HW7: Beam Search Decoding - News Headline Generation

In this exercise, you are going to learn and implement decoding techniques for sequence generation. Usually, the sequence is generated word-by-word from a model. In each step, the model predicted the most likely word based on the predicted words in previous steps (this is called auto-regressive decoding).

As such, it is very important how you decide on what to predicted at each step, as it will be conditioned on to predicted all of the following steps. We will implement two of main decoding techniques introduced in the lecture: **Greedy Decoding** and **Beam Search Decoding**. Greedy Decoding immediately chooses the word with best score at each step, while Beam Search Decoding focuses on the sequence that give the best score overall.

To complete this exercise, you will need to complete the methods for decoding for a text generation model trained on New York Times Comments and Headlines dataset. The model is trained to predict a headline for the news given seed text. You do not need to train any model model in this exercise as we provide both the pretrained model and dictionary.

Download model and vocab and setup

```
!wget -O vocab.txt https://www.dropbox.com/s/ht12ua9vpkep6l8/hw9_vocab.txt
!wget -O model.bin https://www.dropbox.com/s/okmri7cnd729rr5/hw9_model.bin
```

```
--2025-02-22 11:56:18--  https://www.dropbox.com/s/ht12ua9vpkep6l8/hw9_vocab.
Resolving www.dropbox.com (www.dropbox.com)... 162.125.85.18
Connecting to www.dropbox.com (www.dropbox.com)|162.125.85.18|:443... connect
HTTP request sent, awaiting response... 302 Found
Location: https://www.dropbox.com/scl/fi/zlkw3il9cj4c121vtrrxh/hw9_vocab.txt?
--2025-02-22 11:56:19--  https://www.dropbox.com/scl/fi/zlkw3il9cj4c121vtrrxh
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc307f312445e2f20cce032270c1.dl.dropboxusercontent.com/cd/0
--2025-02-22 11:56:19--  https://uc307f312445e2f20cce032270c1.dl.dropboxuserc
Resolving uc307f312445e2f20cce032270c1.dl.dropboxusercontent.com (uc307f31244
Connecting to uc307f312445e2f20cce032270c1.dl.dropboxusercontent.com (uc307f3
HTTP request sent, awaiting response... 200 OK
Length: 78729 (77K) [text/plain]
Saving to: ‘vocab.txt’

vocab.txt           100%[===================>]  76.88K   299KB/s    in 0.3s

2025-02-22 11:56:21 (299 KB/s) - ‘vocab.txt’ saved [78729/78729]

--2025-02-22 11:56:21--  https://www.dropbox.com/s/okmri7cnd729rr5/hw9_model.
Resolving www.dropbox.com (www.dropbox.com)... 162.125.85.18
```

```
Connecting to www.dropbox.com (www.dropbox.com)|162.125.85.18|:443... connect
HTTP request sent, awaiting response... 302 Found
Location: https://www.dropbox.com/scl/fi/es8o6240q6qogbulewk2s/hw9_model.bin?
--2025-02-22 11:56:22--  https://www.dropbox.com/scl/fi/es8o6240q6qogbulewk2s
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uca5258fac172e44c77f633e9f33.dl.dropboxusercontent.com/cd/0
--2025-02-22 11:56:22--  https://uca5258fac172e44c77f633e9f33.dl.dropboxuserc
Resolving uca5258fac172e44c77f633e9f33.dl.dropboxusercontent.com (uca5258fac1
Connecting to uca5258fac172e44c77f633e9f33.dl.dropboxusercontent.com (uca5258
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/Ckl2Xc_cxxdaBWVe5r_40p906bjel0CQwAxPe1_nq37b6MJxQyPOC
--2025-02-22 11:56:23--  https://uca5258fac172e44c77f633e9f33.dl.dropboxuserc
Reusing existing connection to uca5258fac172e44c77f633e9f33.dl.dropboxusercon
HTTP request sent, awaiting response... 200 OK
Length: 8690974 (8.3M) [application/octet-stream]
Saving to: ‘model.bin’

model.bin           100%[===================>]   8.29M  5.25MB/s    in 1.6s

2025-02-22 11:56:26 (5.25 MB/s) - ‘model.bin’ saved [8690974/8690974]
```

```python
import torch
import torch.nn as nn
from tokenizers import Tokenizer
from tokenizers.models import WordLevel
from tokenizers.pre_tokenizers import Whitespace
```

```python
class RNNmodel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, dropout_rate):

        super().__init__()
        self.embedding_dim = embedding_dim

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, 128, num_layers=2,
                     batch_first=True)
        self.dropout = nn.Dropout(dropout_rate)
        self.fc2 = nn.Linear(128, vocab_size)

    def forward(self, src):
        embedding = self.embedding(src)
        output,_ = self.rnn(embedding)
        output = self.dropout(output)
        prediction = self.fc2(output)
        return prediction
```

```python
with open("vocab.txt") as f:
  vocab_file = f.readlines()
embedding_dim = 64
dropout_rate = 0.2

model = RNNmodel(len(vocab_file), embedding_dim, dropout_rate)
```

```
model.load_state_dict(torch.load("model.bin",map_location='cpu'))
model.eval()
```

```
/var/folders/5g/160h3py942nb4pbbgy0dhtz40000gn/T/ipykernel_19190/3839689483.py
  model.load_state_dict(torch.load("model.bin",map_location='cpu'))
RNNmodel(
  (embedding): Embedding(10054, 64)
  (rnn): LSTM(64, 128, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=128, out_features=10054, bias=True)
)
```

```
vocab = [v.strip() for v in vocab_file]
vocab_size = len(vocab)
print(f"Vocab Size: {vocab_size}")
vocab[:10]
```

```
Vocab Size: 10054
['<unk>', '<pad>', '<eos>', 'the', 'a', 'to', 'of', 's', 'in', 'for']
```

```
stoi = { ch:i for i,ch in enumerate(vocab) }
tokenizer = Tokenizer(WordLevel(stoi, unk_token="<unk>"))
tokenizer.pre_tokenizer = Whitespace()
tokenized_text = tokenizer.encode("the a of to unknowns")
print(tokenized_text)
print(tokenized_text.ids)
print(tokenized_text.tokens)
print(tokenizer.decode(tokenized_text.ids))
```

```
Encoding(num_tokens=5, attributes=[ids, type_ids, tokens, offsets, attention_m
[3, 4, 6, 5, 0]
['the', 'a', 'of', 'to', '<unk>']
the a of to <unk>
```

1. TODO: Greedy decode

Normally, in sequence generation task, the model will continue generating tokens until an end-of-sequence symbol appear or the maximum length is reached. For this task:

- The end-of-sequence symbol is "< eos >" and its index is 2
- Use the maximum generation length of 15

```
eos_token = '<eos>'
max_gen_length = 15
```

```
model(torch.Tensor(tokenizer.encode('to encourage').ids).long()).shape
```

```
torch.Size([2, 10054])
```

```
tokenizer.encode('make me <eos>').tokens
```

```
['make', 'me', '<unk>', '<unk>', '<unk>']
```

```python
def greedy_decode(seed_text, tokenizer):
    """Greedy decodes with seed text.

        Args:
        seed_text: The seed string to be used as initial input to the model.
        tokenizer: The tokenizer for converting word to index and back.

        Your code should do the followings:
          1. Convert current_text to sequences of indices
          2. Predict the next token using the model and choose the token with
          3. Append the predicted index to current_text
          4. Loop until completion
          5. Return text prediction and a list of probabilities of each step

        You do not need to stop early when end-of-sequence token is generated
        until max_gen_length is reached. We can filter the eos token out later
    """
    current_indices = torch.Tensor(tokenizer.encode(seed_text).ids).long()
    probs = []
    for _ in range(max_gen_length):
        output = model(current_indices.unsqueeze(0))[0]
        output = torch.softmax(output,dim=-1)

        next_token = torch.argmax(output[-1]).item()
        probs.append(output[-1][next_token].item())

        current_indices = torch.cat((current_indices,torch.tensor([next_token]

        if next_token == 2:
            break

    output = tokenizer.decode(current_indices.tolist())
    return output, probs
```

```python
def clean_output(text, eos_token):
    """Drop eos_token and every words that follow"""
    return text.split(eos_token)[0]
```

```python
sample_seeds = ["to", "america", "people", "next", "picture", "on"]
for seed in sample_seeds:
    output, probs = greedy_decode(seed, tokenizer)
    output = clean_output(output, eos_token)
    print(f"Seed: {seed}\tOutput: {output}")
```

```
Seed: to        Output: to encourage creativity in the new york bill
Seed: america   Output: america s lethal export
Seed: people    Output: people to balloon to make a criminal with a dog with
Seed: next      Output: next phenom english clubs 2 call another deal in the
Seed: picture   Output: picture perfect chapter a spot of view of banning car
Seed: on        Output: on the catwalk in saudi arabia
```

Your output should be:

- to encourage creativity in the new york bill
- america s lethal export
- people to balloon to make a criminal with a dog with a callous rival
- next phenom english clubs 2 call another deal in the same arrivals
- picture perfect chapter a spot of view of banning care
- on the catwalk in saudi arabia

2. TODO: Beam search decode

Another well-known decoding method is beam search decoding that focuses more on the overall sequence score.

Instead of greedily choosing the token with the highest score for each step, beam search decoding expands all possible next tokens and keeps the **k** most likely sequence at each step, where **k** is a user-specified beam size. A sequence score is also calculated according user-specified cal_score() function. The beam with the highest score after the decoding process is done will be the output.

There are a few things that you need to know before implementing a beam search decoder:

- When the eos token is produced, you can stop expanding that beam
- However, the ended beams must be sorted together with active beams
- The decoding ends when every beams are either ended or reached the maximum length, but for this task, you can continue decoding until the max_gen_len is reached
- We usually work with probability in log scale to avoid numerical underflow. You should use np.log(score) before any calculation
- **As probabilities for some classes will be very small, you must add a very small value to the score before taking log e.g np.log(prob + 0.00000001)**

Sequence Score

The naive way to calculate the sequence score is to **multiply every token scores** together. However, doing so will make the decoder prefer shorter sequence as you multiply the sequence score with a value between [0,1] for every tokens in the sequence. Thus, we usually normalize the sequence score with its length by calculating its **geometric mean** instead.

You should do this in log scale

```
import numpy as np
```

```python
def cal_score(score_list: list[float], length: int, normalized: int=False): #c
    score_list = torch.tensor(score_list)
    score_list = torch.log(score_list + 1e-15)
    score = score_list.sum().item()
    if normalized:
        score /= length
    return score
```

```python
tokenizer.encode('to').ids + [1]
```

```
[5, 1]
```

```python
def beam_search_decode(seed_text, max_gen_len, tokenizer, beam_size=5, normali
    """We will do beam search decoing using seed text in this function.

    Output:
    beams: A list of top k beams after the decoding ended, each beam is a list
      [seed_text, list of scores, length]

    Your code should do the followings:
    1.Loop until max_gen_len is reached.
    2.During each step, loop thorugh each beam and use it to predict the next
      If a beam is already ended, continues without expanding.
    3.Sort all hypotheses according to cal_score().
    4.Keep top k hypotheses to be used at the next step.
    """
    # For each beam we will store (generated text, list of scores, and current
    # Add initial beam
    seed_index = tokenizer.encode(seed_text).ids
    beams = [[seed_index, [], 1, False]]
    for _ in range(max_gen_len):
      current_beams = []
      for beam in beams:
        # If beam is finished, we will not expand it
        if beam[-1]:
          current_beams.append(beam)
          continue
        # Generate next token
        current_indices = torch.tensor(beam[0])
        output = model(current_indices)
        probs = torch.softmax(output,dim=-1)[-1].tolist()
        # Expand beam
        for i in range(len(probs)):
          new_beam = [
            current_indices.tolist() + [i],
            beam[1] + [probs[i]],
            beam[2] + 1,
            i == 2
          ]
          current_beams.append(new_beam)

      beams = sorted(current_beams, key=lambda x: cal_score(x[1], x[2], normal

  for beam in beams:
   if beam[-1]:
    beam[0] = beam[0][:-1]
    beam[1] = beam[1][:-1]
    beam[2] -= 1
   # Convert indices back to text
   beam[0] = tokenizer.decode(beam[0])

  return [beam[:-1] for beam in beams]
```

3. Generate!

Generate 6 sentences based on the given seed texts.

Decode with the provided seed texts with beam_size 5. Compare the results between greedy, normalized, and unnormalized decoding.

Print the result using greedy decoding and top 2 results each using unnormalized and normalized decoing for each seed text.

Also, print scores of each candidate according to cal_score(). Use normalization for greedy decoding.

```
sample_seeds = ["to", "america", "people", "next", "picture", "on"]
for seed in sample_seeds:
  print('-Greedy-')
  output, probs = greedy_decode(seed, tokenizer)
  output = clean_output(output, eos_token)
  print(output, round(np.exp(cal_score(probs, len(probs), normalized=True)),

  print('-Unnormalized-')
  for output, probs, _ in beam_search_decode(seed, max_gen_length, tokenizer
    output = clean_output(output, eos_token)
    print(output, round(np.exp(cal_score(probs, len(probs))), 2))

  print('-Normalized-')
  for output, probs, _ in beam_search_decode(seed, max_gen_length, tokenizer
    output = clean_output(output, eos_token)
    print(output, round(np.exp(cal_score(probs, len(probs), normalized=Tru

  print()
```

```
-Greedy-
to encourage creativity in the new york bill  0.12
-Unnormalized-
to consult exploring recipes for new jersey 0.0
to consult exploring recipes up the pacific northwest 0.0
-Normalized-
to consult exploring recipes up the pacific northwest 0.17
to consult exploring recipes for new jersey 0.15

-Greedy-
america s lethal export  0.35
```

```
    -Unnormalized-
    america s lethal export 0.02
    america s desert aisles 0.01
    -Normalized-
    america s lethal export 0.25
    america s desert aisles 0.2

    -Greedy-
    people to balloon to make a criminal with a dog with a callous rival  0.16
    -Unnormalized-
    people to balloon for a criminal 0.0
    people to balloon for a criminal with trump 0.0
    -Normalized-
    people to balloon for a criminal with trump 0.13
    people to balloon for a criminal with a second fiddle 0.13

    -Greedy-
    next phenom english clubs 2 call another deal in the same arrivals  0.15
    -Unnormalized-
    next s blist revue 0.0
    next phenom english clubs 1 a chance to be back 0.0
    -Normalized-
    next s blist revue 0.14
    next phenom english clubs 1 a chance to be back 0.14

    -Greedy-
    picture perfect chapter a spot of view of banning care  0.09
    -Unnormalized-
    picture perfect use coffee 0.0
    picture korean a bonanza of pancakes 0.0
    -Normalized-
    picture korean a bonanza of contemplation times of donald trump 0.13
    picture korean a bonanza of contemplation times of trump s son 0.12

    -Greedy-
    on the catwalk in saudi arabia  0.25
    -Unnormalized-
    on the billboard chart 0.0
    on the catwalk in saudi arabia 0.0
    -Normalized-
    on the billboard chart 0.16
    on the whole30 diet vowing to eat smarter carbs to be insufficient 0.25
```

Your output should be:

```
-Greedy-
to encourage creativity in the new york bill  0.12
-Unnormalized-
To Consult Exploring Recipes For New Jersey 0.00
To Consult Exploring Recipes Up The Pacific Northwest 0.00
-Normalized-
To Consult Exploring Recipes Up The Pacific Northwest 0.17
To Consult Exploring Recipes Up The Least Of The Week 0.16
```

```
-Greedy-
america s lethal export  0.35
-Unnormalized-
America S Lethal Export 0.02
America S Desert Aisles 0.01
-Normalized-
America S Lethal Export 0.25
America S Desert Aisles 0.20

-Greedy-
people to balloon to make a criminal with a dog with a callous rival  0.16
-Unnormalized-
People To Balloon For A Criminal 0.00
People To Balloon For A Criminal With Trump 0.00
-Normalized-
People To Balloon For A Criminal With A Second Fiddle 0.13
People To Balloon For A Criminal With Trump 0.13

-Greedy-
next phenom english clubs 2 call another deal in the same arrivals  0.15
-Unnormalized-
Next S Blist Revue 0.00
Next Phenom English Clubs 1 A Chance To Be Back 0.00
-Normalized-
Next S Blist Revue 0.14
Next Phenom English Clubs 1 A Chance To Be Back 0.14

-Greedy-
picture perfect chapter a spot of view of banning care  0.09
-Unnormalized-
Picture Perfect Use Coffee 0.00
Picture Korean A Bonanza Of Pancakes 0.00
-Normalized-
Picture Korean A Bonanza Of Contemplation Times Of Trump S Son 0.12
Picture Korean A Bonanza Of Pancakes 0.07

-Greedy-
on the catwalk in saudi arabia  0.25
-Unnormalized-
On The Billboard Chart 0.00
On The Catwalk In Saudi Arabia 0.00
-Normalized-
On The Whole30 Diet Vowing To Eat Smarter Carbs To Be 0.27
On The Whole30 Diet Vowing To Eat Smarter Carbs For Because 0.26
```

Answer Questions in MyCourseVille!

Use the seed word "usa" to answer questions in MCV.