Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação

CPS769 - Introdução à Inteligência Artificial e Aprendizagem Generativa

Prof. Dr. Edmundo de Souza e Silva (PESC/COPPE/UFRJ)
Profa. Dra. Rosa M. Leão (PESC/COPPE/UFRJ)
Participação Especial: Gaspare Bruno (Diretor Inovação, ANLIX)

Lista de Exercícios 5

Luiz Henrique Souza Caldas
email: lhscaldas@cos.ufrj.br

15 de agosto de 2024

Questão 1

O objetivo da lista é fazer com que você exercite os conceitos de RAG e *embedding* conforme visto nesta aula. Você deve usar as APIs da OpenAI cujo acesso para todos foi disponibilizado pela startup Anlix.

Escolha um dos artigos estudados até o momento. (Procure escolher um artigo diferente do seu colega.) Usando os conceitos vistos em aula, faça uma aplicação onde o usuário deve fazer perguntas sobre o artigo e as respostas serão obtidas do artigo e repassadas ao usuário.

Já existem aplicações que usam as APIs da OpenAI para fazer o que descrevemos acima (exemplo: *AskYourPDF*). Você deve usar os conceitos de *Embedding* no texto do artigo, que será a fonte de dados. E obviamente os conceitos de RAG. As perguntas serão quaisquer passadas pelo usuário. Um conjunto de perguntas e respostas deve ser incluído no resultado. Seja criativo!

Discutiremos o código de cada um em sala de aula, os problemas e os resultados. O seu código deve estar executando.

Explicação do código implementado

Abaixo será explicado cada trecho do código implementado para essa lista. O código completo encontra-se no final deste reletório 1 e no repositório https://github.com/lhscaldas/cps769-ai-gen.

Extração do Texto

Primeiro, precisamos extrair o texto do PDF ou da URL fornecida. Para isso, foi utilizado o *BeautifulSoup* e o *pdfplumber*.

```
def extract_text(input_path):
    parsed_url = urlparse(input_path)
    if parsed_url.scheme in ['http', 'https']:
        # Se for uma URL, faz o download do conteúdo
        response = requests.get(input_path)
        response.raise_for_status()  # Levanta um erro se a requisição falhar

        # Extraí o texto da página web
        soup = BeautifulSoup(response.text, 'html.parser')
        text = soup.get_text()
    else:
        # Se não for uma URL, assume que é um caminho de arquivo PDF
        with open(input_path, 'rb') as pdf_file:
            with pdfplumber.open(pdf_file) as pdf:
                text = ''
                for page in pdf.pages:
                    text += page.extract_text() or ''

    return text
```

Dividisão do Texto em Partes Menores

Para criar os embeddings, precisamos dividir o texto em partes menores (chunks).

```python
def split_text(text, max_tokens=500):
    # Vamos dividir o texto em parágrafos, assumindo que cada parágrafo seja
       pequeno o suficiente.
    paragraphs = text.split('\n\n')
    chunks = []
    current_chunk = ''

    for paragraph in paragraphs:
        if len(current_chunk) + len(paragraph) <= max_tokens:
            current_chunk += paragraph + '\n\n'
        else:
            chunks.append(current_chunk)
            current_chunk = paragraph + '\n\n'

    # Adiciona o último chunk
    if current_chunk:
        chunks.append(current_chunk)

    return chunks
```

Indexação dos Chunks

Agora, precisamos criar uma indexação para que possamos recuperar as informações relevantes quando uma pergunta for feita. Foi utilizada a classe *OpenAIEmbeddings* para isso.

```python
def create_index(chunks: List[str]) -> FAISS:
    embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)

    # Criar objetos Document
    documents = [Document(page_content=chunk) for chunk in chunks]

    # Criar o índice FAISS
    faiss_index = FAISS.from_documents(documents, embeddings)

    return faiss_index
```

Recuperação dos Chunks

Com a indexação criada, podemos recuperar os trechos relevantes para a pergunta, utilizando o método *similarity_search*.

```python
def retrieve_relevant_chunks(query: str, index: FAISS) -> List[str]:
    results = index.similarity_search(query, k=3)

    relevant_chunks = [result.page_content for result in results]
    return relevant_chunks
```

Geração da Resposta Usando a API da OpenAI

Finalmente, podemos usar a API da OpenAI para gerar a resposta a partir dos trechos recuperados.

```python
# Define a estrutura de resposta para a API da OpenAI
class AnswerResponse(BaseModel):
    """Respond in a conversational manner to answer the question based on the context provided."""
    response: str = Field(description="A response to the user's question based on the context provided")

# Setup do modelo da OpenAI usando LangChain
llm = ChatOpenAI(
    model="gpt-4o-mini",  # Modelo GPT-4o-mini
    openai_api_key=OPENAI_API_KEY
)

# Configurando o modelo LLM para fornecer a resposta estruturada
structured_llm = llm.with_structured_output(AnswerResponse)

# Função para gerar a resposta usando o LangChain
def generate_answer(question: str, relevant_chunks: List[str]) -> str:
    # Combinar os chunks relevantes em um contexto
    context = "\n".join(relevant_chunks)

    # Preparar a entrada do prompt com o contexto e a pergunta
    input_data = f"Context: {context}\n\n Question: {question}\n"

    # Invocar o modelo para gerar a resposta
    answer = structured_llm.invoke(input_data)

    return answer.response
```

Integração Final

Agora, podemos integrar todas as partes para criar uma função que recebe um PDF, uma pergunta, e retorna uma resposta.

```python
def answer_question(pdf_path, question):
    # 1. Extrair texto do PDF
    text = extract_text(pdf_path)

    # 2. Dividir o texto em chunks
    chunks = split_text(text)

    # # 3. Criar o índice e embeddings
    index = create_index(chunks)

    # # 4. Recuperar os trechos relevantes
    relevant_chunks = retrieve_relevant_chunks(question, index)

    # # 5. Gerar a resposta
    answer = generate_answer(question, relevant_chunks)

    return answer
```

Testes

Teste 1: artigo *AI's Language Problem*

- Pergunta 1: Quais são os principais desafios da IA na compreensão do contexto em linguagem humana?

  Resposta: Os principais desafios da IA na compreensão do contexto em linguagem humana incluem:

  1. **Ambiguidade Linguística**: Palavras podem ter múltiplos significados dependendo do contexto. Por exemplo, a palavra "banco"pode referir-se a uma instituição financeira ou a um assento. A IA deve ser capaz de discernir o significado correto com base no contexto em que a palavra é usada.

  2. **Referência Contextual**: A IA precisa entender a que ou quem uma palavra ou frase se refere em uma conversa. Por exemplo, se alguém diz "ele está vindo", a IA deve saber quem é "ele"com base nas informações anteriores.

  3. **Conhecimento de Mundo**: A compreensão da linguagem está profundamente ligada ao conhecimento comum sobre o mundo. A IA precisa ter acesso a informações sobre como o mundo funciona, incluindo relações entre objetos, ações e eventos, o que é um grande desafio.

  4. **Inferência e Contexto Social**: Muitas vezes, o significado de uma conversa depende de inferências sociais e emocionais. A IA precisa reconhecer e interpretar tons, ironias ou puns, que exigem uma compreensão mais profunda do contexto social.

  5. **Complexidade Gramatical**: A linguagem humana é complexa e cheia de regras gramaticais que podem variar significativamente entre diferentes idiomas e contextos. A IA deve ser capaz de entender e processar essa complexidade para responder adequadamente.

  6. **Aprendizado Limitado**: Ao contrário dos humanos, que podem aprender rapidamente a partir de experiências limitadas, as redes neurais geralmente precisam de grandes quantidades de dados para generalizar bem. Isso pode limitar a capacidade da IA de compreender novos contextos de forma eficaz.

  Esses desafios combinados tornam a tarefa de ensinar uma máquina a compreender linguagem e contexto humana uma tarefa complexa e em evolução.

- Pergunta 2: Como combinar aprendizado de máquina com regras linguísticas pode melhorar a compreensão da linguagem pela IA?

  Resposta: Combinar aprendizado de máquina com regras linguísticas pode melhorar a compreensão da linguagem pela IA de várias maneiras. O aprendizado de máquina, especialmente através de redes neurais profundas, permite que os sistemas aprendam padrões complexos e sutilezas na linguagem a partir de grandes quantidades de dados. No entanto, a aplicação de regras linguísticas pode fornecer estrutura e contexto que ajudam a IA a interpretar corretamente os significados das palavras e frases, especialmente em casos onde o significado depende de contexto ou ambiguidade.

  Por exemplo, enquanto o aprendizado de máquina pode ajudar a IA a identificar a semântica de palavras em um texto, as regras linguísticas podem ajudar a IA a entender a gramática e a sintaxe, possibilitando que ela forme frases coerentes e compreensíveis.

Além disso, ao integrar conhecimento comum e regras gramaticais, a IA pode desenvolver uma compreensão mais profunda da linguagem, permitindo que ela resolva ambiguidades, reconheça ironias ou hipérboles e se envolva em conversas de maneira mais natural e contextualizada. Dessa forma, a combinação dessas abordagens pode levar a um avanço significativo na capacidade da IA de entender e interagir com a linguagem humana de forma mais eficaz.

- Pergunta 3: Quais exemplos o artigo dá sobre falhas da IA em entender linguagem, e o que podemos aprender com eles?

  Resposta: O artigo fornece vários exemplos de falhas da IA em entender linguagem. Um exemplo é o programa de resposta automática criado por Quoc Le, que, ao ser perguntado sobre quantas pernas um centopéia tem, respondeu erradamente "oito". Isso mostra que, embora o sistema possa identificar padrões linguísticos e formular respostas, ele não possui compreensão real do mundo ou do significado das palavras que utiliza.

  Outro exemplo refere-se ao sistema que gerava legendas para imagens, que cometia erros bizarros, como descrever um sinal de trânsito como uma "geladeira cheia de comida". Isso indica que, apesar de a IA ser capaz de processar informações visuais e gerar texto, ela não consegue integrar essas informações de maneira lógica ou contextual.

  Essas falhas ensinam que a IA, na sua forma atual, ainda não alcançou uma verdadeira compreensão da linguagem, que envolve não apenas a manipulação de símbolos, mas também um entendimento profundo do contexto, significado e a natureza do mundo. Para avançar, é necessário integrar conhecimento comum e uma melhor capacidade de raciocínio contextual, em vez de apenas depender de dados e algoritmos para gerar respostas.

Teste 2: artigo *The Unreasonable Effectiveness of Recurrent Neural Networks*

- Pergunta 1: Qual é a principal vantagem das Redes Neurais Recorrentes (RNNs) em comparação com redes neurais tradicionais para tarefas de processamento de sequências?

  Resposta: A principal vantagem das Redes Neurais Recorrentes (RNNs) em comparação com redes neurais tradicionais é sua capacidade de operar sobre sequências de vetores, permitindo que processem entradas e saídas que variam em tamanho e são sequenciais. Enquanto redes neurais tradicionais (como redes neurais feedforward e redes convolucionais) aceitam apenas vetores de tamanho fixo como entrada e produzem saídas de tamanho fixo, as RNNs podem lidar com dados sequenciais, como texto ou áudio, onde a ordem e a dependência temporal entre elementos são cruciais. Isso as torna especialmente eficazes em tarefas como transcrição de fala, tradução de idiomas e modelagem de linguagem.

- Pergunta 2: Como as variantes das RNNs, como LSTM e GRU, melhoram o desempenho em tarefas que envolvem dependências de longo prazo?

  Resposta: As variantes das RNNs, como LSTM (Long Short-Term Memory) e GRU (Gated Recurrent Unit), melhoram o desempenho em tarefas que envolvem dependências de longo prazo de várias maneiras. Aqui estão algumas das principais melhorias que essas arquiteturas oferecem:

  1. **Mecanismos de Portas:** Tanto o LSTM quanto o GRU utilizam mecanismos de portas que controlam o fluxo de informações. Isso permite que a rede decida quais informações devem

ser mantidas ou descartadas ao longo das sequências, ajudando a preservar informações relevantes por longos períodos.

2. **Células de Memória:** O LSTM possui células de memória que podem armazenar informações por longos intervalos de tempo. Isso é crucial em tarefas que exigem a lembrança de informações de etapas anteriores da sequência, enquanto o GRU combina a célula de memória e as portas em uma estrutura mais simplificada, o que também ajuda a capturar dependências de longo prazo.

3. **Gradientes Estáveis:** Uma das limitações das RNNs tradicionais é o problema de gradientes que explodem ou desaparecem, o que dificulta o treinamento em sequências longas. LSTMs e GRUs mitigam esse problema ao permitir que os gradientes fluam de maneira mais estável através das camadas, facilitando o aprendizado de dependências de longo prazo.

4. **Estruturas Mais Simples:** O GRU, em particular, é uma versão simplificada do LSTM, com menos parâmetros e uma arquitetura mais compacta. Isso não apenas acelera o treinamento, mas também pode levar a um desempenho competitivo em muitas tarefas, mantendo a capacidade de capturar dependências de longo prazo.

5. **Flexibilidade:** Ambas as arquiteturas são flexíveis e podem ser adaptadas para uma variedade de tarefas, desde processamento de linguagem natural até reconhecimento de fala e visão computacional, onde as dependências temporais são cruciais.

Em resumo, LSTMs e GRUs são projetados para superar as limitações das RNNs tradicionais, permitindo que modelos aprendam e retenham informações importantes em sequências longas, o que é fundamental para muitas aplicações em aprendizado de máquina.

- Pergunta 3: Quais são alguns exemplos de aplicações práticas onde as RNNs demonstraram resultados surpreendentes, conforme discutido no artigo?

  Resposta: Alguns exemplos de aplicações práticas onde as RNNs demonstraram resultados surpreendentes, conforme discutido no artigo, incluem:

  1. **Transcrição de fala para texto**: As RNNs têm sido utilizadas para converter fala em texto de maneira eficaz. 2. **Tradução automática**: Elas têm sido aplicadas em sistemas de tradução de idiomas, mostrando resultados notáveis. 3. **Geração de texto manuscrito**: As RNNs têm sido usadas para criar texto que parece ter sido escrito à mão. 4. **Modelos de linguagem**: Elas têm sido empregadas como poderosos modelos de linguagem, tanto em nível de caracteres quanto de palavras. 5. **Classificação de vídeos**: No campo da visão computacional, as RNNs estão se tornando comuns para a classificação de vídeos em nível de quadro. 6. **Legenda de imagens**: Elas têm sido utilizadas para gerar descrições de imagens, como mencionado na experiência do autor. 7. **Resposta a perguntas visuais**: Recentemente, as RNNs têm sido aplicadas em tarefas de resposta a perguntas baseadas em imagens.

  Esses exemplos mostram como as RNNs têm se destacado em diversas áreas, superando expectativas em termos de desempenho.

Teste 3: livro *Speech and Language Processing*

- Pergunta 1: Por que as funções de ativação não lineares, como a ReLU ou a sigmoide, são essenciais para o funcionamento de redes neurais?

  Resposta: As funções de ativação não lineares, como a ReLU (Rectified Linear Unit) ou a sigmoide, são essenciais para o funcionamento de redes neurais porque permitem que as redes aprendam e representem relações complexas e não lineares nos dados. Sem essas funções não lineares, uma rede neural composta por várias camadas se comportaria de maneira equivalente a uma única camada linear. Isso significa que, independentemente do número de camadas, a rede seria incapaz de capturar a complexidade dos dados, já que todas as transformações poderiam ser reduzidas a uma única transformação linear.

  Por exemplo, a função ReLU mantém a linearidade para valores positivos e zera os valores negativos, enquanto a sigmoide mapeia a saída para um intervalo entre 0 e 1. Essas características permitem que as redes neurais se ajustem a diferentes conjuntos de dados e aprendam representações úteis, facilitando a separação de classes em problemas de classificação e a modelagem de padrões em dados complexos.

- Pergunta 2: O que torna o problema XOR um exemplo importante na compreensão das limitações dos perceptrons e da necessidade de redes neurais multicamadas?

  Resposta: O problema XOR é um exemplo importante porque ilustra claramente as limitações dos perceptrons, que são unidades de rede neural simples que utilizam uma função de ativação linear. O XOR (ou exclusivo ou) é uma função lógica que não pode ser separada linearmente, o que significa que não é possível traçar uma única linha que separe as saídas positivas (1) das negativas (0) para todas as combinações de entradas.

  Os perceptrons podem resolver funções que são linearmente separáveis, como AND e OR, mas não conseguem resolver o XOR. Isso demonstra que, para aprender funções mais complexas que não são linearmente separáveis, é necessário ter redes neurais multicamadas.

  As redes multicamadas (ou redes neurais profundas) combinam múltiplos perceptrons ou unidades não lineares, permitindo que a rede aprenda representações mais complexas dos dados. No caso do XOR, uma rede neural com duas camadas pode efetivamente transformar as entradas em um espaço onde a separação linear se torna possível, permitindo que a rede aprenda a função XOR. Portanto, esse exemplo destaca a importância das redes neurais multicamadas na superação das limitações dos perceptrons simples.

- Pergunta 3: Quais são as principais diferenças entre uma rede neural feedforward e uma rede neural recorrente (RNN)?

  Resposta: As principais diferenças entre uma rede neural feedforward e uma rede neural recorrente (RNN) incluem:

  1. **Estrutura de Conexão**: - **Rede Neural Feedforward**: Os dados se movem em uma única direção, do input para o output, sem ciclos ou loops. Cada camada é alimentada pela camada anterior e não há conexões que retrocedem. - **Rede Neural Recorrente (RNN)**: Permite conexões de feedback, onde as saídas de uma camada podem ser usadas como entradas para a mesma camada ou para camadas anteriores. Isso permite que as RNNs

mantenham informações de entradas anteriores, tornando-as adequadas para sequências de dados.

2. **Tratamento de Dados Sequenciais**: - **Feedforward**: Geralmente não é ideal para dados sequenciais ou temporais, já que cada entrada é tratada de maneira independente. - **RNN**: Projetada especificamente para lidar com sequências de dados, como texto ou séries temporais. As RNNs podem lembrar informações de etapas anteriores, tornando-as eficazes para tarefas como tradução automática e modelagem de linguagem.

3. **Memória**: - **Feedforward**: Não possui memória interna. Cada previsão é baseada apenas na entrada atual. - **RNN**: Possui uma forma de memória, que permite reter informações de entradas passadas, essencial para entender o contexto em tarefas que envolvem sequências.

4. **Complexidade Computacional**: - **Feedforward**: Geralmente mais simples e mais rápido para treinar, uma vez que não precisa lidar com a complexidade de ciclos e estados ocultos. - **RNN**: Mais complexa de treinar, especialmente devido ao problema do gradiente que pode desaparecer ou explodir durante o treinamento, requerendo técnicas como LSTM (Long Short-Term Memory) ou GRU (Gated Recurrent Units) para estabilizar o aprendizado.

Essas diferenças tornam as RNNs mais adequadas para tarefas que envolvem sequências ou dependências temporais, enquanto as redes feedforward são mais utilizadas em classificações independentes ou tarefas de regressão.

Código

O código abaixo encontra-se no repositório https://github.com/lhscaldas/cps769-ai-gen, bem como o arquivo LaTex com o relatório.

Código 1: código implementado no exercício

```
import pdfplumber
from langchain_community.vectorstores import FAISS
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain.schema import Document
import requests
from bs4 import BeautifulSoup
from urllib.parse import urlparse
from typing import List
from env import OPENAI_API_KEY  # Recupera a key armazenada em um arquivo que está no .gitignore

########################################################################
# Extração do Texto
########################################################################
def extract_text(input_path):
    parsed_url = urlparse(input_path)
    if parsed_url.scheme in ['http', 'https']:
        # Se for uma URL, faz o download do conteúdo
        response = requests.get(input_path)
```

```
        response.raise_for_status()  # Levanta um erro se a requisição falhar
        print(response.text)

        # Extraí o texto da página web
        soup = BeautifulSoup(response.text, 'html.parser')
        text = soup.get_text()
    else:
        # Se não for uma URL, assume que é um caminho de arquivo PDF
        with open(input_path, 'rb') as pdf_file:
            with pdfplumber.open(pdf_file) as pdf:
                text = ''
                for page in pdf.pages:
                    text += page.extract_text() or ''

    return text

###########################################################################
# Dividir o Texto em Partes Menores
###########################################################################
def split_text(text, max_tokens=500):
    # Vamos dividir o texto em parágrafos, assumindo que cada parágrafo seja
       pequeno o suficiente.
    paragraphs = text.split('\n\n')
    chunks = []
    current_chunk = ''

    for paragraph in paragraphs:
        if len(current_chunk) + len(paragraph) <= max_tokens:
            current_chunk += paragraph + '\n\n'
        else:
            chunks.append(current_chunk)
            current_chunk = paragraph + '\n\n'

    # Adiciona o último chunk
    if current_chunk:
        chunks.append(current_chunk)

    return chunks

###########################################################################
# Indexação dos Chunks
###########################################################################
def create_index(chunks: List[str]) -> FAISS:
    embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)

    # Criar objetos Document
    documents = [Document(page_content=chunk) for chunk in chunks]

    # Criar o índice FAISS
    faiss_index = FAISS.from_documents(documents, embeddings)

    return faiss_index

###########################################################################
```

```python
# Recuperação dos Chunks
#########################################################################
def retrieve_relevant_chunks(query: str, index: FAISS) -> List[str]:
    results = index.similarity_search(query, k=3)

    relevant_chunks = [result.page_content for result in results]
    return relevant_chunks

#########################################################################
# Geração da Resposta Usando a API da OpenAI
#########################################################################
# Define a estrutura de resposta para a API da OpenAI
class AnswerResponse(BaseModel):
    """Respond in a conversational manner to answer the question based on the context provided."""
    response: str = Field(description="A response to the user's question based on the context provided")

# Setup do modelo da OpenAI usando LangChain
llm = ChatOpenAI(
    model="gpt-4o-mini",  # Modelo GPT-4o-mini
    openai_api_key=OPENAI_API_KEY
)

# Configurando o modelo LLM para fornecer a resposta estruturada
structured_llm = llm.with_structured_output(AnswerResponse)

# Função para gerar a resposta usando o LangChain
def generate_answer(question: str, relevant_chunks: List[str]) -> str:
    # Combinar os chunks relevantes em um contexto
    context = "\n".join(relevant_chunks)

    # Preparar a entrada do prompt com o contexto e a pergunta
    input_data = f"Context: {context}\n\n Question: {question}\n"

    # Invocar o modelo para gerar a resposta
    answer = structured_llm.invoke(input_data)

    return answer.response

#########################################################################
# Integração Final
#########################################################################
def answer_question(pdf_path, question):
    # 1. Extrair texto do PDF
    text = extract_text(pdf_path)

    # 2. Dividir o texto em chunks
    chunks = split_text(text)

    # # 3. Criar o índice e embeddings
    index = create_index(chunks)

    # # 4. Recuperar os trechos relevantes
```

```
    relevant_chunks = retrieve_relevant_chunks(question, index)

    # # 5. Gerar a resposta
    answer = generate_answer(question, relevant_chunks)

    return answer

##########################################################################
# Usando o Programa
##########################################################################

# Teste 1
pdf_path = 'https://www.technologyreview.com/2016/08/09/158125/ais-language-problem/?gad_source=1&gclid=Cj0KCQjwzva1BhD3ARIsADQuPnU8G_YKoOW29JnFezUFmM8zVN36cX0gQkpzktMoJBlpgRkPr4oDlKMaAgUlEA'
question = 'Quais são os principais desafios da IA na compreensão do contexto em linguagem humana?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 1: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'Como combinar aprendizado de máquina com regras linguísticas pode melhorar a compreensão da linguagem pela IA?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 2: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'Quais exemplos o artigo dá sobre falhas da IA em entender linguagem, e o que podemos aprender com eles?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 3: {question}\n")
print(f"Resposta: {resposta}\n")

# Teste 2
pdf_path = 'https://karpathy.github.io/2015/05/21/rnn-effectiveness/'
question = 'Qual é a principal vantagem das Redes Neurais Recorrentes (RNNs) em comparação com redes neurais tradicionais para tarefas de processamento de sequências?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 1: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'Como as variantes das RNNs, como LSTM e GRU, melhoram o desempenho em tarefas que envolvem dependências de longo prazo?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 2: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'Quais são alguns exemplos de aplicações práticas onde as RNNs demonstraram resultados surpreendentes, conforme discutido no artigo?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 3: {question}\n")
print(f"Resposta: {resposta}\n")

# Teste 3
pdf_path = 'lista_5/teste_3.pdf'
question = 'Por que as funções de ativação não lineares, como a ReLU ou a
```

```
    sigmoide, são essenciais para o funcionamento de redes neurais?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 1: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'O que torna o problema XOR um exemplo importante na compreensão das
    limitações dos perceptrons e da necessidade de redes neurais multicamadas?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 2: {question}\n")
print(f"Resposta: {resposta}\n")
question = 'Quais são as principais diferenças entre uma rede neural
    feedforward e uma rede neural recorrente (RNN)?'
resposta = answer_question(pdf_path, question)
print(f"Pergunta 3: {question}\n")
print(f"Resposta: {resposta}\n")
```