Lecture 7

Chatbots, RAG and Tuning

Lars A. Lochstoer

UCLA Anderson School of Management

Today

Chatbots and RAG

- General Chatbot structure
- RAG: Retrieval-Augmented Generation
- Querying your data: Building a simple RAG-Chatbot
- Command Line Interface (CLI) and User Interface (UI)

Fine-tuning

- Overview of tuning an LLM or Embedding Model
  - Evaluations from Chatbot users as “teacher”
- Tuning example: EmbeddingGemma-300M

Chatbots

Chatbot overview

(Obviously) Chatbots are important for many businesses

- Helps customer interface
- Automates call-lines, etc

Also useful for analysis

- Help analyze a database or set of documents

Example: asset manager needs performance evaluation for each investors' portfolio

Generic chatbot structure

User interface layer

- E.g., a chat window

NLP and dialog management layer

- Linking queries and appropriate responses, keeps context of dialog

Backend integration layer

- Linking to company database, booking systems, etc

Database logging layer

- Store conversation history, evaluation of answers, etc

Simplest, rule-based chatbot

We already discussed this in the context of embeddings:

1. Construct database of pre-written (scripted) responses
2. Embed all responses, embed user queries as they arise
3. Choose as output the pre-written answer that is the most related to the question
    - E.g., cosine similarity

Could also be used to route query to the right human customer support agent

Today: AI-powered chatbots

Uses embeddings plus LLM

- Generative AI so answers not scripted but adapts to conversation/dialog

Cheap and easy method: RAG

- Use existing LLM and work on prompt engineering to guide answers to be on-topic
- That is, we are working with the context window to improve answers
- RAG: Retrieval-Augmented Generation
- Combine with embedding and other NLP methods

More difficult, more bespoke

- Tune AI to be highly specialized for the specific content/tasks

RAG: motivation

LLMs are trained on broad set of data

Issues include:

- False information when it does not have answer
- Out-of-date or too generic, while typical chatbot case requires specifics
- Responses could come from web-scraped, non-authoritative sources
- Terminology confusion
  - Example: "Deadhead" refers to a different thing for an airline vs a concert venue

LLM arguably like an over-enthusiastic new employee who refuses to stay informed with current events but will always answer every question with absolute confidence.

- Not great for a chatbot

RAG: benefits

Cost-effective

- Foundational models (like Gemma, LLama, GPT) are very expensive to train
- With RAG we can introduce new data to the LLM

Current information

- LLMs trained on historical data, gets outdated
- Latest research and statistics can be connected

Enhanced user trust

- Can include source attribution
- Citations

RAG: overview of how it works

Create external data

- Files, database records, text, embeddings

Retrieve relevant information

- Embed query, search external data for relevant docs
- "how much annual leave do I have?" – system retrieves leave policy documents and employees' prior leave record

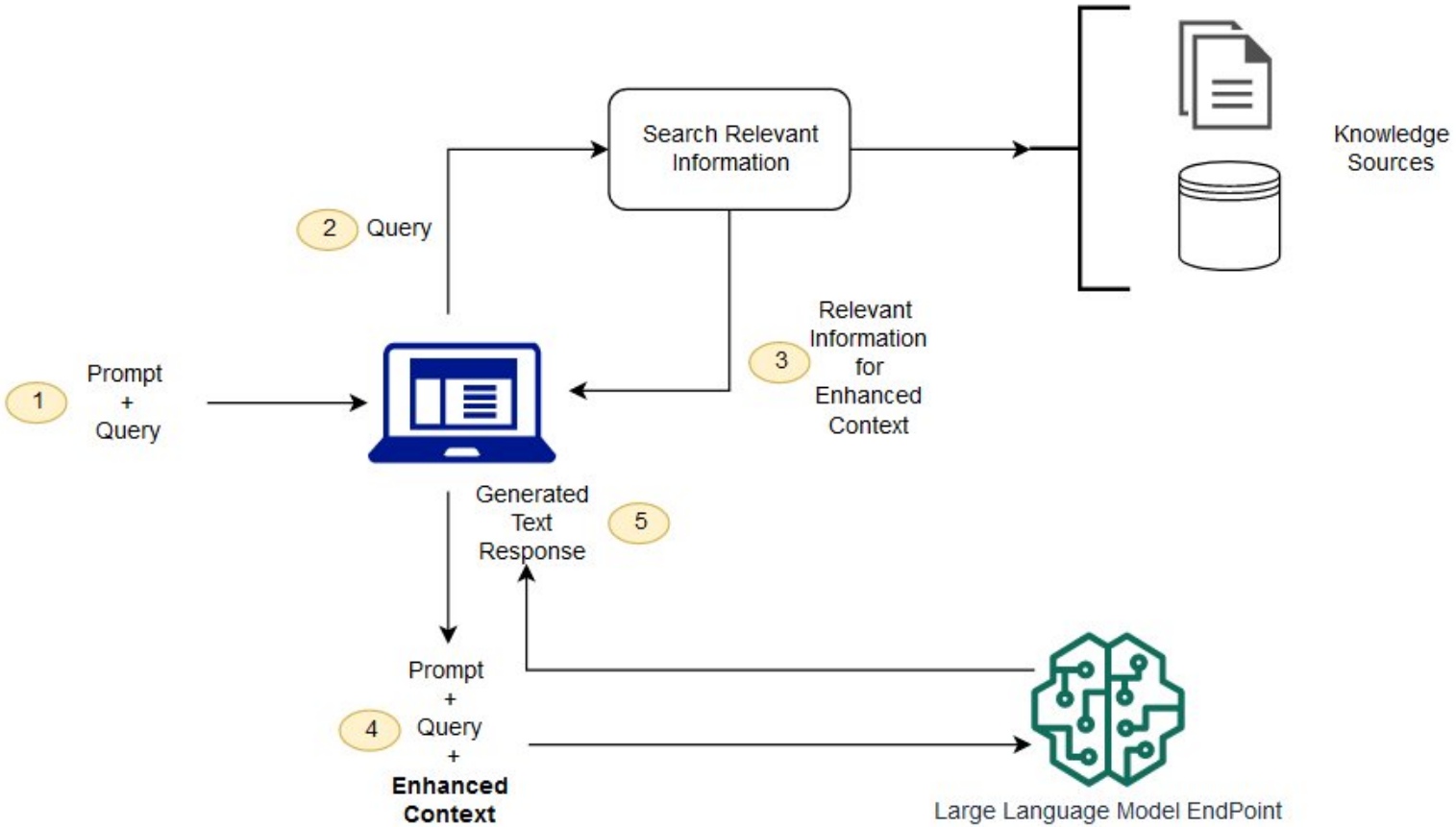
Augment LLM prompt

- Use the extracted relevant information for prompt engineering!

Update external data

- Automatic real-time, or batch processing

Conceptual from of using RAG with LLMs



Mini-case:

RAG-chatbot

Case: equity analyst assistant

"*Query your data!*"

Data: combined earnings calls and 10-K's

- Use the chatbot as a specialist research assistant with knowledge about these

Simple example, simple corpora

- No external APIs or internet references, but serves to illustrate main techniques and approaches
- Use most recent earnings calls and 10-K's for Apple and Dell
  - Obviously, you would have much bigger scale of project in reality

Steps

1. Chunking: splitting each document into lots of pieces
2. Embeddings: GemmaEmbeddings-300m
3. Retrieving relevant documents/chunks
4. Context generation strategy
5. Choice of LLM: Gemma 3 1b-it
6. Generate responses
7. Validation and citations

Chunking

LLMs have fixed context windows

- Whole docs won't fit and takes too long to analyze in scale

Overlap

- Overlapping chunks preserve sentence boundaries to allow verbatim quotes and avoid losing important context

In code

- Window $\approx$ 260 words with 90-word overlap (proxy for 200–400 tokens).
- Simple whitespace split for speed and transparency, all dashes made same '-'
- Result: overlapping chunks labeled (doc_id, chunk_id, text).

Chunking: trade-offs and tips

Smaller chunks $\longrightarrow$ higher precision but may lose cross-sentence context.

Larger chunks $\longrightarrow$ higher recall but waste context window and increase duplication.

Keep overlap 10–20% of window to bridge boundaries.

Embedding

We use EmbeddingGemma-300m to encode all chunks

- Row-normalize using L2 norm to enable cosine via dot product
- Recall:

$$\text{Cosine}\,(q, d) = \frac{q \cdot d}{\|q\|\,\|d\|}$$

Save the embedding matrix

- N chunks by 786 dimensional embedding vector
- Use this for cosine similarity to retrieve relevant chunks of documents to answer query
  - Save top 50 similar chunks to query

Since our corpora is very small, this method is sufficiently fast

- Use eg FAISS: Facebook AI Similarity Search for large embedding matrices

Alternative: lexical ranker

It is often useful to have an alternative similarity metric

- Lexical good for numeric phrases, tickers, expressions like "R&D" and "gross margin"
- Weaknesses: No synonym or semantic awareness, this comes from embedding side

Okapi BM25 lexical ranker

- Chuck d matches a query q by term frequency and document length normalization
- Look up formula if you like, but intuitive understanding is sufficient for now:
  - a chunks that contain similar words as the query is more relevant than a chunk that does not have the same words as query

Save top 50 similar chunks to query

Merged similarity score

We use the merged similarity score

$$\text{merged}\,(i) = \alpha \cdot \text{Cosine}\,(i) + (1-\alpha) \cdot BM25\,(i)$$

- α governs which of the two we weight more
- Can be set so as to deliver best responses in a set of test questions

We rescale each of the two scores so they have the same "size" (between 0 and 1) and thus a fair comparison using a min-max scaler

- For a vector of scores $s = (s_1, ..., s_m)$ from one scorer:

$$s^{norm} = \frac{s_i - \min_j s_j}{\max_j s_j - \min_j s_j + \varepsilon}, \text{ if } \max_j s_j > \min_j s_j; \text{ zero otherwise}$$

- Thus, the final similarity score is actually

$$\text{merged}_i = \alpha \cdot s_i^{\text{cosnorm}} + (1-\alpha) \cdot s_i^{\text{bm25norm}}$$

- If a chunk is not in the top 50 of cosine, its $s_i^{\text{cosnorm}} = 0$, and vice versa for BM25

Anchors and Aliases

Anchors:

We have used anchors before, forcing certain words to be present in the answer

- Get these from the query
- That is, there has to be some literal overlap with the words in the query in the chunk.
  - This is best when questions are likely to be technical in nature (e.g., what is the guidance for gross margins?)

Aliases:

Sometimes certain concepts/words and abbreviated, acronym'ed, etc.

- Particularly true for the finance lingo
- It can be helpful to create aliases such as
  - r&d: randd, rd, research and development
  - fy2025: fiscal 2025, fiscal year 2025
  - capex: capital expenditures
- Changing these to the same tokens help with matching anchors to the right concepts

Maximal Marginal Relevance (MMR)

Iterative re-ranking that balances relevance with diversity to avoid redundant chunks

- First pick grabs the best matched (merged_i)
- Subsequent picks must be **good** and **different**
  - the redundancy penalty grows if a candidate is very similar to any already chosen item
  - spreads selections across distinct sentences/regions that still align with the query

MMR-score used in chunk retrieval

$$\lambda \cdot \text{merged}_j - (1-\lambda) \cdot \max_{s \in S} \text{cosine}\,(j, s)\,, \quad \lambda \in [0,1]$$

-
  - $\lambda = 1$: pure relevance, no diversity
  - $\lambda = 0$: farthest-first traversal, max diversity

Neighbor expansion and Top-k selection

The chunking can artificially cut nearby sentences that would be relevant information for the LLM

- This can happen even though we use overlap

After the MMR selection

- expand the quoted sentence by adding neighboring sentences
- Then send the original chunk plus the neighboring (one before, one after) the the LLM

In the end, we set k

- Send top k chunks from the above procedure to LLM
- including the neighboring sentences for each of the top k

Context gate

Purpose: before you ever call the LLM, the gate decides whether the retrieved context is on-topic enough to justify prompting. If the gate fails, you immediately return JSON: {"answer":"Not in corpus","citations":[]}

The gate requires both:
(A) a local lexical overlap test on the chosen context ctx, and
(B) either a semantic score or a global lexical score to be high

For (A), drop stopwords and verify that the chunks that passed prior filters contains at least two tokens that are the same as in query

For (B) set numerical minimum thresholds for cosine and BM25 for a chunk to be included in prompt

Build the prompt: system message

- **Grounding**: "Answer ONLY using the CONTEXT."
- **Output shape**: "Return exactly one JSON object with keys answer and citations."
- **Envelope**: "Wrap output between <JSON> and </JSON>; no preface, no code fences."
- **Conciseness**: answer $\leq$ 35 words.
- **Citations budget**: citations $\leq$ 2.
- **Verbatim requirement**: each citations[i].quote must be an exact substring of the provided CONTEXT and $\leq$ 160 chars.
- **Fallback**: if the context does not contain the query's keywords, the model must output {"answer":"Not in corpus","citations":[]}.

These constraints reduce drift, keep answers short, and make citation validation mechanical.

Build the prompt: user message (payload)

The user message concatenates:

- **QUESTION**: the raw question text.
- **CONTEXT**: the sequence of chunk blocks from step 2, separated by blank lines.
- **JSON SCHEMA**: a literal JSON Schema for the expected object (with required fields and additionalProperties: false). We tell the model not to repeat the schema; it is there to bias structure.
- **OUTPUT FORMAT**: the literal line <JSON>{...single JSON object...}</JSON>, which demonstrates the required envelope

Call the tokenizer's apply_chat_template(messages, add_generation_prompt=True) to convert the pair of messages (system, user) into the model's preferred prompt format. This keeps compatibility with Gemma 3 chat formatting.

Build the prompt: biasing the model

We want a JSON object output.

Append end of prompt with

\n<JSON>{

This strongly nudges the model to complete a single well-formed JSON object and then close with </JSON>. After generation we prepend the missing { back to the model's text (since it completed after the open brace) and trim at the first </JSON> if present.

Call LLM with greedy (temperature = 0) decoding by setting do_sample = FALSE.

Post-processing, validation, and citation

- **Parsing**: extract_json(...) first looks for a <JSON>{...}</JSON> block, then falls back to finding the largest balanced JSON object in the output text.
- **Repair path**: If parsing fails, we issue a tiny "fix" prompt that forces <JSON>{"answer":"Not in corpus","citations":[]}</JSON>.
- **Citation checks**: validate_citations(...) drops any citation whose quote is not an exact substring of the cited chunk or whose doc_id/chunk_id don't match the provided context. If the model supplied no valid quotes but the answer is supported, autocite(...) extracts ≤ 2 short, relevant quotes from the used context (numbers and keyword overlap are prioritized).
- **Guardrails**: If the retrieval gates deem the context off-topic (semantic + lexical thresholds), we skip prompting and directly return {"answer":"Not in corpus","citations":[]}.

Prompt skeleton: structure the LLM sees

```
SYSTEM:
Answer ONLY using the CONTEXT. Return exactly one JSON object ...
... constraints (≤35 words, ≤2 cites, quotes ≤160, exact substrings) ...
If not in CONTEXT, output Not in corpus with [].
Wrap answer in <JSON>...</JSON>. Escape quotes as \".

USER:
QUESTION:
<q>

CONTEXT:
[DOC=... | CHUNK=...]
<chunk_1 text>

[DOC=... | CHUNK=...]
<chunk_2 text>

... (up to TOP_K chunks)

JSON SCHEMA (do NOT repeat this in your output):
{ "type":"object", "properties":{ "answer":... "citations":[ ... ] }, ... }
```

```
OUTPUT FORMAT:
<JSON>{...single JSON object...}</JSON>

ASSISTANT (generation begins after appending):
<JSON>{
```

CLI and UI

CLI: Command-Line Interface

- What we will be using
- Simple Q and A, where user writes in Q

UI: User interface

- Easy to generate more fancy layout, like a chat window
- Streamlit and Gradio are possibilities
- Lots of options

Code in mini_ChatBot_code.py on BruinLearn

Chatbot output (try yourself)

Device set to use cpu
RAG chatbot ready. Type a question, or 'exit' to quit.
Q> What gross-margin range did Apple guide for the December quarter on the Oct 31, 2024 call? Quote the exact range.
Answer: Apple guided for the December quarter with a gross margin range of 46% to 47%.
Citations:
- APPL_CALL_20241031 | 20: "similar to what we reported in the fiscal year 2024. We expect gross margin to be between 46% and 47%. We expect OpEx to be between $15.3 billion and $15.5 bill"
- APPL_CALL_20241031 | 28: "They are really, for our company, record levels of gross margin and obviously guiding to 46% to 47% for the December quarter with all the new technologies that"

Chatbot output (try yourself)

Q> What is Apple's revenue in the September quarter, and what is the increase since last year?
Answer: Apple's September quarter revenue was $94.9 billion, up 6% from a year ago. iPhone revenue was $46.2 billion, up 6% from a year ago, and services revenue was up 12% year-over-year.
Citations:
- APPL_CALL_20241031 | 1: "fternoon, everyone, and thanks for joining the call. Today, Apple is reporting revenue of $94.9 billion, a September quarter record and up 6% from a year ago. i"
- APPL_CALL_20241031 | 4: "eginning with iPhone. iPhone revenues set a September quarter record of $46.2 billion, up 6% from a year ago with growth in every geographic segment. With the i"

Chatbot output (try yourself)

Q> Report Dell's FY2025 R&D expenses per the 10-K. Quote the sentence with the numbers.
Answer: Dell Technologies reported $3.1 billion in R&D expenses in fiscal 2025, representing $2.8 billion for fiscal 2024 and $20943 million for fiscal 2023.
Citations:
- DELL_10K_2025 | 20: "of our strategic partnerships. We have a global R&D presence, with total R&D expenses of $3.1 billion for Fiscal 2025, and $2.8 billion for both Fiscal 2024 and"
- DELL_10K_2025 | 192: "% $ 13,766 15.6 % (7) % $ 14,790 14.5 % Fiscal 2025 compared to Fiscal 2024 During Fiscal 2025, total operating expenses decreased 4%, due to a decline in selli"

Chatbot output (try yourself)

Q> What guidance or commentary did Dell give on AI server demand on the Feb 27, 2025 call? Quote the exact sentence.
Answer: Dell is projecting significant growth in AI server demand, citing the need to exceed 1x revenue, and anticipates a substantial increase in the $15 billion range.
Citations:
- DELL_CALL_20250227 | 40: "about cash. I do expect it to be greater than 1x, and so yes. And then to your second question, I probably won't give you as much detail as you like because we think some of the changes we're making a"
- DELL_CALL_20250227 | 49: "Our sales force is incented to attach storage with AI opportunities. They will continue to be incented. We inspect that, and we continue to see progress in that area. And our next question will come f"

Chatbot output (try yourself)

Q> What is Microsoft's earnings in 2025?
Answer: Not in corpus
Citations: []

Chatbot output (try yourself)

Q> What was Taylor Swift's first single?
Answer: Not in corpus
Citations:
- APPL_CALL_20241031 | 21: "may we have the first question, please?"
- APPL_CALL_20241031 | 6: "With AirPods 4, we've broken new ground in comfort and design with our best ever open-ear headphones available for the first time with active noise cancellation"

Chatbot: further calibration

Output is not always good

- Sometimes get "not in corpus" when it is
- Sometimes get answers that are not entirely on topic
- Output is slow: need better indexing of embeddings (e.g., FAISS)

Play with all chatbot parameters/models, for instance:

TOP_K = 10 # number of context chunks provided to the LLM
ALPHA = 0.55 # hybrid weight (vector vs BM25)
MMR_LAMBDA = 0.5 # diversity in selection (MMR)
EMB_MODEL = "google/embeddinggemma-300m"
GEN_MODEL = "google/gemma-3-1b-it"
MAX_TOKENS = 384 # space for JSON completion

Maybe embedding needs to be fine-tuned for our application?

Fine-tuning of LLM/Embeddings

Fine-tuning: motivation

Chatbots are specialized to particular tasks

- Same with Agentic AI (AI-based chatbots, essentially)

Embedding model and LLM trained to be general

- We used extensive prompt engineering to guide the LLM

Fine-tuning involves actually changing the parameters of the model

- For this we need a "teacher" dataset
- That is, we need lots of example questions and answers (ranked), that the model can learn from
- These questions and answers would be specialized to be representative of the task at hand

Fine-tuning: Prepare fine-tuning dataset

- Create a dataset that teaches the model what "similar" means in your specific context
- This data is often structured as triplets: (anchor, positive, negative):
  - Anchor: The original query or sentence.
  - Positive: A sentence that is semantically very similar or identical to the anchor.
  - Negative: A sentence that is on a related topic but semantically distinct.

Code on next slide

- Obviously, you will need a way bigger training dataset than the three examples on the next slide

Fine-tuning: Prepare fine-tuning dataset

```
from datasets import Dataset
dataset = [
["How do I open a NISA account?", "What is the procedure for starting a new tax-free investment account?", "I want to check the balance of my regular savings account."],
["Are there fees for making an early repayment on a home loan?", "If I pay back my house loan early, will there be any costs?", "What is the management fee for this investment trust?"],
["What is the coverage for medical insurance?", "Tell me about the benefits of the health insurance plan.", "What is the cancellation policy for my life insurance?"],
]
# Convert the list-based dataset into a list of dictionaries.
data_as_dicts = [ {"anchor": row[0], "positive": row[1], "negative": row[2]} for row in dataset ]
# Create a Hugging Face 'Dataset' object from the list of dictionaries.
train_dataset = Dataset.from_list(data_as_dicts)
```

Fine-tuning: Before fine-tuning values

Let's say you search for "tax-free investment"

Before fine-tuning, using EmbeddingGemma we may have gotten:

1. Document: Opening a NISA account (Score: 0.51)
2. Document: Opening a Regular Saving Account (Score: 0.50) <- Similar score, potentially confusing
3. Document: Home Loan Application Guide (Score: 0.44)

Code on next slide

Fine-tuning: Before fine-tuning values

Get similarities using:

```
def get_scores(query, documents):
# Calculate embeddings by calling model.encode()
query_embeddings = model.encode(query, prompt_name=task_name)
doc_embeddings = model.encode(documents,
prompt_name=task_name)
# Calculate the embedding similarities
similarities = model.similarity(query_embeddings, doc_embeddings)
for idx, doc in enumerate(documents):
print("Document: ", doc, "-> Sim Score: ", similarities.numpy()[0][idx])
query = "I want to start a tax-free installment investment, what should I
do?"
documents = ["Opening a NISA Account", "Opening a Regular Savings
Account", "Home Loan Application Guide"]
get_scores(query, documents)
```

Fine-tuning: Training

SentenceTransformer library contains a training modules

```
from sentence_transformers import SentenceTransformerTrainer,
SentenceTransformerTrainingArguments
from sentence_transformers.losses import MultipleNegativesRankingLoss
from transformers import TrainerCallback
loss = MultipleNegativesRankingLoss(model)
args = SentenceTransformerTrainingArguments(
    # Required parameter:
    output_dir="my-embedding-gemma",
    # Optional training parameters:
    prompts=model.prompts[task_name], # use model's prompt to train
    num_train_epochs=5,
    per_device_train_batch_size=1,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    # Optional tracking/debugging parameters:
    logging_steps=train_dataset.num_rows,
    report_to="none",
)
```

Fine-tuning: Training

```
class MyCallback(TrainerCallback):
"A callback that evaluates the model at the end of eopch"
def __init__(self, evaluate):
self.evaluate = evaluate # evaluate function
def on_log(self, args, state, control, **kwargs):
# Evaluate the model using text generation
print(f"Step {state.global_step} finished. Running evaluation:")
self.evaluate()

def evaluate():
get_scores(query, documents)

trainer = SentenceTransformerTrainer(
model=model,
args=args,
train_dataset=train_dataset,
loss=loss,
callbacks=[MyCallback(evaluate)]
)

trainer.train()
```

Fine-tuning: After fine-tuning is done

Search again for “tax-free investment”

Answer could now be

1. Document: Opening a NISA account (Score: 0.73)
2. Document: Opening a Regular Saving Account (Score: 0.34)
3. Document: Home Loan Application Guide (Score: 0.48)

Perhaps upload your model to HuggingFace Hub:

```
# Push to Hub
model.push_to_hub("my-embedding-gemma")
```

Midterm

1. State-space and HMM models.
2. Hidden layers as function approximators. Feed-forward networks, activation functions, and estimation (batch, backpropagation, etc.)
3. Recurrent Neural Nets and the GRU (not LSTM)
4. Embeddings: DTM, sparse embeddings, dense embeddings
5. Transformers: structure, context and contextual embeddings, cross-entropy, multi-head
6. LLMs: Prompt engineering. How do LLMs work? How are they estimated? Word prediction, sampling, top-k, temperature, context windows

Read the slides, homework answers, and supplement with pdf's of books and other materials I posted on BruinLearn

The exam will be 2hrs on Nov 18 in class, only pen and paper allowed, plus a one letter-sized cheat sheet (you can write on both pages)