**شبکه‌های عصبی بازگشتی و LSTM: راهنمای جامع**

**مقدمه**

شبکه‌های عصبی بازگشتی (Recurrent Neural Networks یا RNN) و به‌ویژه نوع پیشرفته آن‌ها، یعنی شبکه‌های حافظه طولانی کوتاه‌مدت (Long Short-Term Memory یا LSTM)، از مهم‌ترین الگوریتم‌های یادگیری عمیق هستند که برای پردازش داده‌های توالی‌دار مانند سری‌های زمانی، متن، گفتار و ویدئو استفاده می‌شوند. این شبکه‌ها به دلیل داشتن حافظه داخلی، قادر به حفظ اطلاعات از مراحل قبلی و استفاده از آن‌ها برای پیش‌بینی‌های بعدی هستند. در این مقاله، به بررسی چیستی RNN و LSTM، دلیل پیدایش آن‌ها، کاربردها، نحوه عملکرد، مقایسه با سایر روش‌ها، مثال‌های عملی و نکات تخصصی برای استفاده در بازار کار می‌پردازیم.

**چیستی شبکه‌های عصبی بازگشتی (RNN)**

شبکه‌های عصبی بازگشتی نوعی از شبکه‌های عصبی مصنوعی هستند که برای پردازش داده‌های توالی‌دار طراحی شده‌اند. برخلاف شبکه‌های پیش‌خور (Feedforward Neural Networks)، که هر ورودی به‌صورت مستقل پردازش می‌شود، RNN‌ها دارای حلقه‌های بازگشتی هستند که خروجی یک مرحله را به‌عنوان ورودی مرحله بعدی استفاده می‌کنند. این ویژگی به آن‌ها امکان می‌دهد تا وابستگی‌های زمانی و الگوهای ترتیبی را در داده‌ها تشخیص دهند.

**ویژگی‌های کلیدی RNN:**

* **حافظه ترتیبی (Sequential Memory):** RNN‌ها اطلاعات مراحل قبلی را از طریق حالت مخفی (Hidden State) ذخیره می‌کنند.
* **اشتراک پارامترها:** از وزن‌های یکسانی در تمام مراحل زمانی استفاده می‌شود، که تعداد پارامترها را کاهش می‌دهد.
* **مناسب برای داده‌های توالی‌دار:** مانند متن، گفتار، سری‌های زمانی و غیره.

**مشکلات RNN:**

1. **محوشدگی گرادیان (Vanishing Gradient):** در توالی‌های طولانی، گرادیان‌ها ممکن است به صفر نزدیک شوند، که یادگیری وابستگی‌های بلندمدت را دشوار می‌کند.
2. **انفجار گرادیان (Exploding Gradient):** در برخی موارد، گرادیان‌ها ممکن است بیش از حد بزرگ شوند و باعث ناپایداری در یادگیری شوند.
3. **محدودیت در حفظ اطلاعات بلندمدت:** RNN‌های سنتی نمی‌توانند اطلاعات را برای توالی‌های بسیار طولانی حفظ کنند.

**دلیل پیدایش شبکه‌های LSTM**

شبکه‌های LSTM در سال 1997 توسط سپ هوخ‌رایتر (Sepp Hochreiter) و یورگن اشمیدهوبر (Jürgen Schmidhuber) برای حل مشکلات RNN‌های سنتی، به‌ویژه مشکل محوشدگی گرادیان، معرفی شدند. LSTM‌ها با استفاده از ساختار خاصی به نام **سلول حافظه (Cell State)** و **دروازه‌ها (Gates)**، قادر به حفظ اطلاعات برای مدت طولانی‌تر و مدیریت وابستگی‌های بلندمدت در داده‌های توالی‌دار هستند.

**چرا LSTM؟**

* **حل مشکل محوشدگی گرادیان:** LSTM‌ها با استفاده از دروازه‌های خاص، جریان اطلاعات را کنترل می‌کنند و از محو شدن یا انفجار گرادیان جلوگیری می‌کنند.
* **حافظه بلندمدت و کوتاه‌مدت:** LSTM‌ها می‌توانند اطلاعات مهم را برای مدت طولانی حفظ کنند، در حالی که اطلاعات غیرضروری را فراموش می‌کنند.
* **انعطاف‌پذیری در پردازش توالی‌ها:** مناسب برای توالی‌های طولانی و پیچیده مانند متون طولانی یا سری‌های زمانی پیچیده.

**معماری و نحوه عملکرد LSTM**

شبکه‌های LSTM از ماژول‌های تکرارشونده‌ای تشکیل شده‌اند که شامل چهار لایه اصلی هستند که با یکدیگر تعامل دارند. این لایه‌ها عبارتند از:

1. **دروازه فراموشی (Forget Gate):** تصمیم می‌گیرد که کدام اطلاعات از حافظه سلول باید حذف شوند. این دروازه با استفاده از تابع سیگموید، مقادیری بین 0 و 1 تولید می‌کند (0 به معنای فراموش کردن کامل و 1 به معنای حفظ کامل).
2. **دروازه ورودی (Input Gate):** مشخص می‌کند که کدام اطلاعات جدید باید به حافظه سلول اضافه شوند.
3. **دروازه خروجی (Output Gate):** تعیین می‌کند که کدام بخش از حافظه سلول به‌عنوان خروجی استفاده شود.
4. **حالت سلول (Cell State):** به‌عنوان حافظه بلندمدت عمل می‌کند و اطلاعات را در طول توالی منتقل می‌کند.

این ساختار امکان مدیریت دقیق جریان اطلاعات را فراهم می‌کند و از فراموشی اطلاعات مهم جلوگیری می‌کند.

**فرمول‌های اصلی LSTM**

**دروازه فراموشی (Forget Gate)**:  
ft=σ(Wf⋅[ht−1,xt]+bf)*ft*=*σ*(*Wf*⋅[*ht*−1,*xt*]+*bf*)  
این گیت تصمیم می‌گیرد چه بخشی از حافظه قبلی (Ct−1*Ct*−1) باید حفظ یا فراموش شود. خروجی آن بین ۰ و ۱ است[1](https://howsam.org/lstm-neural-network/" \t "_blank)[3](https://blog.faradars.org/lstm-%DA%86%DB%8C%D8%B3%D8%AA/).

**دروازه ورودی (Input Gate)**:  
it=σ(Wi⋅[ht−1,xt]+bi)*it*=*σ*(*Wi*⋅[*ht*−1,*xt*]+*bi*)  
تعیین می‌کند چه اطلاعات جدیدی به حافظه اضافه شود. خروجی آن نیز بین ۰ و ۱ است[3](https://blog.faradars.org/lstm-%DA%86%DB%8C%D8%B3%D8%AA/" \t "_blank)[7](https://hamruyesh.com/understanding-rnn-and-lstm-gudie/).

**کاندیدای حافظه جدید (C~t*C*~*t*)**:  
C~t=tanh⁡(WC⋅[ht−1,xt]+bC)*C*~*t*=tanh(*WC*⋅[*ht*−1,*xt*]+*bC*)  
مقدار جدیدی که ممکن است به حافظه اضافه شود، با استفاده از تابع tanh⁡tanh (محدود به بازه −۱−۱ تا ۱۱) محاسبه می‌شود[6](https://artarasaneh.com/lstm-long-short-term-memory" \t "_blank)[7](https://hamruyesh.com/understanding-rnn-and-lstm-gudie/).

**به‌روزرسانی حالت سلول (Ct*Ct*)**:

|  |  |  |  |
| --- | --- | --- | --- |
| گیت | تابع فعال‌ساز | ورودی‌ها | نقش |
| فراموشی | سیگموید |  | کنترل حفظ/حذف حافظه قبلی |
| ورودی | سیگموید |  | کنترل افزودن اطلاعات جدید |
| خروجی | سیگموید |  | کنترل انتقال حافظه به خروجی |
| کاندیدای حافظه |  |  | تولید مقدار جدید برای حافظه |

این فرمول‌ها چهار برابر پارامترهای یک شبکه RNN ساده دارند و با استفاده از گیت‌ها، جریان اطلاعات را در طول زمان مدیریت می‌کنند[1](https://howsam.org/lstm-neural-network/" \t "_blank)[4](https://alisterta.github.io/2017-07-19/%D8%B3%D8%A7%D9%84-2016-%D8%AF%D8%B1-%DB%8C%DA%A9-%D9%86%DA%AF%D8%A7%D9%87-%D8%B4%D8%A8%DA%A9%D9%87-%D9%87%D8%A7%DB%8C-%D8%B9%D8%B5%D8%A8%DB%8C-LSTM-%D9%88-RNN-%D8%A8%D8%AE%D8%B4-%D8%B3%D9%88%D9%85/).

**کاربردهای LSTM در صنعت و حرفه**

شبکه‌های LSTM به دلیل توانایی در پردازش توالی‌های پیچیده، در حوزه‌های مختلف کاربرد دارند:

|  |  |  |
| --- | --- | --- |
| حوزه | کاربرد | مثال عملی |
| **پردازش زبان طبیعی (NLP)** | ترجمه ماشینی، تحلیل احساسات، تولید متن، تکمیل خودکار متن | پیش‌بینی کلمه بعدی در کیبوردهای هوشمند (مانند Gboard) |
| **تشخیص گفتار** | تبدیل گفتار به متن، تشخیص دستورات صوتی | دستیارهای صوتی مانند Siri و Google Assistant |
| **سری‌های زمانی** | پیش‌بینی بازار سهام، پیش‌بینی آب‌وهوا، تشخیص ناهنجاری در داده‌های صنعتی | پیش‌بینی قیمت سهام با استفاده از داده‌های تاریخی بازار |
| **پردازش ویدئو** | شرح خودکار ویدئو، تشخیص فعالیت در ویدئو | تولید توضیحات متنی برای ویدئوهای یوتیوب |
| **پزشکی** | تحلیل توالی‌های DNA، پیش‌بینی روند بیماری‌ها | تشخیص الگوهای غیرعادی در داده‌های پزشکی مانند نوار قلب |
| **موسیقی و هنر** | تولید موسیقی، خلق متون خلاقانه | تولید ملودی‌های جدید با استفاده از داده‌های موسیقی |

**مقایسه LSTM با سایر روش‌ها**

**مقایسه با RNN سنتی:**

|  |  |  |
| --- | --- | --- |
| ویژگی | RNN | LSTM |
| **حافظه بلندمدت** | ضعیف (مشکل محوشدگی گرادیان) | قوی (مدیریت وابستگی‌های بلندمدت) |
| **تعداد پارامترها** | کمتر | بیشتر (به دلیل دروازه‌ها) |
| **پیچیدگی محاسباتی** | کمتر | بیشتر |
| **کارایی در توالی‌های طولانی** | ضعیف | قوی |

**مقایسه با GRU:**

واحدهای بازگشتی دروازه‌دار (Gated Recurrent Units یا GRU) نوع ساده‌تری از LSTM هستند که در سال 2014 معرفی شدند.

|  |  |  |
| --- | --- | --- |
| ویژگی | LSTM | GRU |
| **تعداد دروازه‌ها** | 3 (ورودی، فراموشی، خروجی) | 2 (به‌روزرسانی، بازنشانی) |
| **تعداد پارامترها** | بیشتر | کمتر |
| **سرعت محاسبات** | کندتر | سریع‌تر |
| **کارایی** | بهتر در توالی‌های پیچیده و طولانی | مناسب برای مسائل ساده‌تر و داده‌های کمتر |

**مقایسه با ترنسفورمرها:**

ترنسفورمرها (Transformers) که در مقاله معروف "Attention is All You Need" معرفی شدند، جایگزین پیشرفته‌تری برای LSTM در بسیاری از کاربردهای NLP هستند.

|  |  |  |
| --- | --- | --- |
| ویژگی | LSTM | ترنسفورمر |
| **پردازش موازی** | خیر (پردازش ترتیبی) | بله (پردازش موازی) |
| **وابستگی‌های بلندمدت** | خوب | بسیار قوی (به دلیل مکانیزم توجه) |
| **پیچیدگی محاسباتی** | کمتر در توالی‌های کوتاه | بیشتر در توالی‌های طولانی |
| **کاربردها** | مناسب برای سری‌های زمانی و داده‌های کوچک | مناسب برای NLP و داده‌های بزرگ |

**مثال عملی ساده: پیش‌بینی کلمه بعدی با LSTM**

**سناریو:**

فرض کنید می‌خواهیم کلمه بعدی در جمله "ابرها در آسمان هستند" را پیش‌بینی کنیم. با استفاده از LSTM، می‌توانیم مدلی بسازیم که بر اساس کلمات قبلی، کلمه بعدی (آسمان) را پیش‌بینی کند.

**کد نمونه در پایتون (با کتابخانه Keras):**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# داده‌های نمونه

sentences = ["ابرها در آسمان هستند", "خورشید در آسمان می‌درخشد"]

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(sentences)

sequences = tokenizer.texts\_to\_sequences(sentences)

word\_index = tokenizer.word\_index

vocab\_size = len(word\_index) + 1

# آماده‌سازی داده‌ها

max\_length = 4

X, y = [], []

for seq in sequences:

    for i in range(1, len(seq)):

        X.append(seq[:i])

        y.append(seq[i])

X = pad\_sequences(X, maxlen=max\_length, padding='pre')

y = np.array(y)

# ساخت مدل LSTM

model = Sequential()

model.add(Embedding(vocab\_size, 10, input\_length=max\_length))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(vocab\_size, activation='softmax'))

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# آموزش مدل

model.fit(X, y, epochs=100, verbose=0)

# پیش‌بینی

test\_sequence = tokenizer.texts\_to\_sequences(["ابرها در"])[0]

test\_sequence = pad\_sequences([test\_sequence], maxlen=max\_length, padding='pre')

predicted = model.predict(test\_sequence)

predicted\_word = tokenizer.index\_word[np.argmax(predicted)]

print(f"کلمه پیش‌بینی‌شده: {predicted\_word}")

**خروجی مورد انتظار:** کلمه پیش‌بینی‌شده: آسمان

**توضیح کد:**

1. **آماده‌سازی داده‌ها:** متن به توکن‌ها تبدیل شده و به صورت توالی‌های عددی درمی‌آید.
2. **ساخت مدل:** از لایه Embedding برای تبدیل کلمات به بردار، لایه LSTM برای پردازش توالی و لایه Dense برای پیش‌بینی کلمه بعدی استفاده شده است.
3. **آموزش و پیش‌بینی:** مدل با داده‌های آموزشی آموزش داده شده و برای پیش‌بینی کلمه بعدی استفاده می‌شود.

**نکات تکمیلی و پیشنهادها**

1. **ترکیب با سایر روش‌ها:**
   * **ترکیب با CNN:** برای پردازش داده‌های چندبعدی (مانند تصاویر و توالی‌ها)، می‌توان LSTM را با شبکه‌های کانولوشنی (CNN) ترکیب کرد. مثال: شرح خودکار تصاویر.
   * **ترکیب با ترنسفورمرها:** در مسائل NLP پیشرفته، می‌توان از LSTM برای پیش‌پردازش و ترنسفورمرها برای مدل‌سازی نهایی استفاده کرد.
   * **استفاده از Attention:** افزودن مکانیزم توجه (Attention) به LSTM می‌تواند کارایی را در وظایف پیچیده مانند ترجمه ماشینی بهبود بخشد.
2. **بهینه‌سازی مدل:**
   * از تکنیک‌هایی مانند Dropout برای جلوگیری از بیش‌برازش استفاده کنید.
   * الگوریتم‌های بهینه‌سازی مانند Adam یا RMSprop را برای آموزش سریع‌تر به کار ببرید.
   * برای داده‌های بزرگ، از معماری‌های سبک‌تر مانند GRU استفاده کنید.
3. **چالش‌ها و راه‌حل‌ها:**
   * **پیچیدگی محاسباتی:** برای کاهش زمان آموزش، از سخت‌افزارهای GPU یا TPU استفاده کنید.
   * **داده‌های ناکافی:** از تکنیک‌های افزایش داده (Data Augmentation) یا یادگیری انتقالی (Transfer Learning) بهره ببرید.
4. **کاربرد در بازار کار:**
   * **مهارت‌های مورد نیاز:** تسلط بر پایتون، کتابخانه‌های Keras/TensorFlow/PyTorch، و مفاهیم یادگیری عمیق.
   * **حوزه‌های شغلی:** تحلیل داده، مهندسی یادگیری ماشین، توسعه هوش مصنوعی در شرکت‌های فناوری، مالی، پزشکی و غیره.
   * **پروژه‌های عملی:** ساخت چت‌بات، پیش‌بینی سری‌های زمانی، یا تحلیل احساسات متن می‌تواند رزومه شما را تقویت کند.

**نتیجه‌گیری**

شبکه‌های عصبی بازگشتی و به‌ویژه LSTM‌ها ابزارهای قدرتمندی برای پردازش داده‌های توالی‌دار هستند. LSTM با حل مشکلات RNN‌های سنتی، مانند محوشدگی گرادیان، امکان پردازش توالی‌های طولانی و پیچیده را فراهم کرده است. این شبکه‌ها در حوزه‌های مختلفی از NLP تا پیش‌بینی سری‌های زمانی کاربرد دارند و با ترکیب با سایر روش‌ها، می‌توانند عملکرد بهتری ارائه دهند. برای متخصصان داده و برنامه‌نویسان، یادگیری LSTM و پیاده‌سازی پروژه‌های عملی با آن می‌تواند گامی مهم در مسیر موفقیت حرفه‌ای باشد.

**🔹 1. Vanilla LSTM (LSTM استاندارد)**

**📌 توضیح:**

مدل پایه‌ی LSTM که شامل دروازه‌های فراموشی، ورودی، خروجی و حالت سلول است.

**✅ کاربرد:**

* پیش‌بینی سری‌های زمانی
* تحلیل احساسات
* ترجمه ماشینی

**🔹 2. Bidirectional LSTM (BiLSTM)**

**📌 توضیح:**

مدل LSTM که هم به جلو و هم به عقب در دنباله نگاه می‌کند (دو LSTM موازی).  
خروجی از هر دو جهت ترکیب می‌شود.

**✅ کاربرد:**

* برچسب‌گذاری توالی (مثل Named Entity Recognition)
* تحلیل معنایی جملات
* تشخیص گفتار

**⚖️ تفاوت با LSTM:**

LSTM فقط اطلاعات گذشته را می‌بیند؛  
BiLSTM هم گذشته و هم آینده را برای هر لحظه در نظر می‌گیرد.

**🔹 3. Stacked LSTM (چندلایه‌ای)**

**📌 توضیح:**

مدل LSTM چندلایه که خروجی لایه پایینی، ورودی لایه بالایی می‌شود.

**✅ کاربرد:**

* مسائل پیچیده‌تر یادگیری عمیق
* ترجمه ماشینی
* تولید متن

**📌 مزیت:**

قدرت مدل بیشتر، ولی هزینه محاسباتی هم بالاتر می‌رود.

**🔹 4. CNN-LSTM**

**📌 توضیح:**

ترکیبی از شبکه‌های کانولوشنی (برای استخراج ویژگی‌ها) و LSTM (برای یادگیری توالی زمانی).

**✅ کاربرد:**

* تحلیل ویدیو (مثلاً تشخیص فعالیت انسانی)
* داده‌های فضایی-زمانی

**🔹 5. ConvLSTM**

**📌 توضیح:**

در این مدل، ضرب‌های ماتریسی با کانولوشن جایگزین می‌شوند.  
مناسب برای داده‌هایی با ساختار دو یا چند بعدی مثل تصاویر متوالی یا ویدیو.

**✅ کاربرد:**

* پیش‌بینی ویدیو فریم بعدی
* پیش‌بینی آب و هوا
* رادار و بینایی ماشین

**🔹 6. Attention-based LSTM**

**📌 توضیح:**

مدل LSTM به همراه مکانیزم توجه (Attention) که به مدل اجازه می‌دهد روی بخش‌های مهم‌تر دنباله تمرکز کند.

**✅ کاربرد:**

* ترجمه ماشینی
* خلاصه‌سازی متن
* مدل‌های مکالمه

**🔹 7. Peephole LSTM**

**📌 توضیح:**

دروازه‌ها به جای اینکه فقط به حالت مخفی نگاه کنند، مستقیماً به **حالت سلول قبلی** نیز دسترسی دارند.

**✅ کاربرد:**

* بهبود در مدل‌سازی الگوهای پیچیده زمانی

**🔹 8. Tree-LSTM**

**📌 توضیح:**

ساختار LSTM که به جای دنباله خطی، در ساختار **درختی** کار می‌کند. برای داده‌های درختی مثل ساختار نحوی زبان طبیعی.

**✅ کاربرد:**

* درک نحو زبان
* درخت تجزیه نحوی
* تحلیل ساختار زبان طبیعی

**🔹 9. Residual LSTM**

**📌 توضیح:**

بین لایه‌های LSTM از مسیرهای میان‌بر (residual connections) استفاده می‌شود.

**✅ کاربرد:**

* جلوگیری از مشکل ناپدید شدن گرادیان در LSTMهای عمیق

**🔹 10. Phased LSTM**

**📌 توضیح:**

نوعی LSTM که زمان‌بندی ورودی‌ها را بهتر مدل می‌کند. برای داده‌هایی با **زمان‌های نامنظم** طراحی شده.

**✅ کاربرد:**

* داده‌های سنسور
* داده‌های سری زمانی با تاخیر نامنظم

**🔹 11. GRU (Gated Recurrent Unit)**

**📌 توضیح:**

GRU یک نوع ساده‌شده از LSTM است که دو دروازه دارد: **دروازه به‌روزرسانی (update gate)** و **دروازه بازنشانی (reset gate)**.

**✅ مزیت‌ها:**

* ساختار ساده‌تر نسبت به LSTM
* تعداد پارامتر کمتر ➜ یادگیری سریع‌تر
* عملکرد مشابه یا بهتر در برخی وظایف

**📐 فرمول‌ها:**

* **دروازه به‌روزرسانی:**
* **دروازه بازنشانی:**
* **کاندیدای حالت جدید:**
* **حالت نهایی:**

**✅ کاربرد:**

* تحلیل احساسات
* ترجمه ماشینی
* تشخیص گفتار
* زمانی که سرعت آموزش مهم‌تر از دقت نهایی است

**⚖️ تفاوت GRU با LSTM:**

|  |  |  |
| --- | --- | --- |
| مورد | LSTM | GRU |
| دروازه‌ها | ۳ (ورودی، فراموشی، خروجی) | ۲ (به‌روزرسانی، بازنشانی) |
| حالت‌ها | حالت سلول (C) و حالت مخفی (h) | فقط حالت مخفی (h) |
| پیچیدگی | بیشتر | کمتر |
| عملکرد | بهتر در مسائل پیچیده | سریع‌تر در آموزش |

**📊 جدول نهایی مقایسه انواع LSTM و GRU**

|  |  |  |
| --- | --- | --- |
| نوع | ویژگی | کاربرد اصلی |
| Vanilla LSTM | پایه‌ای | سری زمانی، NLP |
| BiLSTM | دوجهته | درک زبان، گفتار |
| Stacked LSTM | چند لایه | مدل‌سازی پیچیده |
| CNN-LSTM | کانولوشن + LSTM | ویدیو، تصویر |
| ConvLSTM | LSTM کانولوشنی | بینایی ماشین |
| Attention LSTM | با توجه | ترجمه، خلاصه |
| Peephole LSTM | دید به سلول | الگوهای زمانی |
| Tree-LSTM | ساختار درختی | تحلیل زبان |
| Residual LSTM | با مسیر میان‌بر | LSTM‌های عمیق |
| Phased LSTM | زمان‌بندی نامنظم | سنسور، IoT |
| GRU | ساده‌تر، سریع‌تر | یادگیری سریع، NLP |

**✅ در Keras / TensorFlow**

python

CopyEdit

tf.keras.layers.LSTM(

units,

activation='tanh',

recurrent\_activation='sigmoid',

use\_bias=True,

return\_sequences=False,

return\_state=False,

go\_backwards=False,

stateful=False,

unroll=False,

dropout=0.0,

recurrent\_dropout=0.0,

kernel\_initializer='glorot\_uniform',

recurrent\_initializer='orthogonal',

bias\_initializer='zeros',

unit\_forget\_bias=True,

...

)

**📌 آرگومان‌های مهم:**

|  |  |
| --- | --- |
| آرگومان | توضیح |
| units | تعداد نرون‌ها در لایه LSTM (خروجی حالت مخفی) |
| activation | تابع فعال‌سازی برای خروجی (پیش‌فرض 'tanh') |
| recurrent\_activation | تابع فعال‌سازی برای دروازه‌ها (پیش‌فرض 'sigmoid') |
| use\_bias | آیا از بایاس استفاده شود؟ (پیش‌فرض: True) |
| return\_sequences | اگر True، خروجی برای تمام گام‌های زمانی بازمی‌گردد (برای لایه بعدی RNN مناسب است) |
| return\_state | اگر True، خروجی نهایی و حالت‌ها را بازمی‌گرداند (h\_t, c\_t) |
| go\_backwards | اگر True، دنباله ورودی را از آخر به اول پردازش می‌کند |
| stateful | اگر True، حالت‌ها را بین batchها نگه می‌دارد |
| unroll | اگر True، حلقه زمان باز می‌شود (سریع‌تر ولی حافظه بیشتر) |
| dropout | dropout روی ورودی |
| recurrent\_dropout | dropout روی حالت مخفی در زمان‌های مختلف |
| unit\_forget\_bias | اگر True، مقدار اولیه بایاس در دروازه فراموشی برابر 1 تنظیم می‌شود |

**✅ در PyTorch**

python

CopyEdit

torch.nn.LSTM(

input\_size,

hidden\_size,

num\_layers=1,

bias=True,

batch\_first=False,

dropout=0.0,

bidirectional=False,

...

)

**📌 آرگومان‌های مهم:**

|  |  |
| --- | --- |
| آرگومان | توضیح |
| input\_size | تعداد ویژگی‌های ورودی |
| hidden\_size | تعداد نرون‌های حالت مخفی |
| num\_layers | تعداد لایه‌های LSTM روی هم (stacked) |
| bias | استفاده از بایاس یا نه |
| batch\_first | اگر True، ورودی باید به شکل (batch, seq, feature) باشد |
| dropout | dropout بین لایه‌ها (فقط اگر num\_layers > 1) |
| bidirectional | اگر True، یک LSTM دوجهته خواهد بود |

**📌 خروجی در Keras و PyTorch:**

* در Keras:

output = LSTM(units)(input)

* در PyTorch:

output, (h\_n, c\_n) = lstm(input)

**✅ جدول مقایسه آرگومان‌های مهم RNN, LSTM, GRU در Keras**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| آرگومان | توضیح | LSTM | GRU | SimpleRNN |
| units | تعداد نرون‌ها در لایه | ✅ | ✅ | ✅ |
| activation | تابع فعال‌سازی خروجی | 'tanh' (پیش‌فرض) | 'tanh' | 'tanh' |
| recurrent\_activation | تابع فعال‌سازی دروازه‌ها | 'sigmoid' | 'sigmoid' | ندارد |
| use\_bias | آیا بایاس استفاده شود؟ | ✅ | ✅ | ✅ |
| return\_sequences | خروجی برای همه زمان‌ها؟ | ✅ | ✅ | ✅ |
| return\_state | خروجی حالت نهایی (h, c) | ✅ (h\_t, c\_t) | ✅ (h\_t) | ✅ (h\_t) |
| go\_backwards | توالی را برعکس پردازش کند؟ | ✅ | ✅ | ✅ |
| stateful | حفظ حالت بین batchها | ✅ | ✅ | ✅ |
| unroll | بازکردن حلقه زمان | ✅ | ✅ | ✅ |
| dropout | روی ورودی‌ها | ✅ | ✅ | ✅ |
| recurrent\_dropout | روی حالت مخفی | ✅ | ✅ | ✅ |
| reset\_after | تنظیم زمان بازنشانی (برای GRU) | ❌ | ✅ | ❌ |

**✅ جدول مقایسه در PyTorch (nn.RNN, nn.LSTM, nn.GRU)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| آرگومان | توضیح | LSTM | GRU | RNN |
| input\_size | ابعاد ورودی | ✅ | ✅ | ✅ |
| hidden\_size | تعداد نرون‌ها | ✅ | ✅ | ✅ |
| num\_layers | تعداد لایه‌ها (stacked) | ✅ | ✅ | ✅ |
| bias | استفاده از بایاس | ✅ | ✅ | ✅ |
| batch\_first | شکل ورودی (batch, seq, feature) | ✅ | ✅ | ✅ |
| dropout | بین لایه‌ها (اگر num\_layers > 1) | ✅ | ✅ | ✅ |
| bidirectional | دو جهته بودن | ✅ | ✅ | ✅ |
| proj\_size | برای LSTM فقط، فشرده‌سازی خروجی | ✅ | ❌ | ❌ |

**✅ خروجی‌ها**

**🔷 Keras:**

python

output = LSTM(units)(inputs) # if return\_sequences=False

output, state = LSTM(units, return\_state=True)(inputs)

**🔷 PyTorch:**

CopyEdit

output, (h\_n, c\_n) = nn.LSTM(input\_size, hidden\_size)(input)

output, h\_n = nn.GRU(input\_size, hidden\_size)(input)

**✅ جدول انواع RNNها + لایه مورد استفاده در Keras**

|  |  |  |  |
| --- | --- | --- | --- |
| نوع | لایه در Keras | پیاده‌سازی مستقیم؟ | توضیح |
| Simple RNN | SimpleRNN | ✅ | نسخه پایه RNN |
| LSTM | LSTM | ✅ | دارای دروازه‌های ورودی، فراموشی، خروجی |
| GRU | GRU | ✅ | ساده‌تر از LSTM، دو دروازه‌ای |
| Bidirectional LSTM | Bidirectional(LSTM(...)) | ✅ | دو LSTM معکوس و مستقیم |
| Stacked LSTM | چند LSTM(..., return\_sequences=True) پشت سر هم | ✅ | چندلایه‌ای |
| Peephole LSTM | سفارشی با LSTMCell(peephole=True) یا دستی | 🔶 محدود/سفارشی | اتصال مستقیم به cell state |
| Attention LSTM | ترکیب LSTM + Attention layer | 🔶 دستی | تمرکز روی ورودی مهم |
| CNN-LSTM | Conv2D ➜ Reshape ➜ LSTM | ✅ | برای ویدیوها و تصاویر |
| ConvLSTM | ConvLSTM2D | ✅ | کانولوشنی-زمانی |
| Tree-LSTM | ندارد | ❌ فقط دستی یا در PyTorch | ساختار درختی |
| Phased LSTM | ندارد | ❌ فقط با کتابخانه خارجی | زمان‌بندی نامنظم |
| Residual LSTM | دستی با skip-connection | 🔶 سفارشی | اضافه کردن خروجی لایه قبل |

**✅ مثال‌های کوتاه برای هر نوع (Keras)**

**🔸 Simple RNN**

python

CopyEdit

from tensorflow.keras.layers import SimpleRNN

model.add(SimpleRNN(64, input\_shape=(100, 32)))

**🔸 LSTM**

python

CopyEdit

from tensorflow.keras.layers import LSTM

model.add(LSTM(64, input\_shape=(100, 32)))

**🔸 GRU**

python

CopyEdit

from tensorflow.keras.layers import GRU

model.add(GRU(64, input\_shape=(100, 32)))

**🔸 Bidirectional LSTM**

python

CopyEdit

from tensorflow.keras.layers import Bidirectional

model.add(Bidirectional(LSTM(64), input\_shape=(100, 32)))

**🔸 Stacked LSTM**

python

CopyEdit

model.add(LSTM(64, return\_sequences=True, input\_shape=(100, 32)))

model.add(LSTM(32))

**🔸 CNN-LSTM**

python

CopyEdit

model.add(Conv2D(16, (3,3), activation='relu', input\_shape=(10, 64, 64, 1)))

model.add(Reshape((10, -1)))

model.add(LSTM(32))

**🔸 ConvLSTM**

python

CopyEdit

from tensorflow.keras.layers import ConvLSTM2D

model.add(ConvLSTM2D(filters=32, kernel\_size=(3,3), input\_shape=(10, 64, 64, 1)))

**🔸 Attention + LSTM (سفارشی)**

python

CopyEdit

from tensorflow.keras.layers import Attention

query = LSTM(64, return\_sequences=True)(x)

value = LSTM(64, return\_sequences=True)(x)

attn\_out = Attention()([query, value])