**به نام خدا**

**جلسه ۱: آشنایی عمیق با RNN و مشکلات آن :**

**شبکه‌های عصبی بازگشتی (RNN - Recurrent Neural Networks)** یک نوع خاص از شبکه‌های عصبی هستند که برای **پردازش** **داده‌های متوالی (Sequential Data)** طراحی شده‌اند. در اینجا به صورت مفصل ولی با زبان ساده توضیح می‌دهم:

### ****مفهوم اصلی RNN:****

RNN ها دارای **حافظه داخلی** هستند که اطلاعات مراحل قبلی را در محاسبات فعلی استفاده می‌کنند.  
🔹 مثل وقتی که شما یک جمله می‌خوانید و برای درک کلمه جدید، از کلمات قبلی کمک می‌گیرید.

### ****ساختار پایه RNN:****

from tensorflow.keras.layers import SimpleRNN

model.add(SimpleRNN(units=64, input\_shape=(10, 50))) *# 10 گام زمانی، 50 ویژگی در هر گام*

1. **واحدهای بازگشتی (Units)**:
   * تعداد نورون‌های لایه (مثلاً 64)
   * هر واحد هم‌زمان:
     + ورودی فعلی (X\_t) را پردازش می‌کند
     + حالت قبلی (h\_{t-1}) را به خاطر می‌آورد
2. **حلقه بازگشتی**:  
   [https://miro.medium.com/v2/resize:fit:1400/1\*XICrq5q3E-6mHsa36KJQVw.gif](https://miro.medium.com/v2/resize:fit:1400/1*XICrq5q3E-6mHsa36KJQVw.gif)
   * در هر گام زمانی (t):
     + حالت جدید (h\_t) = تابعی از (X\_t و h\_{t-1})
     + خروجی (y\_t) = تابعی از (h\_t)

### ****انواع RNN:****

1. **یک به یک (One-to-One)**:
   * معماری پایه (غیرمتوالی)
   * مثال: طبقه‌بندی تصاویر
2. **یک به چند (One-to-Many)**:
   * یک ورودی، چند خروجی
   * مثال: تولید کپشن از تصویر
3. **چند به یک (Many-to-One)**:
   * چند ورودی، یک خروجی
   * مثال: تحلیل احساسات متن
4. **چند به چند (Many-to-Many)**:
   * دو حالت:
     + همزمان (مانند ترجمه ماشینی)
     + با تاخیر (مانند پیش‌بینی سری‌های زمانی)

### ****پارامترهای کلیدی در پیاده‌سازی:****

SimpleRNN(

units=128, *# تعداد نورون‌ها*

activation='tanh', *# تابع فعال‌سازی (معمولاً tanh یا relu)*

return\_sequences=False, *# آیا همه خروجی‌های متوالی را برگرداند؟*

input\_shape=(time\_steps, features) *# شکل ورودی*

)

### ****مقایسه با سایر مدل‌ها:****

| **ویژگی** | **RNN معمولی** | **LSTM** | **GRU** |
| --- | --- | --- | --- |
| **حافظه کوتاه‌مدت** | ضعیف | قوی | قوی |
| **سرعت آموزش** | سریع‌تر | کندتر | متوسط |
| **پیچیدگی** | ساده | پیچیده | متوسط |
| **مقابله با Vanishing Gradient** | ❌ | ✅ | ✅ |

### ****مثال کاربردی (پیش‌بینی متن):****

from tensorflow.keras.models import Sequential

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

model = Sequential([

Embedding(input\_dim=10000, output\_dim=64), *# لایه تبدیل کلمات به بردار*

SimpleRNN(128, return\_sequences=True), *# RNN با برگرداندن تمام خروجی‌ها*

SimpleRNN(64), *# لایه RNN دوم*

Dense(10000, activation='softmax') *# پیش‌بینی کلمه بعدی*

])

**توضیح کد :**

### ****لایه****Embedding:

Embedding(input\_dim=10000, output\_dim=64)

* **کاربرد**: تبدیل کلمات به بردارهای عددی (کدگذاری معنی‌دار).
* **پارامترها**:
  + input\_dim=10000: اندازه دایره‌واژگان (۱۰۰۰۰ کلمه منحصر به فرد).
  + output\_dim=64: هر کلمه به یک بردار ۶۴ بعدی تبدیل می‌شود.
* **مثال**:
  + کلمه "گربه" → [0.2, -0.5, 0.7, ..., 0.1] (۶۴ عدد).

### ****لایه****SimpleRNN****اول****:

SimpleRNN(128, return\_sequences=True)

* **کاربرد**: پردازش دنباله کلمات با حفظ حافظه کوتاه‌مدت.
* **پارامترها**:
  + 128: تعداد نورون‌های لایه.
  + return\_sequences=True: **تمام خروجی‌های متوالی** را نگه می‌دارد (برای اتصال به لایه RNN بعدی ضروری است).
* **مثال**:
  + برای جمله "من امروز ..."، حالت RNN بعد از پردازش "من" را به محاسبات "امروز" منتقل می‌کند.

### ****لایه****SimpleRNN****دوم****:

* **کاربرد**: استخراج ویژگی‌های سطح بالاتر از دنباله.
* **تفاوت با لایه اول**:
  + return\_sequences=False (پیش‌فرض): فقط **آخرین خروجی** را برمی‌گرداند (نه تمام مراحل زمانی).
  + ۶۴ نورون برای کاهش ابعاد داده.

### ****لایه****Dense****(خروجی)****:

1. Dense(10000, activation='softmax')
2. **کاربرد**: پیش‌بینی کلمه بعدی با توزیع احتمال.
3. **پارامترها**:
   1. 10000: تعداد کلاس‌ها (همان اندازه دایره‌واژگان).
   2. activation='softmax': احتمال هر کلمه را محاسبه می‌کند (جمع همه احتمالات = ۱).
4. **مثال خروجی**:
   1. پس از پردازش "من امروز"، مدل پیش‌بینی می‌کند:
      1. "رفتم" → ۷۰% احتمال
      2. "خوشحالم" → ۲۰% احتمال
      3. ...

### ****جریان کلی داده****:

کلمات → Embedding → RNN (128) → RNN (64) → Dense → پیش‌بینی کلمه بعدی

### ****هدف مدل****:

پس از آموزش، مدل می‌تواند:

* متن ناتمام را کامل کند (مثال: "هوا امروز بسیار ..." → "گرم است").
* در **چت‌بات‌ها** یا **پیشنهاد خودکار متن** استفاده شود

### ****محدودیت‌ها****:

1. **مشکل Vanishing Gradient**: در جملات طولانی، ارتباط کلمات ابتدایی فراموش می‌شود.
   * **راه‌حل**: استفاده از **LSTM** یا **GRU** به جای SimpleRNN.
2. **محاسبات سنگین**: به دلیل تعداد پارامترهای زیاد (مخصوصاً در لایه Dense).

### ****بهبود مدل****:

* جایگزینی SimpleRNN با **LSTM**:

from tensorflow.keras.layers import LSTM

model = Sequential([

Embedding(10000, 64),

LSTM(128, return\_sequences=True),

LSTM(64),

Dense(10000, activation='softmax')

])

### ****مشکلات RNN معمولی:****

1. **Vanishing Gradient**:
   * در دنباله‌های طولانی، اطلاعات اولیه فراموش می‌شود.
   * راه‌حل: استفاده از **LSTM** یا **GRU**
2. **حافظه محدود**:
   * برای وابستگی‌های خیلی طولانی مناسب نیست.

### ****کاربردهای RNN:****

1. پردازش زبان طبیعی (NLP)
   * ترجمه ماشینی
   * تولید متن
2. پیش‌بینی سری‌های زمانی
   * قیمت سهام
   * پیش‌بینی آب و هوا
3. تشخیص گفتار
4. تحلیل احساسات

**مکانیزم RNN:**

* نحوه پردازش داده‌های متوالی
* فرمول ریاضی حاکم بر RNN:

h\_t = tanh(W\_{hh}h\_{t-1} + W\_{xh}x\_t + b\_h)

* نمایش گرافیکی گردش اطلاعات در RNN
* **محدودیت‌های RNN:**
  + مشکل Vanishing/Exploding Gradients
  + عدم توانایی در یادگیری وابستگی‌های بلندمدت
  + مثال کاربردی: چرا RNN در متن‌های طولانی عملکرد ضعیفی دارد؟

### ****جمع‌بندی:****

* RNN برای داده‌های **وابسته به زمان/ترتیب** عالی است.
* نسخه‌های پیشرفته‌تر (**LSTM** و **GRU**) مشکلات RNN پایه را حل می‌کنند.
* در **پردازش متن** و **سری‌های زمانی** کاربرد فراوان دارد.