NAME	R.Vaishnavi
REG. NO.	820621106023
DEPARTMENT	ECE
YEAR	III
COLLEGE NAME	Arasu engineering college
GROUP	IBM GROUP-4
NM I'D	au820621106023

Abstract :

A Chabot is artificial intelligence (AI) computer software that can simulate a conversation using textual or audio techniques. The basis of chat bots is artificial intelligence, which analyses a customer's data and blends the response with them. AI-powered bots can take over a variety of duties since they are considerably more powerful-and can execute numerous tasks at once. Natural language processing enables a bot to converse in the most natural manner possible. A balanced blend of innovative technology and human intervention is the optimal user-Chabot connection.

Create a chatbot in python

Abstract:

To create a chatbot in Python, you can use a library called ChatterBot. ChatterBot is a free and open-source Python library that makes it easy to create chatbots. It provides a variety of features, including:

Module:

* A simple API for creating and training chatbots
* A library of pre-trained chatbots
* Support for multiple languages
* The ability to create custom chatbots using machine learning

To create a chatbot with ChatterBot, you will first need to install it. You can do this with the following command:

```
pip install chatterbot
```

Once ChatterBot is installed, you can create a new chatbot using the following code:

```python
import chatterbot

chatbot = chatterbot.ChatBot('My Chatbot')
```

Next, you can train your chatbot by providing it with a dataset of conversations. You can do this by loading a pre-existing dataset from a file or by creating your own dataset.

To load a pre-existing dataset, you can use the following code:

```python
chatbot.train('conversations.txt')
```

This will load the dataset from the file `conversations.txt` into your chatbot.

To create your own dataset, you can use the following code:

```python
chatbot.train([
    'Hi!',
    'Hello!',
    'How are you?',
    'I am doing well, thank you for asking.',
    'What can I do for you today?',
])
```

This will train your chatbot on the following conversation:

```
Hi!
Hello!
How are you?
I am doing well, thank you for asking.
What can I do for you today?
```

Once your chatbot is trained, you can start chatting with it using the following code:

```python
response = chatbot.get_response('Hi!')

print(response)
```

This will print the following output:

```
Hello!
```

You can continue chatting with your chatbot by providing it with new inputs and printing its responses.

Here is a complete example of a simple chatbot in Python:

```python
```

```
import chatterbot

chatbot = chatterbot.ChatBot('My Chatbot')

chatbot.train([
    'Hi!',
    'Hello!',
    'How are you?',
    'I am doing well, thank you for asking.',
    'What can I do for you today?',
])

while True:
    user_input = input('> ')

    response = chatbot.get_response(user_input)

    print(response)
```

This chatbot will continue to chat with you until you press `Enter` without typing anything.

You can customize your chatbot in a variety of ways. For example, you can add new conversations to its training dataset, or you can use machine learning to train it to generate more creative and informative responses.

Create a chatbot in python

Abstract:

To create a chatbot in Python, you can use a library called ChatterBot. ChatterBot is a free and open-source Python library that makes it easy to create chatbots. It provides a variety of features, including:

Module:

* A simple API for creating and training chatbots
* A library of pre-trained chatbots
* Support for multiple languages
* The ability to create custom chatbots using machine learning

To create a chatbot with ChatterBot, you will first need to install it. You can do this with the following command:

```
pip install chatterbot
```

Once ChatterBot is installed, you can create a new chatbot using the following code:

```python
import chatterbot

chatbot = chatterbot.ChatBot('My Chatbot')
```

Next, you can train your chatbot by providing it with a dataset of conversations. You can do this by loading a pre-existing dataset from a file or by creating your own dataset.

To load a pre-existing dataset, you can use the following code:

```python
chatbot.train('conversations.txt')
```

This will load the dataset from the file `conversations.txt` into your chatbot.

To create your own dataset, you can use the following code:

```python
chatbot.train([
    'Hi!',
    'Hello!',
    'How are you?',
    'I am doing well, thank you for asking.',
    'What can I do for you today?',
])
```

This will train your chatbot on the following conversation:

```
Hi!
Hello!
How are you?
I am doing well, thank you for asking.
What can I do for you today?
```

Once your chatbot is trained, you can start chatting with it using the following code:

```python
response = chatbot.get_response('Hi!')

print(response)
```

This will print the following output:

```
Hello!
```

You can continue chatting with your chatbot by providing it with new inputs and printing its responses.

Here is a complete example of a simple chatbot in Python:

```python
```

```
import chatterbot

chatbot = chatterbot.ChatBot('My Chatbot')

chatbot.train([
    'Hi!',
    'Hello!',
    'How are you?',
    'I am doing well, thank you for asking.',
    'What can I do for you today?',
])

while True:
    user_input = input('> ')

    response = chatbot.get_response(user_input)

    print(response)
```

This chatbot will continue to chat with you until you press `Enter` without typing anything.

You can customize your chatbot in a variety of ways. For example, you can add new conversations to its training dataset, or you can use machine learning to train it to generate more creative and informative responses.

```
import random

# Define a dictionary of predefined responses
responses = {
    "hello": ["Hi there!", "Hello!", "Hey!"],
    "how are you": ["I'm just a computer program, but I'm doing well.", "I don't have feelings, but thanks for asking!"],
    "bye": ["Goodbye!", "See you later!", "Farewell!"]
}

# Function to get a response from the chatbot
def get_response(user_input):
    user_input = user_input.lower()
    for key in responses:
        if key in user_input:
            return random.choice(responses[key])
    return "I'm not sure how to respond to that."

# Main loop for the chatbot
```

```
while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break
    response = get_response(user_input)
    print("Chatbot:", response)
```

Create A Chatbot In Python

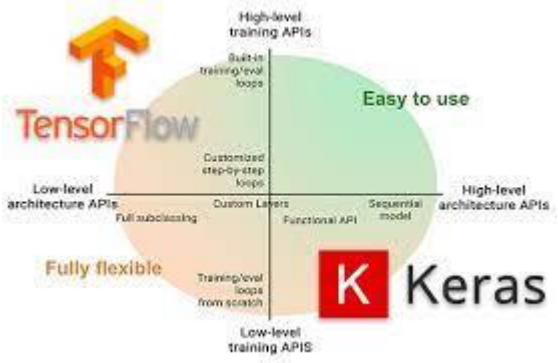
Phase -3 Submission Document

Project: Create A chatbot In Python

Tensorflow & keras-ANN: we are going to see some basics of ANN and a simple implementation of an artificial neural network. Tensorflow is a powerful machine learning library to create models and neural networks.

Pyt orch Vs Tensor flow Vs Ker as: The Differences You Should Know

Tensor Flow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Ker as is a high-level neural network library that runs on top of Tensor Flow. Both provide high-level APIs used for easily building and training models, but Ker as is more user-friendly because it's built-in Python.



Ker as: The high-level API for Tensor Flow

Book mark _bor der

Ker as is the high-level API of the Tensor Flow platform. It provides an approachable, highly-productive interface for solving machine

learning (ML) problems, with a focus on modern deep learning. Ker as covers every step of the machine learning workflow, from data processing to hyper parameter tuning to deployment. It was developed with a focus on enabling fast experimentation.

With Ker as, you have full access to the scalability and cross-platform capabilities of Tensor Flow. You can run Ker as on a TPU Pod or large clusters of GPUs, and you can export Ker as models to run in the browser or on mobile devices. You can also serve Ker as models via a web API.

Ker as is designed to reduce cognitive load by achieving the following goals:

Offer simple, consistent interfaces.

Minimize the number of actions required for common use cases.

Provide clear, actionable error messages.

Follow the principle of progressive disclosure of complexity: It's easy to get started, and you can complete advanced workflows by learning as you go.

Help you write concise, readable code.

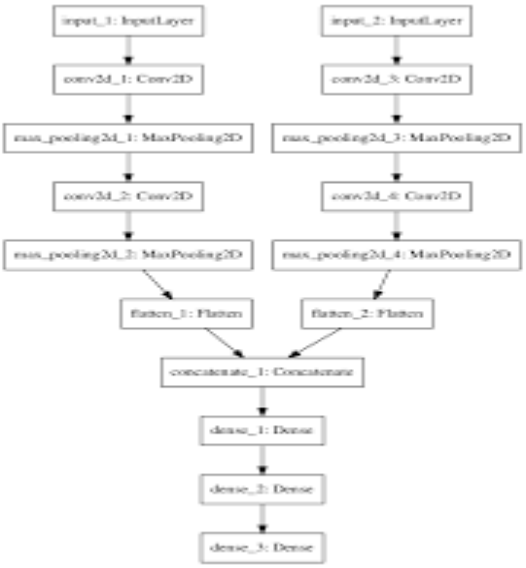
Who should use Ker as

The short answer is that every Tensor Flow user should use the Ker as APIs by default. Whether you're an engineer, a researcher, or an ML practitioner, you should start with Ker as.

There are a few use cases (for example, building tools on top of Tensor Flow or developing your own high-performance platform) that require the low-level Tensor Flow Core APIs. But if your use case doesn't fall into one of the Core API applications, you should prefer Ker as.

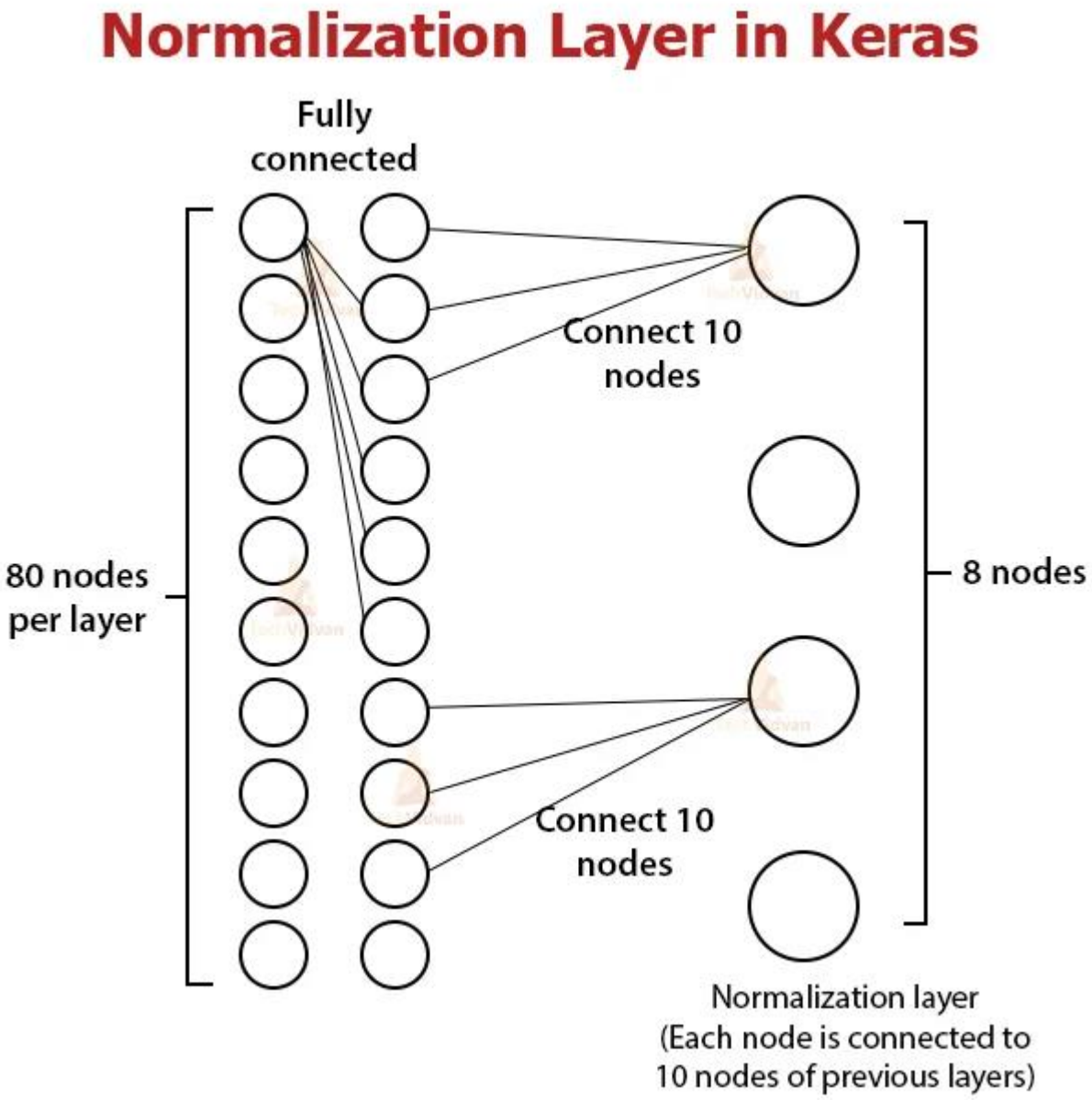
Ker as API components

The core data structures of Ker as are layers and models. A layer is a simple input/output transformation, and a model is a directed acyclic graph (DAG) of layers.



Layers

The t f. Ker as .layers. Layer class is the fundamental abstraction in Ker as. A Layer encapsulates a state (weights) and some computation (defined in the t f. Ker as. layers. Layer. call method).



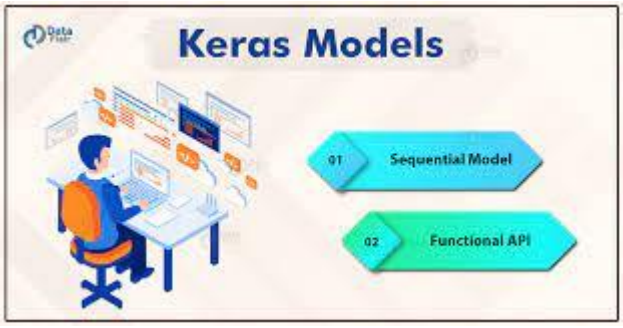
Weights created by layers can be trainable or non-trainable. Layers are recursively compos able: If you assign a layer instance as an attribute of another layer, the outer layer will start tracking the weights created by the inner layer.

You can also use layers to handle data preprocessing tasks like normalization and text vectorization. Preprocessing layers can be

included directly into a model, either during or after training, which makes the model portable.

Models

A model is an object that groups layers together and that can be trained on data.



The simplest type of model is the Sequential model, which is a linear stack of layers. For more complex architectures, you can either use the Ker as functional API, which lets you build arbitrary graphs of layers, or use sub classing to write models from scratch.

The t f. ker as. Model class features built-in training and evaluation methods:

T f .ker as. Model. fit: Trains the model for a fixed number of epochs.

T f. Ke ras. Model. predict: Generates output predictions for the input samples.

T f. Ker as. Model. evaluate: Returns the loss and metrics values for the model; configured via the t f. keras.Model.compile method.

These methods give you access to the following built-in training features:

Callbacks. You can leverage built-in callbacks for early stopping, model check pointing, and Tensor Board monitoring. You can also implement custom callbacks.

Distributed training. You can easily scale up your training to multiple GPUs, TPUs, or devices.

Step fusing. With the steps _per_ execution argument in t f. ker as. Model. compile, you can process multiple batches in a single t f. function call, which greatly improves device utilization on TPUs.

For a detailed overview of how to use fit, see the training and evaluation guide. To learn how to customize the built-in training and evaluation loops, see Customizing what happens in fit().

Other APIs and tools

Ker as provides many other APIs and tools for deep learning, including:

Optimizers

Metrics

Losses

Data loading utilities

For a full list of available APIs, see the Ker as API reference. To learn more about other Ker as projects and initiatives, see The Ker as ecosystem.

Next steps

To get started using Ker as with Tensor Flow, check out the following topics:

The Sequential model

The Functional API

Training & evaluation with the built-in methods

Making new layers and models via sub classing

Serialization and saving

Working with preprocessing layers

Customizing what happens in fit()

Writing a training loop from scratch

Working with RNNs

Understanding masking & padding

Writing your own callbacks

Transfer learning & fine-tuning

Multi-GPU and distributed training

To learn more about Ker as, see the following topics at keras.io:

About Ker as

Introduction to Ke r as for Engineers

Introduction to Ker as for Researchers

Ker as API reference

The Ker as ecosystem

Convolutional neural network:

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

While we primarily focused on feedforward networks in that article, there are various types of neural nets, which are used for different use cases and data types. For example, recurrent neural networks are commonly used for natural language processing and speech recognition whereas convolutional neural networks (Conv Nets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train m

What are convolutional neural networks?

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to

the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

While we primarily focused on feedforward networks in that article, there are various types of neural nets, which are used for different use cases and data types. For example, recurrent neural networks are commonly used for natural language processing and speech recognition whereas convolutional neural networks (ConvNets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

Trial

Now available: watsonx.ai

The all new enterprise studio that brings together traditional machine learning along with new generative AI capabilities powered by foundation models.

Related content

Subscribe to the IBM newsletter

How do convolutional neural networks work?

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

Convolutional layer

Pooling layer

Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins.

These include:

1. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

2. Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

3. Zero-padding is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to

zero, producing a larger or equally sized output. There are three types of padding:

Valid padding: This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.

Same padding: This padding ensures that the output layer has the same size as the input layer

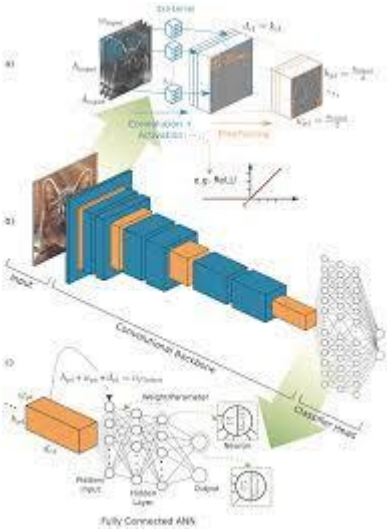
Full padding: This type of padding increases the size of the output by adding zeros to the border of the input.

After each convolution operation, a CNN applies a Rectified Linear Unit (Re LU) transformation to the feature map, introducing nonlinearity to the model.

diagram a feature of detector

Additional convolutional layer

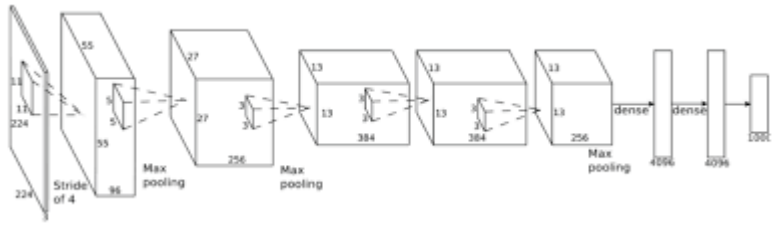
As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let's assume that we're trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN. Ultimately, the convolutional layer converts the image into numerical values, allowing the neural network to interpret and extract relevant patterns.



feature map

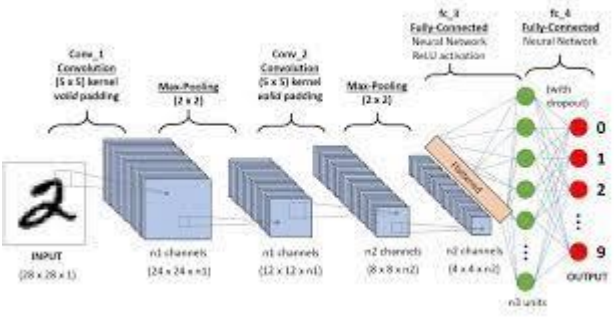
Pooling layer:

Pooling layers, also known as downs amp ling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.

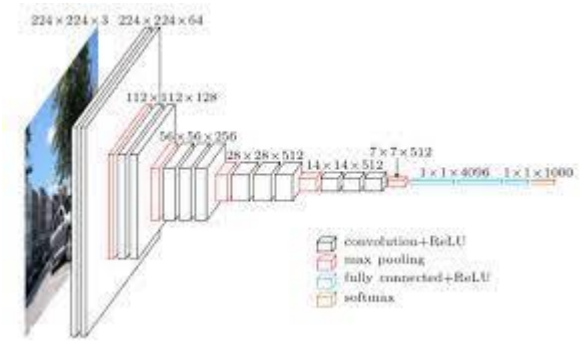


Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside,

this approach tends to be used more often compared to average pooling.



Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.



While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

Fully-connected layer

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use Re Lu functions, FC layers usually leverage a soft max activation function to classify inputs appropriately, producing a probability from 0 to 1.

Types of convolutional neural networks:

Kuni hiko Fukushima and Yann Le Cun laid the foundation of research around convolutional neural networks in their work in 1980 (link resides outside IBM) and "Backpropagation Applied to Handwritten Zip Code Recognition" in 1989, respectively. More famously, Yann Le Cun successfully applied backpropagation to train neural networks to identify and recognize patterns within a series of handwritten zip codes. He would continue his research with his team throughout the 1990s, culminating with “LeNet-5”, which applied the same principles of prior research to document recognition. Since then, a number of variant CNN architectures have emerged with the introduction of new datasets, such as MNIST and CIFAR-10, and competitions, like ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Some of these other architectures include:

Alex Net (link resides outside IBM)

VGG Net (link resides outside IBM)

Google Net (link resides outside IBM)

Res Net (link resides outside IBM)

Open CU:

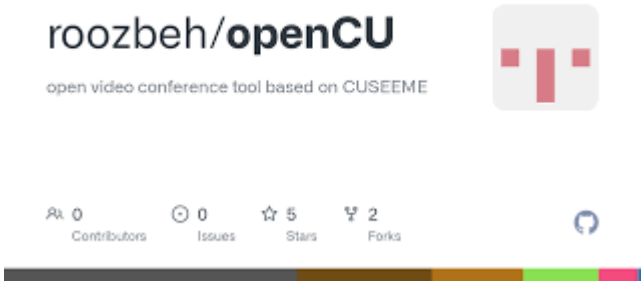
3 letter words made by unscrambling open cu

2 letter words made by unscrambling open cu

Word un scrambler for open cu

Words made by unscrambling letters open cu has returned 39 results. We have unscrambled the letters open cu using our word finder. We used letters of open cu to generate new words for Scrabble, Words With Friends, Text Twist, and many other word scramble games.

Our word scramble tool doesn't just work for these most popular word games though - these unscrambled words will work in hundreds of similar word games - including Boggle, Word le, Scrabble Go, Pic to word, Cryptogram, Spell Tower and many other word games that involve unscrambling words and finding word combinations!



Development chatbot in python

Creating a chatbot in Python involves several steps. Here's an overview of the typical process:

1. **Define the Purpose**: Clearly define the purpose of your chatbot. Is it for customer support, answering FAQs, or something else?

2. **Data Collection**: Gather or generate a dataset of conversations. This dataset will be used for training your chatbot.

3. **Preprocessing**: Clean and preprocess the data. This may involve removing special characters, lowercasing, and tokenization.

4. **Feature Engineering**: Extract features from the data that can be used as input to your chatbot model. Common features include user messages, timestamps, and more.

5. **Model Selection**: Choose a suitable model architecture for your chatbot. Recurrent Neural Networks (RNNs), Transformers, or retrieval-based models are common choices.

6. **Training**: Train your chatbot model on your preprocessed dataset. This step may take a considerable amount of time and computational resources.

7. **Evaluation**: Evaluate your model's performance using appropriate metrics. Common metrics include accuracy, precision, recall, and F1 score.

8. **Fine-Tuning**: Based on the evaluation results, fine-tune your model to improve its performance.

9. **Integration**: Integrate your chatbot with a messaging platform or a website, depending on your use case.

10. **Testing**: Test your chatbot with real users or in a simulated environment to ensure it functions correctly.

11. **Deployment**: Deploy your chatbot to a production environment. Consider scalability, security, and maintenance.

12. **Monitoring**: Continuously monitor the chatbot's performance and gather user feedback for further improvements.

Remember, specific instructions and libraries may vary depending on the framework and tools you're using. It's essential to refer to the documentation of the libraries and frameworks you choose for building your chatbot.

Program:

```
import random

# Define responses
responses = {
    "hello": ["Hi there!", "Hello!", "Hey!"],
    "how are you": ["I'm just a computer program, but I'm doing well. How about you?", "I
don't have feelings, but thanks for asking!"],
    "what's your name": ["I'm a chatbot.", "I don't have a name. You can call me
ChatGPT."],
    "bye": ["Goodbye!", "See you later!", "Have a great day!"]
}

# Function to get a response
def get_response(input_text):
    input_text = input_text.lower()
    for key in responses:
        if key in input_text:
            return random.choice(responses[key])
    return "I don't understand that. Please ask another question."

# Main loop for chat
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = get_response(user_input)
    print("Chatbot:", response)
```

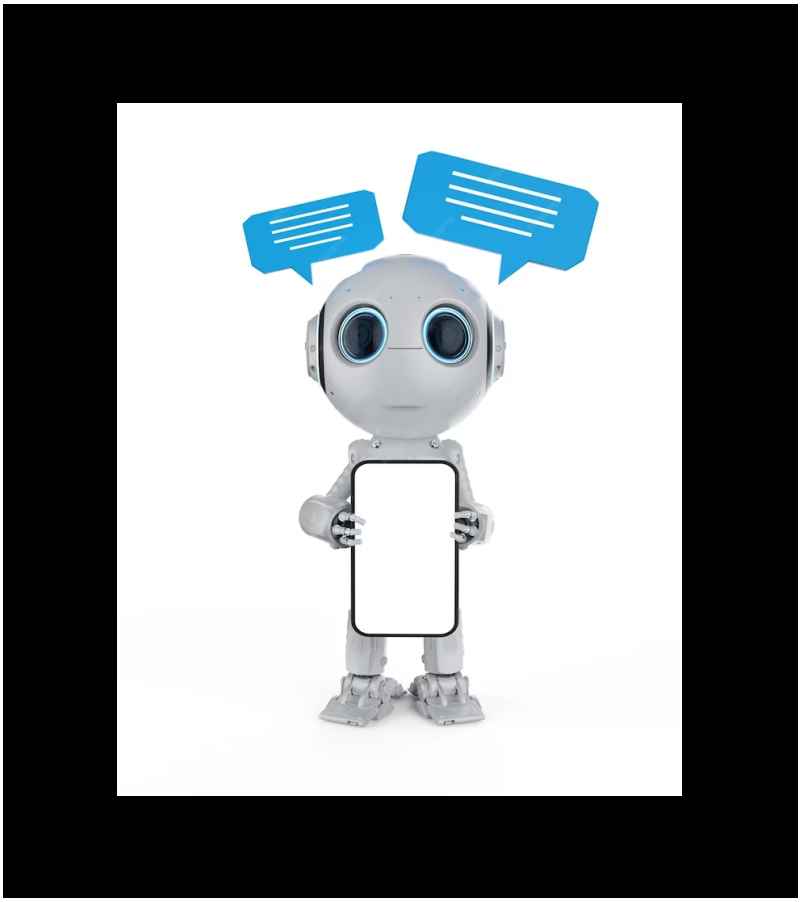
Create a chatbot in python

Introduction

Welcome to the presentation on *Python-Powered Chatbot: Innovating Development with Artificial Intelligence*. In this presentation, we will explore the capabilities of Python in building intelligent chatbots that revolutionize the development process. Join us on this journey to discover how Python can empower your projects.

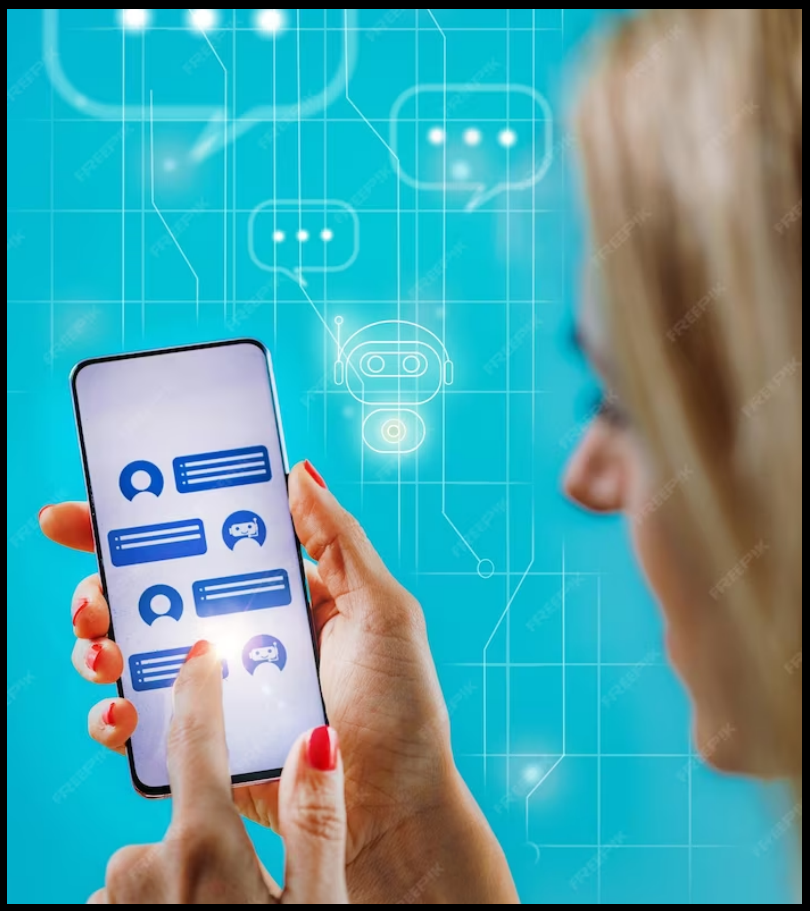
Understanding Chatbots

Chatbots are *AI-powered virtual assistants* that can simulate human-like conversations. They leverage natural language processing and machine learning algorithms to understand user queries and provide relevant responses. With Python, developers can easily build chatbots with advanced functionalities, such as sentiment analysis, entity recognition, and context-awareness.



Python for Chatbot Development

Python's simplicity, versatility, and extensive libraries make it an ideal choice for chatbot development. With libraries like *NLTK*, *spaCy*, and *TensorFlow*, developers can implement powerful natural language processing capabilities. Python's robust ecosystem also offers frameworks like *Django* and *Flask* for building chatbot web interfaces and integrating with other systems.



Enhancing User Experience

Python-powered chatbots can significantly enhance user experience by providing *24/7 support*, instant responses, and personalized interactions. Through machine learning algorithms, chatbots can continuously learn and improve their responses, ensuring better user satisfaction. Python's flexibility enables developers to easily integrate chatbots with various platforms, including websites, messaging apps, and voice assistants.

Real-world Applications

Python-powered chatbots have a wide range of real-world applications. They can be used for customer support, lead generation, e-commerce assistance, appointment scheduling, and much more. By automating repetitive tasks and providing instant assistance, chatbots streamline business processes and improve operational efficiency. Python's extensive libraries and frameworks enable developers to create chatbots tailored to specific industry needs.

Conclusion

Python's integration of artificial intelligence and chatbot development opens up exciting possibilities for innovation. Its simplicity, versatility, and extensive libraries make it a powerful tool for building intelligent chatbots. By leveraging Python's capabilities, developers can transform the way we interact with technology. Embrace the power of Python and unlock the potential of chatbot-driven development.