Recurrent

[source]

```
keras.layers.recurrent.Recurrent(weights=None, return_sequences=False, go_backwards=False, stat
```

Abstract base class for recurrent layers. Do not use in a model -- it's not a valid layer! Use its children classes `LSTM`, `GRU` and `SimpleRNN` instead.

All recurrent layers (`LSTM`, `GRU`, `SimpleRNN`) also follow the specifications of this class and accept the keyword arguments listed below.

Example

```
# as the first layer in a Sequential model
model = Sequential()
model.add(LSTM(32, input_shape=(10, 64)))
# now model.output_shape == (None, 32)
# note: `None` is the batch dimension.

# the following is identical:
model = Sequential()
model.add(LSTM(32, input_dim=64, input_length=10))

# for subsequent layers, not need to specify the input size:
model.add(LSTM(16))
```

Arguments

- **weights**: list of Numpy arrays to set as initial weights. The list should have 3 elements, of shapes: `[(input_dim, output_dim), (output_dim, output_dim), (output_dim,)]`.
- **return_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **go_backwards**: Boolean (default False). If True, process the input sequence backwards.
- **stateful**: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **unroll**: Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. When using TensorFlow, the network is always unrolled, so this argument does not do anything. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.
- **consume_less**: one of "cpu", "mem", or "gpu" (LSTM/GRU only). If set to "cpu", the RNN will use an implementation that uses fewer, larger matrix products, thus running faster on CPU but consuming more memory. If set to "mem", the RNN will use more matrix products, but smaller ones, thus running slower (may actually be faster on GPU) while consuming less memory. If set to "gpu" (LSTM/GRU only), the RNN

will combine the input gate, the forget gate and the output gate into a single matrix, enabling more time-efficient parallelization on the GPU. Note: RNN dropout must be shared for all gates, resulting in a slightly reduced regularization.

- **input_dim**: dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.
- **input_length**: Length of input sequences, to be specified when it is constant. This argument is required if you are going to connect `Flatten` then `Dense` layers upstream (without it, the shape of the dense outputs cannot be computed). Note that if the recurrent layer is not the first layer in your model, you would need to specify the input length at the level of the first layer (e.g. via the `input_shape` argument)

Input shape

3D tensor with shape `(nb_samples, timesteps, input_dim)`.

Output shape

- if `return_sequences`: 3D tensor with shape `(nb_samples, timesteps, output_dim)`.
- else, 2D tensor with shape `(nb_samples, output_dim)`.

Masking

This layer supports masking for input data with a variable number of timesteps. To introduce masks to your data, use an Embedding layer with the `mask_zero` parameter set to `True`.

TensorFlow warning

For the time being, when using the TensorFlow backend, the number of timesteps used must be specified in your model. Make sure to pass an `input_length` int argument to your recurrent layer (if it comes first in your model), or to pass a complete `input_shape` argument to the first layer in your model otherwise.

Note on using statefulness in RNNs

You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness: - specify `stateful=True` in the layer constructor. - specify a fixed batch size for your model, by passing a `batch_input_shape=(...)` to the first layer in your model. This is the expected shape of your inputs *including the batch size*. It should be a tuple of integers, e.g. `(32, 10, 100)`.

To reset the states of your model, call `.reset_states()` on either a specific layer, or on your entire model.

Note on using dropout with TensorFlow

When using the TensorFlow backend, specify a fixed batch size for your model following the notes on statefulness RNNs.

SimpleRNN

[source]

```
keras.layers.recurrent.SimpleRNN(output_dim, init='glorot_uniform', inner_init='orthogonal', ac
```

Fully-connected RNN where the output is to be fed back to input.

Arguments

- **output_dim**: dimension of the internal projections and the final output.
- **init**: weight initialization function. Can be the name of an existing function (str), or a Theano function (see: initializations).
- **inner_init**: initialization function of the inner cells.
- **activation**: activation function. Can be the name of an existing function (str), or a Theano function (see: activations).
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the input weights matrices.
- **U_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the recurrent weights matrices.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **dropout_W**: float between 0 and 1. Fraction of the input units to drop for input gates.
- **dropout_U**: float between 0 and 1. Fraction of the input units to drop for recurrent connections.

References

- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

---

GRU

[source]

```
keras.layers.recurrent.GRU(output_dim, init='glorot_uniform', inner_init='orthogonal', activati
```

Gated Recurrent Unit - Cho et al. 2014.

Arguments

- **output_dim**: dimension of the internal projections and the final output.
- **init**: weight initialization function. Can be the name of an existing function (str), or a Theano function (see: initializations).
- **inner_init**: initialization function of the inner cells.
- **activation**: activation function. Can be the name of an existing function (str), or a Theano function (see: activations).
- **inner_activation**: activation function for the inner cells.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the input weights matrices.
- **U_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the recurrent weights matrices.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **dropout_W**: float between 0 and 1. Fraction of the input units to drop for input gates.
- **dropout_U**: float between 0 and 1. Fraction of the input units to drop for recurrent connections.

References

- On the Properties of Neural Machine Translation: Encoder–Decoder Approaches
- Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

---

LSTM

[source]

```
keras.layers.recurrent.LSTM(output_dim, init='glorot_uniform', inner_init='orthogonal', forget_
```

Long-Short Term Memory unit - Hochreiter 1997.

For a step-by-step description of the algorithm, see this tutorial.

Arguments

- **output_dim**: dimension of the internal projections and the final output.
- **init**: weight initialization function. Can be the name of an existing function (str), or a Theano function (see: initializations).
- **inner_init**: initialization function of the inner cells.
- **forget_bias_init**: initialization function for the bias of the forget gate. Jozefowicz et al. recommend initializing with ones.
- **activation**: activation function. Can be the name of an existing function (str), or a Theano function (see: activations).

- **inner_activation**: activation function for the inner cells.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the input weights matrices.
- **U_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the recurrent weights matrices.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **dropout_W**: float between 0 and 1. Fraction of the input units to drop for input gates.
- **dropout_U**: float between 0 and 1. Fraction of the input units to drop for recurrent connections.

References

- Long short-term memory (original 1997 paper)
- Learning to forget: Continual prediction with LSTM
- Supervised sequence labelling with recurrent neural networks
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks