

CSC-496/696: Natural Language Processing and Text as Data

Lecture 16: Large Language Models

Patrick Wu

Tuesday, October 29, 2024

Lecture Contents

1. Announcements
2. Review of Self-Attention and Transformers
3. Large Language Models
4. Training a Large Language Model
5. Tokenization
6. Fine-Tuning LLMs
7. Ethical Considerations and Potential Harms with LLMs

Announcements

Assignment 3

- Due today at 11:59pm
- Assignment 4 will be released later today, too
- Remember, you have late days—you won't get extra credit for holding onto them. Let me know if you need an update on how many late days you have left.

Final Project

- I again encourage you to come to me early to talk about potential project ideas
- If you are doing a senior project or doing a capstone, you can use that project for this class *if* that project involves NLP
 - Or you can add an NLP dimension to that project

Review of Self-Attention and Transformers

Self-Attention

- Language is not something that is *relationally* left to right
- The English language is read left to right, but words relate to each other in both directions

Self-Attention (Simplified)

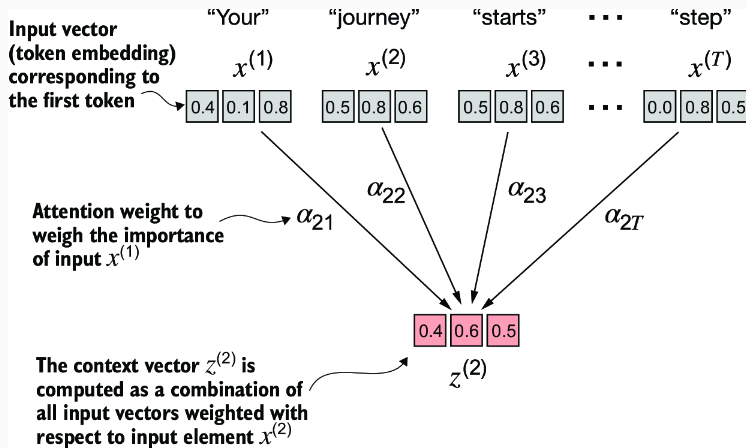


Figure from Raschka (2024). $z^{(2)}$ is calculated using attention weights with respect to input $x^{(2)}$ and all other inputs.

Self-Attention (Simplified)

Specifically (in this simplified version of self-attention),

$$z^{(2)} = \sum_{t=1}^T \alpha_{2t} x^{(t)}$$

where

$$\alpha_{2t} = \text{softmax}(\text{score}(x^{(2)}, x^{(t)}))$$

for all $t \in \{1, \dots, T\}$, and

$$\text{score}(x^{(2)}, x^{(t)}) = x^{(2)} \cdot x^{(t)}$$

Example of Calculating A Transformed Output

Let's say we have the sentence, "Dodgers win!"

Let

$$\overrightarrow{\text{Dodgers}} = x^{(1)} = \begin{bmatrix} 0.2 & 0.4 \end{bmatrix}$$

$$\overrightarrow{\text{win}} = x^{(2)} = \begin{bmatrix} 1.0 & 0.5 \end{bmatrix}$$

Let's say that we want to calculate $z^{(1)}$.

Example of Calculating a Transformed Output

Let's first calculate the dot-product scores:

$$\text{score}(x^{(1)}, x^{(1)}) = 0.2 \times 0.2 + 0.4 \times 0.4 = 0.2$$

$$\text{score}(x^{(1)}, x^{(2)}) = 0.2 \times 1.0 + 0.4 \times 0.5 = 0.4$$

Then,

$$\alpha = \begin{bmatrix} \frac{\exp(0.2)}{\exp(0.2) + \exp(0.4)} & \frac{\exp(0.4)}{\exp(0.2) + \exp(0.4)} \end{bmatrix} = \begin{bmatrix} 0.45 & 0.55 \end{bmatrix}$$

Example of Calculating a Transformed Output

Now that we have our attention weights, we can calculate $z^{(1)}$:

$$\begin{aligned} z^{(1)} &= 0.45 \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} + 0.55 \begin{bmatrix} 1.0 & 0.5 \end{bmatrix} \\ &= \begin{bmatrix} 0.09 & 0.18 \end{bmatrix} + \begin{bmatrix} 0.55 & 0.275 \end{bmatrix} \\ &= \begin{bmatrix} 0.64 & 0.455 \end{bmatrix} \end{aligned}$$

Self-Attention (Simplified)

- We end up with features $z^{(1)}, z^{(2)}, \dots, z^{(T)}$ that are transformed features of the input features
- Each transformed feature is a weighted sum of all the other features

Self-Attention: More Details

In self-attention, each input vector plays three roles:

- Query: the current element being compared to all other inputs
 - In the previous example, $x^{(2)}$ is the query
- Key: the input being compared to the query to determine the similarity weight
 - In the previous example, this is the x_t in $\text{score}(x^{(2)}, x^{(t)}) = x^{(2)} \cdot x^{(t)}$
- Value: the input that gets weighted and summed up to compute the output for the current element
 - In the previous example, recall that $z^{(2)} = \sum_{t=1}^T \alpha_{2t} x^{(t)}$. $x^{(t)}$ is called the value.

Self-Attention (Full Version)

Because each vector plays three roles, we can add three weight matrices that will dramatically increase the ability to capture even more nuanced patterns

$$q^{(i)} = x^{(i)} W^q$$

$$k^{(i)} = x^{(i)} W^k$$

$$v^{(i)} = x^{(i)} W^v$$

Eventually, we will learn each weight matrix through backpropagation.

Self-Attention (Full Version)

$$q^{(i)} = x^{(i)} W^q; k^{(i)} = x^{(i)} W^k; v^{(i)} = x^{(i)} W^v$$

$$\text{score}(x^{(i)}, x^{(t)}) = \frac{q_i \cdot k_t}{\sqrt{d_k}}$$

$$\alpha_{it} = \text{softmax}(\text{score}(x^{(i)}, x^{(t)})) \quad \forall t \in \{1, \dots, T\}$$

$$z^{(i)} = \sum_{t=1}^T \alpha_{it} x^{(t)}$$

Self-Attention (Full Version)

- Notice that we calculate the score by using the dot product, but also dividing by the square root of the key
- This is called **scaled dot-product attention**

Multi-head Self-Attention

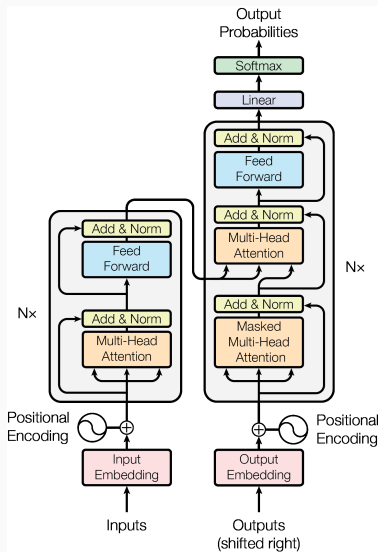
- In transformers, we can use multiple heads; we can have even greater diversity in representing the inputs because the weight matrices associated with the query, key, and value can all be different
- Intuition: multi-head attention allows each self-attention head to learn distinct patterns within the sentence
- Equations (9.14) to (9.19) in SLP Ch. 9 shows all the mathematical details of multi-headed attention

Unmasked vs. Masked Attention

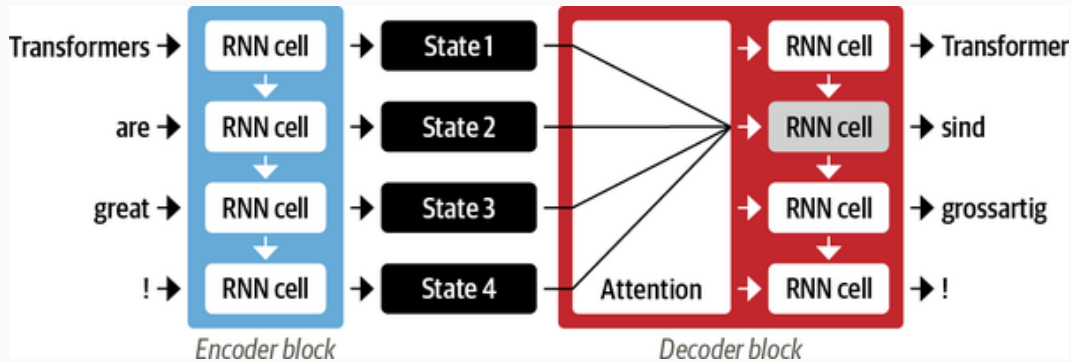
- In the above, we allowed each query to attend to all preceding *and* subsequent outputs
- But sometimes we don't want to do that—we want the model to only learn from the preceding input
- We can use masked attention, where the model is only allowed to attend to previous inputs
- Using our example, $z^{(2)}$ would only be calculated using $x^{(2)}$ and $x^{(1)}$

Any questions before we move on?

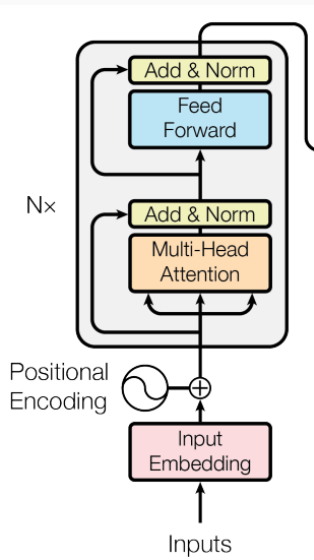
Transformers (Full Diagram)



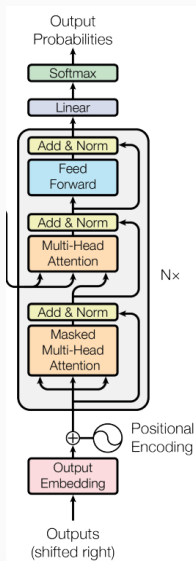
Encoder-Decoder Architecture of the Transformer



Transformer Encoder



Transformer Decoder



Decoder: Masked Multi-Head Self-Attention

- Ensures that the tokens we generate at each time step are only based on the past outputs and the current token being predicted
- We don't want the decoder to cheat by “peeking” ahead
- In other words, the current input can only attend to the previous inputs and itself, but not the subsequent inputs

Decoder: Encoder-Decoder Attention Layer

- Performs multi-head attention over the output values of the encoder stack, which act as the keys and values
- The intermediate representations of the decoder act as the queries
- That is, we will compare the intermediate representations of the decoder to all the outputs of the encoder
- This is the part of the decoder that relates the encoder output with the decoder input

Large Language Models

Large Language Models

- Modern large language models (LLMs) are all based on transformers
- There are encoder-only models (e.g., BERT), decoder-only models (e.g., GPT), and encoder-decoder models (e.g., T5)
- Decoder-only models are currently the center of attention at this moment, but all approaches dramatically improved benchmark results
- Large language models also allow us to *pretrain* a model on huge amounts of data, which then allows us to either use them directly for tasks of interest or we can *finetune* a model with a small amount of labeled data

Encoder Only Models

- These models convert an input sequence of text into a numerical representation that is well-suited for tasks like text classification and named entity recognition
- Uses full (or bidirectional) self-attention
- BERT was the first encoder-only model based on the transformer architecture; you can read the paper [here](#) (it has 116,305 citations!)

Encoder Only Models

- BERT
- DistilBERT
- RoBERTa
- XLM
- XLM-RoBERTa
- ALBERT
- ELECTRA
- DeBERTa

Decoder Only Models

- GPT
- CTRL
- GPT-NEO/GPT-J
- Claude
- Gemini
- Llama
- Mistral

Encoder-Decoder Models

- T5
- BART
- M2M-100
- BigBird

Training a Large Language Model

Teaching a Transformer

- One of the key advantages is that we don't have to train a transformer from scratch each time we use it
- For example, when you use ChatGPT, we don't need to re-train the model each time before we use it
- It already has *embedded knowledge* in these models

Teaching a Transformer

- We use a **self-supervised** algorithm to train a large language model
- We take a corpus of text as training material, and at each time step t , we ask the model to predict the next word
- That is, we might ask the model to predict the next word: “A journey of a thousand miles starts with one”
- This is called self-supervised because we don't need any gold standard labels

Teaching a Transformer

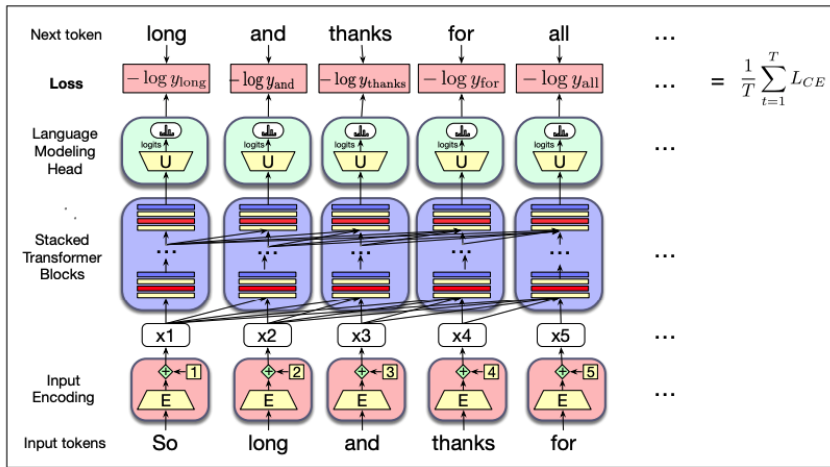


Figure from SLP, Ch. 10.

Teaching a Transformer

- Notice that we always feed into the model the correct sequence of words, rather than using as previous history the model's predictions
- This is known as *teacher forcing*

Teaching a Transformer

- During training, the probability assigned to the correct word is used as the loss function
 - We want this probability to be high
- The loss for a training sequence is the average cross-entropy loss function over the entire sequence
- Can then adjust the weights of the network to minimize this average cross-entropy loss via gradient descent

Corpora Used for Training

- LLMs are typically trained using text scraped from the internet
- Common Crawl: a collection of web text
- Colossal Clean Crawled Corpus (C4): a corpus of English internet text that is filtered in various ways, such as non-natural language removed, sentences with offensive words, etc.
- Other corpora include the Pile (Wikipedia and other academic pages) and Dolma (which includes social media)

Issues Around Crawling

- The need for these models to use massive amounts of data has triggered discussions around the ethics of using this kind of data
- Usually, this data is filtered and cleaned in some way, such as removing personal identifying information (PII)
- Toxicity detection models are also often used to remove offensive material
 - But this can be problematic—who gets to define toxicity? What about false positives?
- Fair use and copyright: in some countries, the fair use doctrine may allow copyrighted material to be used for “transformative” uses. But does an LLM count?

Other Training Tasks

- Predicting the next word is a good task for a decoder, but it is not a good training task for an encoder
- Recall that an encoder uses bidirectional self-attention
- If the attention mechanism can attend to both tokens before and after a word, the next word prediction task is moot
- Instead, models such as BERT are trained on other training tasks

Training Tasks for Encoder-Only Models

- Masked language modeling (MLM): predicts a masked word in a sentence
- Next sentence prediction (NSP): give the model two sentences, and ask it to predict whether the two sentences belong together in the right order

Training Tasks for Encoder-Only Models

- Masked language modeling (MLM): predicts a masked word in a sentence
- Next sentence prediction (NSP): give the model two sentences, and ask it to predict whether the two sentences belong together in the right order
- NSP has not been used as much, because the paper on the RoBERTa model (Robustly Optimized BERT Pretraining Approach) showed that NSP was not very useful

Training Tasks for Encoder-Only Models

- Masked language modeling (MLM): predicts a masked word in a sentence
- Next sentence prediction (NSP): give the model two sentences, and ask it to predict whether the two sentences belong together in the right order
- NSP has not been used as much, because the paper on the RoBERTa model (Robustly Optimized BERT Pretraining Approach) showed that NSP was not very useful
- Common theme in NLP research: lots of engineering and experimentation!

Pausing

Any questions before we move on?

Tokenization

Tokenization

- Tokenization: the process of breaking down the text into smaller units called **tokens**
- Tokens can be words, subwords, characters, or punctuation marks
- Tokenization is a crucial step for LLMs to “understand” the data we are inputting into these models

Importance of Tokenization

- As we have seen with previous models, tokenization is incredibly important
- This is still true with LLMs
- Tokenizing “White House” as [white, house] vs. [white_house]
- Tokenization determines how efficiently the model represents text inputs and generalizes language models
- Want to strike a balance between computational efficiency (not too many tokens) and flexible tokens (avoiding tokens that are unrecognized by the model)

Types of Tokenization: Word-Based Tokenization

- Can split on spaces or punctuation. For example, “Let’s do tokenization!” can be broken up into [Let’s, do, tokenization] or [let, ’s, do, tokenization, !]
- What we have been doing for most of this class so far
- Ideally, we would not be ignoring punctuation—think about sentiment analysis
- But we also don’t want to index every unique word. For example, we want the model to know that the words “cat” and “cats” are related

Types of Tokenization: Character-Based Tokenization

- We can also split at the character level: [L,e,t,',s,d,o,t,o,k,e,n,i,z,a,t,i,o,n]
- Now the vocabulary is much smaller and there are much fewer out-of-vocabulary terms, since every word is built from characters
- This works well for some languages, such as Chinese
- But in English, each character carries less meaning
- It also creates a lot of input at once: this three word sentence has 19 (if I counted correctly) inputs

Types of Tokenization: Subword Tokenization

- We can split up less frequently occurring words into *subwords*
- We could split up “Let’s do tokenization!” into: [Let’s</w>, do </w>, token, ization</w>, !<w>]
- </w> indicates the end of a word
- Notice that we split up a complex word “tokenization” into just two subwords

Subword Tokenization

- Builds on the idea that oftentimes, complex words share similar prefixes or suffixes

Subword Tokenization

- Builds on the idea that oftentimes, complex words share similar prefixes or suffixes
- Example: biology, psychology, audiology, cardiology, ecology, hydrology, paleontology, topology, etc.
- They all represent fields of study

Subword Tokenization

- Builds on the idea that oftentimes, complex words share similar prefixes or suffixes
- Example: biology, psychology, audiology, cardiology, ecology, hydrology, paleontology, topology, etc.
- They all represent fields of study
- We could break up “psychology” into [psych, ology</w>]

Subword Tokenization

- Builds on the idea that oftentimes, complex words share similar prefixes or suffixes
- Example: biology, psychology, audiology, cardiology, ecology, hydrology, paleontology, topology, etc.
- They all represent fields of study
- We could break up “psychology” into [psych, ology</w>]
- This allows us to relate the first subword “psych” to a lot of other words, and “ology” to a lot of other words too

Subword Tokenization

- Builds on the idea that oftentimes, complex words share similar prefixes or suffixes
- Example: biology, psychology, audiology, cardiology, ecology, hydrology, paleontology, topology, etc.
- They all represent fields of study
- We could break up “psychology” into [psych, ology</w>]
- This allows us to relate the first subword “psych” to a lot of other words, and “ology” to a lot of other words too
- Example of how subword tokenization can often be very efficient

Algorithms for Tokenization

- Usually, an LLM will use an existing tokenization algorithm
- Examples include: byte-level encoding (BPE), WordPiece, and SentencePiece
- These algorithms usually combine word-level, subword-level, and character-level tokenization to ensure that all words can be tokenized

Special Tokens

- We also add additional special tokens such as “[BOS]” (beginning of sequence), “[EOS]” (end of sequence), or “[PAD]” (padding)
- “[BOS]”: token signifies the start of the text, where content should begin
- “[EOS]”: token signifies the end of the text; often used if we include two pieces of text in a single training example where the two texts are unrelated
- “[PAD]”: often used to make short texts the same length as the longest text in a batch

Byte-Pair Encoding (BPE)

- Tokenization scheme used by GPT models (including ChatGPT)
- BPE is used to handle rare words and subwords by iteratively merging the most frequent pairs of characters and subwords in a text corpus
- Results in a vocabulary that includes whole words, subwords, and characters
- Often used when we need to tokenize a corpus with a very large vocabulary (like the Common Crawl)

Byte-Pair Encoding (BPE)



- BPE first adds all characters to the vocabulary (“a”, “b”, etc.)
- It then merges these character combinations that frequently occur together and adds them to the vocabulary
- Then continues to merge these character combinations so subwords or even whole words are formed
- It can even take into account white spaces! That is, “turtles” and “ turtles” are considered separate words.

Example of Tokenization Using GPT

Tiktokenizer

User

I like turtles.



X

Add message

```
<|im_start|>user<|im_sep|>I like turtles.<|im_end|>  
<|im_start|>assistant<|im_sep|>
```

gpt-4o

Token count
11

```
<|im_start|>user<|im_sep|>I like turtles.<|im_end|><|im_start|>  
assistant<|im_sep|>
```

200264, 1428, 200266, 40, 1299, 129606, 13, 200265, 200264,
173781, 200266


☐ Show whitespace

Example of Tokenization Using GPT

Tiktokenizer

User


I like turtles. turtles



×

Add message

```
<|im_start|>user<|im_sep|>turtles like turtles.<|im_end|>  
<|im_start|>assistant<|im_sep|>
```



gpt-4o

Token count
12

```
<|im_start|>user<|im_sep|>turtles like turtles.<|im_end|><|  
im_start|>assistant<|im_sep|>
```

200264, 1428, 200266, 83, 94958, 1299, 129606, 13, 200265,
200264, 173781, 200266

☐ Show whitespace

Tokenization in Other Models

Tiktokenizer

meta-llama/Meta-Llama-3-70B

I like turtles|

Token count

3

I like turtles

40, 1093, 72503

☐ Show whitespace

Tokenization in Other Models

Tiktokenizer

meta-llama/Meta-Llama-3-70B

nonsense
Nonsense
nonsense
asldkjfa;lkjdf laksjdf;lkajd



Token count
23

nonsense
Nonsense
nonsense
asldkjfa;lkjdf laksjdf;lkajd

77, 98833, 271, 45, 98833, 271, 41902, 271, 300, 509, 9379
7, 3716, 26, 42848, 44490, 1517, 10011, 73, 3013, 26, 4284
8, 1662, 67

☐ Show whitespace

Problems with Tokenization

Tiktokenizer

meta-llama/Meta-Llama-3-70B

Is 9.4 less than 9.14?

Token count

12

Is 9.4 less than 9.14?

3957, 220, 24, 13, 19, 2753, 1109, 220, 24, 13, 975, 30

☐ Show whitespace

Tokenization: An Area of Ongoing Research

- Tokenization is *incredibly* important to improving (or making worse) the performance of LLMs
- You can watch a 2.5 hour video on how to build the GPT tokenizer [here](#)
- There is ongoing research on how to improve tokenization—or maybe do away with it entirely

Fine-Tuning LLMs

Fine-Tuning a LLM

- LLMs are often good at general tasks, but maybe we want to apply the LLM to a new domain or have it behave in a specific manner
- We can do that using **fine-tuning**, which takes the parameters of a pre-trained model and updates them on a new, specialized task
- For example, we can fine-tune a model so it is very good at making political-related classifications
- Many different types of fine-tuning

Continued Pre-Training

- We can retrain all the parameters of the model on new, additional text data using the same self-supervised word prediction task
- This is as if our new data were at the tail end of the pre-training data
- This is often called “continued pretraining”

Masked Language Modeling Fine-Tuning

- Recall that BERT is an example of an encoder-only language model, which means the attention mechanism can look forward and backward
- One common approach is to fine-tune this language model to output predictions
- To do this, we can add extra neural circuitry after the last layer of the model to produce classifications
- A very efficient way to produce high-quality predictions, but you do need labeled data to fine-tune these models

Supervised Fine-Tuning

- Supervised fine-tuning, or SFT, is often used for instruction fine-tuning, where we want a pre-trained language model to learn to follow text instructions
- How ChatGPT was partially trained to follow instructions well
- We do need supervised responses to each command, so it is truly supervised (not self-supervised)

Ethical Considerations and Potential Harms with LLMs

Bias in Training Corpora

- Even if we filter out harmful content and PII from training corpora, biases still exist in these models
- Even reading a normal Twitter thread, for example, yields countless examples of biased responses
- Because LLMs are ingesting these texts and trying to predict the next word, it can learn from these biases and stereotypes
- An entire course can be taught on biases in LLMs!
- This can even affect tokenization, given that BPE tokenization is based on how frequently pairs of characters or subwords co-occur together

Biases in Training Corpora

In 2016, Microsoft released a chatbot called Tay. It did not end well. You can read about the story [here](#).

Privacy Concerns

- Pre-training information can contain information like phone numbers and addresses
- These are often filtered out
- But you can also often reverse engineer private information

Netflix Prize Debacle

- In 2006, Netflix set up a competition called the “Netflix Prize”
- They released 100 million anonymized movie ratings, each which includes a unique subscriber ID (so you could connect movie reviews on a user-level basis)
- The goal was to predict how those users would then rate other movies, which could then be used to improve the recommendation algorithm
- Just 16 days into the competition, two University of Texas researchers said they could identify a large portion of the users in the data
- They linked together IMDb accounts (public information) with the Netflix data (private but anonymized information), which allowed them to see what other movies these users had watched without giving an IMDb rating

Privacy Concerns

- Given that the LLMs can learn highly complex relationships between words, LLMs can potentially automatically learn these relationships between different data sources, leading to private information being leaked