LSTM layer

LSTM class

```
tf.keras.layers.LSTM(
    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,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

See [the Keras RNN API guide] for details about the usage of RNN API.

Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. If a GPU is available and all the arguments to the layer meet the requirement of the cuDNN kernel (see below for details), the layer will use a fast cuDNN implementation.

The requirements to use the cuDNN implementation are:

1. `activation` == `tanh`
2. `recurrent_activation` == `sigmoid`
3. `recurrent_dropout` == 0
4. `unroll` is `False`
5. `use_bias` is `True`
6. Inputs, if use masking, are strictly right-padded.
7. Eager execution is enabled in the outermost context.

For example:

```
>>> inputs = tf.random.normal([32, 10, 8])
>>> lstm = tf.keras.layers.LSTM(4)
>>> output = lstm(inputs)
>>> print(output.shape)
(32, 4)
>>> lstm = tf.keras.layers.LSTM(4, return_sequences=True, return_state=True)
>>> whole_seq_output, final_memory_state, final_carry_state = lstm(inputs)
>>> print(whole_seq_output.shape)
(32, 10, 4)
>>> print(final_memory_state.shape)
(32, 4)
>>> print(final_carry_state.shape)
(32, 4)
```

Arguments

- **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`.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). 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.
- **return_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence. Default: `False`.

- **return_state**: Boolean. Whether to return the last state in addition to the output. Default: `False`.
- **go_backwards**: Boolean (default `False`). If True, process the input sequence backwards and return the reversed sequence.
- **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.
- **time_major**: The shape format of the `inputs` and `outputs` tensors. If True, the inputs and outputs will be in shape `[timesteps, batch, feature]`, whereas in the False case, it will be `[batch, timesteps, feature]`. Using `time_major = True` is a bit more efficient because it avoids transposes at the beginning and end of the RNN calculation. However, most TensorFlow data is batch-major, so by default this function accepts input and emits output in batch-major form.
- **unroll**: Boolean (default `False`). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

Call arguments

- **inputs**: A 3D tensor with shape `[batch, timesteps, feature]`.
- **mask**: Binary tensor of shape `[batch, timesteps]` indicating whether a given timestep should be masked (optional, defaults to `None`). An individual `True` entry indicates that the corresponding timestep should be utilized, while a `False` entry indicates that the corresponding timestep should be ignored.
- **training**: Python boolean indicating whether the layer should behave in training mode or in inference mode. This argument is passed to the cell when calling it. This is only relevant if `dropout` or `recurrent_dropout` is used (optional, defaults to `None`).
- **initial_state**: List of initial state tensors to be passed to the first call of the cell (optional, defaults to `None` which causes creation of zero-filled initial state tensors).