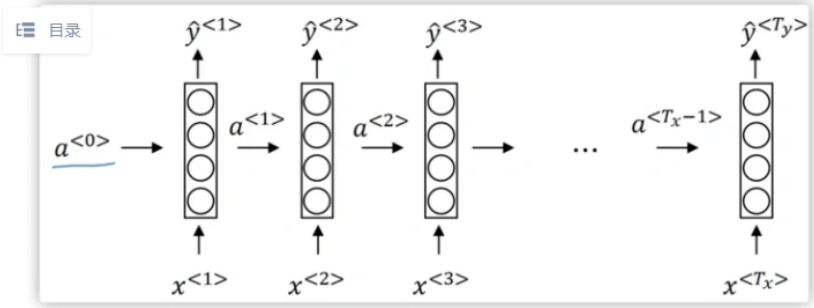
RNN layer

目录



- $a^{<t>} = g\left(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a\right)$
  - activation function is usually tahn, sometimes ReLU
  - W_{ax} 中"ax"的a表示输出为a-like vector, x表示要乘的是x-like vector, 其他的matrix也类似
- $\hat{y}^{<t>} = g\left(W_{ya}a^{<t>} + b_y\right)$
  - activation function depends on the problem/task, could be sigmoid or softmax
- Simplication:
  - $a^{\langle t\rangle} = g\left(W_a\left[a^{(t-1)}, x^{(t)}\right] + b_a\right)$
  - $\hat{y}^{(t)} = g\left(W_y a^{(t)} + b_y\right)$

Class

Python

```
torch.nn.RNN(*args, **kwargs)
```

Applies a multi-layer Elman RNN with tanh or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh\left(W_{ih}x_t + b_{ih} + W_{hh}h(t-1) + b_{hh}\right)$$

where h_t is the hidden state at time t, x_t is the input at time t, and $h(t-1)$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time o. If nonlinearity is ' $relu$ ', then ReLU is used instead of tanh.

Parameters

该资源由于格式或者网络问题，无法播放!

- `input_size` – The number of expected features in the input x
  等于 embedding vector 的维度
- `hidden_size` – The number of features in the hidden state h
  hidden state h 向量的维度
- `num_layers` – Number of recurrent layers. E.g., setting num_layers=2 would mean stacking two RNNs together to form a stacked RNN, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- `nonlinearity` – The non-linearity to use. Can be either ' tanh ' or ' relu '. Default: ' $\tanh$ '
  激活函数

- `bias` – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- `batch_first` – If True, then the input and output tensors are provided as (batch, seq, feature) instead of (seq, batch, feature). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: False
- `dropout` – If non-zero, introduces a Dropout layer on the outputs of each RNN layer except the last layer, with dropout probability equal to dropout. Default: 0
  保留每个神经元的概率
- `bidirectional` – If True, becomes a bidirectional RNN. Default: False

Inputs

该资源由于格式或者网络问题，无法播放!

- `input`: tensor of shape (L, N, H_{in}) when batch_first=False or (N, L, H_{in}) when batch_first=True containing the features of the input sequence. The input can also be a packed variable length sequence. See torch. nn . utils. mnn . pack_padded_sequence () or torch. nn . utils. rnn. pack_sequence () for details.
- `h_0`: tensor of shape $(D*$ num_layers, $N, H_{\text{out}})$ containing the initial hidden state for each element in the batch. Defaults to zeros if not provided.

where:

$$
\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{out} &= \text{hidden\_size}
\end{aligned}
$$

`input_size`

`hidden_size`

Outputs

该资源由于格式或者网络问题，无法播放!

- `output`: tensor of shape $(L, N, D * H_{\text{out}})$ when batch_first=False or $(N, L, D * H_{\text{out}})$ when batch_first=True containing the output features (h_t) from the last layer of the RNN, for each **input word** x_t . If a torch. nn. utils. rnn. PackedSequence has been given as the input, the output will also be a packed sequence.
- `h_n`: tensor of shape $(D*$ num_layers, $N, H_{Out})$ containing the final hidden state for each **sequence** in the batch.

Variables

- **~RNN.weight_ih_l[k]** – the learnable input-hidden weights of the k-th layer, of shape (*hidden_size, input_size*) for *k = 0*. Otherwise, the shape is (*hidden_size, num_directions * hidden_size*)
- **~RNN.weight_hh_l[k]** – the learnable hidden-hidden weights of the k-th layer, of shape (*hidden_size, hidden_size*)
- **~RNN.bias_ih_l[k]** – the learnable input-hidden bias of the k-th layer, of shape (*hidden_size*)
- **~RNN.bias_hh_l[k]** – the learnable hidden-hidden bias of the k-th layer, of shape (*hidden_size*)

example

Python

```python
import torch
import torch.nn as nn

rnn = nn.RNN(input_size=10,hidden_size=20,num_layers=2,batch_first=True)
inputs = torch.randn(3, 5, 10)  # batch_size,seq_length,embedding_dim
h0 = torch.randn(2, 3, 20)      # D*num_layers, batch_size, hidden_size
output, hn = rnn(inputs, h0)
```

该资源由于格式或者网络问题，无法播放!