```
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
)
```

```
# type: ignore
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input , Dense , LSTM , Embedding , Dropout , add, Conv2D, TimeDistributed
```

Formula to calculate the number of parameters of LSTM

The formula for trainable parameter in the LSTM layer is is

$4 * (n + m + 1) * m$

n is the dimension of the input vector

m is the number of LSTM units in a layer

1 is the bias parameter

```
inputs = Input(shape=(400, 3))
lstm = LSTM(50, activation='relu')(inputs)
outputs = Dense(1)(lstm)
model = Model(inputs=inputs, outputs=outputs)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 400, 3)]          0

 lstm (LSTM)                 (None, 50)                10800

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 10,851
```

```
Trainable params: 10,851
Non-trainable params: 0
_________________________________________________________________
```

In this case, the number of parameters of this LSTM layer can be calculated as: $4 * (3 + 50 + 1) * 50 = 10800$

```
def gen_lstm_model(lstm_param=LSTM(50)):
    inputs = Input(shape=(10, 3))
    lstm = lstm_param(inputs)
    # outputs = Dense(1)(lstm)
    model = Model(inputs=inputs, outputs=lstm)
    # model.summary()
    print('Output shape', model.output_shape)
    print(model.count_params())
    print('*'*100)

gen_lstm_model(LSTM(50))
# use_bias: wheter the model uses a bias vector b: y = Wx + b
gen_lstm_model(LSTM(50, use_bias=False))
# return_sequences: return output (time_steps, units), output at each timestamp
gen_lstm_model(LSTM(50, return_sequences=True))
# return_states: return [output (units), state_h (units), state_c (units)]. In this case, state_h == output
gen_lstm_model(LSTM(50, return_state=True))
# return_sequences and return_states:
# return [output (time_steps, units), state_h (units), state_c (units)]. In this case, state_h == output[0,-1]
gen_lstm_model(LSTM(50, return_state=True, return_sequences=True))
```

```
Output shape (None, 50)
10800
****************************************************************************************************
Output shape (None, 50)
10600
****************************************************************************************************
Output shape (None, 10, 50)
10800
****************************************************************************************************
Output shape [(None, 50), (None, 50), (None, 50)]
10800
****************************************************************************************************
Output shape [(None, 10, 50), (None, 50), (None, 50)]
```

```
10800
********************************************************************************************************************
```

```
# inputs (n_samples, n_time_steps, feature_vector)
inputs = tf.random.normal([1,3,1])
print(LSTM(2)(inputs))
print('-'*80)
print(LSTM(2, use_bias = False)(inputs))
print('-'*80)
print(LSTM(2, return_sequences=True)(inputs))
print('-'*80)
print(*LSTM(2, return_state=True)(inputs), sep='\n')
print('-'*80)
print(*LSTM(2, return_state=True, return_sequences=True)(inputs), sep='\n')
```

```
tf.Tensor([[0.19843383 0.24309134]], shape=(1, 2), dtype=float32)
--------------------------------------------------------------------------------
tf.Tensor([[0.26354963 0.0401829 ]], shape=(1, 2), dtype=float32)
--------------------------------------------------------------------------------
tf.Tensor(
[[[ 0.0546216  -0.02551219]
  [ 0.00309311  0.00942754]
  [-0.06628668  0.16456468]]], shape=(1, 3, 2), dtype=float32)
--------------------------------------------------------------------------------
tf.Tensor([[-0.05335176  0.18356399]], shape=(1, 2), dtype=float32)
tf.Tensor([[-0.05335176  0.18356399]], shape=(1, 2), dtype=float32)
tf.Tensor([[-0.21267909  0.26048866]], shape=(1, 2), dtype=float32)
--------------------------------------------------------------------------------
tf.Tensor(
[[[-0.03435385 -0.00614802]
  [ 0.00335012 -0.00221255]
  [ 0.18249846  0.02452812]]], shape=(1, 3, 2), dtype=float32)
tf.Tensor([[0.18249846 0.02452812]], shape=(1, 2), dtype=float32)
tf.Tensor([[0.33608383 0.07505616]], shape=(1, 2), dtype=float32)
```

```
model = Sequential()
model.add(Input(shape=(10, 3)))
model.add(LSTM(64, return_sequences=True))
# TimeDistributed layer will perform aplly the SAME Dense(2) on each of the time steps
model.add(tf.keras.layers.TimeDistributed(Dense(2)))

model.compile(optimizer='adam', loss='mse')

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_12 (LSTM)              (None, 10, 64)            17408

 time_distributed (TimeDistr  (None, 10, 2)            130
 ibuted)

=================================================================
Total params: 17,538
Trainable params: 17,538
Non-trainable params: 0
_________________________________________________________________
```

units parameter refers to dimensionality of the output. Each LSTM cell (present at a given time_step) takes in input x and forms a hidden state vector a, the length of this hidden vector is what is called the **units** in LSTM.

When **units** is increased, width of the network increase, increases the number of parameters thus takes longer to train. If the **units** is too large, it might lead to overfitting.

use_bias parameter in an LSTM layer is a boolean that determines whether the layer uses a bias vector.
use_bias set to True can help the model fit the data better, but it also increases the number of parameters in the model, which can potentially lead to overfitting.