape as (batch, state_size). In the case of timestep 0, it will be the initial state user specified, or zero filled tensor otherwise.

training Python boolean indicating whether the layer should behave in training mode or in inference mode. Only relevant when dropout or recurrent_dropout is used.

Returns —-= A tuple of two tensors: 1. output tensor for the current timestep, with size output_size. 2. state tensor for next step, which has the shape of state_size.

Class `MultiviewGatedMemory`

```
class MultiviewGatedMemory(
    mem_dim,
    **kwargs
)
```

Multi-view Gated Memory is the neural component that stores a history of cross-view interactions over time. It acts as a unifying memory for the memories in System of LSTMs. This class has been implemented as a keras Layer and overloads the functions of the Layer class. state_size: size(s) of state(s) used by this cell. It can be represented by an Integer, a TensorShape or a tuple of Integers or TensorShapes.

Ancestors (in MRO)

- tensorflow.python.keras.engine.base_layer.Layer
- tensorflow.python.module.module.Module
- tensorflow.python.training.tracking.tracking.AutoTrackable
- tensorflow.python.training.tracking.base.Trackable
- tensorflow.python.keras.utils.version_utils.LayerVersionSelector

Methods

Method `build`

```
def build(
    self,
    input_shape
)
```

Creates the variables of the layer (optional, 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. This is typically used to create the weights of Layer subclasses.

Args —-= **input_shape** : Instance of TensorShape, or list of instances of TensorShape if the layer expects a list of inputs (one instance per input).

Method `call`

```
def call(
    self,
    inputs,
    states
)
```

The function that contains the logic for one RNN step calculation.

Args —-= **inputs** : the input tensor, which is a slide from the overall RNN input by the time dimension (usually the second dimension).

states the state tensor from previous step, which has the same shape as (batch, state_size). In the case of timestep 0, it will be the initial state user specified, or zero filled tensor otherwise.

Returns —-= A tuple of two tensors: 1. output tensor for the current timestep, with size output_size. 2. state tensor for next step, which has the shape of state_size.

Method `get_config`

```
def get_config(
    self
)
```

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).

Returns —-= Python dictionary.
