LSTM layer

Editado por: Dr. Arnaldo de Carvalho Junior
Data: Junho 21, 2024.

LSTM class

```
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,
    seed=None,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    unroll=False,
    use_cudnn="auto",
    **kwargs
)
```

Com base no *hardware* e nas restrições de tempo de execuções disponíveis, esta camada escolherá diferentes implementações (baseadas em cuDNN ou nativas de *backend*) para maximizar o desempenho. Se uma GPU estiver disponível e todos os argumentos da camada atenderem aos requisitos do kernel cuDNN (veja detalhes abaixo), a camada usará uma implementação rápida de cuDNN ao usar o back-end do TensorFlow.

Os requisitos para usar a implementação cuDNN são:

1. activation == tanh
2. recurrent_activation == sigmoid
3. dropout == 0 e recurrent_dropout == 0
4. unroll é False
5. use_bias é True
6. As entradas (Inputs), se usarem mascaramento, são estritamente preenchidas à direita.
7. A execução rápida (Eager) é habilitada no contexto mais externo.

Por exemplo:

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

Argumentos

• **units:** inteiro positivo, dimensionalidade do espaço de saída (*output*).

• **activation**: Função de ativação a ser utilizada. Padrão (*Default*): tangente hiperbólica (tanh). Se colocar None, nenhuma ativação é aplicada (ie. "linear" activation: a(x) = x).

• **recurrent_activation**: Função de ativação para ser utilizada para o passo recorrente. Padrão: sigmoide (sigmoid). Se colocar None, nenhuma ativação é aplicada (ex.: "linear" activation: a(x) = x).

• **use_bias**: Boleano, (padrão True), se a camada deve usar um vetor de polarização (*bias*).

• **kernel_initializer**: Inicializador para a matriz de pesos do kernel, usado para a transformação linear das entradas. Padrão: "glorot_uniform".

• **recurrent_initializer**: Initializador para a matriz de pesos do recurrent_kernel, usado para a transformação linear do estado recorrente. Padrão: "orthogonal".

• **bias_initializer**: Initializador para o vetor *bias*. Padrão: "zeros".

• **unit_forget_bias**: Boleano (padrão True). Se True (verdadeiro), adiciona 1 ao bias da porta de esquecimento (*forget gate*) na inicialização. Configurando ele para True também forçará bias_initializer="zeros". Isso é recomendado em [2].

• **kernel_regularizer**: Função regularizadora aplicada à matriz de pesos do kernel. Padrão: None.

• **recurrent_regularizer**: Função regularizadora aplicada à matriz de pesos do recurrent_kernel. Padrão: None.

• **bias_regularizer**: Função regularizadora aplicada ao vetor bias. Padrão: None.

• **activity_regularizer**: Função regularizadora aplicada à saída da camada (sua "ativação"). Padrão: None.

• **kernel_constraint**: Função limitadora (*constraint*) aplicada à matriz de pesos do kernel. Padrão: None.

• **recurrent_constraint**: Função limitadora (*constraint*) aplicada à matriz de pesos do recurrent_kernel. Padrão: None.

• **bias_constraint**: Função limitadora (*constraint*) aplicada ao vetor de bias. Padrão: None.

• **dropout**: Flutuante (*Float*) entre 0 e 1. Fração das unidades a serem eliminadas para a transformação linear das entradas. Padrão: 0.

• **recurrent_dropout**: Flutuante (*Float*) entre 0 e 1. Fração das unidades a serem eliminadas para a transformação linear do estado recorrente. Padrão: 0.

• **seed**: Semente (*seed*) aleatória para abandono (*dropout*).

• **return_sequences**: Boleano. Seja para retornar a última saída na sequência de saída ou a sequência completa. Padrão:False.

• **return_state**: Boleano. Seja para retornar o último estado na adição para a saída. Padrão: False.

•**go_backwards**: Boleano (padrão: False). Se True, processa a sequência de entrada para trás e retorna a sequência invertida.

• **stateful**: Boleano (padrão: False). Se True, o último estado para cada amostra no índice i em um lote (*batch*) será usado como estado inicial para a amostra do índice i no lote seguinte.

• **unroll**: Boleano (padrão: False). Se True, a rede será desenrolada, caso contrário, um loop simbólico será usado. O desenrolamento pode acelerar um RNN, embora tenda a consumir mais memória. O desenrolar só é adequado para sequências curtas.

• **use_cudnn**: Se deve usar uma implementação por cuDNN-backed. Valor "auto" tentará usar cuDNN quando possível e, caso contrário, retornará à implementação padrão.

Argumentos de Chamada (*Call arguments*)

• **inputs**: Um tensor 3D, com formato (batch, timesteps, feature).

• **mask**: Tensor binário de formato (samples, timesteps) indicando se um dado intervalo de tempo (*timestep)* deve ser mascarado (opcional). Uma entrada (*entry*) individual True indica que o correspondente *timestep* deve ser utilizado, enquanto uma entrada False indica que o *timestep* correspondente deve ser ignorado. Padrão para None.

• **training**: Boleano Python indicando se a camada deve boolean indicating whether the layer should comportar-se no modo de treinamento (*training*) ou no modo de inferência (*inference*). Este argumento é passado para a célula ao chamá-la. Isto só é relevante se dropout ou recurrent_dropout é usado (opcional). Padrão para None.

• **initial_state**: Lista de tensores de estado inicial a serem passados para a primeira chamada da célula (opcional, None causa criação de tensores de estado inicial preenchidos por zero). Padrão para None.

Referências

[1] Keras, LSTM Layer, Keras 3 API documentation / Layers API / Recurrent layers. Disponível em <https://keras.io/api/layers/recurrent_layers/lstm/>. Acessado em Junho 21, 2024.

[2] JOZEFOWICZ, Rafal; ZAREMBA, Wojciech; SUTSKEVER, Ilya. An empirical exploration of recurrent network architectures. In: International conference on machine learning. PMLR, 2015. p. 2342-2350. Disponível em <https://proceedings.mlr.press/v37/jozefowicz15.pdf>. Acessado em Junho 21, 2024.