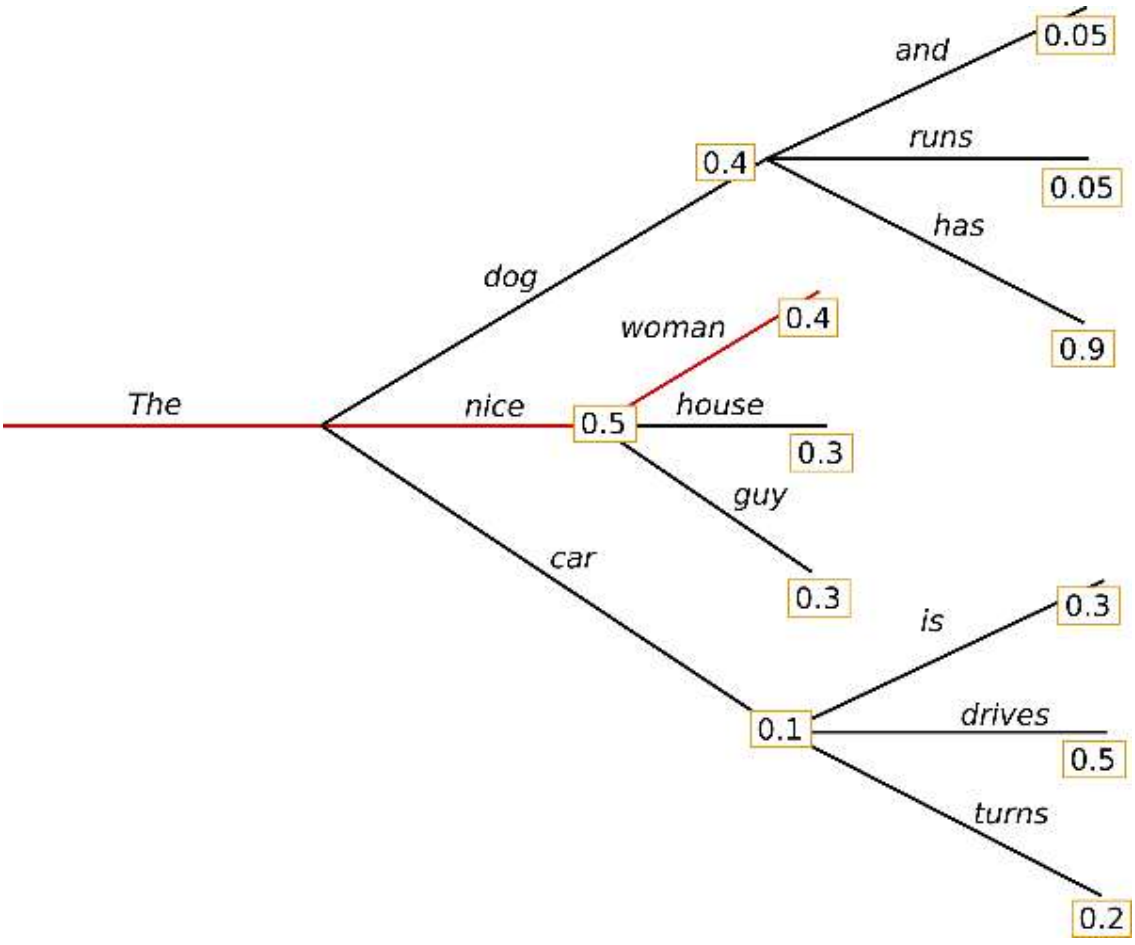
Nama saya Syahrul Apriansyah dengan NPM 2106708311 dari kelas IR izin untuk mengemukakan apa yang saya dapat dari setelah saya membaca referensi mengenai Beam Search yang ada di slice.

Berikut referensi yang saya gunakan:
- https://www.width.ai/post/what-is-beam-search
- https://huggingface.co/blog/how-to-generate

- https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

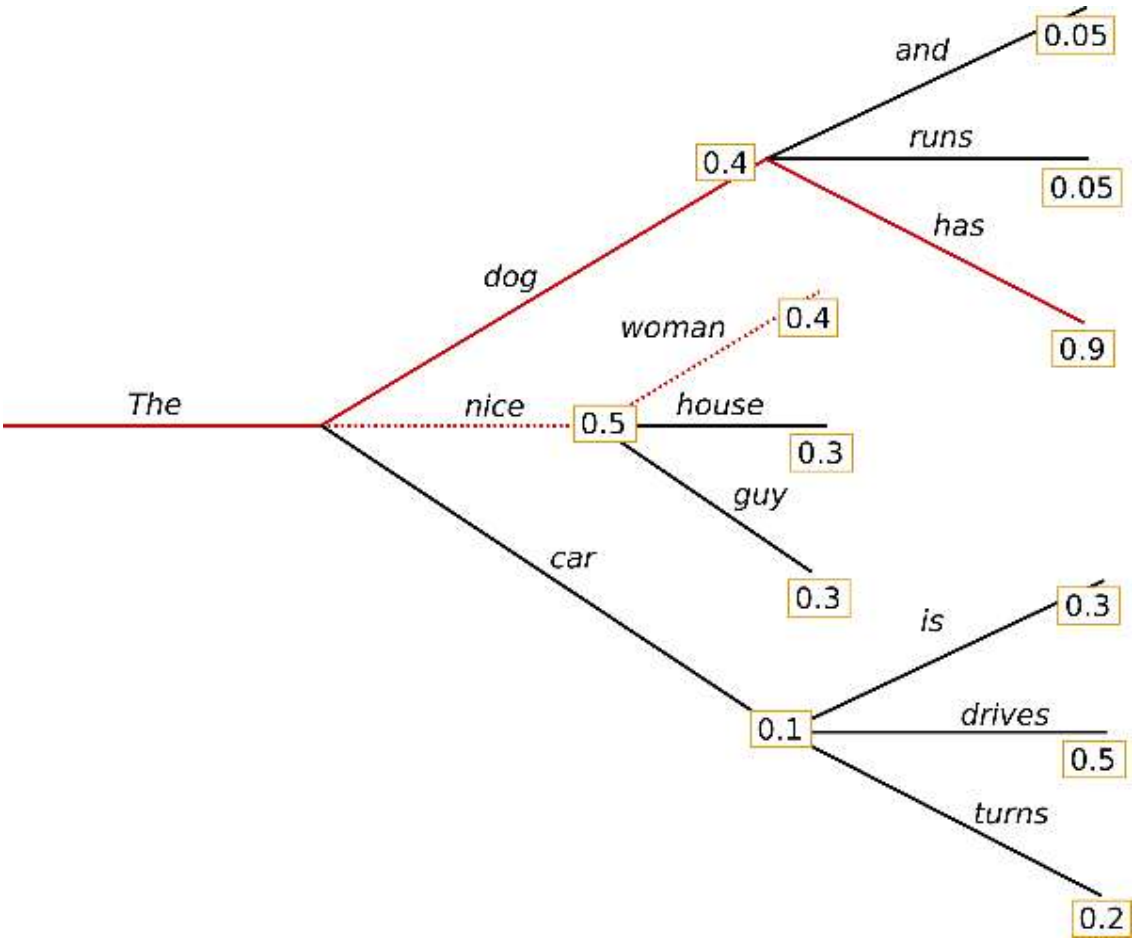
Artikel di width.ai membahas mengenai algoritma Beam Search yang merupakan metode penting dalam NLP dan speech recog. Algoritma ini dipilih karena kemampuannya untuk menghasilkan hasil yang lebih akurat dengan mempertimbangkan berbagai kemungkinan keluaran pada setiap tahap dan memilih kombinasi yang memiliki probabilitas tertinggi secara keseluruhan.

Dibandingkan dengan Greedy Search yang hanya memilih kata dengan probabilitas tertinggi pada setiap langkah tanpa memperhatikan konteks yang lebih luas, Beam Search menggunakan parameter yang disebut num_beams untuk mempertimbangkan beberapa hipotesis terbaik pada setiap langkahnya. Ini memungkinkan algoritma untuk menjaga opsi yang mungkin lebih baik di langkah-langkah berikutnya, daripada hanya memilih opsi terbaik secara instan pada satu langkah seperti yang dilakukan Greedy Search.



Greedy Search

Greedy search adalah metode decoding sederhana. metode ini memilih kata dengan probabilitas tertinggi sebagai kata berikutnya di setiap langkah waktu. Seperti contoh di atas di imulai dari kata "The", algoritma memilih kata berikutnya dengan probabilitas tertinggi "nice", dan seterusnya, sehingga sekuens kata akhir yang dihasilkan adalah "The nice woman" dengan probabilitas keseluruhan
0.5×0.4=0.2



Beam Search

Beam search mengurangi risiko melewatkan sequence kata dengan probabilitas tinggi yang tidak tertangkap pada dengan cara greedt dengan mempertahankan num_beams hipotesis yang paling mungkin di setiap timesteps dan pada akhirnya memilih hipotesis yang memiliki probabilitas keseluruhan tertinggi. Sebagai ilustrasi dengan num_beams=2:
Pada timesteps 1, selain hipotesis paling mungkin "The nice", beam search juga mempertahankan yang kedua paling mungkin "The dog".

Pada timesteps 2, beam search menemukan bahwa sequence kata "The dog has", dengan probabilitas lebih tinggi 0.36 dibandingkan dengan "The nice woman", yang memiliki 0.2

Namun ada walau hasilnya mungkin lebih bagus, output dari beam search masih sering menghasilkan pengulangan sequence kata yang sama. Salah satu solusi yang tersedia adalah memperkenalkan penalti n-gram, yang memastikan tidak ada n-gram yang muncul dua kali dengan secara manual mengatur probabilitas kata berikutnya yang dapat menciptakan n-gram yang sudah terlihat menjadi 0.

Berikut ini adalah kode sederhana untuk beam search:

```
def beam_search_decoder(data, num_beams):
  sequences = [[list(), 0.0]]
  for row in data:
    all_candidates = list()

    for i in range(len(sequences)):
      seq, score = sequences[i]

      for j in range(len(row)):
        candidate = [seq + [j], score - log(row[j])]
        all_candidates.append(candidate)

    # order all candidates by score
    ordered = sorted(all_candidates, key=lambda tup:tup[1])

    # select num_beams best
    sequences = ordered[:num_beams]

  return sequences
```

Berikut adalah penjelasan saya mengenai kode tersebut:

1. Fungsi beam_search_decoder menerima dua parameter: data, yang merupakan matriks probabilitas kata-kata dan num_beams, yang menentukan jumlah hipotesis terbaik yang akan dipertahankan setiap saat.
2. sequences diinisialisasi sebagai list yang berisi satu entri, yaitu list kosong untuk sequence kata-kata dan skor 0. Ini mewakili titik awal search.
3. Fungsi kemudian melakukan iterasi melalui setiap 'baris' dalam data. Setiap 'baris' mewakili probabilitas kata-kata pada suatu langkah waktu tertentu dalam sequence.
4. Untuk setiap sequence yang ada, fungsi melakukan iterasi lagi setiap elemen 'baris' (yaitu probabilitas setiap kata) dan membuat 'kandidat' sequence baru dengan menambahkan kata tersebut ke sequence yang ada dan mengurangi skor sequence dengan logaritma negatif dari probabilitas kata tersebut. Logaritma negatif digunakan karena probabilitas yang lebih tinggi (lebih dekat ke 1) harus memberikan skor yang lebih rendah (lebih baik).
5. Setelah itu, semua kandidat (dari semua sequence yang ada dan semua kata yang mungkin) diurutkan berdasarkan skornya, dari yang terbaik (skor terendah) ke yang terburuk (skor tertinggi).
6. Kemudian, num_beams kandidat teratas dipilih untuk menjadi sequence yang dipertahankan untuk iterasi berikutnya.
7. Setelah iterasi terakhir, fungsi mengembalikan num_beams sequence teratas sebagai output.

Berikut link code percobaan saya:
https://colab.research.google.com/drive/1EyiD6vK-DOChjmGUJMo38kmQcuxBL7Lr