Table of Contents

LSTM

CLASS torch.nn.LSTM(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *proj_size=0*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer long short-term memory (LSTM) RNN to an input sequence. For each element in the input sequence, each layer computes the following function:

$$
\begin{array}{ll}
i_t = \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}) \\
f_t = \sigma(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}) \\
g_t = \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}) \\
o_t = \sigma(W_{io} x_t + b_{io} + W_{ho} h_{t-1} + b_{ho}) \\
c_t = f_t \odot c_{t-1} + i_t \odot g_t \\
h_t = o_t \odot \tanh(c_t)
\end{array}
$$

it = σ(Wiixt + bii + Whiht−1 + bhi)ft = σ(Wifxt + bif + Whfht−1 + bhf)gt = tanh(Wigxt + big + Whght−1 + bhg)ot = σ(Wioxt + bio + Whoht−1 + bho)ct = ft ⊙ ct−1 + it

⊙ gtht = ot ⊙ tanh(ct)

where h_t is the hidden state at time *t*, c_t is the cell state at time *t*, x_t is the input at time *t*, h_{t-1} is the hidden state of the layer at time *t-1* or the initial hidden state at time *0*, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product.

In a multilayer LSTM, the input $x^{(l)}_t$ of the l -th layer ($l \ge 2$) is the hidden state $h^{(l-1)}_t$ of the previous layer multiplied by dropout $\delta^{(l-1)}_t$ where each $\delta^{(l-1)}_t$ is a Bernoulli random variable which is 0 with probability `dropout`.

If `proj_size > 0` is specified, LSTM with projections will be used. This changes the LSTM cell in the following way. First, the dimension of h_t will be changed from `hidden_size` to `proj_size` (dimensions of W_{hi} will be changed accordingly). Second, the output hidden state of each layer will be multiplied by a learnable projection matrix: $h_t = W_{hr} h_t$. Note that as a consequence of this, the output of LSTM network will be of different shape as well. See Inputs/Outputs sections below for exact dimensions of all variables. You can find more details in https://arxiv.org/abs/1402.1128.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **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 LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Inputs: input, (h_0, c_0)

- **input**: tensor of shape (L, H_{in}) for unbatched input, (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.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.
- **c_0**: tensor of shape $(D * \text{num\_layers}, H_{cell})$ for unbatched input or $(D * \text{num\_layers}, N, H_{cell})$ containing the initial cell state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.

where:

$$
\begin{aligned}
N ={} & \text{batch size} \\
L ={} & \text{sequence length} \\
D ={} & 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} ={} & \text{input\_size} \\
H_{cell} ={} & \text{hidden\_size} \\
H_{out} ={} & \text{proj\_size if } \text{proj\_size} > 0 \text{ otherwise hidden\_size}
\end{aligned}
$$

N = L = D = Hin = Hcell = Hout = batch sizesequence length2 if bidirectional=True otherwise 1input_sizehidden_sizeproj_size if proj_size > 0 otherwise hidden_size

Outputs: output, (h_n, c_n)

- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features *(h_t)* from the last layer of the LSTM, for each *t*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence. When `bidirectional=True`, *output* will contain a concatenation of the forward and reverse hidden states at each time step in the sequence.
- **h_n**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the final hidden state for each element in the sequence. When `bidirectional=True`, *h_n* will contain a concatenation of the final forward and reverse hidden states, respectively.
- **c_n**: tensor of shape $(D * \text{num\_layers}, H_{cell})$ for unbatched input or $(D * \text{num\_layers}, N, H_{cell})$ containing the final cell state for each element in the sequence. When `bidirectional=True`, *c_n* will contain a concatenation of the final forward and reverse cell states, respectively.

Variables

- **weight_ih_l[k]** – the learnable input-hidden weights of the k^{th} layer *(W_ii|W_if|W_ig|W_io)*, of shape *(4*hidden_size, input_size)* for *k = 0*. Otherwise, the shape is *(4*hidden_size, num_directions * hidden_size)*. If `proj_size > 0` was specified, the shape will be *(4*hidden_size, num_directions * proj_size)* for *k > 0*
- **weight_hh_l[k]** – the learnable hidden-hidden weights of the k^{th} layer *(W_hi|W_hf|W_hg|W_ho)*, of shape *(4*hidden_size, hidden_size)*. If `proj_size > 0` was specified, the shape will be *(4*hidden_size, proj_size)*.
- **bias_ih_l[k]** – the learnable input-hidden bias of the k^{th} layer *(b_ii|b_if|b_ig|b_io)*, of shape *(4*hidden_size)*
- **bias_hh_l[k]** – the learnable hidden-hidden bias of the k^{th} layer *(b_hi|b_hf|b_hg|b_ho)*, of shape *(4*hidden_size)*
- **weight_hr_l[k]** – the learnable projection weights of the k^{th} layer of shape *(proj_size, hidden_size)*. Only present when `proj_size > 0` was specified.
- **weight_ih_l[k]_reverse** – Analogous to *weight_ih_l[k]* for the reverse direction. Only present when `bidirectional=True`.
- **weight_hh_l[k]_reverse** – Analogous to *weight_hh_l[k]* for the reverse direction. Only present when `bidirectional=True`.
- **bias_ih_l[k]_reverse** – Analogous to *bias_ih_l[k]* for the reverse direction. Only present when `bidirectional=True`.
- **bias_hh_l[k]_reverse** – Analogous to *bias_hh_l[k]* for the reverse direction. Only present when `bidirectional=True`.
- **weight_hr_l[k]_reverse** – Analogous to *weight_hr_l[k]* for the reverse direction. Only present when `bidirectional=True` and `proj_size > 0` was specified.

• NOTE

All the weights and biases are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{hidden\_size}}$

U(− k √ , k √) where k = hidden_size1

• NOTE

For bidirectional LSTMs, forward and backward are directions 0 and 1 respectively. Example of splitting the output layers when `batch_first=False`: `output.view(seq_len, batch, num_directions, hidden_size)`.

• NOTE

For bidirectional LSTMs, *h_n* is not equivalent to the last element of *output*; the former contains the final forward and reverse hidden states, while the latter contains the final forward hidden state and the initial reverse hidden state.

• NOTE

`batch_first` argument is ignored for unbatched inputs.

• NOTE

`proj_size` should be smaller than `hidden_size`.

• WARNING

There are known non-determinism issues for RNN functions on some versions of cuDNN and CUDA. You can enforce deterministic behavior by setting the following environment variables:

On CUDA 10.1, set environment variable `CUDA_LAUNCH_BLOCKING=1`. This may affect performance.

On CUDA 10.2 or later, set environment variable (note the leading colon symbol) `CUBLAS_WORKSPACE_CONFIG=:16:8` or `CUBLAS_WORKSPACE_CONFIG=:4096:2`.

See the cuDNN 8 Release Notes for more information.

• NOTE

If the following conditions are satisfied: 1) cudnn is enabled, 2) input data is on the GPU 3) input data has dtype `torch.float16` 4) V100 GPU is used, 5) input data is not in `PackedSequence` format persistent algorithm can be selected to improve performance.

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```