Text prediction/text generation

Now that was have established an understanding of how text embeddings work, we will use those embeddings to try to guess the next word in a sequence

In [ ]:
```python
from transformers import GPT2Tokenizer, AutoModelForCausalLM
import numpy as np

# Here we grab gpt2 tokenizer and model from the hub. You can also use your own model.
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
tokenizer.pad_token_id = tokenizer.eos_token_id


# Make the sequence we are trying to get the next token for. Feel free to change this to whatever you want and see wh
current_sequence = "The Boston Red"
inputs = tokenizer(current_sequence, return_tensors="pt")

# Here we ask the model what the next token should be. We get the top 5 items and probabilities.
num_to_gen = 5
outputs = model.generate(**inputs, max_new_tokens=1, return_dict_in_generate=True, output_scores=True,num_beams=num_t
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, normalize_logits=True,
)

print(f"Given Sequence:\n{current_sequence}\nWe have the following next token probabilities:")

# We can now look at the top 5 tokens and their probabilities.
for i in range(num_to_gen):
    this_sequence = outputs.sequences[i]
    last_token = this_sequence[-1]
    token_score = transition_scores[i].item()
    print (f"Token:{tokenizer.decode(last_token)} Probability: {np.exp(token_score):.2%}")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
Given Sequence:
The Boston Red
We have the following next token probabilities:
Token: Sox Probability: 97.25%
Token: Wings Probability: 0.90%
Token: Bulls Probability: 0.85%
Token: Cross Probability: 0.22%
Token: Wing Probability: 0.11%
```

Sequence Generation

Sequence generation is simply running the above procedure repeatedly to create more than 1 word

In [3]:
```python
import torch
from transformers import pipeline


def continue_sequence(input, model, temperature=0.7, top_k=50, top_p=0.95, max_new_tokens=50):
    """
    Generate new text based on the input sequence.
    """
    # We set up the pipeline to use the model and generate text
    # The torch dtype is set to bfloat16 for a smaller memory footprint
    text_continuation = pipeline("text-generation", model=model, do_sample=True, temperature=temperature, top_k=top_

    # Generate text with a maximum length of 40 tokens
    outputs = text_continuation(input, max_new_tokens = max_new_tokens)

    return outputs

# Here we are trying out a different model.
# There are many models at https://huggingface.co/models
# This is a smaller model that is more efficient to run on a CPU
model = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

# Generate new text based on this sequence
stub = "The Boston Red Sox are"

# Generate new text based on the input sequence
```

```
model_output = continue_sequence(stub, model)

print(f"Model Output: {model_output[0]['generated_text']}")
```

```
Device set to use mps:0
Model Output: The Boston Red Sox are a professional baseball team based in Boston, Massachusetts. They are currently
members of the American League East division. The Red Sox have won three World Series championships, in 1904, 1912, a
nd 20
```

Multiple candidate sequences

Since the models have some randomness in the next word they select, we can see that they generate different options based on the same output

In [4]:
```
# Reusing the model from above
model = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

# Feel free to change this text!
stub = "The Boston Red Sox are"

# Generate new text based on the input sequence
num_sequences_to_generate = 5
for i in range(num_sequences_to_generate):
    model_output = continue_sequence(stub, model)
    print(f"Model Output {i}: {model_output[0]['generated_text']}")
```

```
Device set to use mps:0
Model Output 0: The Boston Red Sox are taking on the Toronto Blue Jays in the American League Championship Series, an
d here's what you need to know about the teams and their matchups:

1. Boston Red Sox: Boston has a 3-1 lead in the
Device set to use mps:0
Model Output 1: The Boston Red Sox are 4-1 against the Los Angeles Angels in the regular season, and the teams split
their two postseason matchups. The Red Sox won the first two games of the series, but the Angels won the next three.
The Ang
Device set to use mps:0
Model Output 2: The Boston Red Sox are seeking to end their nine-year playoff drought and return to the World Series.
They've done it before, but it hasn't happened yet.
The team, which has won 28 National League East division titles,
Device set to use mps:0
```

```
Model Output 3: The Boston Red Sox are a professional baseball team based in Boston, Massachusetts. They play in the
Eastern Division of Major League Baseball's American League. The team was founded in 1901 and has won five World Seri
es championships, most recently in 20
Device set to use mps:0
Model Output 4: The Boston Red Sox are 3-1 on their current 10-game homestand and are still in first place in the Ame
rican League East. The Red Sox are just three games back of the Yankees for first place in the division.
The
```

Removing Randomness

We can also just generate the most likely items and remove this randomness. This is not generally advised but is instructuve

```
In [ ]: # Reusing the model from above
        model = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

        # Feel free to change this text!
        stub = "The Boston Red Sox are"

        # Generate new text based on the input sequence
        num_sequences_to_generate = 2
        for i in range(num_sequences_to_generate):
            # Here we are setting the top_k to 1 to get the most likely next token which removes the randomness and makes it
            model_output = continue_sequence(stub, model, top_k=1)
            print(f"Model Output {i}: {model_output[0]['generated_text']}")
```

```
Device set to use mps:0
Model Output 0: The Boston Red Sox are a professional baseball team based in Boston, Massachusetts. They are a member
club of the American League (AL) East division of Major League Baseball (MLB). The Red Sox have won 27 World Series c
hampionships, more than any other
Device set to use mps:0
Model Output 1: The Boston Red Sox are a professional baseball team based in Boston, Massachusetts. They are a member
club of the American League (AL) East division of Major League Baseball (MLB). The Red Sox have won 27 World Series c
hampionships, more than any other
```

Chatting with models

The sequence generation is interesting but let's set up an actual Q&A model like we see in popular product such as chat GPT

In [ ]:
```python
import torch
from transformers import pipeline

model = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

# Download the model and specify the task
# The torch dtype is set to bfloat16 for a smaller memory footprint
pipe = pipeline("text-generation", model=model, torch_dtype=torch.bfloat16)

# Set the system prompt and user message
system_prompt = "You are a friendly chatbot who always responds with extremely relevant information and provides conc
user_message = "How do I best use AI for education?"

messages = [
    {
        "role": "system",
        "content": system_prompt,
    },
    {
        "role": "user",
        "content": user_message,
    },
]

prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
outputs = pipe(prompt, max_new_tokens=400, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
print(outputs[0]["generated_text"])
```

```
Device set to use mps:0
```

```
<|system|>
You are a friendly chatbot who always responds with extremely relevant information and provides concise answers.</s>
<|user|>
How do I best use AI for education?</s>
<|assistant|>
AI has become an essential tool for education, providing various benefits and enhancing the quality and relevance of
learning experiences for students. Here are a few ways in which AI can be used for education:

1. Personalized Learning: AI-based tools can analyze student data and provide personalized learning experiences. For
instance, students can access content based on their learning style, progress, and interests.

2. Automated Assessments: AI-based tools can automate assessments, freeing up teachers' time for more meaningful task
s. Students can receive feedback on their performance immediately after the assessment.

3. Teacher Training: AI-based tools can provide teachers with data-driven insights on their students' learning. This
data can be used to improve teaching techniques, such as classroom management, differentiated instruction, and person
alized learning.

4. Assistive Technologies: AI-based tools can assist students with disabilities, such as visual and auditory impairme
nts. For example, assistive technology can help students with reading disabilities read more effectively.

5. Gamification: AI-based tools can be used to make learning more engaging and fun. For instance, games can be used t
o teach coding, math, and other subjects.

6. Personalized Learning Paths: AI-based tools can create personalized learning paths for students, based on their in
terests and learning styles. This approach can help students stay motivated and engaged in their learning.

Overall, AI can be a valuable tool for educators, helping them provide more effective and engaging learning experienc
es for students.
```

Augmenting based on user data

One neat trick that is frequently used to get more specific content out of a model is to add relevant information is to attach is to the system prompt.

For our example we will take the description from this session and add it to the prompt.

Sometimes there is a retreival step that is used to select the most relevant information from some corpus and attach that to the system propmt. That is called retrieval augmented generation and is the basis of a lot of tools. Here is a helpful Wikipedia entry on retrieval augmented generation (RAG) in case you want to read more.

In [8]:
```
import torch
from transformers import pipeline

model = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

# Download the model and specify the task
# The torch dtype is set to bfloat16 for a smaller memory footprint
pipe = pipeline("text-generation", model=model, torch_dtype=torch.bfloat16)

# This is the session description that we will add to the system prompt
session_description = "This session is for any data analyst who wants to build a foundation for machine learning and

# Set the system prompt and user message
system_prompt = "You are a friendly chatbot who always responds with extremely relevant information and provides conc
user_message = "What are the learning outcomes of this session?"

messages = [
    {
        "role": "system",
        "content": system_prompt,
    },
    {
        "role": "user",
        "content": user_message,
    },
]

prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
outputs = pipe(prompt, max_new_tokens=400, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
print(outputs[0]["generated_text"])
```

```
Device set to use mps:0
```

```
<|system|>
You are a friendly chatbot who always responds with extremely relevant information and provides concise answers. Use
the following session description to inform your responses: This session is for any data analyst who wants to build a
foundation for machine learning and AI using what they already know with traditional analytic methods. It provides a
broad overview and motivating examples across different tasks including analyzing textual data, predicting which of t
wo or more groups individuals will belong to in the future, and forecasting future metrics plus next word prediction
just like fancy generative AI models. Importantly, the information will be presented in a way to build a bridge betwe
en traditional analytic methods and artificial intelligence: spelling out what is common and what is unique, “transla
ting” AI jargon, and pointing out where AI methods are similar to other tried-and-true methods familiar to data analy
sts. For the areas covered, the session will show how the methods have evolved to reach their current state, what AI
is doing behind the scenes with the data, and how these methods can be useful when working with educational data. The
session will end with worked examples featuring simple code in Python, but no prior experience in Python is needed. A
lthough the session covers a lot of ground, attendees can work at their own pace thanks to instructions, code example
s, and other resources that they can return to after convening. Facilitators are data scientists with a combined 15 y
ears of industry experience working with real world data—primarily K-12 educational data—who want to prove that if yo
u have written code in any language to analyze quantitative data before, you are well situated to apply methods used
in machine learning and AI.  This is the session for data analysts who have wanted to try their hand at using AI in t
heir work but would like guidance in determining how, where, or when to start.</s>
<|user|>
What are the learning outcomes of this session?</s>
<|assistant|>
The learning outcomes of this session are:

1. Understand the basics of data analytics and machine learning, including how to use traditional analytic methods an
d AI approaches to solve real-world problems.

2. Appreciate the similarities and differences between traditional analytic methods and AI approaches in the context
of analyzing educational data.

3. Gain practical skills in data analysis using Python and apply them to solve real-world problems using AI technique
s.

4. Learn how to work with data in Python and use code to analyze and interpret data.

5. Understand the limitations and challenges of using AI techniques to solve problems with educational data, and how
to address them.

6. Learn how to build a foundation for machine learning and AI using traditional analytic methods.

7. Be able to apply these knowledge and skills to practical situations and solve real-world problems in the field of
education.
```

Bonus: How to implement RAG

We had some conversations after the session and it seems that interacting with data using RAG is an important ability that we would like to support. We have added the below code as a very minimal example of how RAG is used

In [11]:
```
from langchain_huggingface import HuggingFaceEmbeddings
import numpy as np

# Here we will get a sentence embedder to embed the text we want to retrieve
# Note: This is the same model we used for sentence embeddings in the embed text notebook
model_name = "sentence-transformers/all-mpnet-base-v2"
model_kwargs = {'device': 'cpu'}
encode_kwargs = {'normalize_embeddings': False}
embedding_model = HuggingFaceEmbeddings(
    model_name=model_name,
    model_kwargs=model_kwargs,
    encode_kwargs=encode_kwargs
)

# Here we will embed the text we want to retrieve
# If you want to embed different data, plug in your own text here
doc_embeddings = embedding_model.embed_documents(
    [
        "John Hosmer works as a data scientist at Panorama Education.",
        "Tara is great at making complicated models easy to understand.",
        "The Harvard Strategic Data Porject has a Convening that provides a lot of value for attendees"
    ]
)

# Here we will ask a question and embed it
# If you want to ask a different question, plug in your own text here
query_embedding = embedding_model.embed_query("Who is John Hosmer?")


# Here is our similariry metric, this is how we will measure the similarity between the query and the documents
def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

# Here we are again demonstrating embedding similarities as we are using it for RAG, let's compare our query embeddir
for i in range(len(doc_embeddings)):
    this_doc_embedding = doc_embeddings[i]
    similarity = cosine_similarity(query_embedding, this_doc_embedding)
    print(f"Similarity between query and document {i}: {similarity:.2f}")
```

```
Similarity between query and document 0: 0.65
Similarity between query and document 1: 0.11
Similarity between query and document 2: 0.13
```

Now we know how to get the most similar document to our query, let's use that to Augment the query and improve the results!

In [18]:
```
# Let's text what we would get out of our model if we didn't use RAG, this is very similar to the text generation exc

import torch
from transformers import pipeline

model = "Qwen/Qwen3-1.7B"

# Download the model and specify the task
# The torch dtype is set to bfloat16 for a smaller memory footprint
pipe = pipeline("text-generation", model=model, torch_dtype=torch.bfloat16)


# Set the system prompt and user message
system_prompt = "You are a friendly chatbot who always responds with extremely relevant information and provides conc
user_message = "Who is John Hosmer?"

print("User message: " + user_message)

messages = [
    {
        "role": "system",
        "content": system_prompt,
    },
    {
        "role": "user",
        "content": user_message,
    },
]

# Get the non-RAG output
prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
outputs = pipe(prompt, max_new_tokens=400, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
print("Non RAG output: " + outputs[0]["generated_text"])

# Now get the most relevant document from our list
# This is more copy paste from above but I'm trying to keep theses atomic
model_name = "sentence-transformers/all-mpnet-base-v2"
model_kwargs = {'device': 'cpu'}
encode_kwargs = {'normalize_embeddings': False}
embedding_model = HuggingFaceEmbeddings(
    model_name=model_name,
    model_kwargs=model_kwargs,
    encode_kwargs=encode_kwargs
)

# Here we will embed the text we want to retrieve
# If you want to embed different data, plug in your own text here
docs = [
        "John Hosmer works as a data scientist at Panorama Education who got a masters in Machine Learning from Georg
        "Tara is great at making complicated models easy to understand.",
        "The Harvard Strategic Data Porject has a Convening that provides a lot of value for attendees"
]

doc_embeddings = embedding_model.embed_documents(docs)
query_embedding = embedding_model.embed_query(user_message)

best_doc = None
best_similarity = -1
for i in range(len(doc_embeddings)):
    this_doc_embedding = doc_embeddings[i]
    similarity = cosine_similarity(query_embedding, this_doc_embedding)
    if similarity > best_similarity:
        best_similarity = similarity
        best_doc = docs[i]



# Set the system prompt and user message
system_prompt_RAG = "You are a friendly chatbot who always responds with extremely relevant information and provides
```

```
print("User message: " + user_message)

messages = [
    {
        "role": "system",
        "content": system_prompt_RAG,
    },
    {
        "role": "user",
        "content": user_message,
    },
]

# Get the non-RAG output
prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
outputs = pipe(prompt, max_new_tokens=400, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
print("Non RAG output: " + outputs[0]["generated_text"])
outputs
```

```
config.json:   0%|          | 0.00/726 [00:00<?, ?B/s]
model.safetensors.index.json:   0%|          | 0.00/25.6k [00:00<?, ?B/s]
Fetching 2 files:   0%|          | 0/2 [00:00<?, ?it/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP downloa
d. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP downloa
d. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
model-00001-of-00002.safetensors:   0%|          | 0.00/3.44G [00:00<?, ?B/s]
model-00002-of-00002.safetensors:   0%|          | 0.00/622M [00:00<?, ?B/s]
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
generation_config.json:   0%|          | 0.00/239 [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/9.68k [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/2.78M [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/1.67M [00:00<?, ?B/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP downloa
d. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
tokenizer.json:   0%|          | 0.00/11.4M [00:00<?, ?B/s]
Device set to use mps:0
```

```
User message: Who is John Hosmer?
Non RAG output: <|im_start|>system
You are a friendly chatbot who always responds with extremely relevant information and provides concise answers<|im_e
nd|>
<|im_start|>user
Who is John Hosmer?<|im_end|>
<|im_start|>assistant
<think>
Okay, the user is asking about John Hosmer. Let me think. First, I need to figure out who he is. I know that John Hos
mer is a name that might be associated with some historical figures or people in different fields.

Starting with the most common possibilities. There's John Hosmer, the American politician. He was a Republican from T
exas, served in the U.S. House of Representatives, and was known for his work in agriculture and rural development. H
e was also a member of the Texas Senate. His career spanned several decades, and he was a key figure in Texas politic
s.

Another possibility is John Hosmer from the UK. There's a John Hosmer who was a British soldier and politician. He se
rved in the British Army, was involved in the Napoleonic Wars, and later became a member of the British Parliament. H
e was known for his military service and his role in the British political landscape.

There's also a John Hosmer in the context of sports, maybe a player or coach. However, I don't recall a prominent fig
ure in sports with that name.

Additionally, there's a John Hosmer in the field of education or academia. I'm not sure about any specific contributi
ons in that area.

I should check if there's a John Hosmer in the entertainment industry, like a movie or TV actor. But I don't have inf
ormation on that.

Wait, the user might be referring to a specific person, but without more context, it's hard to tell. The answer shoul
d mention the possible identities and ask for more details if needed.

I should present the most likely candidates first, then note that there could be others. Also, mention that if the us
er has more context, like a specific field or time period, it could help narrow down the answer.
</think>

John Hosmer is a name associated with several individuals in different fields, but the most prominent figure is **Joh
n Hosmer (1849–1
User message: Who is John Hosmer?
Non RAG output: <|im_start|>system
You are a friendly chatbot who always responds with extremely relevant information and provides concise answers. Only
use the following information for your responses: John Hosmer works as a data scientist at Panorama Education who got
```

```
a masters in Machine Learning from Georgia Tech. He has a background in education and has worked with data for over 1
0 years. John Hosmer like to play disc golf and is a big fan of the Boston Red Sox.<|im_end|>
<|im_start|>user
Who is John Hosmer?<|im_end|>
<|im_start|>assistant
<think>
Okay, the user is asking who John Hosmer is. Let me start by recalling the information I have. John Hosmer works as a
data scientist at Panorama Education. He has a masters in Machine Learning from Georgia Tech. His background is in ed
ucation and he's worked with data for over 10 years. Also, he likes disc golf and the Boston Red Sox.

I need to present this information clearly. The user might be interested in his professional role, educational backgr
ound, experience, and personal interests. I should make sure to mention all these points in a concise manner. Avoid a
ny unnecessary details. Let me structure the answer step by step: first his job, then his education, his experience,
and his hobbies. Keep each point brief. Also, check if there's any missing info, but based on the given data, that's
all. Make sure the response is friendly and relevant.
</think>

John Hosmer is a data scientist at Panorama Education with a Master's degree in Machine Learning from Georgia Tech. H
e has over 10 years of experience in education and data analysis. Additionally, he enjoys disc golf and follows the B
oston Red Sox.
```

Out[18]:

```
[{'generated_text': "<|im_start|>system\nYou are a friendly chatbot who always responds with extremely relevant info
rmation and provides concise answers. Only use the following information for your responses: John Hosmer works as a
data scientist at Panorama Education who got a masters in Machine Learning from Georgia Tech. He has a background in
education and has worked with data for over 10 years. John Hosmer like to play disc golf and is a big fan of the Bos
ton Red Sox.<|im_end|>\n<|im_start|>user\nWho is John Hosmer?<|im_end|>\n<|im_start|>assistant\n<think>\nOkay, the u
ser is asking who John Hosmer is. Let me start by recalling the information I have. John Hosmer works as a data scie
ntist at Panorama Education. He has a masters in Machine Learning from Georgia Tech. His background is in education
and he's worked with data for over 10 years. Also, he likes disc golf and the Boston Red Sox.\n\nI need to present t
his information clearly. The user might be interested in his professional role, educational background, experience,
and personal interests. I should make sure to mention all these points in a concise manner. Avoid any unnecessary de
tails. Let me structure the answer step by step: first his job, then his education, his experience, and his hobbies.
Keep each point brief. Also, check if there's any missing info, but based on the given data, that's all. Make sure t
he response is friendly and relevant.\n</think>\n\nJohn Hosmer is a data scientist at Panorama Education with a Mast
er's degree in Machine Learning from Georgia Tech. He has over 10 years of experience in education and data analysi
s. Additionally, he enjoys disc golf and follows the Boston Red Sox."}]
```

In [ ]: