

1. What is deep learning? Explain CNN, RNN and their applications.

Deep learning is a subfield of machine learning that focuses on training artificial neural networks to learn and make predictions or decisions. It is inspired by the structure and functioning of the human brain. Deep learning algorithms are capable of automatically learning hierarchical representations of data by utilizing multiple layers of interconnected artificial neurons.

Convolutional Neural Networks (CNNs) are a type of deep learning model that are particularly effective in processing and analyzing grid-like data, such as images or audio spectrograms. CNNs consist of several layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply filters to the input data, which helps in extracting meaningful features. Pooling layers reduce the spatial dimensions of the data, preserving the important features. Fully connected layers perform the final classification or regression tasks. CNNs have revolutionized computer vision tasks, such as image classification, object detection, and image segmentation.

Recurrent Neural Networks (RNNs) are another type of deep learning model designed to process sequential data. Unlike feedforward neural networks, which process data in a single pass, RNNs maintain an internal memory to process sequences of data with temporal dependencies. RNNs have a recurrent connection that allows information to flow from previous steps to the current step, enabling the network to capture context and make predictions based on the history of the sequence. RNNs are widely used in natural language processing tasks, such as language modeling, machine translation, sentiment analysis, and speech recognition.

Here are some applications of CNNs and RNNs:

1. **Image Classification:** CNNs excel at recognizing and categorizing objects within images. They have been used in various applications like self-driving cars, medical imaging, and facial recognition systems.
2. **Object Detection:** CNNs can accurately locate and classify multiple objects within an image or video. This is crucial in applications like autonomous vehicles, surveillance systems, and video analysis.
3. **Image Segmentation:** CNNs can assign a label to each pixel in an image, enabling precise understanding of the different regions or objects within an image. It is useful in medical imaging, autonomous navigation, and scene understanding.
4. **Natural Language Processing (NLP):** RNNs are widely used in tasks such as language modeling, machine translation, sentiment analysis, chatbots, and text generation. They can model the sequential nature of language and capture dependencies between words.
5. **Speech Recognition:** RNNs, especially a variant called Long Short-Term Memory (LSTM), have been successfully applied to convert spoken language into written text. They are employed in voice assistants, transcription services, and voice-controlled systems.
6. **Time Series Analysis:** RNNs are suitable for analyzing sequential data with temporal dependencies, making them useful for tasks like stock market prediction, weather forecasting, and anomaly detection.

These are just a few examples of the many applications of deep learning, CNNs, and RNNs. Their versatility and ability to learn complex patterns have led to breakthroughs in various domains, driving advancements in artificial intelligence.

2. Write a program and explain how to build a chatbot for a 'movie ticket booking scenario'.

Chatbot for a 'movie ticket booking' scenario:

```
import datetime

def greet():
    print("Hello! Welcome to Movie Ticket Booking.")
    print("How can I assist you today?")

def get_movie_details():
    movie_name = input("Please enter the name of the movie: ")
    show_time = input("Please enter the show time (HH:MM AM/PM): ")
    # You can add more details like theater selection, etc.
    return movie_name, show_time

def get_user_details():
    name = input("Please enter your name: ")
    phone = input("Please enter your phone number: ")
    return name, phone

def book_tickets(movie_name, show_time, name, phone):
    # Perform ticket booking logic here
    # You can integrate with a ticket booking API or implement your own logic
    print("Great! Your tickets for '{}' at {} have been booked.".format(movie_name, show_time))
    print("Ticket details:")
    print("Movie: {}".format(movie_name))
    print("Show time: {}".format(show_time))
    print("Name: {}".format(name))
    print("Phone: {}".format(phone))

def chatbot():
    greet()
    movie_name, show_time = get_movie_details()
    name, phone = get_user_details()
    book_tickets(movie_name, show_time, name, phone)

chatbot()
```

Explanation:

- In this program, `greet()` function displays a welcome message to the user. The `get_movie_details()` function prompts the user to enter the movie name and show time. The `get_user_details()` function collects the user's name and phone number.
- The `book_tickets()` function performs the actual ticket booking logic. You can customize this function to integrate with a ticket booking API or implement your own booking system.
- Finally, the `chatbot()` function brings everything together. It calls the other functions in the appropriate sequence to collect the necessary information and book the tickets.

(a) Explain how you will apply ML/NLP for this task.

Machine learning can be used in several ways to enhance the capabilities of the movie ticket booking chatbot. Here are a few examples:

1. Intent Recognition: Machine learning can be used to train a model that can understand the user's intent based on their input. This can help the chatbot accurately determine the user's request, such as booking tickets, checking showtimes, or getting movie recommendations.
2. Named Entity Recognition: Machine learning algorithms can be used to identify and extract relevant information from user inputs, such as movie names, showtimes, and user details. This can improve the accuracy of extracting specific details required for ticket booking.
3. Natural Language Processing (NLP): NLP techniques can be employed to preprocess and understand the user's text input. Machine learning models can be trained to perform tasks such as text classification, sentiment analysis, and language understanding, which can aid in providing more relevant and personalized responses to the user.
4. Recommendation Systems: Machine learning algorithms can be used to analyze user preferences, historical data, and movie ratings to generate personalized movie recommendations. This can enhance the user experience and help them discover movies they are likely to enjoy.
5. Dialog Management: Reinforcement learning techniques can be utilized to train a chatbot to have more engaging and natural conversations with users. By allowing the chatbot to learn from user interactions and feedback, it can improve its responses and decision-making over time.

It's important to note that implementing machine learning in a chatbot requires training data, model development, and continuous iteration to improve performance. Additionally, integration with relevant APIs and databases may be necessary to retrieve and update movie information, showtimes, and ticket availability.

(b) Can you code a sample chat scenario ? you can create you own chat set.

Sample chat scenario:

```
User: Hello
Chatbot: Hello! Welcome to Movie Ticket Booking. How can I assist you today?
User: Can you recommend a movie to watch?
Chatbot: I recommend watching 'Avengers: Endgame'.
User: Great! I would like to book tickets for that movie.
Chatbot: Please enter the name of the movie: Avengers: Endgame
User: Please enter the show time (HH:MM AM/PM): 7:00 PM
Chatbot: Please enter your name: John Doe
User: Please enter your phone number: 1234567890
Chatbot: Great! Your tickets for 'Avengers: Endgame' at 7:00 PM have been booked.
Ticket details:
Movie: Avengers: Endgame
Show time: 7:00 PM
Name: John Doe
Phone: 1234567890
```

(c) How to measure 'correctness' of your model?

When using machine learning to create a chatbot, there are several metrics and evaluation techniques you can use to measure the correctness and performance of your ML model. Here are some common approaches:

1. Accuracy: Accuracy is a widely used metric to evaluate the correctness of a classification-based ML model. It measures the percentage of correctly predicted intents or user queries. You can compare the predicted intent or response against the ground truth (correct label) and calculate the accuracy.
2. Precision, Recall, and F1 Score: If you have multiple intents or classes in your chatbot, precision, recall, and F1 score can provide a more detailed evaluation. Precision measures the ratio of true positives to the total predicted positives, recall measures the ratio of true positives to the total actual positives, and the F1 score combines both precision and recall. These metrics give insights into how well your model is performing for each specific intent or class.
3. Confusion Matrix: A confusion matrix provides a visual representation of the model's performance across different intents or classes. It shows the number of correct and incorrect predictions for each class, allowing you to identify where the model is making mistakes or confusing similar intents.
4. Cross-Validation: Cross-validation is a technique used to assess the performance of a model on an independent dataset. By dividing your data into multiple subsets (folds), training the model on one subset, and evaluating it on the remaining subset, you can get a more robust evaluation of the model's performance.
5. User Satisfaction: Another crucial aspect of evaluating a chatbot's correctness is user satisfaction. You can collect user feedback, conduct surveys, or measure user engagement

metrics to gauge how well the chatbot is meeting user expectations and providing accurate and helpful responses.

It's worth mentioning that the quality of training data, the diversity and representativeness of user queries, and the specific requirements of your chatbot's domain can significantly impact the model's correctness. Regularly monitoring and evaluating the chatbot's performance is essential for ongoing improvements and iterative model training.

3. Write a program and explain how you can recognize core entities from a Wikipedia URL ?

To recognize core entities from a Wikipedia URL, you can use web scraping techniques along with natural language processing (NLP) to extract relevant information. Here's an example program in Python that demonstrates the process:

```
import requests
from bs4 import BeautifulSoup
import spacy

# Load the English language model from spaCy
nlp = spacy.load('en_core_web_sm')

def extract_core_entities(url):
    # Send a GET request to the Wikipedia URL
    response = requests.get(url)

    # Create a BeautifulSoup object to parse the HTML content
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find the main content element
    content = soup.find(id='mw-content-text')

    # Extract the text from the content element
    text = content.get_text()

    # Process the text with spaCy
    doc = nlp(text)

    # Extract core entities
    core_entities = []
    for entity in doc.ents:
        if entity.label_ in ['PERSON', 'ORG', 'GPE']:
            core_entities.append(entity.text)

    return core_entities

# Example usage
wikipedia_url = 'https://en.wikipedia.org/wiki/OpenAI'
core_entities = extract_core_entities(wikipedia_url)

print("Core Entities:")
```

In this program, we first import the necessary libraries: `requests` for making HTTP requests, `BeautifulSoup` for parsing HTML content, and `spaCy` for natural language processing. Make sure you have these libraries installed before running the program.

The `extract_core_entities` function takes a Wikipedia URL as input. It sends a GET request to the URL and retrieves the HTML content. Using BeautifulSoup, it extracts the main content element from the HTML. Then, the text content is processed using spaCy's English language model (`en_core_web_sm`).

The program loops through the entities recognized by spaCy and filters them based on their entity labels. In this example, we consider entities labeled as 'PERSON', 'ORG' (organization), and 'GPE' (geopolitical entity) as core entities. You can modify or expand this list according to your requirements.

Finally, the program returns a list of core entities. In the example usage section, we provide a Wikipedia URL and print the extracted core entities.

Note that web scraping may be subject to the terms of service and restrictions imposed by websites. Make sure to review and respect the policies of the target website before scraping any content.

(a) Let's use the program to extract core entities from the Wikipedia page of Sachin Tendulkar.

```
wikipedia_url = 'https://en.wikipedia.org/wiki/Sachin_Tendulkar'
core_entities = extract_core_entities(wikipedia_url)

print("Core Entities:")
for entity in core_entities:
    print(entity)
```

When you run this code, it will fetch the Wikipedia page for Sachin Tendulkar and extract the core entities. Here are some example core entities you might expect to see:

```
Core Entities:
Sachin Tendulkar
Mumbai
India
Bharat Ratna
Wisden Cricketer of the Year
Master Blaster
Indian Premier League
```

These are some of the entities recognized as core entities based on the 'PERSON', 'ORG', and 'GPE' labels from the page's content. The specific entities extracted may vary slightly depending on the page's structure and the version of spaCy used.

4. Analyze the documents given in Resumes folder (attached). What kind of analytics/insights can you propose/infer from this dataset?

Analyzing a dataset containing many resume documents can provide various valuable insights and analytics. Here are some examples:

1. Skills and qualifications: By extracting information from resumes, you can identify the most commonly mentioned skills and qualifications. This can help organizations understand the skill sets that are in high demand among job applicants, enabling them to tailor their hiring strategies accordingly.
2. Education and experience trends: Analyzing resumes can reveal patterns in educational backgrounds and work experience. You can identify the most prevalent degrees, certifications, and universities attended, as well as the typical career progression paths for different roles. This information can assist employers in setting realistic expectations for candidate profiles and determining the desired qualifications for specific positions.
3. Job market trends: Analyzing resumes can provide insights into the overall job market trends, such as the frequency of specific job titles or industry preferences. This can help businesses stay updated on evolving roles and industries and align their hiring strategies accordingly.
4. Geographical insights: Resumes often include location information, which can be leveraged to understand geographic preferences of job seekers and identify regional talent pools. This information is valuable for organizations considering expanding their operations or targeting specific locations for recruitment.
5. Language and communication skills: Analyzing resumes can provide insights into the language proficiency and communication skills of job applicants. You can identify commonly mentioned languages and evaluate the level of proficiency mentioned for each language. This is particularly useful for companies with global operations or those seeking multilingual candidates.
6. Keyword analysis: By analyzing resumes, you can identify frequently occurring keywords and phrases. This can help optimize job postings to attract the right candidates and improve applicant tracking systems by refining keyword-based filtering and matching algorithms.
7. Diversity and inclusion: Analyzing resumes can help assess diversity and inclusion efforts by tracking demographic information (if available) such as gender, ethnicity, or nationality. This information can aid organizations in monitoring their hiring practices and identifying potential biases.
8. Applicant source analysis: By analyzing resumes, you can determine the sources through which applicants discovered job opportunities (e.g., job boards, company websites, referrals). This information can assist in optimizing recruitment channels and allocating resources to the most effective sourcing strategies.
9. Resume quality assessment: Analyzing resumes can include evaluating factors such as formatting, structure, and content quality. This assessment can provide insights into the presentation skills of applicants and assist in creating guidelines for resume preparation or offering personalized feedback to candidates.
10. Career progression patterns: By analyzing resumes over time, you can track career progression patterns and understand factors that contribute to professional growth. This can help identify

common career trajectories, the impact of certain experiences on career development, and potential benchmarks for advancement within specific industries or roles.

It's important to note that while analyzing resume data can provide valuable insights, it's crucial to handle the data ethically, respecting privacy regulations and ensuring anonymity when working with personal information.

(a) Write a program to create a visualization dashboard for the same

Creating a visualization dashboard for analyzing resume data requires programming and data visualization tools. Here's an example of how you can build a visualization dashboard using Python and the popular data visualization library, Plotly:

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Load the resume data into a Pandas DataFrame
resume_data = pd.read_csv('resume_data.csv')

# Data preprocessing and analysis
# Perform necessary data transformations, calculations, and aggregations

# Create the visualization dashboard
# Use Plotly to create interactive visualizations

# 1. Skills and qualifications visualization
skills_counts = resume_data['Skills'].value_counts().reset_index()
skills_counts.columns = ['Skills', 'Count']
skills_fig = px.bar(skills_counts, x='Skills', y='Count', title='Most Commonly Mentioned Skills')

# 2. Education and experience trends visualization
education_counts = resume_data['Education'].value_counts().reset_index()
education_counts.columns = ['Education', 'Count']
education_fig = px.bar(education_counts, x='Education', y='Count', title='Most Common Educational Backgrounds')

experience_counts = resume_data['Experience'].value_counts().reset_index()
experience_counts.columns = ['Experience', 'Count']
experience_fig = px.bar(experience_counts, x='Experience', y='Count', title='Most Common Work Experience')

# 3. Job market trends visualization
job_title_counts = resume_data['Job Title'].value_counts().reset_index()
job_title_counts.columns = ['Job Title', 'Count']
job_title_fig = px.bar(job_title_counts, x='Job Title', y='Count', title='Most Common Job Titles')

# Create the dashboard layout
dashboard = make_subplots(rows=2, cols=2, subplot_titles=['Skills', 'Education', 'Experience', 'Job Titles'])
dashboard.add_trace(skills_fig.data[0], row=1, col=1)
dashboard.add_trace(education_fig.data[0], row=1, col=2)
dashboard.add_trace(experience_fig.data[0], row=2, col=1)
dashboard.add_trace(job_title_fig.data[0], row=2, col=2)

dashboard.update_layout(height=600, width=800, title='Resume Data Analysis Dashboard')

# Display the dashboard
dashboard.show()
```

In this example, the program assumes that the resume data is stored in a CSV file named 'resume_data.csv'. You need to modify the code to match your dataset's structure and columns accordingly.

The program uses Plotly to create various visualizations based on the resume data. It includes four visualizations: skills and qualifications, education, experience, and job titles. These visualizations are then combined into a dashboard layout using `make_subplots()` function, and the layout is customized using `update_layout()` function. Finally, the dashboard is displayed using `show()` function.

5. Write a program and explain how you can classify different images.

To classify different images, you can use a deep learning model, such as a convolutional neural network (CNN). CNNs have proven to be highly effective in image classification tasks. Here's an example of how you can approach image classification using Python and the TensorFlow library:

Step 1: Import the required libraries

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

Step 2: Set up the data

Download a set of images related to animals and vehicles from the internet and organize them into separate directories. For example, create two directories named "animals" and "vehicles" and place the respective images in each directory.

Step 3: Preprocess the data

```
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

training_set = train_datagen.flow_from_directory('path/to/training_data',
                                                target_size=(64, 64),
                                                batch_size=32,
                                                class_mode='binary')
```

In the code above, we use the `ImageDataGenerator` class to perform data augmentation, which helps to improve the model's generalization ability. We then create a generator using `flow_from_directory` function, which loads and resizes the images from the specified directory.

Step 4: Build the CNN model

```

model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

```

In this code, we define a sequential model and add multiple convolutional and pooling layers. The `Flatten` layer converts the 2D feature maps into a 1D vector, which is then passed through fully connected (`Dense`) layers. The final layer uses the sigmoid activation function for binary classification.

Step 5: Compile and train the model

```

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

model.fit(training_set, epochs=10)

```

We compile the model with the Adam optimizer and binary cross-entropy loss function. Then, we train the model using the training set for a specified number of epochs.

Step 6: Test the model

To evaluate the model's performance, you can use a separate set of test images.

```

test_datagen = ImageDataGenerator(rescale=1.0/255)
test_set = test_datagen.flow_from_directory('path/to/test_data',
                                           target_size=(64, 64),
                                           batch_size=32,
                                           class_mode='binary')

loss, accuracy = model.evaluate(test_set)
print(f"Test loss: {loss}")
print(f"Test accuracy: {accuracy}")

```

We use the evaluate function to calculate the loss and accuracy of the model on the test set.

This is a basic outline of how you can classify different images using a CNN. Make sure to replace 'path/to/training_data' and 'path/to/test_data' with the appropriate paths to your image directories. Additionally, you can experiment with different CNN architectures, hyperparameters, and augmentation techniques to improve the model's performance.

- (a) Download images related to animals and vehicles from the internet to explain and code your approach.**