Deep RNNs (Stacked RNNs)

🔁 What is a Deep RNN?

A **Deep RNN** is simply a **stack** (or **multi-layer**) of RNN layers.

In a **basic RNN**, you have **one recurrent layer**, which processes sequences step by step. But with **deep RNNs**, you stack **multiple RNN layers** on top of each other, allowing the model to learn **more complex patterns** and **abstract representations**.

```
Input ➡ RNN ➡ RNN ➡ RNN ➡ Output
```

🤖 Why use Deep RNNs?

Advantage	Meaning
🧠 More representation power	Learns deeper patterns in sequences
🔁 Better temporal modeling	Captures long-term dependencies better
💬 Handles complex language tasks	Like translation, speech recognition, etc.
🔼Captures the hierarchy	

Deep LSTM in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=64))
model.add(LSTM(128, return_sequences=True))  # 1st LSTM layer
model.add(LSTM(64))  # 2nd LSTM layer
model.add(Dense(1, activation='sigmoid'))  # Output layer

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accur
```

```
acy'])
model.summary()
```

`input_dim` → How many unique words we have

`output_dim` → Size of each word vector

> 🔄 **Note: `return_sequences=True` is required for all layers except the last, so the output is still a sequence that the next RNN layer can read.**

Sentiment analysis:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense,LSTM,GR
U
```

```
# Load the IMDb dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# Pad sequences to have the same length
x_train = pad_sequences(x_train, maxlen=100)
x_test = pad_sequences(x_test, maxlen=100)
```

Build a model:

```
# Define the LSTM model
model = Sequential([
    Embedding(10000, 32),
    LSTM(5, return_sequences=True),
```

```
    LSTM(5),
    Dense(1, activation='sigmoid')
])

model.summary()
```

Compile:

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
acy'])
```

Fit:

```
# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split=
0.2)
```

```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 27ms/step - accuracy: 0.6614 - loss: 0.5871 - val_accuracy: 0.8258 - val_loss: 0.3948
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 26ms/step - accuracy: 0.8882 - loss: 0.2965 - val_accuracy: 0.8396 - val_loss: 0.3659
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 25ms/step - accuracy: 0.9260 - loss: 0.2044 - val_accuracy: 0.8402 - val_loss: 0.3792
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 25ms/step - accuracy: 0.9507 - loss: 0.1505 - val_accuracy: 0.8372 - val_loss: 0.4409
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 25ms/step - accuracy: 0.9638 - loss: 0.1155 - val_accuracy: 0.8298 - val_loss: 0.4808
```

You can also use GRU:

```
# Define the GRU model
model = Sequential([
    Embedding(10000, 32),
    GRU(5, return_sequences=True),
    GRU(5),
    Dense(1, activation='sigmoid')
```

```
])

model.summary()
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 27ms/step - accuracy: 0.6448 - loss: 0.6085 - val_accuracy: 0.8254 - val_loss: 0.4096
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 26ms/step - accuracy: 0.8750 - loss: 0.3155 - val_accuracy: 0.8382 - val_loss: 0.3747
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 26ms/step - accuracy: 0.9180 - loss: 0.2229 - val_accuracy: 0.8380 - val_loss: 0.3786
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 25ms/step - accuracy: 0.9479 - loss: 0.1555 - val_accuracy: 0.8334 - val_loss: 0.4212
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 17s 26ms/step - accuracy: 0.9653 - loss: 0.1119 - val_accuracy: 0.8350 - val_loss: 0.4947
```

When to use Deep RNNs?

- Complex tasks
    - eg. Speech Recognition, Machine Translation
- Large datasets
- Sufficient Computational power
- Not satisfied with simple models