Parâmetros LSTM

Parâmetros Comuns

1. **Units (units)**:
   - Number of LSTM units in the layer.
     - Exemplo: `LSTM(50)` define uma camada LSTM com 50 unidades.
2. **Input Shape (input_shape)**:
   - Three-dimensional format: (batch_size, timesteps, features).
     - `batch_size`: Number of samples in each data batch.
     - `timesteps`: Number of time steps in each input sequence.
     - `features`: Number of features in each time step.
3. **Activation (activation)**:
   - Activation function applied to the layer outputs.
4. **Recurrent Activation (recurrent_activation)**:
   - Activation function used for the recurrent activation gate.
5. **Use Bias (use_bias)**:
   - Indicates whether the layer uses a bias in its operations.
6. **Kernel Initializer (kernel_initializer)**:
   - Initialization scheme for the layer weights.
7. **Recurrent Initializer (recurrent_initializer)**:
   - Initialization scheme for the recurrent connection weights.
8. **Bias Initializer (bias_initializer)**:
   - Initialization scheme for the biases.

EXEMPLO:

```
lstm_layer = LSTM(units=64,
                  input_shape=(10, 32),
                  activation='tanh',
                  recurrent_activation='sigmoid',
                  use_bias=True,
                  dropout=0.2,
                  recurrent_dropout=0.2,
                  return_sequences=True,
                  return_state=False,
                  kernel_regularizer=None,
```

```
                    recurrent_regularizer=None,
                    bias_regularizer=None)
```

Config Usada

- `model = Sequential()` inicializa um novo modelo sequencial vazio.
- `model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))` adiciona uma camada LSTM ao modelo
  - Esta camada LSTM tem 50 unidades, o que significa que haverá 50 células LSTM na camada.
  - `(X_train.shape[1], X_train.shape[2])` corresponde ao número de passos de tempo e ao número de características em cada passo de tempo dos dados de entrada
  - `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`
  - `X` e `y` são os dados de entrada e saída, respectivamente.
  - `test_size=0.2` especifica que 20% dos dados serão usados como conjunto de teste, enquanto os restante (80%) serão usados como conjunto de treino.
  - `random_state=42` define a seed para a randomização dos dados antes da divisão. Isso garante que a divisão dos dados seja sempre a mesma, o que é útil para garantir reprodutibilidade nos resultados.
  - `X_train`: Dados de entrada para treino.
  - `X_test`: Dados de entrada para teste.
  - `y_train`: Rótulos correspondentes aos dados de entrada de treino.
  - `y_test`: Rótulos correspondentes aos dados de entrada de teste.
- `model.add(Dense(X_train.shape[2]))` adiciona uma camada dense ao modelo
  - Esta camada dense tem o mesmo número de unidades que o número de características nos dados de entrada.
  - A camada dense produz a saída final do modelo.
- `model.compile(optimizer='adam', loss='mse')` compila o modelo para prepará-lo para o treino.
  - Adam é um algoritmo de otimização que ajusta os pesos do modelo durante o treino para minimizar a função de perda.
  - A função de perda é definida como 'mse', que significa *Mean Square Error*. Esta métrica é usada para problemas de regressão.
- `model.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=(X_test, y_test))` treina o modelo
  - O parâmetro `epochs=100` especifica o número de épocas de treino, ou seja, quantas vezes o modelo verá o conjunto de dados de treino completo durante o treino
- O parâmetro `batch_size=16` especifica o número de amostras que serão propagadas pela rede antes que os pesos sejam atualizados

- Os dados de teste `(X_test, y_test)` são usados como dados de validação para monitorar o desempenho do modelo durante o treino
- `return model` retorna o modelo treinado após o treino ser concluído.

Código:

```
def build_and_train_model(X_train, X_test, y_train, y_test):

    model = Sequential()
    model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(X_train.shape[2]))
    model.compile(optimizer='adam', loss='mse')

    model.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=
(X_test, y_test))
    return model
```