Networks

The networks module contains pieces of neural network that combine multiple layers.

NLP

sequence_conv_pool

`paddle.trainer_config_helpers.networks.`**`sequence_conv_pool`**(**args*, ***kwargs*)

Text convolution pooling layers helper.

Text input => Context Projection => FC Layer => Pooling => Output.

Parameters:
- **name** (*basestring*) – name of output layer(pooling layer name)
- **input** (*LayerOutput*) – name of input layer
- **context_len** (*int*) – context projection length. See context_projection's document.
- **hidden_size** (*int*) – FC Layer size.
- **context_start** (*int or None*) – context projection length. See context_projection's context_start.
- **pool_type** (*BasePoolingType.*) – pooling layer type. See pooling_layer's document.
- **context_proj_layer_name** (*basestring*) – context projection layer name. None if user don't care.
- **context_proj_param_attr** (*ParameterAttribute or None.*) – context projection parameter attribute. None if user don't care.
- **fc_layer_name** (*basestring*) – fc layer name. None if user don't care.
- **fc_param_attr** (*ParameterAttribute or None*) – fc layer parameter attribute. None if user don't care.
- **fc_bias_attr** (*ParameterAttribute or None*) – fc bias parameter attribute. False if no bias, None if user don't care.
- **fc_act** (*BaseActivation*) – fc layer activation type. None means tanh
- **pool_bias_attr** (*ParameterAttribute or None.*) – pooling layer bias attr. None if don't care. False if no bias.
- **fc_attr** (*ExtraLayerAttribute*) – fc layer extra attribute.
- **context_attr** (*ExtraLayerAttribute*) – context projection layer extra attribute.
- **pool_attr** (*ExtraLayerAttribute*) – pooling layer extra attribute.

Returns: output layer name.

Return type: LayerOutput

text_conv_pool

`paddle.trainer_config_helpers.networks.`**`text_conv_pool`**(**args*, ***kwargs*)

Text convolution pooling layers helper.

Text input => Context Projection => FC Layer => Pooling => Output.

Parameters:
- **name** (*basestring*) – name of output layer(pooling layer name)
- **input** (*LayerOutput*) – name of input layer
- **context_len** (*int*) – context projection length. See context_projection's document.
- **hidden_size** (*int*) – FC Layer size.
- **context_start** (*int or None*) – context projection length. See context_projection's context_start.
- **pool_type** (*BasePoolingType.*) – pooling layer type. See pooling_layer's document.
- **context_proj_layer_name** (*basestring*) – context projection layer name. None if user don't care.
- **context_proj_param_attr** (*ParameterAttribute or None.*) – context projection parameter attribute. None if user don't care.
- **fc_layer_name** (*basestring*) – fc layer name. None if user don't care.
- **fc_param_attr** (*ParameterAttribute or None*) – fc layer parameter attribute. None if user don't care.
- **fc_bias_attr** (*ParameterAttribute or None*) – fc bias parameter attribute. False if no bias, None if user don't care.
- **fc_act** (*BaseActivation*) – fc layer activation type. None means tanh
- **pool_bias_attr** (*ParameterAttribute or None.*) – pooling layer bias attr. None if don't care. False if no bias.
- **fc_attr** (*ExtraLayerAttribute*) – fc layer extra attribute.
- **context_attr** (*ExtraLayerAttribute*) – context projection layer extra attribute.
- **pool_attr** (*ExtraLayerAttribute*) – pooling layer extra attribute.

Returns: output layer name.

Return type: LayerOutput

Images

img_conv_bn_pool

`paddle.trainer_config_helpers.networks.`**`img_conv_bn_pool`**(**args*, ***kwargs*)

Convolution, batch normalization, pooling group.

Parameters:
- **name** (*basestring*) – group name
- **input** (*LayerOutput*) – layer's input
- **filter_size** (*int*) – see img_conv_layer's document
- **num_filters** (*int*) – see img_conv_layer's document
- **pool_size** (*int*) – see img_pool_layer's document.
- **pool_type** (*BasePoolingType*) – see img_pool_layer's document.
- **act** (*BaseActivation*) – see batch_norm_layer's document.
- **groups** (*int*) – see img_conv_layer's document
- **conv_stride** (*int*) – see img_conv_layer's document.
- **conv_padding** (*int*) – see img_conv_layer's document.
- **conv_bias_attr** (*ParameterAttribute*) – see img_conv_layer's document.
- **num_channel** (*int*) – see img_conv_layer's document.

- **conv_param_attr** (*ParameterAttribute*) – see img_conv_layer's document.
- **shared_bias** (*bool*) – see img_conv_layer's document.
- **conv_layer_attr** (*ExtraLayerOutput*) – see img_conv_layer's document.
- **bn_param_attr** (*ParameterAttribute.*) – see batch_norm_layer's document.
- **bn_bias_attr** – see batch_norm_layer's document.
- **bn_layer_attr** – ParameterAttribute.
- **pool_stride** (*int*) – see img_pool_layer's document.
- **pool_padding** (*int*) – see img_pool_layer's document.
- **pool_layer_attr** (*ExtraLayerAttribute*) – see img_pool_layer's document.

Returns: Layer groups output

Return type: LayerOutput

img_conv_group

`paddle.trainer_config_helpers.networks.`**`img_conv_group`**(**args*, ***kwargs*)

Image Convolution Group, Used for vgg net.

TODO(yuyang18): Complete docs

Parameters:

- **conv_batchnorm_drop_rate** –
- **input** –
- **conv_num_filter** –
- **pool_size** –
- **num_channels** –
- **conv_padding** –
- **conv_filter_size** –
- **conv_act** –
- **conv_with_batchnorm** –
- **pool_stride** –
- **pool_type** –

Returns:

simple_img_conv_pool

`paddle.trainer_config_helpers.networks.`**`simple_img_conv_pool`**(**args*, ***kwargs*)

Simple image convolution and pooling group.

Input => conv => pooling

Parameters:

- **name** (*basestring*) – group name
- **input** (*LayerOutput*) – input layer name.
- **filter_size** (*int*) – see img_conv_layer for details
- **num_filters** (*int*) – see img_conv_layer for details
- **pool_size** (*int*) – see img_pool_layer for details
- **pool_type** (*BasePoolingType*) – see img_pool_layer for details

- **act** (*BaseActivation*) – see img_conv_layer for details
- **groups** (*int*) – see img_conv_layer for details
- **conv_stride** (*int*) – see img_conv_layer for details
- **conv_padding** (*int*) – see img_conv_layer for details
- **bias_attr** (*ParameterAttribute*) – see img_conv_layer for details
- **num_channel** (*int*) – see img_conv_layer for details
- **param_attr** (*ParameterAttribute*) – see img_conv_layer for details
- **shared_bias** (*bool*) – see img_conv_layer for details
- **conv_layer_attr** (*ExtraLayerAttribute*) – see img_conv_layer for details
- **pool_stride** (*int*) – see img_pool_layer for details
- **pool_padding** (*int*) – see img_pool_layer for details
- **pool_layer_attr** (*ExtraLayerAttribute*) – see img_pool_layer for details

Returns: Layer's output

Return type: LayerOutput

vgg_16_network

`paddle.trainer_config_helpers.networks.`**`vgg_16_network`**(*input_image*, *num_channels*, *num_classes=1000*)

Same model from https://gist.github.com/ksimonyan/211839e770f7b538e2d8

Parameters:
- **num_classes** –
- **input_image** (*LayerOutput*) –
- **num_channels** (*int*) –

Returns:

Recurrent

LSTM

lstmemory_unit

`paddle.trainer_config_helpers.networks.`**`lstmemory_unit`**(**args*, ***kwargs*)

Define calculations that a LSTM unit performs in a single time step. This function itself is not a recurrent layer, so that it can not be directly applied to sequence input. This function is always used in recurrent_group (see layers.py for more details) to implement attention mechanism.

Please refer to **Generating Sequences With Recurrent Neural Networks** for more details about LSTM. The link goes as follows: .. _Link: https://arxiv.org/abs/1308.0850

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t tanh(c_t)
\end{aligned}
$$

The example usage is:

```
lstm_step = lstmemory_unit(input=[layer1],
                           size=256,
                           act=TanhActivation(),
                           gate_act=SigmoidActivation(),
                           state_act=TanhActivation())
```

Parameters:

- **input** (*LayerOutput*) – input layer name.
- **name** (*basestring*) – lstmemory unit name.
- **size** (*int*) – lstmemory unit size.
- **param_attr** (*ParameterAttribute*) – Parameter config, None if use default.
- **act** (*BaseActivation*) – lstm final activiation type
- **gate_act** (*BaseActivation*) – lstm gate activiation type
- **state_act** (*BaseActivation*) – lstm state activiation type.
- **mixed_bias_attr** (*ParameterAttribute|False*) – bias parameter attribute of mixed layer. False means no bias, None means default bias.
- **lstm_bias_attr** (*ParameterAttribute|False*) – bias parameter attribute of lstm layer. False means no bias, None means default bias.
- **mixed_layer_attr** (*ExtraLayerAttribute*) – mixed layer's extra attribute.
- **lstm_layer_attr** (*ExtraLayerAttribute*) – lstm layer's extra attribute.
- **get_output_layer_attr** (*ExtraLayerAttribute*) – get output layer's extra attribute.

Returns: lstmemory unit name.

Return type: LayerOutput

lstmemory_group

`paddle.trainer_config_helpers.networks.`**`lstmemory_group`**(**args*, ***kwargs*)

lstm_group is a recurrent layer group version Long Short Term Memory. It does exactly the same calculation as the lstmemory layer (see lstmemory in layers.py for the maths) does. A promising benefit is that LSTM memory cell states, or hidden states in every time step are accessible to for the user. This is especially useful in attention model. If you do not need to access to the internal states of the lstm, but merely use its outputs, it is recommended to use the lstmemory, which is relatively faster than lstmemory_group.

NOTE: In PaddlePaddle's implementation, the following input-to-hidden multiplications: $W_{xi}x_t$, $W_{xf}x_t$, $W_{xc}x_t$, $W_{xo}x_t$ are not done in lstmemory_unit to speed up the calculations. Consequently, an additional mixed_layer with full_matrix_projection must be included before lstmemory_unit is called.

The example usage is:

```
lstm_step = lstmemory_group(input=[layer1],
                            size=256,
                            act=TanhActivation(),
                            gate_act=SigmoidActivation(),
                            state_act=TanhActivation())
```

Parameters:
- **input** (*LayerOutput*) – input layer name.
- **name** (*basestring*) – lstmemory group name.
- **size** (*int*) – lstmemory group size.
- **reverse** (*bool*) – is lstm reversed
- **param_attr** (*ParameterAttribute*) – Parameter config, None if use default.
- **act** (*BaseActivation*) – lstm final activiation type
- **gate_act** (*BaseActivation*) – lstm gate activiation type
- **state_act** (*BaseActivation*) – lstm state activiation type.
- **mixed_bias_attr** (*ParameterAttribute|False*) – bias parameter attribute of mixed layer. False means no bias, None means default bias.
- **lstm_bias_attr** (*ParameterAttribute|False*) – bias parameter attribute of lstm layer. False means no bias, None means default bias.
- **mixed_layer_attr** (*ExtraLayerAttribute*) – mixed layer's extra attribute.
- **lstm_layer_attr** (*ExtraLayerAttribute*) – lstm layer's extra attribute.
- **get_output_layer_attr** (*ExtraLayerAttribute*) – get output layer's extra attribute.

Returns: the lstmemory group.

Return type: LayerOutput

simple_lstm

`paddle.trainer_config_helpers.networks.`**`simple_lstm`**(**args*, ***kwargs*)

Simple LSTM Cell.

It just combine a mixed layer with fully_matrix_projection and a lstmemory layer. The simple lstm cell was implemented as follow equations.

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)\\
c_t &= f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)\\
h_t &= o_t tanh(c_t)
\end{aligned}
$$

Please refer **Generating Sequences With Recurrent Neural Networks** if you want to know what lstm is. Link is here.

Parameters:
- **name** (*basestring*) – lstm layer name.
- **input** (*LayerOutput*) – input layer name.
- **size** (*int*) – lstm layer size.

- **reverse** (*bool*) – whether to process the input data in a reverse order
- **mat_param_attr** (*ParameterAttribute*) – mixed layer's matrix projection parameter attribute.
- **bias_param_attr** (*ParameterAttribute|False*) – bias parameter attribute. False means no bias, None means default bias.
- **inner_param_attr** (*ParameterAttribute*) – lstm cell parameter attribute.
- **act** (*BaseActivation*) – lstm final activiation type
- **gate_act** (*BaseActivation*) – lstm gate activiation type
- **state_act** (*BaseActivation*) – lstm state activiation type.
- **mixed_layer_attr** (*ExtraLayerAttribute*) – mixed layer's extra attribute.
- **lstm_cell_attr** (*ExtraLayerAttribute*) – lstm layer's extra attribute.

Returns: lstm layer name.

Return type: LayerOutput

bidirectional_lstm

`paddle.trainer_config_helpers.networks.`**`bidirectional_lstm`**(**args*, ***kwargs*)

A bidirectional_lstm is a recurrent unit that iterates over the input sequence both in forward and bardward orders, and then concatenate two outputs form a final output. However, concatenation of two outputs is not the only way to form the final output, you can also, for example, just add them together.

Please refer to **Neural Machine Translation by Jointly Learning to Align and Translate** for more details about the bidirectional lstm. The link goes as follows: .. _Link: https://arxiv.org/pdf/1409.0473v3.pdf

The example usage is:

```
bi_lstm = bidirectional_lstm(input=[input1], size=512)
```

Parameters:
- **name** (*basestring*) – bidirectional lstm layer name.
- **input** (*LayerOutput*) – input layer.
- **size** (*int*) – lstm layer size.
- **return_seq** (*bool*) – If set False, outputs of the last time step are concatenated and returned. If set True, the entire output sequences that are processed in forward and backward directions are concatenated and returned.

Returns: LayerOutput object accroding to the return_seq.

Return type: LayerOutput

GRU

gru_unit

`paddle.trainer_config_helpers.networks.`**`gru_unit`**(**args*, ***kwargs*)

Define calculations that a gated recurrent unit performs in a single time step. This function itself is not a recurrent layer, so that it can not be directly applied to sequence input. This function is almost always used in the recurrent_group (see layers.py for more details) to implement attention mechanism.

Please see grumemory in layers.py for the details about the maths.

Parameters:

- **input** (*LayerOutput*) – input layer name.
- **name** (*basestring*) – name of the gru group.
- **size** (*int*) – hidden size of the gru.
- **act** (*BaseActivation*) – type of the activation
- **gate_act** (*BaseActivation*) – type of the gate activation
- **gru_layer_attr** (*ParameterAttribute|False*) – Extra parameter attribute of the gru layer.

Returns: the gru output layer.

Return type: LayerOutput

gru_group

paddle.trainer_config_helpers.networks.**gru_group**(**args*, ***kwargs*)

gru_group is a recurrent layer group version Gated Recurrent Unit. It does exactly the same calculation as the grumemory layer does. A promising benefit is that gru hidden sates are accessible to for the user. This is especially useful in attention model. If you do not need to access to any internal state, but merely use the outputs of a GRU, it is recommanded to use the grumemory, which is relatively faster.

Please see grumemory in layers.py for more detail about the maths.

The example usage is:

```
gru = gur_group(input=[layer1],
                size=256,
                act=TanhActivation(),
                gate_act=SigmoidActivation())
```

Parameters:

- **input** (*LayerOutput*) – input layer name.
- **name** (*basestring*) – name of the gru group.
- **size** (*int*) – hidden size of the gru.
- **reverse** (*bool*) – whether to process the input data in a reverse order
- **act** (*BaseActivation*) – type of the activiation
- **gate_act** (*BaseActivation*) – type of the gate activiation
- **gru_bias_attr** (*ParameterAttribute|False*) – bias. False means no bias, None means default bias.
- **gru_layer_attr** (*ParameterAttribute|False*) – Extra parameter attribute of the gru layer.

Returns: the gru group.

Return type: LayerOutput

simple_gru

paddle.trainer_config_helpers.networks.**simple_gru**(**args*, ***kwargs*)

You maybe see gru_step_layer, grumemory in layers.py, gru_unit, gru_group, simple_gru in network.py. The reason why there are so many interfaces is that we have two ways to implement recurrent neural network. One way is to use one complete layer to implement rnn (including simple rnn, gru and lstm) with multiple time steps, such as recurrent_layer, lstmemory, grumemory. But, the multiplication operation Wx_t is not computed in these layers. See details in their interfaces in layers.py. The other implementation is to use an recurrent group which can ensemble a series of layers to compute rnn step by step. This way is flexible for attenion mechanism or other complex connections.

- gru_step_layer: only compute rnn by one step. It needs an memory as input and can be used in recurrent group.
- gru_unit: a wrapper of gru_step_layer with memory.
- gru_group: a GRU cell implemented by a combination of multiple layers in recurrent group. But Wx_t is not done in group.
- gru_memory: a GRU cell implemented by one layer, which does same calculation with gru_group and is faster than gru_group.
- simple_gru: a complete GRU implementation inlcuding Wx_t and gru_group. W contains W_r, W_z and W, see formula in grumemory.

The computational speed is that, grumemory is relatively better than gru_group, and gru_group is relatively better than simple_gru.

The example usage is:

```
gru = simple_gru(input=[layer1], size=256)
```

Parameters:
- **input** (*LayerOutput*) – input layer name.
- **name** (*basestring*) – name of the gru group.
- **size** (*int*) – hidden size of the gru.
- **reverse** (*bool*) – whether to process the input data in a reverse order
- **act** (*BaseActivation*) – type of the activiation
- **gate_act** (*BaseActivation*) – type of the gate activiation
- **gru_bias_attr** (*ParameterAttribute|False*) – bias. False means no bias, None means default bias.
- **gru_layer_attr** (*ParameterAttribute|False*) – Extra parameter attribute of the gru layer.

Returns: the gru group.

Return type: LayerOutput

simple_attention

paddle.trainer_config_helpers.networks.**simple_attention**(**args*, ***kwargs*)

Calculate and then return a context vector by attention machanism. Size of the context vector equals to size of the encoded_sequence.

$$a(s_{i-1}, h_j) = v_a f(W_a s_{t-1} + U_a h_j)$$
$$e_{i,j} = a(s_{i-1}, h_j)$$
$$a_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{T_x} exp(e_{i,k})}$$
$$c_i = \sum_{j=1}^{T_x} a_{i,j} h_j$$

where h_j is the jth element of encoded_sequence, $U_a h_j$ is the jth element of encoded_proj s_{i-1} is decoder_state f is weight_act, and is set to tanh by default.

Please refer to **Neural Machine Translation by Jointly Learning to Align and Translate** for more details. The link is as follows: https://arxiv.org/abs/1409.0473.

The example usage is:

```
context = simple_attention(encoded_sequence=enc_seq,
                           encoded_proj=enc_proj,
                           decoder_state=decoder_prev,)
```

Parameters:
- **name** (*basestring*) – name of the attention model.
- **softmax_param_attr** (*ParameterAttribute*) – parameter attribute of sequence softmax that is used to produce attention weight
- **weight_act** (*Activation*) – activation of the attention model
- **encoded_sequence** (*LayerOutput*) – output of the encoder
- **encoded_proj** (*LayerOutput*) – attention weight is computed by a feed forward neural network which has two inputs : decoder's hidden state of previous time step and encoder's output. encoded_proj is output of the feed-forward network for encoder's output. Here we pre-compute it outside simple_attention for speed consideration.
- **decoder_state** (*LayerOutput*) – hidden state of decoder in previous time step
- **transform_param_attr** (*ParameterAttribute*) – parameter attribute of the feed-forward network that takes decoder_state as inputs to compute attention weight.

Returns: a context vector

Miscs

dropout_layer

`paddle.trainer_config_helpers.networks.`**`dropout_layer`**(**args*, ***kwargs*)

@TODO(yuyang18): Add comments.

Parameters:
- **name** –
- **input** –
- **dropout_rate** –

Returns:

outputs

`paddle.trainer_config_helpers.networks.`**`outputs`**(*layers*, **args*)

Declare the outputs of network. If user have not defined the inputs of network, this method will calculate the input order by dfs travel.

Parameters: **layers** (*list|tuple|LayerOutput*) – Output layers.

Returns: