Artificial intelligence  
  
  
  
Homework-questions 2-3-4  
  
  
Dr.Asaeei

Arman Tayebi

تمرین:

با مسائل غیر قطعی چگونه رفتار میکنیم؟  
  
  
راه حل مسائل غیر قطعی در هوش مصنوعی مرتبط با مدیریت و تصمیم‌گیری در شرایطی که دارای عدم قطعیت هستند می‌باشد. برای حل این گونه مسائل، می‌توان از رویکردها و تکنیک‌های زیر استفاده کرد:

1. احتمالات و آمار: استفاده از مفاهیم احتمالات و آمار برای مدل‌سازی و پیش‌بینی وقوع رویدادها در شرایط عدم قطعیت.

2. مدل‌سازی بیزی: استفاده از مدل‌های بیزی برای نمایش علاقه‌مندی‌ها و توزیع‌های احتمالی در مسائل غیر قطعی.

3. تئوری تصمیم‌گیری: اعمال تکنیک‌های تصمیم‌گیری چون مدل‌های مارکوف تصمیم‌گیری (MDP) و فرآیندهای تصمیم‌گیری مارکوف (Markov Decision Processes) برای تعیین تصمیم‌های بهینه در شرایطی که دارای عدم قطعیت هستند.

4. اطلاعات فازی: استفاده از اطلاعات فازی برای مدل‌سازی عدم قطعیت و عدم دقت در داده‌ها و تصمیم‌گیری‌ها.

5. تکنیک‌های ترکیبی: ترکیب اطلاعات احتمالی و داده‌های مشاهده شده با دانش پیشین و تجربی به منظور بهبود تصمیم‌گیری در شرایط عدم قطعیت.

6. الگوریتم‌های بهینه‌سازی: استفاده از الگوریتم‌های بهینه‌سازی برای یافتن راه‌حل‌های بهینه در مسائل غیر قطعی.

7. تکنیک‌های تحلیل حساسیت: تجزیه و تحلیل حساسیت برای درک تأثیر پارامترها و عوامل مختلف بر نتایج تصمیم‌گیری در شرایط عدم قطعیت.

8. شبکه‌های عصبی: استفاده از شبکه‌های عصبی برای مدل‌سازی و پیش‌بینی در شرایط عدم قطعیت.

ترکیبی از این رویکردها و تکنیک‌ها بسته به مسئله مورد نظر و میزان عدم قطعیت می‌تواند به راه‌حل‌های موثری در مسائل غیر قطعی در هوش مصنوعی منجر شود.

تمرین 2:   
  
کد ۸ وزیر رو پیدا کنید و به عنوان تکلیف هم تو ‌گروه گذاشته بشه

def is\_safe(board, row, col, n):

# چک کردن آیا می‌توان وزیری را در سلول (row, col) قرار داد یا خیر

# چک کردن ردیف افقی (سمت چپ)

for i in range(col):

if board[row][i] == 1:

return False

# چک کردن قطر بالا به چپ

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

# چک کردن قطر پایین به چپ

for i, j in zip(range(row, n, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens\_util(board, col, n):

# حالت پایه: اگر تمام وزیرها قرار گرفته باشند

if col >= n:

return True

# برای هر سلول در ستون فعلی

for i in range(n):

# چک کردن آیا می‌توان وزیر را در این سلول قرار داد

if is\_safe(board, i, col, n):

# قرار دادن وزیر در این سلول

board[i][col] = 1

# ادامه به جستجوی ستون بعدی

if solve\_n\_queens\_util(board, col + 1, n):

return True

# اگر قرار گرفتن وزیر در این سلول به حل مسئله منجر نشود، آن را از صفحه حذف می‌کنیم

board[i][col] = 0

# اگر هیچ یک از سلول‌ها منجر به حل مسئله نشود

def solve\_n\_queens(n):

return False

# ایجاد صفحه شطرنج خالی

board = [[0 for \_ in range(n)] for \_ in range(n)]

# حل مسئله با فراخوانی اولیه از ستون اول

if not solve\_n\_queens\_util(board, 0, n):

print("هیچ راه حلی وجود ندارد.")

return False

# نمایش جواب

for i in range(n):

for j in range(n):

print(board[i][j], end=" ")

print()

return True

# تابع را فراخوانی می‌کنیم با n=8 برای حل مسئله 8 وزیر

solve\_n\_queens(8)

تمرین3:

با مسائل غیر قطعی چگونه رفتار میکنیم؟

بستگی به نوع مسئله و الگوریتمی که برای حل آن استفاده می‌شود دارد که پیچیدگی حافظه یا زمانی کدامیک بیشتر است. در برخی موارد، مسائلی وجود دارند که پیچیدگی حافظه آن‌ها بسیار بالاست و نیاز به استفاده از منابع حافظه بالا دارند. برای مثال، الگوریتم‌هایی که برای پردازش تصویر و صدا استفاده می‌شوند، به دلیل بزرگی حجم داده‌های ورودی نیاز به استفاده از حافظه بالا دارند.

در مقابل، در بسیاری از مسائل، پیچیدگی زمانی بیشتر از پیچیدگی حافظه است. به عنوان مثال، الگوریتم‌هایی که برای مرتب‌سازی اعداد استفاده می‌شوند، نیاز به حافظه کمتری دارند ولی زمان بیشتری برای اجرای آن‌ها لازم است.

بنابراین، برای انتخاب بهترین الگوریتم برای حل یک مسئله، باید به دو پیچیدگی حافظه و زمانی توجه کرد و الگوریتمی را انتخاب کرد که برای آن مسئله پیچیدگی کمتری داشته باشد.