**آموزش جامع و تخصصی لایه SimpleRNN در Keras/TensorFlow**

**مقدمه**

در دنیای امروز که تحلیل داده‌های سری زمانی و دنباله‌ای (sequential data) اهمیت زیادی پیدا کرده‌اند، مدل‌های یادگیری عمیق مانند شبکه‌های عصبی بازگشتی (Recurrent Neural Networks - RNN) نقش کلیدی ایفا می‌کنند. یکی از ابتدایی‌ترین اما پایه‌ای‌ترین لایه‌های این خانواده، لایه SimpleRNN است که در کتابخانه Keras (از زیرمجموعه‌های TensorFlow) ارائه شده است.

**چرا لایه‌های RNN به وجود آمدند؟**

شبکه‌های عصبی کلاسیک (Dense, MLP) توانایی یادگیری وابستگی‌های زمانی یا ترتیبی را ندارند، زیرا فرض آن‌ها بر این است که تمام داده‌ها مستقل از یکدیگرند. برای مثال:

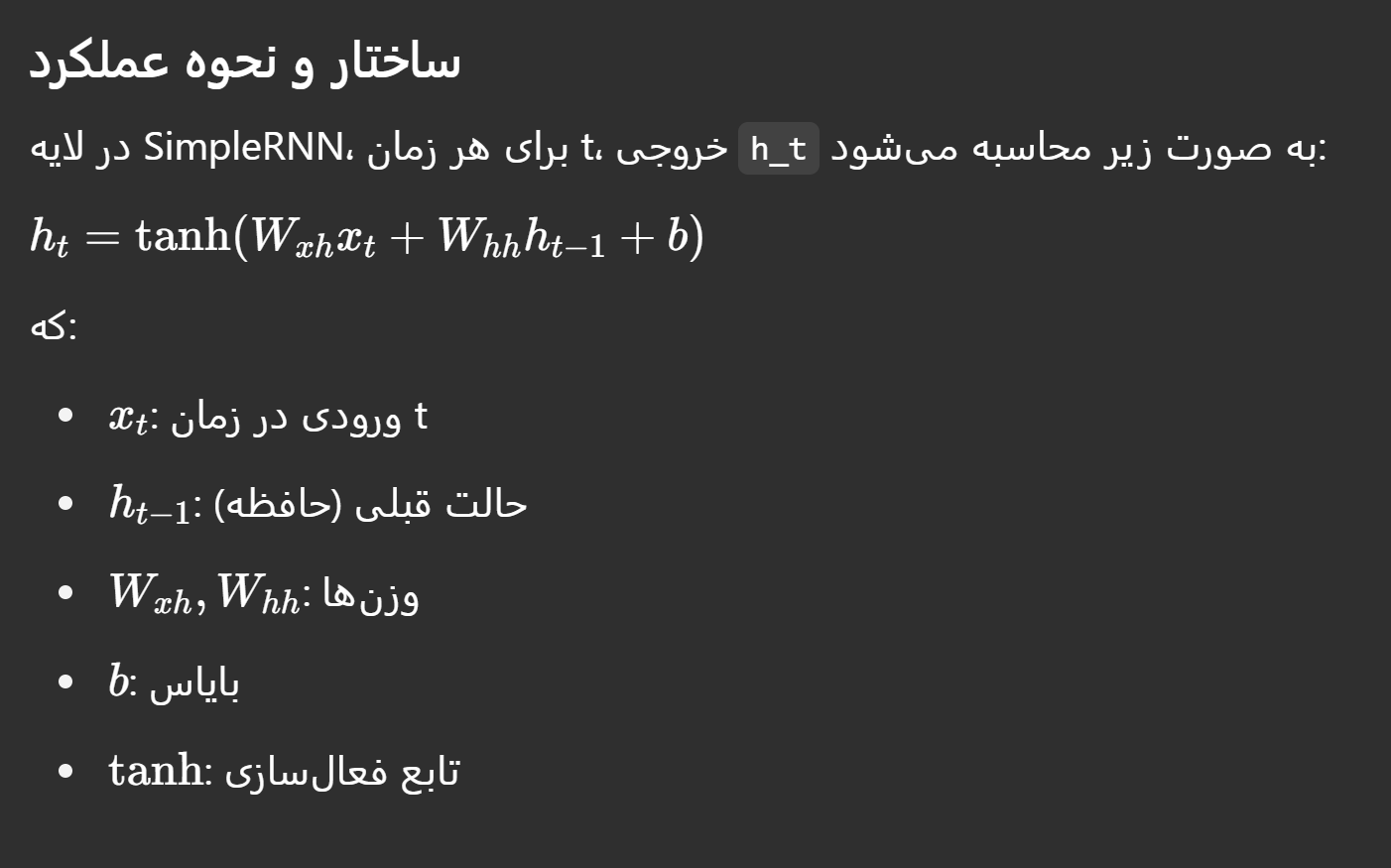
* در پیش‌بینی قیمت سهام، داده‌ی روز گذشته بر داده‌ی امروز تأثیر دارد.
* در پردازش زبان طبیعی (NLP)، ترتیب کلمات اهمیت دارد.

RNN با طراحی خاص خود (داشتن حافظه داخلی) این وابستگی زمانی را درک می‌کند.

**لایه SimpleRNN چیست؟**

SimpleRNN ابتدایی‌ترین نوع RNN در Keras است. این لایه به هر ورودی ترتیبی نگاه می‌کند و خروجی را بر اساس ورودی فعلی و حالت (state) قبلی تولید می‌کند.

from tensorflow.keras.layers import SimpleRNN

****

**پارامترهای مهم SimpleRNN**

|  |  |
| --- | --- |
| پارامتر | توضیح |
| units | تعداد نورون‌های داخلی (یعنی اندازه حالت مخفی h) |
| activation | تابع فعال‌سازی (پیش‌فرض tanh است) |
| return\_sequences | اگر True باشد، خروجی در هر گام زمانی را برمی‌گرداند |
| return\_state | اگر True باشد، علاوه بر خروجی نهایی، حالت نهایی را نیز برمی‌گرداند |
| stateful | آیا حالت بین batchها حفظ شود؟ (برای مدل‌های آنلاین مهم است) |
| dropout | درصد دراپ‌اوت روی ورودی‌ها |
| recurrent\_dropout | درصد دراپ‌اوت روی حالت‌های بازگشتی |

**مثال ساده عملی**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

import numpy as np

model = Sequential([

SimpleRNN(50, input\_shape=(10, 1), return\_sequences=False),

Dense(1)

])

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

X = np.random.rand(100, 10, 1)

y = np.random.rand(100, 1)

model.fit(X, y, epochs=10)

**کاربردهای صنعتی SimpleRNN**

| **حوزه** | **کاربرد** |
| --- | --- |
| مالی | پیش‌بینی قیمت سهام، ارز، بیت‌کوین |
| سلامت | تحلیل سیگنال ECG و EEG |
| انرژی | پیش‌بینی مصرف برق |
| NLP | تکمیل متن، تحلیل احساسات |

**مقایسه با سایر لایه‌های RNN**

| **لایه** | **مزیت** | **عیب** |
| --- | --- | --- |
| SimpleRNN | ساده و سریع | حافظه کوتاه مدت، مشکل gradient vanishing |
| LSTM | حافظه بلندمدت، جلوگیری از gradient vanishing | سنگین‌تر و کندتر |
| GRU | عملکرد نزدیک LSTM ولی سریع‌تر | پیچیدگی بالاتر نسبت به SimpleRNN |

**ترکیب با سایر روش‌ها**

* **ترکیب با CNN:** برای داده‌های صوتی یا متنی (مثلاً تبدیل صدا به ویژگی با CNN و سپس استفاده از RNN)
* **Stacked RNN:** چند لایه SimpleRNN روی هم برای افزایش توان مدل
* **Bidirectional RNN:** خواندن داده هم از ابتدا به انتها و هم برعکس
* **Attention Layer:** برای تمرکز بیشتر روی نقاط مهم دنباله

**نکات حرفه‌ای و پیشنهادی**

* اگر دنباله‌های بلند دارید، **LSTM یا GRU** پیشنهاد می‌شود.
* برای داده‌های با noise زیاد، از **dropout و recurrent\_dropout** استفاده کنید.
* در داده‌های live و real-time، از حالت **stateful=True** استفاده کنید.
* برای پردازش زبان طبیعی بهتر است از Embedding و سپس RNN استفاده شود.

**منابع پیشنهادی برای یادگیری بیشتر**

* [مستندات رسمی TensorFlow SimpleRNN](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN)
* دوره‌های Coursera و DeepLearning.AI درباره Sequence Models
* مقالات Medium با کلیدواژه SimpleRNN + Keras
* ویدیوهای YouTube از کانال‌هایی مثل Codebasics، deeplizard، The AI Guy

**جمع‌بندی**

لایه SimpleRNN پایه‌ای‌ترین روش برای مدل‌سازی داده‌های ترتیبی است. اگرچه محدودیت‌هایی دارد، اما برای آموزش مفاهیم، مدل‌های سبک و پروژه‌های آزمایشی بسیار مفید است. درک درست از این لایه، قدم اول برای ورود به دنیای مدل‌های پیچیده‌تر مانند LSTM، GRU و Transformer است.

**در ادامه، همه آرگومان‌ها (arguments) و نکات کاربردی مهم مربوط به لایه SimpleRNN در Keras/TensorFlow به‌صورت کاملاً جامع، تخصصی و به زبان فارسی ارائه شده:**

**📘 آرگومان‌های لایه SimpleRNN**

**tf.keras.layers.SimpleRNN(**

**units,**

**activation='tanh',**

**use\_bias=True,**

**kernel\_initializer='glorot\_uniform',**

**recurrent\_initializer='orthogonal',**

**bias\_initializer='zeros',**

**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,**

**unroll=False,**

**time\_major=False,**

**\*\*kwargs**

**)**

**📌 شرح کامل آرگومان‌ها**

|  |  |  |  |
| --- | --- | --- | --- |
| آرگومان | نوع داده | پیش‌فرض | توضیح |
| units | **int** | **—** | **تعداد نرون‌های موجود در لایه (حجم بردار وضعیت مخفی یا hidden state)** |
| activation | **str** | **'tanh'** | **تابع فعال‌سازی؛ می‌توان مثلاً 'relu' یا 'sigmoid' هم استفاده کرد** |
| use\_bias | **bool** | **True** | **آیا از بایاس استفاده شود یا نه** |
| kernel\_initializer | **str یا Initializer** | **'glorot\_uniform'** | **نحوه مقداردهی اولیه به وزن‌های ورودی** |
| recurrent\_initializer | **str** | **'orthogonal'** | **مقداردهی اولیه وزن‌های بازگشتی (یعنی وزن‌هایی که بین حالت‌های زمانی هستند)** |
| bias\_initializer | **str** | **'zeros'** | **نحوه مقداردهی اولیه به بایاس** |
| kernel\_regularizer | **Regularizer** | **None** | **اعمال جریمه به وزن‌های ورودی برای جلوگیری از overfitting** |
| recurrent\_regularizer | **Regularizer** | **None** | **جریمه برای وزن‌های بازگشتی** |
| bias\_regularizer | **Regularizer** | **None** | **جریمه برای بایاس** |
| activity\_regularizer | **Regularizer** | **None** | **جریمه بر مبنای خروجی لایه** |
| kernel\_constraint | **Constraint** | **None** | **محدودسازی مقادیر وزن‌ها** |
| recurrent\_constraint | **Constraint** | **None** | **محدودسازی وزن‌های بازگشتی** |
| bias\_constraint | **Constraint** | **None** | **محدودسازی بایاس** |
| dropout | **float** | **0.0** | **دراپ‌اوت روی ورودی‌ها به لایه RNN** |
| recurrent\_dropout | **float** | **0.0** | **دراپ‌اوت روی وزن‌های بازگشتی (مربوط به حافظه)** |
| return\_sequences | **bool** | **False** | **اگر True باشد، خروجی در تمام گام‌های زمانی بازگردانده می‌شود؛ اگر False، فقط خروجی آخر** |
| return\_state | **bool** | **False** | **اگر True، حالت نهایی (h\_t) هم برگردانده می‌شود (مثلاً برای مدل‌های stateful یا encoder-decoder)** |
| go\_backwards | **bool** | **False** | **اگر True، دنباله زمانی به صورت برعکس خوانده می‌شود** |
| stateful | **bool** | **False** | **اگر True، وضعیت (hidden state) بین batchها حفظ می‌شود** |
| unroll | **bool** | **False** | **آیا گراف زمانی unroll شود یا نه (برای دنباله‌های کوتاه مفید است)** |
| time\_major | **bool** | **False** | **اگر True، ترتیب ورودی باید (timesteps, batch, features) باشد به جای پیش‌فرض (batch, timesteps, features)** |

**💡 نکات حرفه‌ای و تجربی در استفاده از SimpleRNN**

1. **مدل‌های بسیار سبک:  
   مناسب برای آموزش‌های ساده، داده‌های کوچک و زمانی که حافظه‌ی سیستم محدود است.**
2. **در دنباله‌های بلند (بیشتر از 50 گام زمانی)  
   از SimpleRNN استفاده نکن چون با مشکل vanishing gradient مواجه می‌شود. به‌جای آن از LSTM یا GRU بهره بگیر.**
3. **در پروژه‌های real-time یا stream-based  
   از stateful=True استفاده کن. اما دقت کن که باید batch\_size و طول دنباله ثابت باشد.**
4. **اگر بخواهی خروجی مدل به‌صورت گام‌به‌گام (Sequence-to-Sequence) باشد  
   باید return\_sequences=True تنظیم شود.**
5. **برای استفاده در Encoder-Decoder (مثل ترجمه ماشینی):  
   از return\_state=True برای گرفتن حالت نهایی encoder استفاده کن.**
6. **استفاده از dropout در داده‌های noisy:  
   مقادیر 0.2 یا 0.3 برای dropout و recurrent\_dropout مناسب است.**

**✅ بهترین تنظیمات پیشنهادی برای کاربردهای مختلف**

|  |  |
| --- | --- |
| کاربرد | تنظیمات پیشنهادی |
| پیش‌بینی تک‌نقطه‌ای (مثلاً قیمت روز آینده) | **return\_sequences=False, units=64** |
| پیش‌بینی چندمرحله‌ای | **return\_sequences=True, همراه با TimeDistributed(Dense(...))** |
| استفاده در encoder | **return\_state=True, return\_sequences=False** |
| تحلیل توالی متنی | **ابتدا Embedding و سپس SimpleRNN یا بهتر LSTM/GRU** |
| یادگیری آنلاین (real-time) | **stateful=True, batch\_input\_shape=(batch, time, features)** |

**شکل ورودی پیش‌فرض SimpleRNN**

**(input\_shape) = (batch\_size, timesteps, features)**

**⬅️ جزئیات:**

|  |  |
| --- | --- |
| پارامتر | توضیح |
| batch\_size | **تعداد نمونه‌ها در هر batch (در حالت training، معمولاً مشخص در fit)** |
| timesteps | **تعداد گام‌های زمانی در هر دنباله (مثلاً 30 روز یا 50 کلمه)** |
| features | **تعداد ویژگی‌ها در هر گام زمانی (مثلاً فقط قیمت = 1، یا چند ویژگی مثل Open/Close/Volume = 3)** |

**🔍 مثال واقعی:**

**فرض کن:**

* **می‌خواهی قیمت سهام 60 روز گذشته را پیش‌بینی کنی.**
* **فقط از ستون Close استفاده می‌کنی (یعنی 1 ویژگی)**

**در این صورت:**

**input\_shape = (60, 1)**

**در مدل Keras:**

**python**

**CopyEdit**

**model = Sequential([**

**SimpleRNN(64, input\_shape=(60, 1)),**

**Dense(1)**

**])**

**💡 نکته درباره batch\_size:**

* **نیازی به تعیین batch\_size در input\_shape نیست مگر اینکه stateful=True باشد.**
* **اگر stateful=True، باید batch\_input\_shape=(batch\_size, timesteps, features) بدهی.**

**مثال:**

**SimpleRNN(64, stateful=True, batch\_input\_shape=(32, 60, 1))**

**✅ جمع‌بندی نهایی:**

|  |  |
| --- | --- |
| **حالت** | **شکل ورودی مورد نیاز** |
| **پیش‌فرض (stateless)** | **input\_shape=(timesteps, features)** |
| **حالت stateful** | **batch\_input\_shape=(batch\_size, timesteps, features)** |
| **خروجی گام‌به‌گام** | **return\_sequences=True** |
| **پیش‌بینی نهایی فقط** | **return\_sequences=False (پیش‌فرض)** |

**اگر بخوای مثالی هم با TimeDistributed, Embedding, یا CNN-RNN ترکیبی داشته باشی، حتماً بگو تا کامل بنویسم.**