edureka!
a Veranda Enterprise | ILLINOIS TECH

POST GRADUATE PROGRAM IN GENERATIVE AI AND ML

Deep Learning and Neural Network Architectures

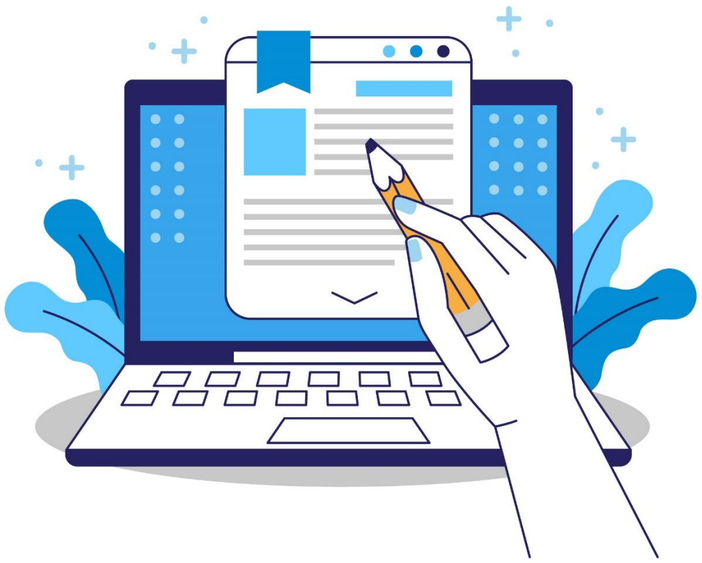
edureka!
a Veranda Enterprise

ILLINOIS TECH

Recurrent Neural Networks (RNNs)

Topics

- Introduction to RNN
- RNN and Unrolled RNNs (Architecture)
- Input/output Formats in RNNs
- Forward and Backward Pass in RNNs
- Vanishing gradient problem
- Teacher forcing
- Sequence data and time series basics
- Time Series Forecasting with RNNs
- Tokenization and embedding layers
- Building RNNs using Keras
- Padding and masking
- Stateful vs. stateless RNNs
- Using GRUs vs. vanilla RNNs



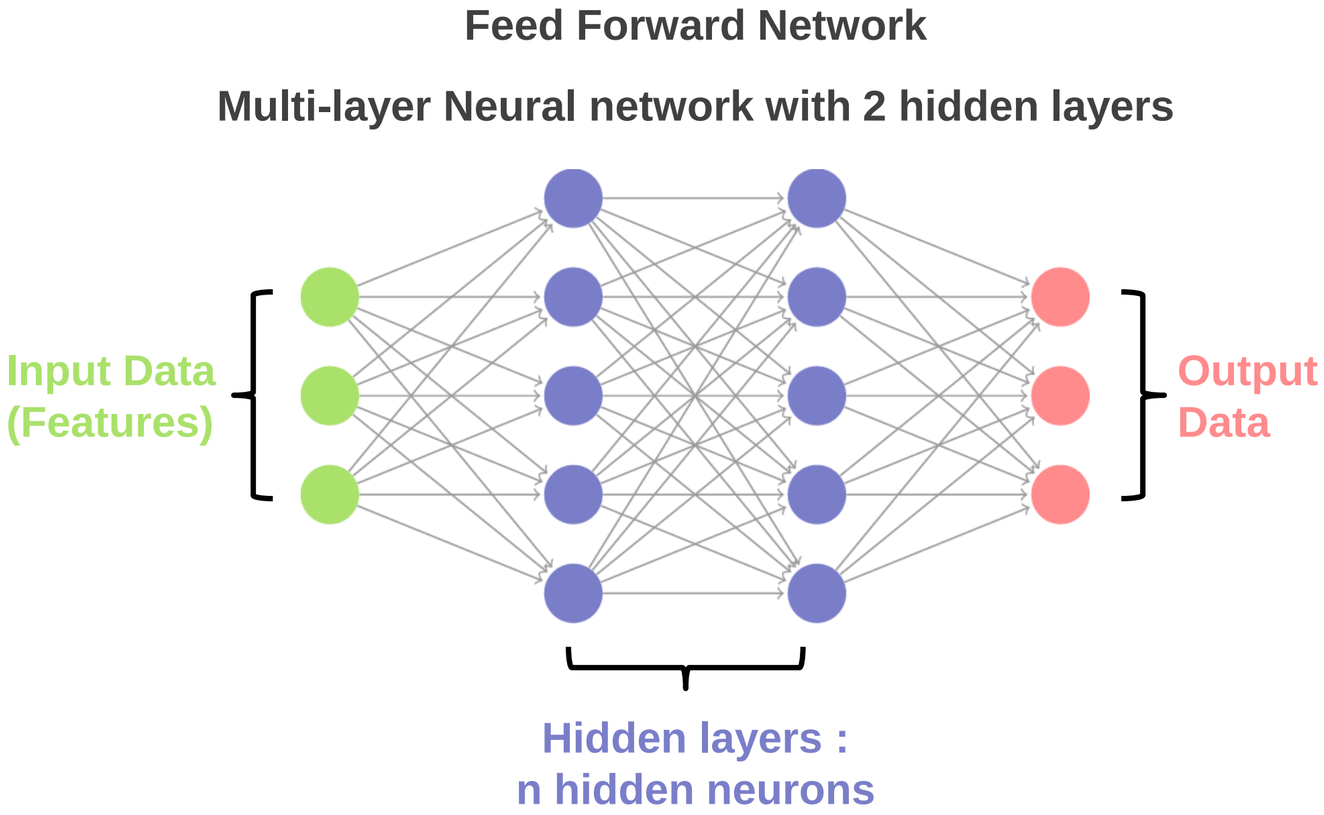
Learning Objectives

By the end of this lesson, you will be able to:

- Describe the basics of RNNs and their unrolled architecture.
- Explain input/output formats and the forward/backward pass in RNNs.
- Identify the vanishing gradient problem and apply teacher forcing, padding, and masking.
- Compare vanilla RNNs, GRUs, and stateful vs stateless models.
- Use tokenization and embedding layers to prepare sequence data.
- Build and train RNNs in Keras for time series tasks and evaluate results.

Introduction to RNN

Issues With Feed Forward Network

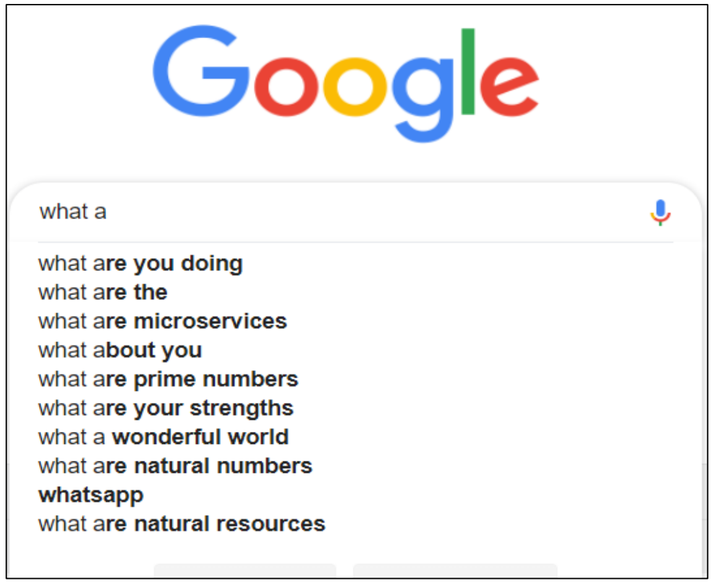


Issues with Feed Forward Network

- Inability to handle sequential data (Sentences, Stock Prices, Video Stream, etc.)
- Only the current input is considered
- Previous inputs are not memorized
- Assumes that the data points are independent of each other

Introduction To RNN

How do you think Google's Autocomplete feature work?



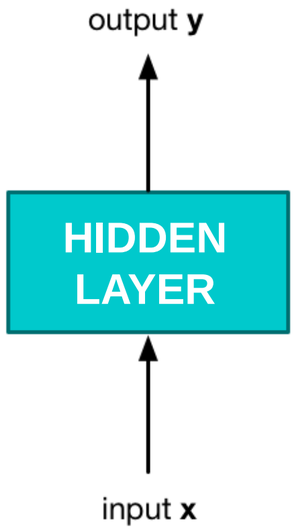
Using RNN

It analyses the data by finding the sequence of frequently occurring words, thereby building a model to predict the next word in the sentence.

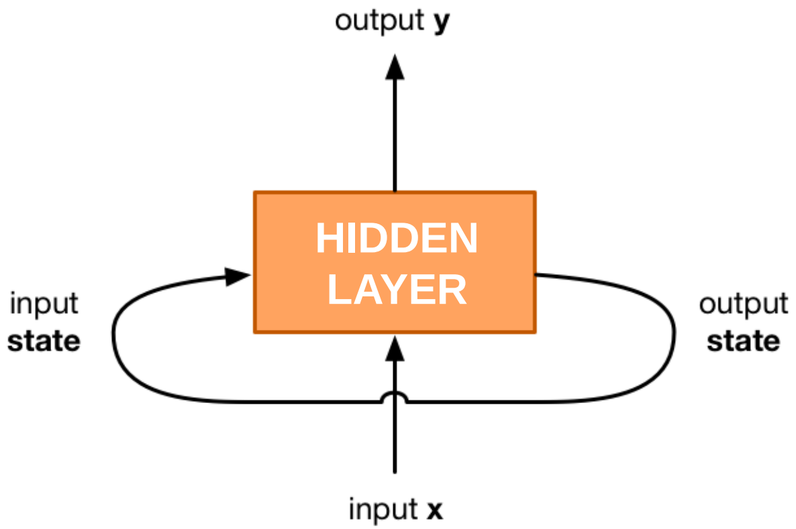
Introduction To RNN (Contd.)

Recurrent neural networks have **networks with loops in them**, allowing information to persist.

It's a special type of neural network designed for **sequence problems**.



feed-forward network



recurrent network

RNN and Unrolled RNNs (Architecture)

RNN Architecture

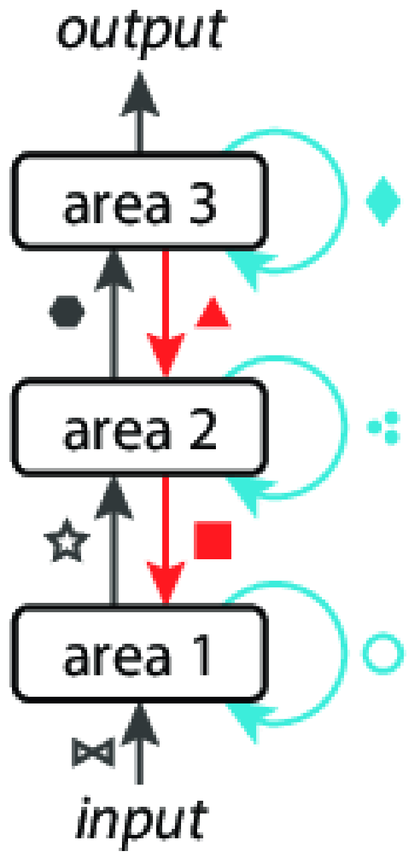
Visual input enters the first area and gets transformed by fixed neuron operations.

The **output flows** through higher areas, each adding further processing.

Lateral connections feed an area's output back into itself to guide future steps.

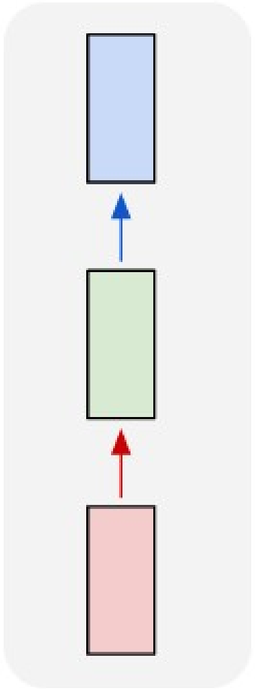
Feedback connections let higher areas influence lower ones for better adjustment.

The **network starts** with a feedforward pass and **updates activations** over time using previous results.

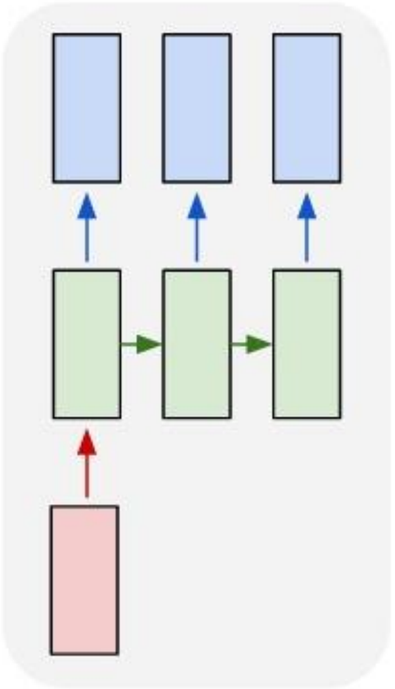


Different Architecture in RNN

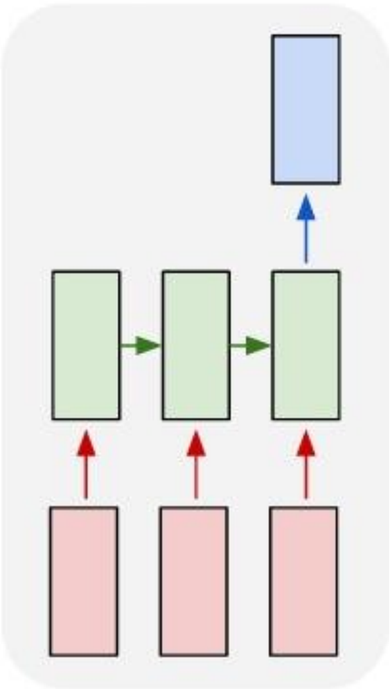
one to one



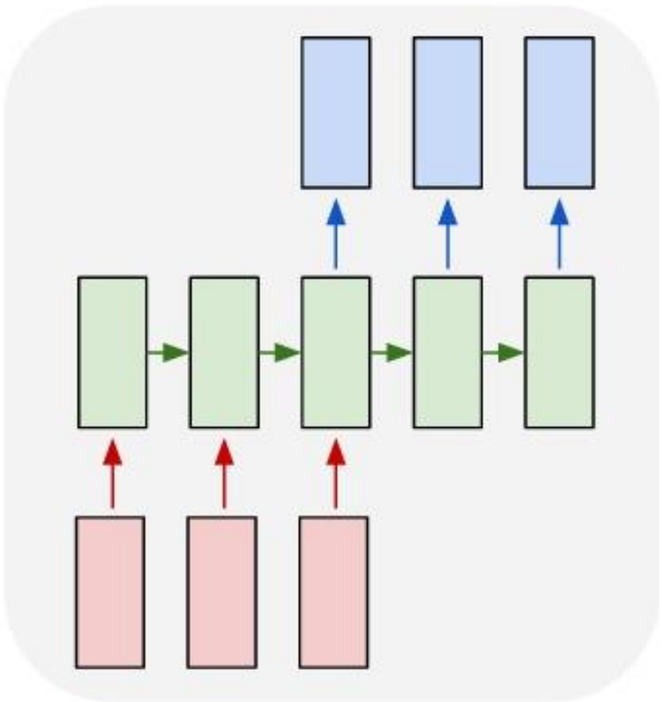
one to many



many to one



many to many



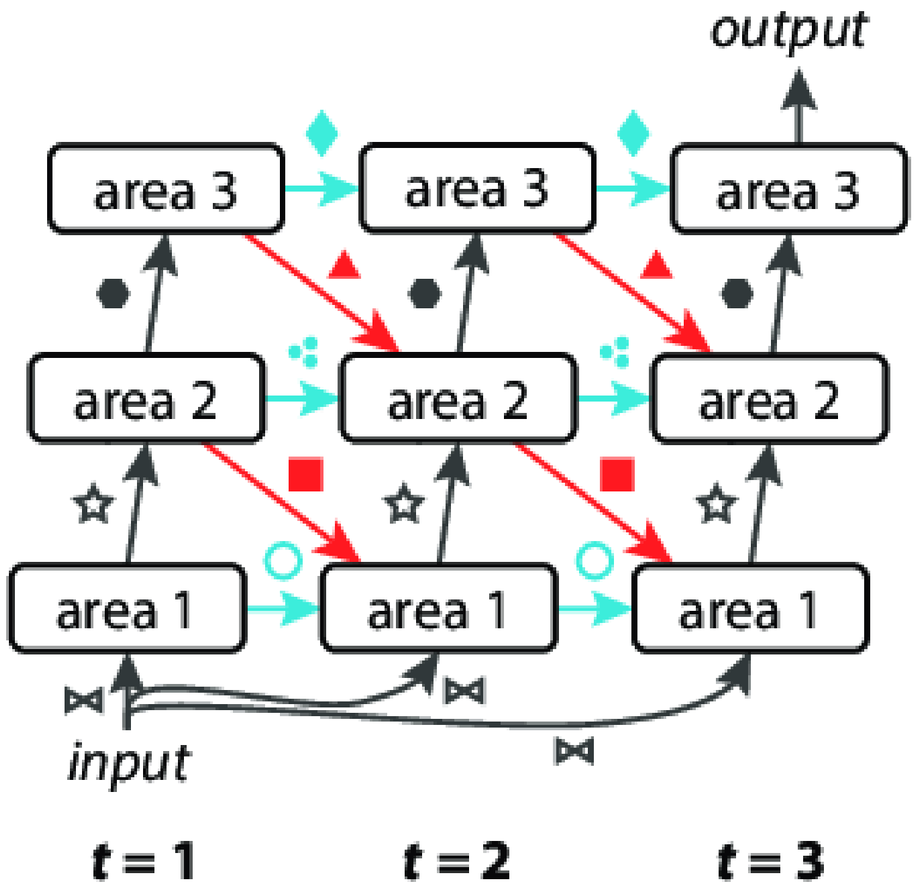
RNN Unrolled in Time

The RNN is unrolled over a **fixed number of time steps** for processing.

It can **run for any number** of time steps before producing an output.

The unrolled form visually resembles a **deeper feedforward network.**

Each time step acts like a new layer, **increasing the network's depth** and **connections**.



RNN Unrolled in Space

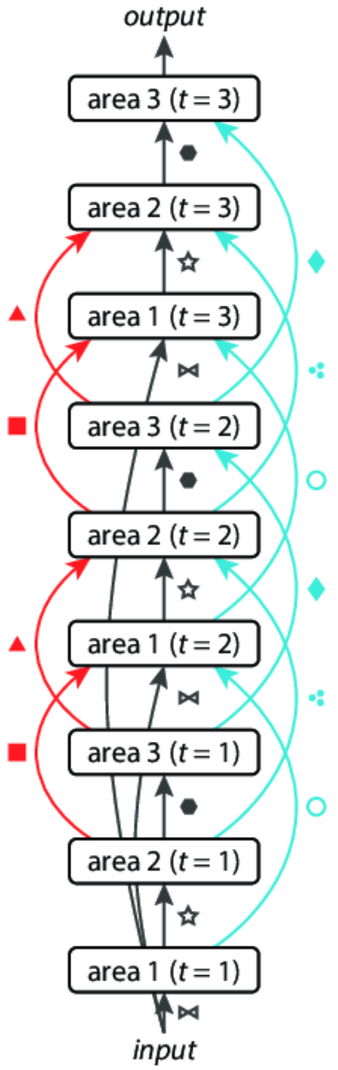
The RNN's time-based computation is reinterpreted as a **spatial structure.**

This is done by arranging the steps linearly, called unrolling in space.

The result looks like a **deep feedforward network** with many layers.

Skip connections **link non-adjacent areas** across this spatial layout.

Many of these **connections are identical**, sharing the **same weights** throughout.



Input/Output Formats in RNNs

Input Format in RNNs

RNNs take sequential data as input. The input is usually a 3D tensor with the following shape:

`(batch_size, sequence_length, input_dim)`

- **batch_size:** Number of sequences processed at once.
- **sequence_length:** Number of time steps in each sequence.
- **input_dim:** Number of features per time step (e.g., word embedding size).

Example: If you have 32 sentences (batch size) where each sentence has 10 words (sequence length), and each word is represented by a 50-dimensional vector (embedding), then: **`input_shape = (32, 10, 50)`**

Output Formats in RNNs

The output shape of an RNN depends on:

- Whether it returns the full sequence or just the final output.
- The number of units in the hidden layer.

There are two major output modes:

Many-to-One (sentiment classification)

You only care about the final output from the last time step.

Output shape: `(batch_size, hidden_units)`

Many-to-Many (translation, time series prediction)

You want output at each time step.

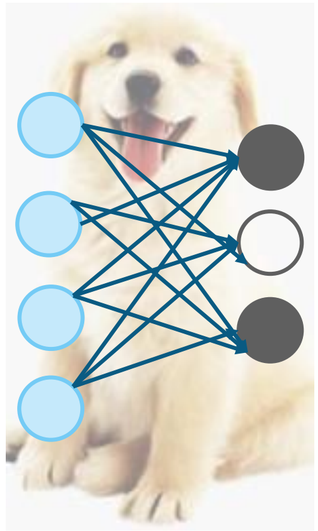
Output shape: `(batch_size, sequence_length, hidden_units)`

Forward and Backward pass in RNNs

Forward and Backward Pass in RNN

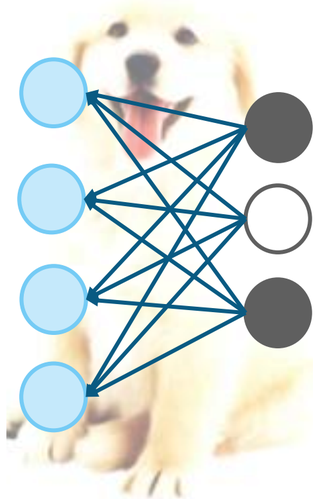
RNNs process sequences by maintaining hidden states across time steps. In training, they perform two main phases:

- Forward Phase
- Backward Phase



Step 1

Forward Pass



Step 2

Backward Pass

=

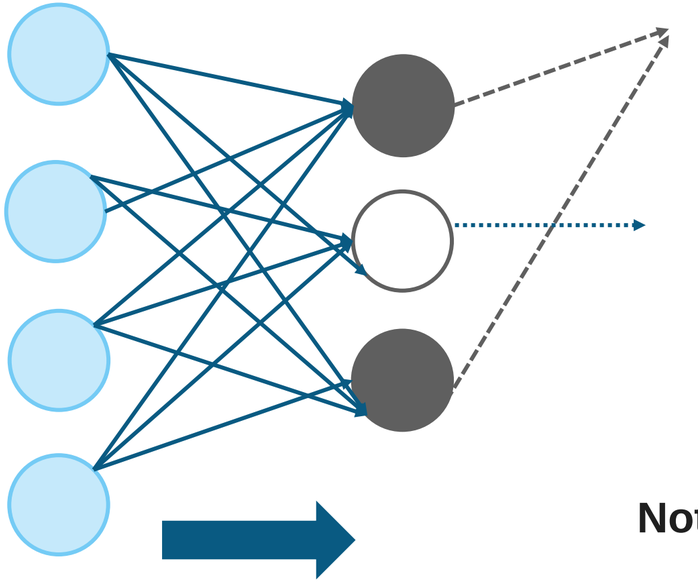
Quality assessment
(error calculation)

Step 3

Quality assessment
(Error calculation)

Forward Pass

- The first step is the forward pass
- Every neuron is added with an individual weight (stochastic way) and an overall bias
- Result will go to hidden layers through an activation function
- Activation of the neurons depend upon the result of the activation function



Active neurons

Non-active neuron

(w1……wn) + b

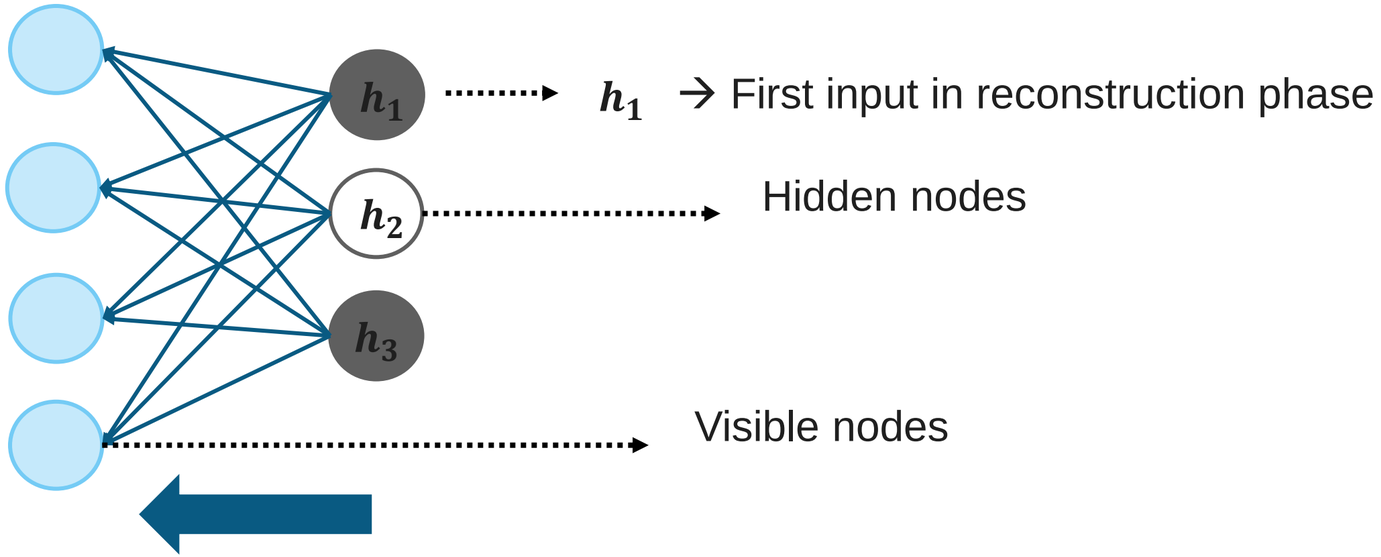
W → weight

b→ bias

Note: Stochastic → "randomly determined"

Backward Pass

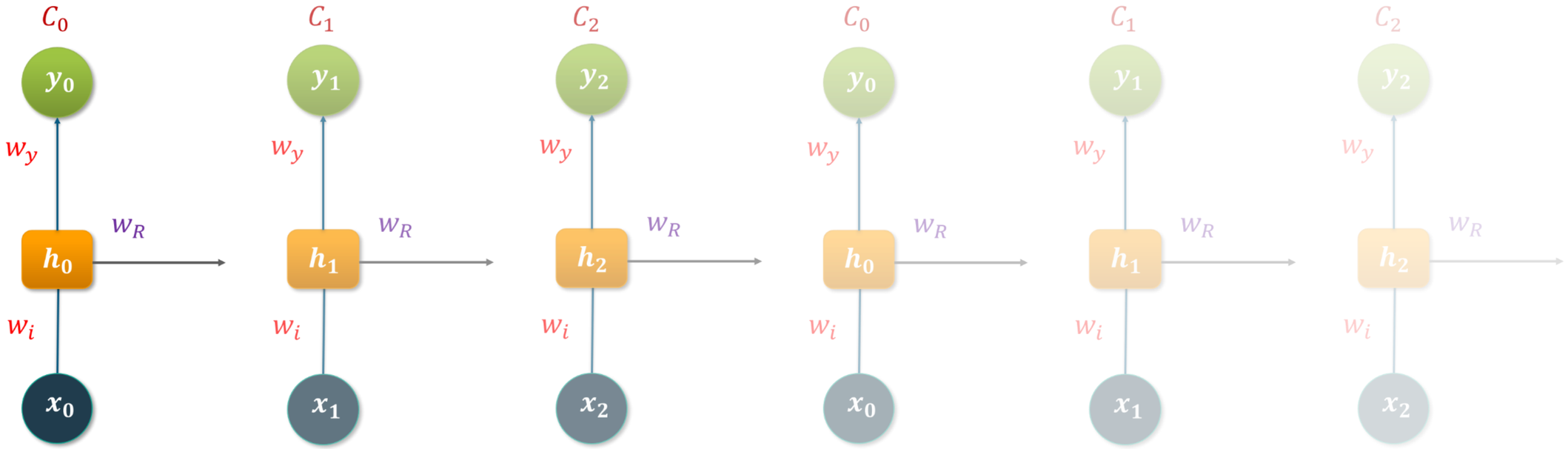
- Reconstruction phase
- First, the hidden neuron becomes the first input
- These hidden layers are multiplied with the same weights
- Sum of those products are added with bias at each visible node
- Output will be the reconstruction of the actual input image



Vanishing Gradient Problem

Vanishing Gradient Problem

The vanishing gradient problem occurs when gradients become extremely small as they propagate backward through the network during training.



As the gradient flows backward from the output layer to the input layer, it gets repeatedly multiplied by small values, causing it to diminish exponentially.

Teacher Forcing

Teacher Forcing

Teacher forcing in RNNs is a training technique used in sequence prediction tasks, especially in recurrent neural networks (RNNs). It helps the model by providing the correct output instead of using the model's predictions.

Without Teacher Forcing (Standard RNN)

- At each time step, the RNN feeds its own predicted output as input to the next step.
- This can lead to error accumulation in early training.

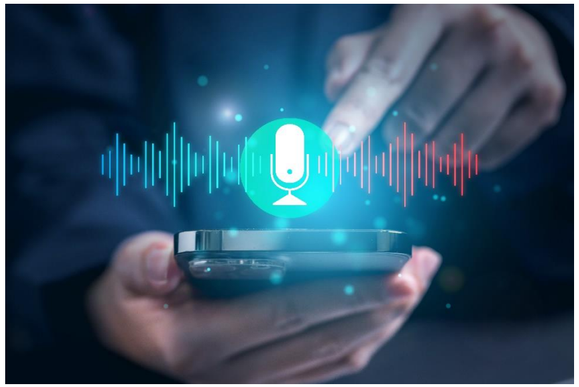
With Teacher Forcing

- At each time step, the actual target value (from training data) is used as the next input.
- Helps the model stay on track during early learning.

Applications of RNN

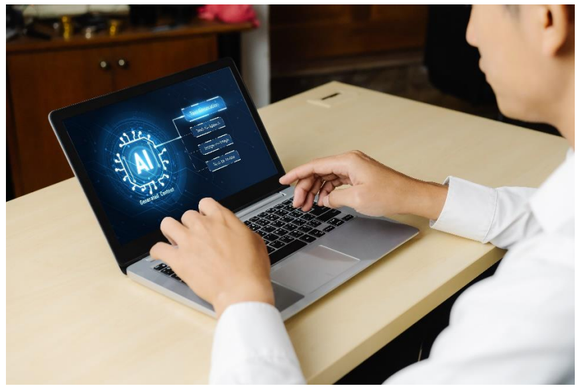
Applications of RNN

Sentiment Analysis

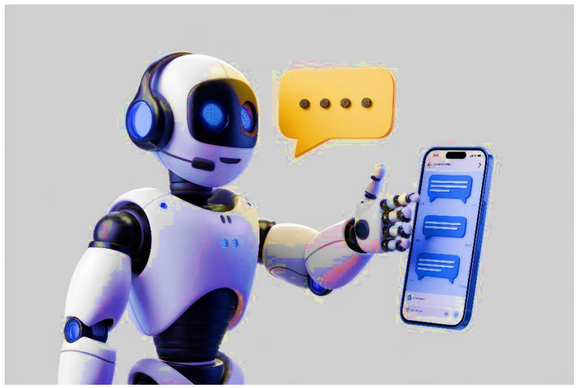


Speech Recognition

Time Series Forecasting



Text Generation

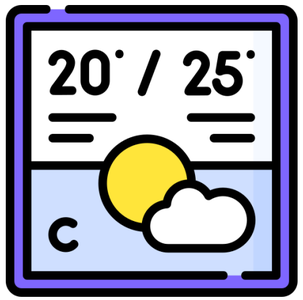


Chatbots & Dialogue Systems

Time Series in RNNs

What is Time Series Data?

Time series data consists of points recorded at specific time intervals, where each observation is influenced by prior ones, unlike regular datasets, where data points are independent.



Daily Temperature Readings

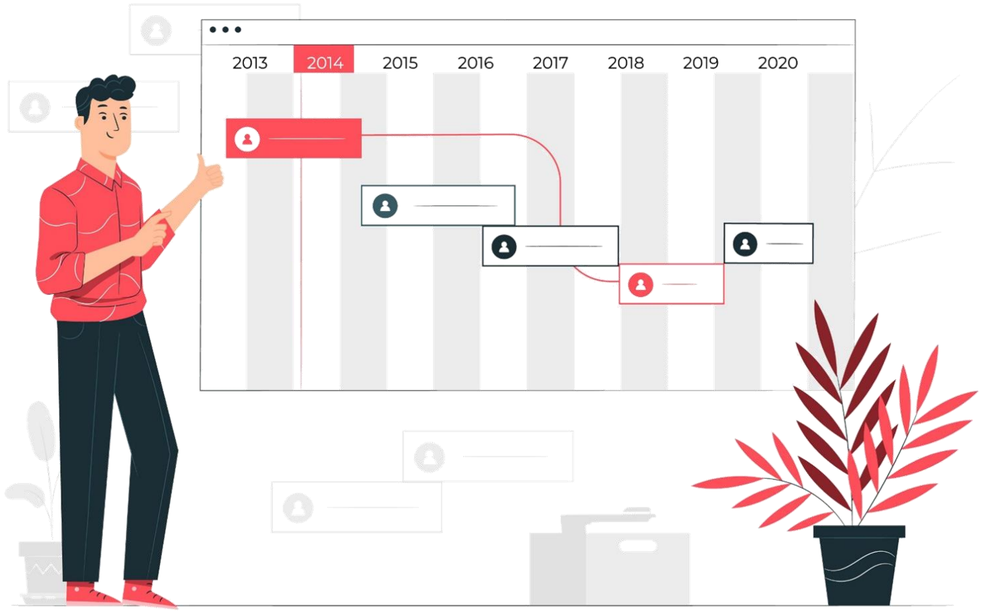
Hourly Stock Market Prices

Monthly Sales Revenue

Weekly Website Traffic

Sequence and Time Series in RNNs

- Sequence data is any ordered data where position matters (e.g., text, audio, DNA).
- Time series is a type of sequence data indexed by time (e.g., stock prices, weather).
- RNNs are built to handle such data by maintaining memory of past inputs using hidden states.
- RNNs process one element at a time, making them ideal for predicting next steps in sequences or time series.
- In both sequence and time series tasks, RNNs capture temporal dependencies, whether the data is about time or just order.



Time Series vs. Regular Data

Feature	Time Series Data	Tabular Data
Ordering	Essential	Not important
Index	Timestamp	Row ID / No order
Dependency	Temporal (lagged)	Usually independent
Use Case	Forecasting	Classification/Clustering

What is Time Series Analysis?

TSA refers to analyzing the time-dependent data to make further predictions.

Example: Bitcoin Price Prediction

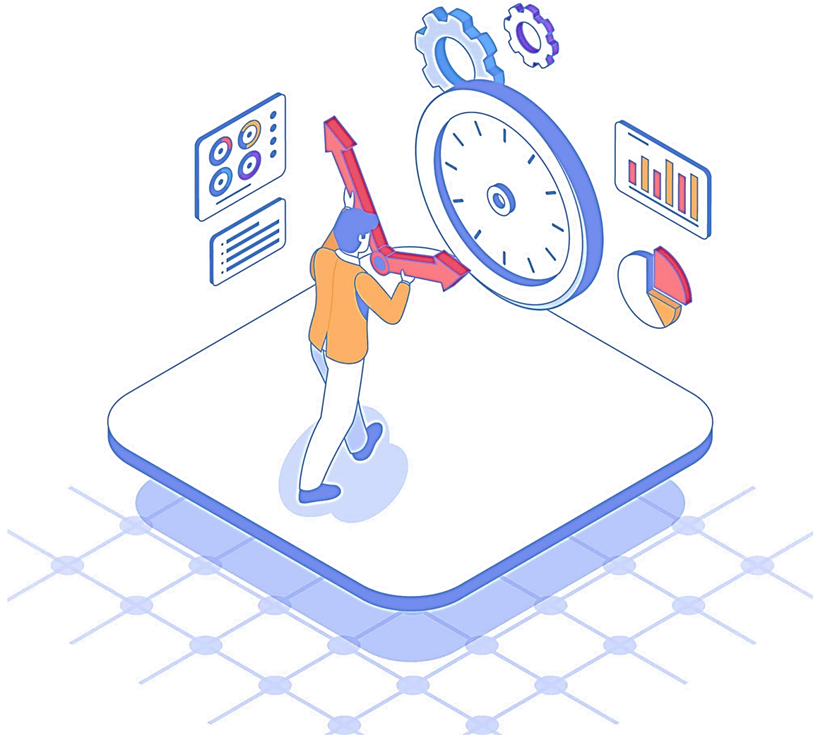
Time Series Forecasting with RNNs

Time Series Forecasting with RNNs

A time series is a sequence of data points indexed in time order, usually taken at successive, equally spaced points in time. Examples: stock prices, weather data, electricity usage, etc.

Challenges in time series:

- Data is sequential.
- Past values influence future values (temporal dependency).
- Data can be noisy or irregular.



Implementation

```
[1]  import numpy as np
     import pandas as pd
     from sklearn.preprocessing import MinMaxScaler
     from keras.models import Sequential
     from keras.layers import Dense, SimpleRNN
```

```
# Load dataset
data = pd.read_csv('https://raw.githubusercontent.com/datasets/finance-vix/main/data/vix-daily.csv')
prices = data['CLOSE'].values

# Normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices.reshape(-1, 1))

# Prepare sequences
sequence_length = 50
X, y = [], []
for i in range(len(prices_scaled) - sequence_length):
    X.append(prices_scaled[i:i+sequence_length])
    y.append(prices_scaled[i+sequence_length])
X, y = np.array(X), np.array(y)
```

Implementation (Contd.)

```
model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(sequence_length, 1)),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=20, batch_size=32)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarn
  super().__init__(**kwargs)
Epoch 1/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.0030
Epoch 2/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - loss: 4.8673e-04
Epoch 3/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 5.2373e-04
Epoch 4/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 5.7234e-04
Epoch 5/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 5.0924e-04
Epoch 6/20
278/278 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 5.1765e-04
```

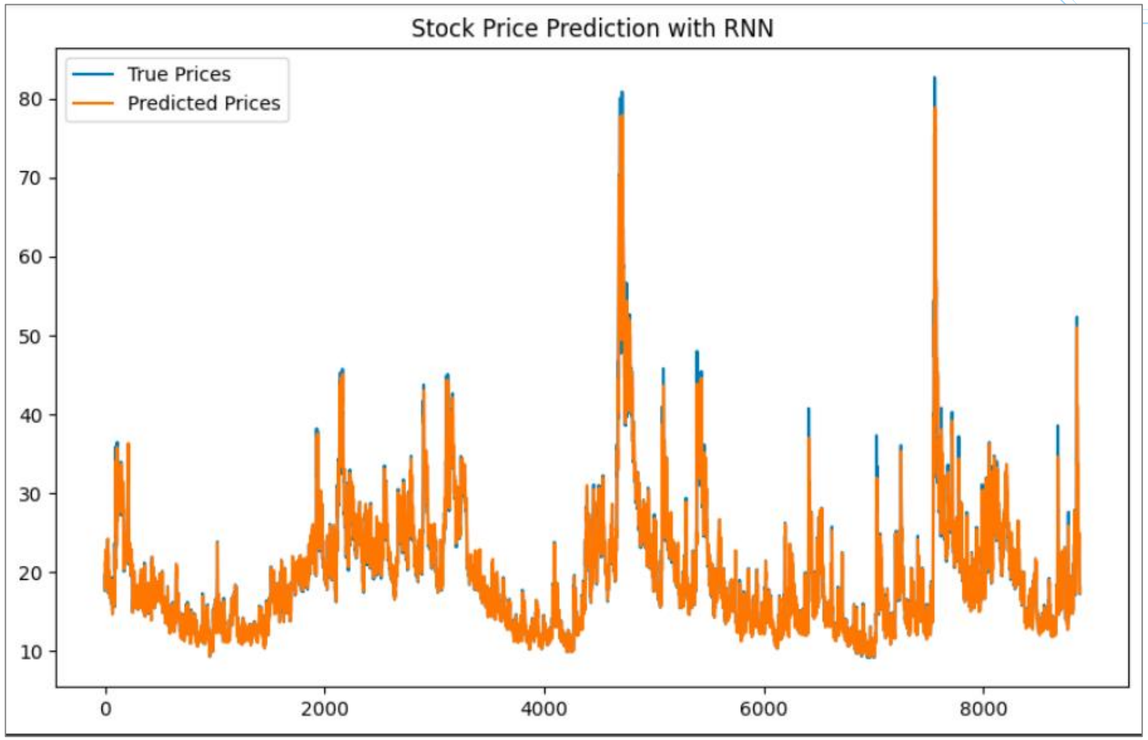
```
[9] # Predict and rescale
    predictions = model.predict(X)
    predictions_rescaled = scaler.inverse_transform(predictions)
```

```
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```

Implementation (Contd.)

```
[10] import matplotlib.pyplot as plt

     plt.figure(figsize=(10, 6))
     plt.plot(prices[sequence_length:], label='True Prices')
     plt.plot(predictions_rescaled, label='Predicted Prices')
     plt.legend()
     plt.title("Stock Price Prediction with RNN")
     plt.show()
```



Tokenization and Embedding Layers

Tokenization in RNNs

Tokenization is the process of converting raw text into a sequence of tokens (usually words or subwords), and then mapping them to integers that the model can understand.

```
[ ]  from tensorflow.keras.preprocessing.text import Tokenizer

     texts = ["I love RNNs", "They process sequences"]
     tokenizer = Tokenizer()
     tokenizer.fit_on_texts(texts)

     sequences = tokenizer.texts_to_sequences(texts)
     print(sequences)
```

```
[[1, 2, 3], [4, 5, 6]]
```

Embeddings Layer in RNNs

An Embedding layer maps these integer tokens to dense vectors of fixed size, capturing semantic meaning. Instead of treating "RNN" as just a number (like 3), it’s mapped to a trainable vector like [0.2, -0.1, ..., 0.5].

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Simulated token sequences
sequences = [[1, 2, 3], [4, 5]]
padded = pad_sequences(sequences, maxlen=5)

# Build RNN model with embedding
model = Sequential([
    Embedding(input_dim=10, output_dim=8, input_length=5),  # vocab size=10, embedding dim=8
    SimpleRNN(16),
    Dense(1, activation='sigmoid')
])

model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning:

Argument `input_length` is deprecated. Just remove it.

Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
simple_rnn (SimpleRNN)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

Building RNNs using Keras

Visualizing RNN Behavior on Sequential Data

Visualizing RNN Behavior on Sequential Data (IMDB Sentiment Analysis)

We build an RNN to classify IMDB reviews and visualize how its hidden states change over time, revealing how the model processes sequences.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, SimpleRNN, Dense

# Load dataset (only top 5000 words)
vocab_size = 5000
maxlen = 100
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Visualizing RNN Behavior on Sequential Data

```
[8]  input_layer = Input(shape=(maxlen,))
     embed = Embedding(input_dim=vocab_size, output_dim=32)(input_layer)
     rnn_layer = SimpleRNN(units=16, return_sequences=True, name="rnn_layer")(embed)
     output_layer = Dense(1, activation='sigmoid')(rnn_layer[:, -1, :])  # Use last timestep output

     model = Model(inputs=input_layer, outputs=output_layer)
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
     model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 100)	0
embedding (Embedding)	(None, 100, 32)	160,000
rnn_layer (SimpleRNN)	(None, 100, 16)	784
get_item (GetItem)	(None, 16)	0
dense (Dense)	(None, 1)	17

Total params: 160,801 (628.13 KB)
Trainable params: 160,801 (628.13 KB)
Non-trainable params: 0 (0.00 B)

```
model.fit(x_train, y_train, epochs=1, batch_size=64, validation_split=0.2)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 58ms/step - accuracy: 0.5532 - loss: 0.6797 - val_accuracy: 0.7588 - val_loss: 0.5111
<keras.src.callbacks.history.History at 0x7edc831d5e10>
```

Visualizing RNN Behavior on Sequential Data

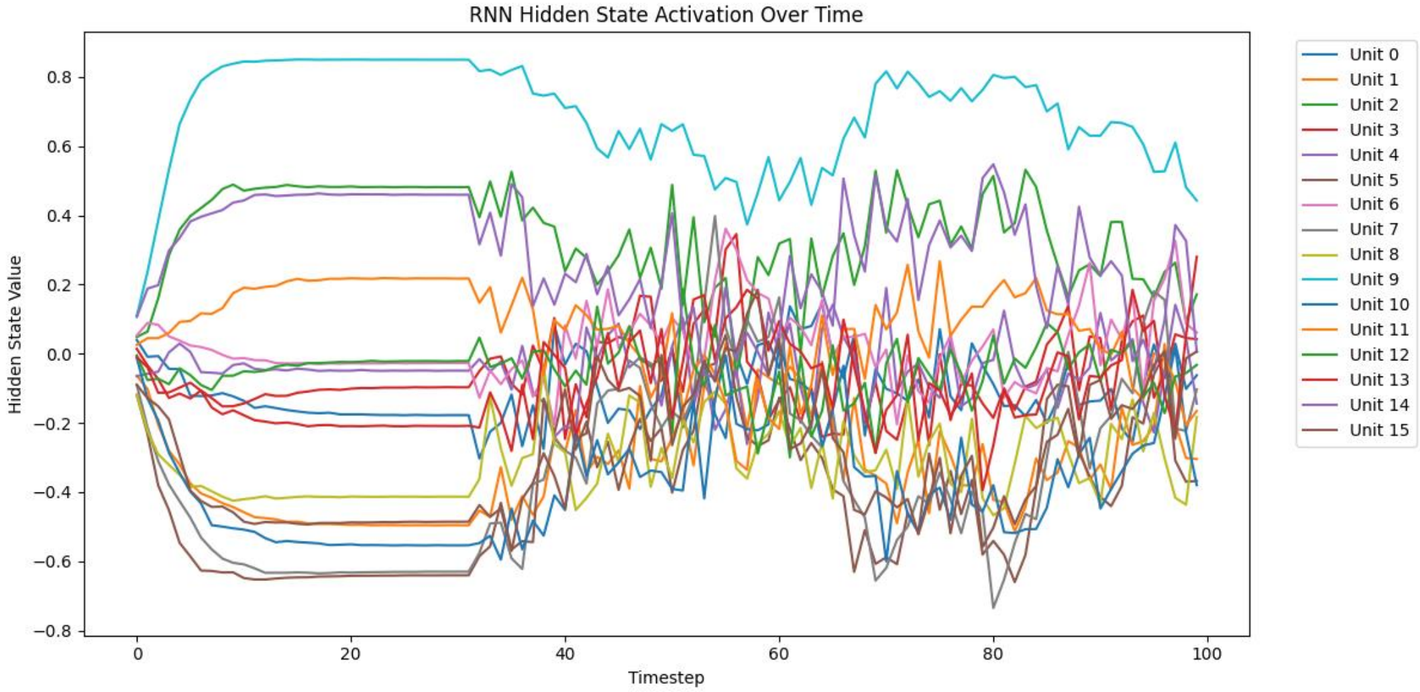
```
[10] # Define a new model to get RNN outputs
     rnn_output_model = Model(inputs=model.input, outputs=model.get_layer("rnn_layer").output)

     # Predict RNN hidden states for a sample
     sample_idx = 0
     sample_input = x_test[sample_idx:sample_idx+1]
     rnn_outputs = rnn_output_model.predict(sample_input)[0]  # shape: (sequence_len, hidden_dim)

     1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
```

```
plt.figure(figsize=(12, 6))
for i in range(rnn_outputs.shape[1]):  # Loop over hidden dimensions
    plt.plot(rnn_outputs[:, i], label=f'Unit {i}')

plt.title("RNN Hidden State Activation Over Time")
plt.xlabel("Timestep")
plt.ylabel("Hidden State Value")
plt.legend(loc='upper right', bbox_to_anchor=(1.15, 1))
plt.tight_layout()
plt.show()
```



Padding and Masking

Padding

Padding is the process of adding zeros to sequences to make them the same length, which is required for efficient batch processing in RNNs.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

sequences = [[1, 5, 7], [3, 8]]
padded = pad_sequences(sequences, maxlen=3)
print(padded)
```

```
[[1 5 7]
 [0 3 8]]
```

RNNs process input in batches, and all sequences in a batch must be the same length. Padding ensures that shorter sequences align with longer ones.

Masking

Masking tells the RNN to ignore padded values (usually 0s) when processing the sequence. This prevents padded zeros from affecting learning.

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

model = Sequential([
    Embedding(input_dim=10, output_dim=4, input_length=3, mask_zero=True),
    SimpleRNN(8),
    Dense(1, activation='sigmoid')
])

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
simple_rnn_1 (SimpleRNN)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

Stateful vs. Stateless RNNs

Stateful vs. Stateless RNNs

Feature	Stateless RNN	Stateful RNN
Hidden State Between Batches	Reset after every batch	Maintained across batches
Sequence Dependency	Handles **independent** sequences	Handles **dependent/long** sequences
Batch Size	Can vary between epochs or batches	Must remain **constant** during training
Memory of Previous Batch	No memory of previous inputs	Retains memory from previous batch
Reset Requirement	Automatically resets state	Must manually call model.reset_states()
Use Cases	Sentiment analysis, document classification	Time series forecasting, language modeling over long texts
Implementation Simplicity	Easier to train and manage	More complex to train and tune

Using GRUs vs. Vanilla RNNs

Using GRUs vs. Vanilla RNNs

Feature	Vanilla RNN	GRU (Gated Recurrent Unit)
Architecture	Single tanh layer	Uses update and reset gates
Gating Mechanism	No gates	Yes, controls flow of information
Long-Term Dependency Handling	Poor, due to vanishing gradient	Better at capturing long-term dependencies
Training Stability	Less stable	More stable and faster convergence
Complexity	Simpler, fewer parameters	More complex, more parameters
Computational Cost	Lower	Slightly higher
Memory Control	Cannot selectively keep or discard past info	Can control memory via gates

Sentiment Analysis on Movie Reviews (Demonstration)

Note: Refer to the Module 5: Demo 1 on LMS for detailed steps.

Sequence Model for Character-Level Text Generation (Demonstration)

Note: Refer to the Module 5: Demo 2 on LMS for detailed steps.

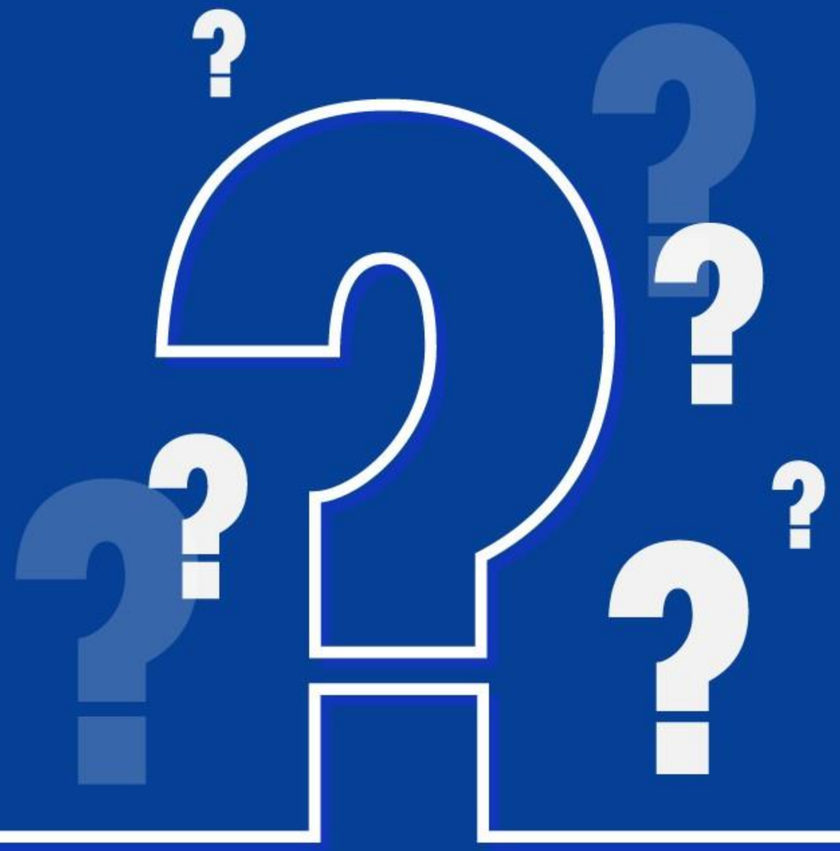
Summary

In this lesson, you have learned to:

- Understand basics of RNNs and its input/output formats and the forward/backward pass in RNNs.
- Explain vanishing gradient problem and apply teacher forcing, padding, and masking.
- Compare vanilla RNNs, GRUs, and stateful vs stateless models.

edureka! a Veranda ENTERPRISE

Questions



edureka! a Veranda ENTERPRISE

Feedback

edureka! a Veranda ENTERPRISE

Thank You

For information, Please Visit our Website
www.edureka.co