

SPEECH RECOGNITION

NAME -
SUDHANSU KAKKAR
STUDENT ID -
20036779

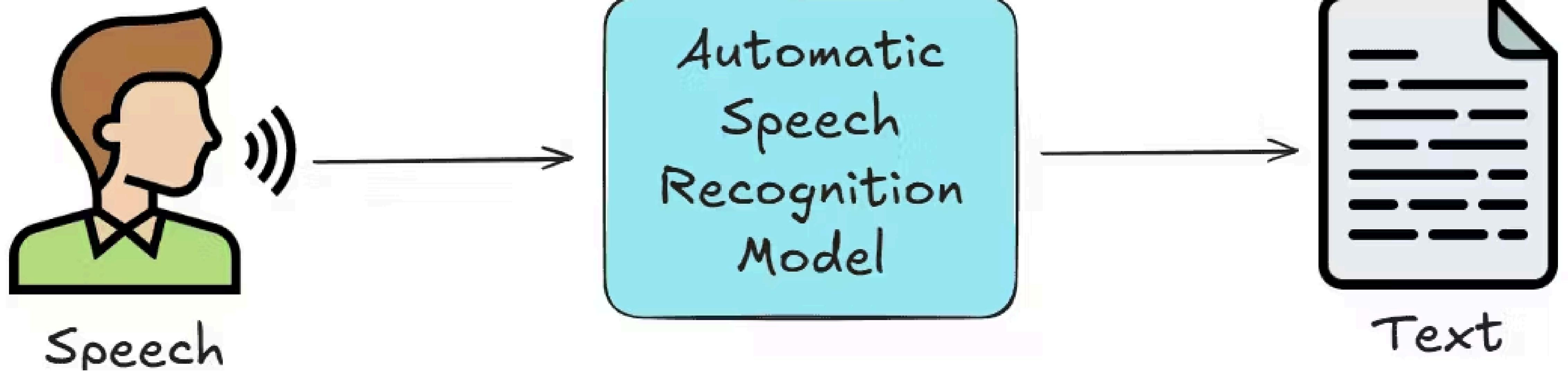
ENABLING MACHINES TO UNDERSTAND AND PROCESS HUMAN SPEECH

INTRODUCTION

Speech recognition is the process of converting spoken language into text using AI-powered systems.

APPLICATIONS

- Virtual assistants (Siri, Alexa, Google Assistant)
- Healthcare (medical dictation, voice-driven diagnostics)
- Customer service (call center automation, chatbots)
- Accessibility (voice control for differently-abled users)
- Transcription (meetings, lectures, media production)



MACHINE LEARNING (ML):

Role: Learns from speech datasets to improve recognition of accents, dialects, and context.

Example: Logistic regression or SVM models used for phoneme classification

DEEP LEARNING (DL)

Role: Neural networks (e.g., RNNs, Transformers) model sequences of speech for high accuracy.

Example: End-to-end systems like DeepSpeech achieving near-human transcription performance.

ARTIFICIAL INTELLIGENCE (AI):

Role: Provides the foundation for speech-to-text pipelines by combining language understanding and signal processing.

Example: AI systems that filter background noise to improve recognition accuracy.

CHATGPT

Role: Processes recognized text into meaningful dialogue, generating natural responses.

Example: Integrated with voice assistants for conversational AI.

LARGE LANGUAGE MODELS (LLMs)

Role: Add contextual understanding to speech recognition, correcting errors and predicting intent.

Example: GPT-4 used to enhance accuracy in domain-specific voice transcription.



MATLAB = MATH + CODE + VISUALIZATION

MATLAB

*Powerful, tool for engineers & scientists to analyze,
simulate, and visualize real-world problems.*

AVAILABLE: WINDOWS, MACOS, LINUX
LANGUAGE: C++, C, JAVA, MATLAB

KEY FEATURES

- Matrix-based computing
- 2D & 3D visualization
- Toolboxes for AI, Image Processing, Control Systems, Finance
- Easy integration with Python, C/C++, hardware

APPLICATIONS

- Data Analysis & Visualization 
- Signal & Image Processing 
- Machine Learning & AI 
- Robotics & Control 
- Financial Modeling 

HW-Topic-3

Data Acquisition, Modeling and Analysis: Big Data Analytics

**Submitted By – Sudhanshu Kakkar
CWID – 20036779**

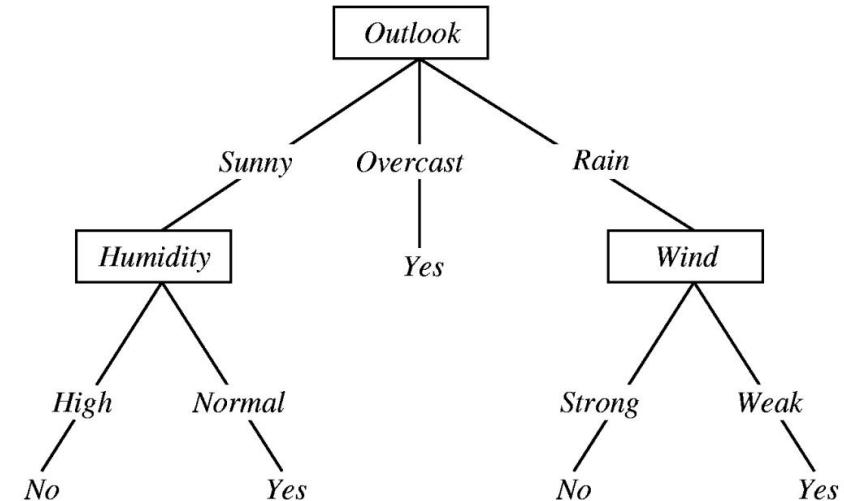
DLT – Decision Tree Learning

WHAT IS IT?

- A **supervised machine learning algorithm** used for **classification and regression**.
- Represents decisions as a **tree structure**.

KEY CONCEPTS

- **Splitting:** Dividing dataset based on feature values.
- **Entropy:** Measure of randomness or impurity.
- **Information Gain:** Reduction in entropy after a split.
- **Pruning:** Removing branches to reduce overfitting.



TYPES

- **Classification Tree:** Predicts discrete labels (e.g., Yes/No)
- **Regression Tree:** Predicts continuous values

Pros	Cons
<ul style="list-style-type: none">• Easy to understand & visualize• Handles categorical & numerical data	<ul style="list-style-type: none">• Prone to overfitting• Sensitive to small changes in data

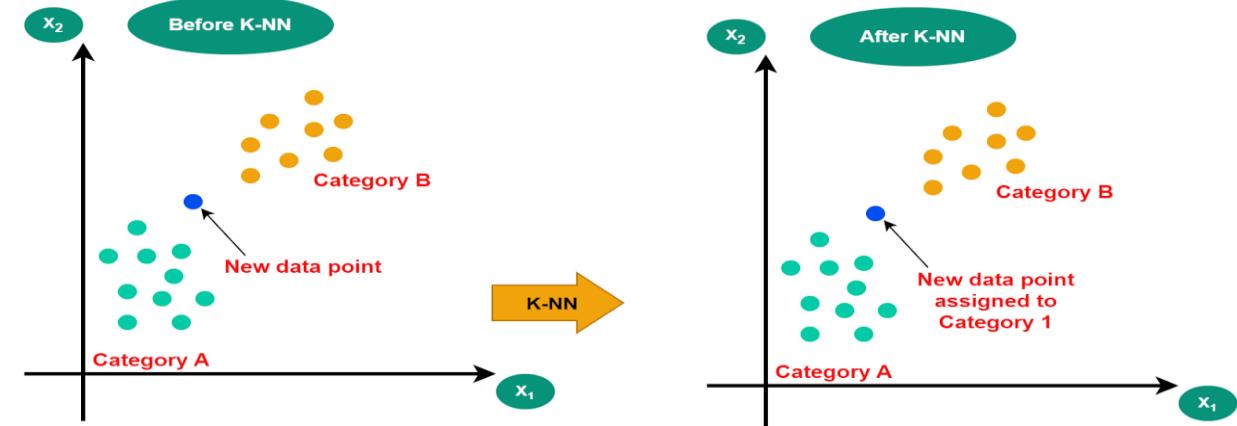
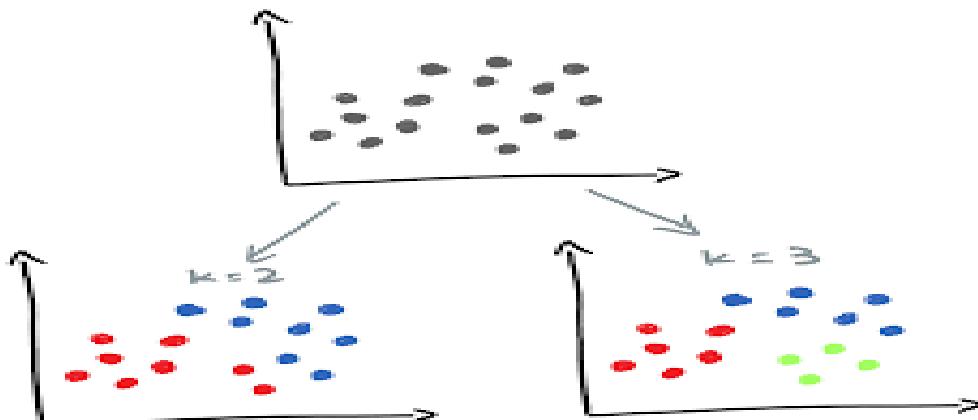
KNN – K-Nearest Neighbours

WHAT IS IT?

- A **supervised machine learning algorithm** used for **classification and regression**.
- Classifies a data point based on the **majority class of its K nearest neighbors**

KEY CONCEPTS

- **K Value:** Number of neighbors considered
- **Distance Metrics:** How similarity is measured
 - **Euclidean Distance**
 - **Manhattan Distance**



Pros	Cons
<ul style="list-style-type: none">• Simple and easy to understand• Works well with small datasets	<ul style="list-style-type: none">• Computationally expensive for large datasets• Poor performance on high-dimensional data

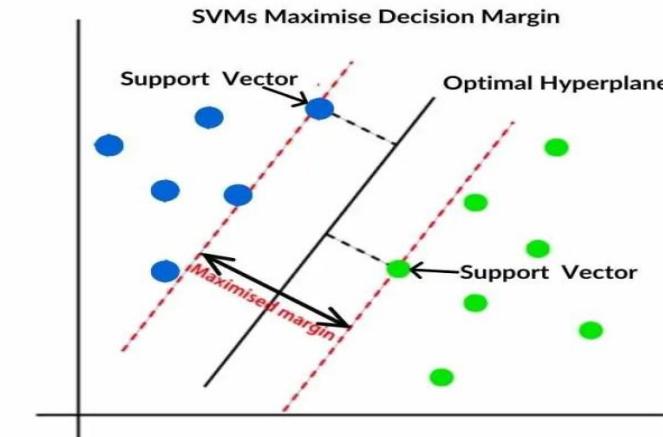
SVM – Support Vector Machine

WHAT IS IT?

- A **supervised machine learning algorithm** used for **classification and regression**.
- Finds the **best hyperplane** that separates classes with the **maximum margin**.
- Works well for **linear and non-linear** data using kernel functions.

KEY CONCEPTS

- **Hyperplane:** Decision boundary separating classes
- **Support Vectors:** Data points closest to the hyperplane
- **Margin:** Distance between support vectors of different classes (maximizing this improves generalization)



TYPES

- **Linear SVM:** Data is linearly separable
- **Non-linear SVM:** Uses kernels for complex datasets

Pros	Cons
<ul style="list-style-type: none">• Effective in high-dimensional spaces• Works well for linearly and non-linearly separable data	<ul style="list-style-type: none">• Computationally expensive for very large datasets• Choosing the right kernel can be tricky

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# declaring points for training
X = np.array([
    [1,2], [2,3], [3,1], [2,1.5], [3,2.5],
    [1.5,1], [2.5,2.2], [3,3.2], [2,0.5], [1,1.5],
    [6,5], [7,7], [8,6], [7,5], [6.5,6.5],
    [8,7], [7.2,6.2], [6.8,5.5], [7.5,6.5], [6,6.2]
])
# declaring labels for points
y = np.array([0]*10 + [1]*10) # 0 = Class A, 1 = Class B

# Train SVM
model = svm.SVC(kernel='linear')
model.fit(X, y)

# New points to classify
new_points = np.array([
    [2,2], [3,3], [6,6], [7,5.5], [1.2,1.5]
])
predictions = model.predict(new_points)

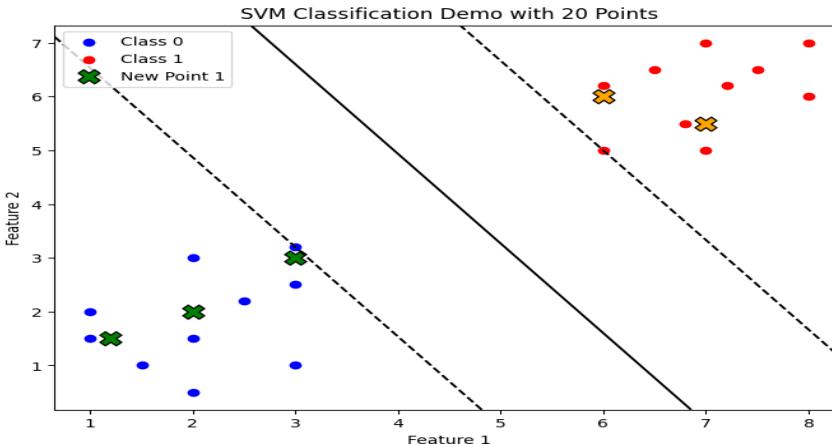
# Print classifications
for point, label in zip(new_points, predictions):
    print(f"Point {point} is classified as Class {label}")

# Plot
plt.figure(figsize=(8,6))

# Plot original points
plt.scatter(X[:10,0], X[:10,1], color='blue', label='Class 0')
plt.scatter(X[10:,0], X[10:,1], color='red', label='Class 1')


```

SVM Code Demo (python)



```

# Plot new points with classification
for i, point in enumerate(new_points):
    plt.scatter(point[0], point[1], color='green' if predictions[i]==0 else 'orange',
               edgecolors='k', s=150, marker='X', label=f'New Point {i+1}' if i==0 else
               "")

# Plot decision boundary
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx = np.linspace(xlim[0], xlim[1], 50)
yy = np.linspace(ylim[0], ylim[1], 50)
YY, XX = np.meshgrid(yy, xx)
Z = model.decision_function(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
ax.contour(XX, YY, Z, levels=[0], colors='k') # hyperplane
ax.contour(XX, YY, Z, levels=[-1,1], colors='k', linestyles='--') # margins

plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM Classification Demo with 20 Points")
plt.legend()
plt.show()


```

HW-Topic-4

Data Acquisition, Modeling and Analysis: Big Data Analytics

**Submitted By – Sudhanshu Kakkar
CWID – 20036779**

Deep Learning

WHAT IS IT?

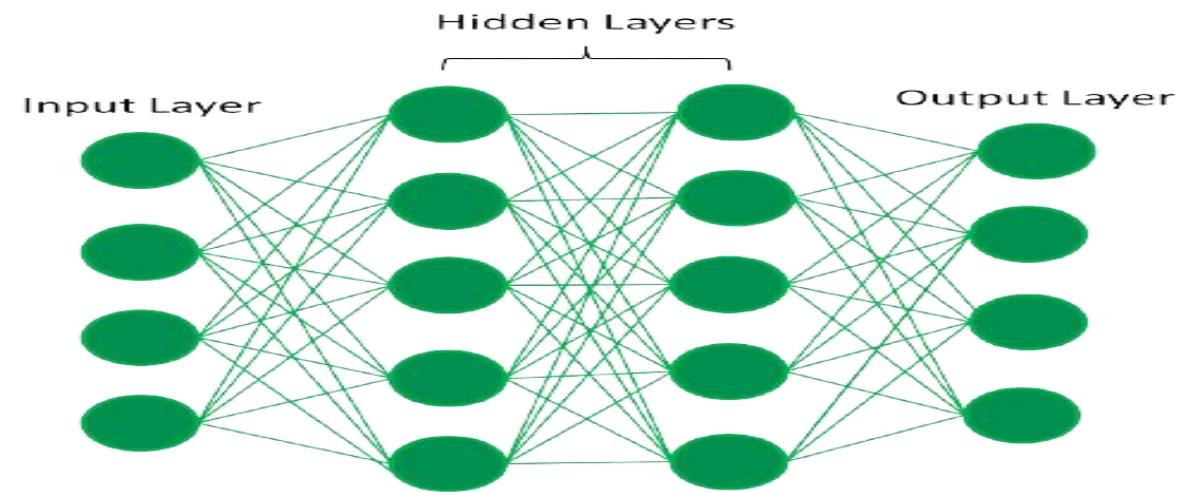
Deep learning is a subfield of **machine learning** that uses **artificial neural networks** with multiple layers (hence "deep") to analyze data, learn patterns, and make decisions or predictions.

KEY CONCEPTS

- **Neural Networks:** Artificial neural networks with multiple hidden layers (deep) form the computational core, inspired by the human brain.
- **Feature Extraction:** Deep learning automatically learns the most important features directly from raw data, eliminating manual feature engineering.

APPLICATIONS

- Self Driving Cars
- Natural Language Processing
- Movie Recommendation Systems
- Natural Language Processing
- Computer Vision
- Financial Fraud Detection
- Healthcare



THE PROCESS

1. **Forward Propagation:** The input data is processed through the network's layers, where initial weights are applied to make the model's first prediction (the "guess").
2. **Measure Loss:** A mathematical function calculates the difference (the "loss") between the model's prediction and the actual correct answer, measuring how wrong the guess was.
3. **Backpropagation:** The error signal is sent backward through the network layers, revealing how much each internal connection (weight) contributed to the total error.
4. **Weight Adjustment:** Using the error feedback, an optimizer (Gradient Descent) makes small, iterative adjustments to the weights and biases to ensure the next guess is more accurate.

Convolution Neural Network

WHAT IS IT?

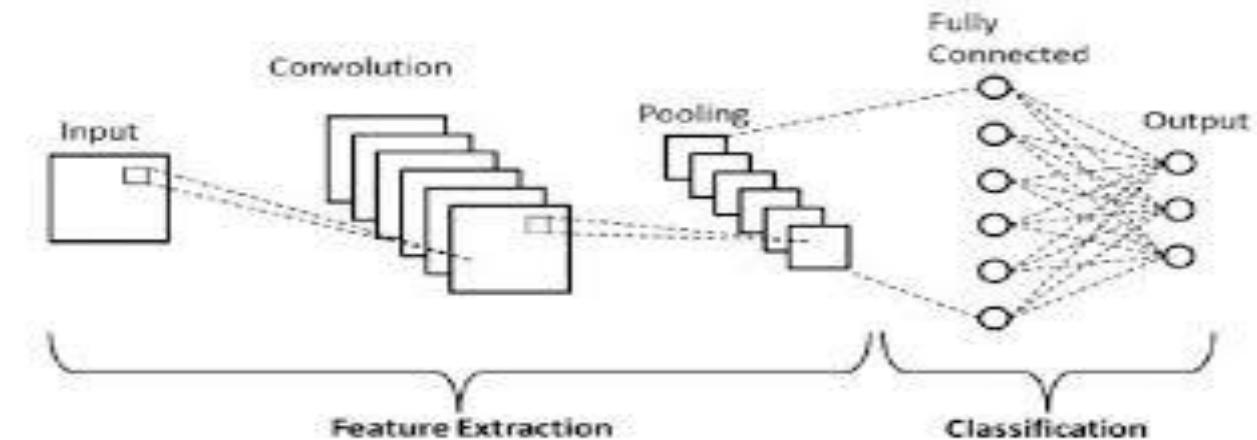
A **Convolutional Neural Network (CNN)** is a type of deep learning network specifically designed to process data with a known grid-like structure, such as **images**. Its process is specialized in automatically extracting visual features.

KEY CONCEPTS

- **Convolution:** Filter slides over input to extract features (edges, textures).
- **Filter / Kernel:** A small learnable weight matrix that detects one specific feature
- **Parameter Sharing:** A single filter's weights are reused across the entire image.
- **Pooling (Max):** Down samples feature map size, adding robustness to shifts.
- **Flatten:** Converts 3D features into a 1D vector for final classification

APPLICATIONS

- Facial Recognition
- Autonomous Vehicles
- Image Classification
- OCR / Handwriting recognition
- Product recommendation engine



THE PROCESS

1. **Convolution:** Extracts features (edges, textures, patterns) by applying a filter (kernel) across the input.
2. **Activation (ReLU):** Introduces non-linearity to the extracted features, enabling the model to learn complex data maps.
3. **Pooling:** Reduces the spatial dimensions (width and height) of the feature data.
4. **Flattening:** Transforms the 2D/3D feature maps into a single 1D vector.
5. **Fully Connected:** Learns high-level global patterns from the flattened feature vector.
6. **Output (Softmax):** Calculates the final probability for each potential class label.

Recurrent Neural Network

WHAT IS IT?

A **Recurrent Neural Network (RNN)** is a type of deep learning model specifically designed to process **sequential data** or data where the order matters. Unlike a traditional neural network, an RNN has an internal memory that allows it to remember information from previous steps in the sequence.

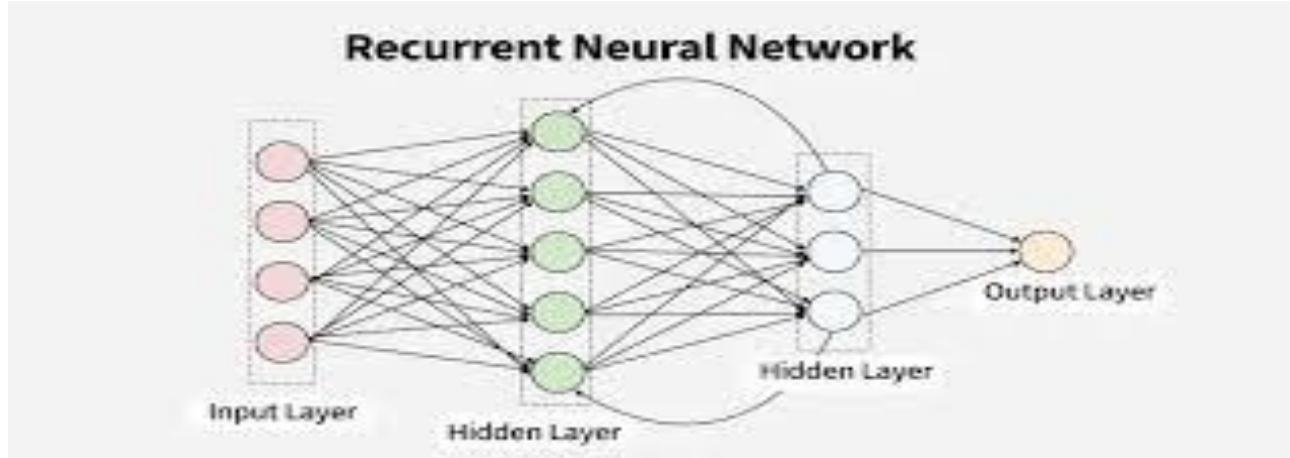
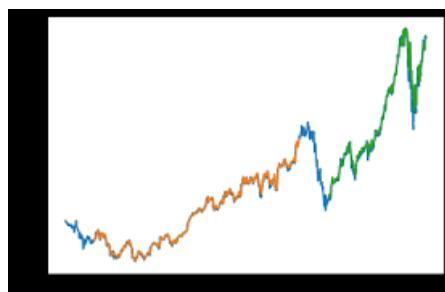
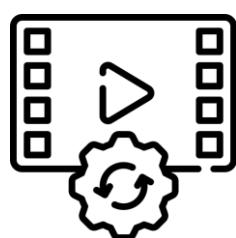
KEY CONCEPTS

- **Contextual Understanding:** Uses internal memory to process current data based on its entire history.
- **Variable Length Input:** Naturally handles sequences of any length (e.g., short or long sentences).
- **Sequence Mapping:** Great at mapping one sequence to another (like translation) or to a single label.
- **Long-Term Memory (LSTM):** Advanced versions use gates to remember critical info over extended periods.

APPLICATIONS



- Sentiment Analysis
- Video Processing
- Stock Prices Prediction
- Text Prediction



THE PROCESS

1. **Input Embedding:** Converts the sequential element (like a word) into a **dense numerical vector** for the network to understand.
2. **Initial Hidden State (Old Memory):** Sets the starting **internal memory** (context) to an empty state.
3. **Recurrence (Calculation):** Takes the **old memory** and the **current input** to compute the **new, updated memory**.
4. **Weight Sharing:** Reuses the same set of weights (rules) for every single time step in the sequence.
5. **Final Hidden State (Full Context):** The memory vector that holds the **complete understanding** of the entire sequence.
6. **Output (Prediction):** Uses the memory (from a specific step or the final one) to make a final **prediction** (e.g., translation, sentiment).

CNN Code Demo (python)

Training and Model Building Step

```
# initializing the image generators each for training and validation set

#using 20% of train images for validation and rest 80% for training

train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training',
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation',
    shuffle=False
)
```

In the above code snippet, we are dividing the data set into 20% validation and rest 80% for training.

```
# building the model here
#as per the approach 3 I am using 3 convolution layers and respective maxpooling layers

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)), #1st conv layer
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'), #2nd conv layer
    MaxPooling2D(2,2),
    Conv2D(128, (3,3), activation='relu'), #3rd conv layer
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])

model.compile(
    optimizer=Adam(learning_rate=1e-5, clipnorm=1.0),
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

In the above code snippet, we are building the model with 3 convolution layers.

Testing Step

```
1  from tensorflow.keras.models import load_model
2  import numpy as np
3  from tensorflow.keras.preprocessing import image
4
5  #loading the model for image label testing
6  model = load_model('./cat_dog_model.h5')
7
8  def predict_image(model, img_path):
9      # processing the image to make it ready to feed into the model for prediction
10     img = image.load_img(img_path, target_size=(150, 150))
11     img_array = image.img_to_array(img)
12     img_array = img_array / 255.0 # normalize
13     img_array = np.expand_dims(img_array, axis=0) # add batch dimension
14
15     prediction = model.predict(img_array)[0][0] # get prediction value
16
17     if prediction > 0.5:
18         print(f"Prediction: dog ({prediction:.2f})")
19     else:
20         print(f"Prediction: Cat ({1 - prediction:.2f})")
21
22 imgPath = "./cat.14.jpg" # input image 1
23 predict_image(model, imgPath);
24
25 imgPath = "./dog.5.jpg" # image image 2
26 predict_image(model, imgPath);
```

Input – Dog image



Output –

Prediction: dog (0.9101)

In the above code snippet, we are testing the trained model with images of cat and dog.