CMPUT 653: Foundations of Reasoning in LLMs Lecture Notes

Siddhartha Chitrakar

September 16, 2025

Contents

Preface

Symbols and Remark on Notation

Insert Preface.

Acknowledgments

Insert Acknowledgments.

CMPUT 653: Foundations of Reasoning in LLMs **Fall 2025**

Lecture 1 Video, Lecture 2 Video

Lecturer: Csaba Szepesvári *Sep. 3 and Sep. 5 2025* *Scribe: Siddhartha Chitrakar*

Lecture 1 and 2: Modelling Sequences

Note: Lecture 1 contains errors due to reusing the notation p for different meanings. Lecture 2 corrected this; [*timestamp] used to link second lecture.

We use **probabilistic sequence models** to model sequences. Exercise 1.1, 1.2 intuitively explain why we use this, but first we introduce some notation:

- Σ - Alphabet Characters (set of all 1 length sequences)
  - English lowercase alphabet: $\Sigma = \{a, b, c, d, \dots, y, z\}$
  - DNA alphabet $\Sigma = \{A, C, G, T\}$
  - LLM alphabet usually has at least 30,000 unique characters
- Concatenation - An operation (denoted by $\oplus$) that joins two sequences
  - $a \oplus b = ab$

Thus, we define n length sequences by concatenation:

- $\Sigma^2 \coloneqq \{a_1 a_2 : a_1, a_2 \in \Sigma\}$, (set of all 2 length sequences)
  - If $\Sigma = \{a, b\}$ then, $\Sigma^2 = \{aa, ab, ba, bb\}$
- $\Sigma^n \coloneqq \{a_1 a_2 \dots a_n : a_1, a_2, \dots a_n \in \Sigma\}$, (set of all n length sequences)
  - $w = (w_1 w_2 \dots w_n) \in \Sigma^n$ (sequence of length n in order from w_1 to w_n)
- $\Sigma^0 \coloneqq \emptyset \coloneqq \{\perp\}$, ($\perp$ denotes the empty length sequence)

We can find the size (cardinality) of these n length sequences sets:

- $|\Sigma| = N$
- $|\Sigma^2| = N^2$
- $|\Sigma^n| = N^n$

Remark 1.1. Keep note that the number of sequences of length n grows exponential since $|\Sigma|^n = N^n$

Definition 1.1. We define the set of **all possible sequences** as

$$\Sigma^* := \bigcup_{n=0}^{\infty} \Sigma^n$$

Exercise 1.1. Let $\mathcal{L} \subseteq \Sigma^*$ be a language. We deterministically define a recognizer function r such that $r : \Sigma^* \to \{\text{yes}, \text{no}\}$. **Why is this problematic?**

Exercise 1.2. Consequently to Exercise 1.1, what are some advantages of using probabilistic sequence models?

1.1 Defining a probability distribution [32:24]

We define a probability distribution p over the set of all possible sequences Σ^*.

$$p : \Sigma^* \to [0, 1]$$

such that the following conditions are met:

1. $0 \leq p(w) \leq 1$ for all sequences $w \in \Sigma^*$.

2. $\sum_{w \in \Sigma^*} p(w) = 1$

With this distribution, we perform text completion tasks. Let $w \in \Sigma^*$ be a sequence, called **prefix/prompt**. Let $U \in \Sigma^*$ be a **random completion**.

We define a completed sequence W' as the concatenation of the prompt and the random completion:

$$W' := w \oplus U$$

We consider the probabilities for all possible completions, $\forall u \in \Sigma^*$:

$$P(W' = w \oplus u) = p(w \oplus u)$$

Example 1.1. Consider prefix w = “The weather is...”. The random completion, U, could be u_1 =“hot”, u_2 =“tornado”, u_3=“biking”. Based on what our model is trained on it would likely output decreasing probabilities respectively.

Remark 1.2. This is how LLMs works. When you give a prompt w, it samples from all possible completions u. However, Σ^* is countably infinte, so how do we learn a probability for every sequence?

1.2 Decomposition by Chain Rule [*00:29]

This section deals with the problem of Σ^* being countably infinite.

Definition 1.2 (Chain Rule)**.** Let $w = (w_1, w_2, \ldots, w_n)$ (sequence of length n. The joint probability of the sequence, $p(w)$, can be decomposed:

$$p(w) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_1, w_2) \cdots p(w_n|w_1, \ldots, w_{n-1})$$

We will **exploit** what is the next character given what we have though Chain Rule. First, we ask how can a probabilistic model know when to stop?

Definition 1.3 (STOP Symbol)**.** A character $\langle STOP \rangle$ that signals the model when to stop.

Intuition: To define conditional probabilities we...

1. Start from the distribution of all possible sequences. (Section 1.1)
2. Now define conditional probability from this (Section 1.2.1)

Remark 1.3 (Bottom-up approach)**.** We could have started from the conditional distribution, then use chain rule to get the distribution of all sequences.

1.2.1 Defining the conditional distribution [*1:45]

Let $W \sim p$, be a random variable sample of a random string. Formally,

$$\forall w \in \Sigma^*, \mathbb{P}(W_n = w_n) = p(w)$$

Goal: Derive a conditional distribution of the next character from known info:

$$\mathbb{P}(W_n = w_n | W_{1:n-1} = w_{1:n-1})$$

Exercise 1.3. What is the problem about this "Goal"? Why would we need to extend $\langle STOP \rangle$ to W_n to infinite sequences?

Definition 1.4 (Extend Stop to Infinite Sequences)**.** Define $\hat{\Sigma} \coloneqq \Sigma \cup \langle STOP \rangle$. Additionally, pad the random variable, W, to an infinite sequence:

$$\hat{W} = W \langle STOP \rangle \langle STOP \rangle \langle STOP \rangle ...$$

Theorem 1.1 (Conditional Distribution of p)**.** Let $\hat{p} : \bigcup_{n\geq 1} \Sigma^{n-1} \times \hat{\Sigma} \to [0,1]$,

$$\begin{aligned}\hat{p}(w_n|w_{1:n-1}) &\coloneqq \mathbb{P}(\hat{W}_n = w_n|\hat{W}_{1:n-1} = w_{1:n-1}) \\ &= \begin{cases} \frac{\mathbb{P}(\hat{W}_{1:n}=w_{1:n})}{\mathbb{P}(\hat{W}_{1:n-1}=w_{1:n-1})} & \text{if } \mathbb{P}(\hat{W}_{1:n-1} = w_{1:n-1}) \neq 0 \\ p_0(w_n) & \text{if } \mathbb{P}(\hat{W}_{1:n-1} = w_{1:n-1}) = 0 \end{cases}\end{aligned}$$

by Chain Rule Definition 1.2 and let p_0 be any distribution.

Exercise 1.4. Prove Theorem 1.1 is a probability distribution

Remark 1.4. Theorem 1.1, $\tilde{p}$, suffices to model completion tasks like Definition 1.5 (Autoregressive models). We generate a character by character completion sequence until the model samples $\langle$STOP$\rangle$.

Definition 1.5 (**Autoregressive Model**)**.** Autoregressive models predict the next value in a sequence based on the values that came before it.

Lecture 1 and 2 Exercises Solutions (Tentative, will be updated)

Solution 1.1. This forces a yes/no decision, and languages are too complex to be deterministically decided like this. For example, language can be interpreted or explained in many different ways.

Solution 1.2. Compared to Exercise 1.1, this relaxes the deterministic clause and instead we ask how likely is a sequence part of a language. This allows us to navigate the complexities and nuances of languages better.

Solution 1.3. First, we look at **finite sequences**, $\Sigma^{n*} := \bigcup_{i=0}^{n} \Sigma^i$.

Consider a conditional distribution on finite sequences, $\hat{p} : \Sigma^{n*} \times \Sigma \to [0, 1]$,

$$\hat{p}(w_1 w_2 \dots w_{n+1}) := p(w_{n+1}|w) = \frac{p(w_1 w_2 \dots w_{n+1})}{p(w_1 w_2 \dots w_n)}$$

Claim: For any sequence w $\in \Sigma^n$, we must extend $\langle \text{STOP} \rangle$ so $|w| = n$.

Proof. Contradiction: Assume we don't extend $\langle \text{STOP} \rangle$. Start with $n = 2$, then, the outcomes are $\{\emptyset, a, b, ab, ba, aa, ba\}$. Suppose $p(b) = \frac{1}{3}$ and $p(ba) = \frac{2}{3}$, then $p(ba|b) = \frac{p(ba)}{p(b)} = \frac{2/3}{1/3} = 2$!? So what went wrong?

$p(b)$ is exactly b. But, it should be the prob. that the first letter seen is b.

Formally, we want $p(b)$ as $p(b) + p(ba) + p(bb)$. To do this, we extend $\langle \text{STOP} \rangle$ so $p(b) = p(b \oplus \langle \text{STOP} \rangle) + p(ba) + p(bb)$. If we want to know the prob. that the sequence is exactly b, it is now $p(b \oplus \langle \text{STOP} \rangle)$

We can inductively show that $\forall n \geq 2$, we must extend $\langle \text{STOP} \rangle$. □

To end, Σ^* is countably infinite, thus we must extend $\langle \text{STOP} \rangle$ to infinite sequences. Importantly, this makes the conditional dependencies compatible!

Intuition explanation: Consider w = "The weather is" and w_{n+1} = "hot". Then, the sentence, "The weather is", is not valid and rarely seen.

$p(\text{"The weather is"}) < p(\text{"The weather is hot"}) \implies \tilde{p}(\text{"The weather is hot"}) > 1$. Clearly, without extending, the conditional dependencies is not compatible.

Solution 1.4 (Prove this and the compatibility yourself...)**.**

CMPUT 653: Foundations of Reasoning in LLMs **Fall 2025**

Lecture 2 Video [27:10], Lecture 3 Video

Lecturer: Csaba Szepesvári *Sep. 5 and Sep. 9 2025* *Scribe: Siddhartha Chitrakar*

Lecture 2 and 3: Transformer Architecture

Note: Lecture 3 recapped and continued the discussion of Transformer Architecture; [*timestamp] used to link Lecture 3.

Assumption: From now on and for convenience $\Sigma := \hat{\Sigma} = \Sigma \cup \langle STOP \rangle$

Exercise 2.1. Read over this lecture: Provide the intuition and re-define the transformer architecture to the specific well-known euclidean space $\mathbb{R}^n$.

Transformers map strings to strings, and we generalize them to abstract vector spaces for alternatives to Euclidean space *(e.g language RASP)*. Why are vector spaces important? The transformer acts in the workspace $\mathcal{W}$.

Definition 2.1 (Workspace)**.** Let $\mathcal{W} \neq \emptyset$ be a **vector space** over scalars $\{0, 1\}$ called a **workspace**. The elements $\mathbf{x} \in \mathcal{W}$ are called vectors.

Recall vector spaces are equipped with two operations, addition $(+)$ and scalar multiplication $(\cdot)$, which for $\forall \mathbf{x}, \mathbf{y} \in \mathcal{W}$:

1. **Closure under Addition:** $\mathbf{x} + \mathbf{y} \in W$.
2. **Scalar Multiplication:** $1 \cdot \mathbf{x} = \mathbf{x}$ and $0 \cdot \mathbf{x} = \mathbf{0}$

Definition 2.2 (Embedding Map)**.** Let $e : \Sigma \to \mathcal{W}$ be an embedding map that maps each $x \in \Sigma$ to a vector representation $\mathbf{w} \in \mathcal{W}$.

$$e(x) = \mathbf{w}$$

Definition 2.3 (Unembedding Map)**.** Let $u : \mathcal{W} \to \Sigma$ be an unembedding map that maps each vector $\mathbf{w} \in \mathcal{W}$ back to a character $x \in \Sigma$

$$u(\mathbf{w}) = x$$

Remark 2.1. The embedding/unembedding maps aren't inverses. Sometimes they reverse the same info, but these are not bijections (different dimensions).

Before we define transformers, here is **notation** for component map extension

- If given $(a_1, a_2, \dots, a_n) \in A^n$, then the component map extension is
  - $s^{[n]}((a_1, a_2, \dots, a_n)) = (s(a_1), s(a_2), \dots, s(a_n))$
  - $s(a_1, a_2, ..., a_n) \coloneqq s^{[n]}((a_1, a_2, \dots, a_n))$ (Drop the [n] for brevity)

2.1 Intuition of Transformer Architecture

Intuition: [Talk by 3B1B] is a good source to visualize the intuition

Transformers, f, maps an input $\vec{w} \in \Sigma^*$, to an output $\vec{w} \in \Sigma^*$ by refining the representation of $\vec{w}$. Goal: "Enrich" the vector with information. The "enriching" process is a sequential composition of functions:

$$f = u^{[n]} \circ f_L^{(n)} \circ \cdots \circ f_1^{(n)} \circ e^{[n]}$$

Step 1. Embedding ($e^{[n]}$)

First, $e^{[n]} : \Sigma^n \to \mathcal{W}^n$, embeds a input sequence $\vec{w} \coloneqq (w_1, ..., w_n) \in \Sigma^*$ component wise to a vector sequence $(e(w_1), ..., e(w_n)) = (\mathbf{x_1}, ..., \mathbf{x_n}) \in \mathcal{W}^n$.

Step 2. Transformer Layers ($f_\ell^{(n)}$)

The vector sequence $(\mathbf{x_1}, ..., \mathbf{x_n})$ is then processed through finitely many L layers. Each layer, $f_\ell^{(n)}$, refines every vector by two sub-steps:

Step 2.1 Attention ($A_\ell^{(n)}$): Updates vectors $\mathbf{x}_i$ by aggregating "useful" info from the entire sequence. How do you determine the "useful" info:

Query (q_ℓ): Asks current vector $\mathbf{x}_i$, "What info do I need?"

Key (k_ℓ): Answers from other vector $\mathbf{x}_j$, "This is the info I represent."

Attention Score (a_ℓ): Compares Query ($\mathbf{x}_i$), with Key ($\mathbf{x}_j$) for relevance.

Value (v_ℓ): The actual info of vector $\mathbf{x}_j$ if deemed relevant (key/query matches)

Step 2.2 Multilayer perceptron ($m_\ell^{[n]}$): The multilayer perceptron $m_\ell^{[n]}$ performs a non-linear transformation on each vector independently.

Step 3. Unembedding ($u^{[n]}$)

After L layers, the "enriched" vectors is unembeded by $u^{[n]} : \mathcal{W}^n \to \Sigma^n$. This maps the last vectors to the vocab space Σ to produce an output sequence.

2.2 Definition of Transformer Architecture [37:39], [*6:13]

Definition 2.4 (Transformer Map)**.** $f : \Sigma^* \to \Sigma^*$ is a regular **transformer map** over the sets of functions $\mathcal{M}, \mathcal{A}, \mathcal{Q}, \mathcal{K}, \mathcal{V}$ if $\forall n \geq 1$,

$$f|_{\Sigma^n} = u^{[n]} \circ f_L^{(n)} \circ \cdots \circ f_1^{(n)} \circ e^{[n]}$$

such that for each layer $\ell \in \{1, \ldots, L\}$:

1. The layer function $f_\ell^{(n)}$ is a composition $f_\ell^{(n)} = m_\ell^{[n]} \circ A_\ell^{(n)}$, s.t $m_\ell^{[n]} \in \mathcal{M}$.
    - The compositions are a multilayer perceptron and attention map.
2. The attention map $A_\ell^{(n)} : \mathcal{W}^n \to \mathcal{W}^n$ is defined $\forall \mathbf{x} \in \mathcal{W}^n$,

$$(A_\ell^{(n)}(\vec{x}))_i = \mathbf{x}_i + \sum_{j=1}^{n} a_\ell(q_\ell(\mathbf{x}_i), k_\ell(\mathbf{x}_j))v_\ell(\mathbf{x}_j)$$

   where $a_\ell \in \mathcal{A}$, $q_\ell \in \mathcal{Q}$, $k_\ell \in \mathcal{K}$, and $v_\ell \in \mathcal{V}$.

   2.1. Function $a_\ell \in \mathcal{A}$ is the attention pattern which determines whether information from vector j (via $v_\ell(\mathbf{x}_j)$) is passed to vector i:

$$a_\ell : \mathcal{W} \times \mathcal{W} \to \{0, 1\}$$

   2.2. In practice, the summation is **normalized**

$$\frac{\sum_{j=1}^{n} a_\ell(q_\ell(\mathbf{x}_i), k_\ell(\mathbf{x}_j))v_\ell(\mathbf{x}_j)}{\sum_{j=1}^{n} a_\ell(q_\ell(\mathbf{x}_i), k_\ell(\mathbf{x}_k))}$$

   2.3 However, we need a **probabilistic unembedding** to predict the next sequence defined in Theorem 1.1. The unembedding (u) maps the final vector representation, $\mathbf{x}_i \in \mathcal{W}$, to a probability distribution $\pi_i := \frac{\exp(\mathbf{w}_j^T \mathbf{z}_i)}{\sum_{j \in \Sigma} \exp(\mathbf{w}_g j T \mathbf{z}_i)}$ over the alphabet Σ. [*19:44]

Note: Summation range can be modified, $j < i$ for **causal attention** or $j \neq i$
.

2.2.1 Discussion About Transformer Architecture [*25:43]

Note: The discussion is to introduce different aspects about transformers.

Sparse Attention [*25:43]: The attention normalization in Definition 2.4 creates a global dependency that's recalculated for every update, leading to $O(n^2)$ complexity and poor caching. This motivates **sparse attention** where each vector only attends to a limited subset of vectors.

Positional Embeddings [*29:23]: The attention summation is permutation-invariant. However, in langauge, word order absolutely matters!

- Initial solution adds a position vector $\mathbf{p}_1, ...\mathbf{p_n} \in \mathcal{W}$ to each input $\mathbf{x}_i$. The query and key become $q_\ell(\mathbf{x}_i + \mathbf{p}_i)$ and $k_\ell(\mathbf{x}_j + \mathbf{p}_j)$. This struggles with infinite sequences $\mathbf{p}_1, \mathbf{p}_2, ...$; thus, we now use rotational embeddings

- **Rotational Positional Embeddings [*1:13:17]:** Positional dependent matrix, R_i acting on queries and keys.

$$\mathbf{q}'_i = R_i q_\ell(\mathbf{x}_i) \quad \text{and} \quad \mathbf{k}'_j = R_j k_\ell(\mathbf{x}_j)$$

  The attention score depends on the relative distance between vectors, due to the rotation matrix property ($R_i^T R_j = R_{i-j}$):

$$\langle \mathbf{q}'_i, \mathbf{k}'_j \rangle = (R_i q_\ell(\mathbf{x}_i))^T (R_j k_\ell(\mathbf{x}_j)) = q_\ell(\mathbf{x}_i)^T R_{i-j} k_\ell(\mathbf{x}_j)$$

  This allows the model to generalize far better to unseen sequence lengths!

Multi-headed Attention [*41:30]: This addresses different types of information simultaneously by running attention patterns in parallel.

Mixtures of Experts [*1:06:50]: Analogous to multi-headed attention where we have a set of k parallel "expert" MLPs, denoted as $m_i(x)$, and a trainable gating network p. For each input, the gating network selects a subset of the experts.

- **Expert Networks (m_i):** A collection of k parallel MLPs.

- **Gating Network (p):** A trainable network that outputs a probability distribution over the experts. The weight for the i-th expert is $p_i(x)$.

- For sequence x, y is the weighted sum of the outputs from all experts.

$$y = \sum_{i=1}^{k} p_i(x) \cdot m_i(x)$$

- Load balancing prevents the gating network from using the same few favorite experts. For a fixed context sequence $x_1, ..., x_T$, the cumulative weight to each expert is roughly equal:

$$\sum_{t=1}^{T} p_i(x_t) \approx \frac{T}{k}$$

Other Discussions:

- Decoding Strategies [*50:55]
- Complexity of Matrices [*1:01:22]
- Precision [*1:03:53]
- **Layered Norm [*43:42]:** Randomly thrown in to rescale the vectors to have a consistent mean and variance. This is done to stabilize the transformer.

Lecture 2 and 3 Exercises Solutions (Tentative, will be updated)

Solution 2.1. Why in practice do we use $\mathbb{R}^n$? So we can encode the sequences to vectors and provide some meaning to it with numbers.

Redefining Transformers Now the abstract workspace $\mathcal{W} = \mathbb{R}^d$

Embedding Map: $e^{[n]} : \Sigma^n \to \mathbb{R}^d$ encodes sequences to vectors

Unembedding Map: $u^{[n]} : \mathbb{R}^d \to \Sigma^n$ encodes vectors back to sequences.

- In practice, there is a **softmax function** to get a probability distribution.

Query, Key, Value (q_ℓ, k_ℓ, v_ℓ These are linear transformations meaning matrix multiplication! $q_\ell(\mathbf{x}_i) = W_Q\mathbf{x}_i$, $k_\ell(\mathbf{x}_j) = W_K\mathbf{x}_j$, and $v_\ell(\mathbf{x}_j) = W_V\mathbf{x}_j$, where W_Q, W_K, W_V are learned weight matrices.

Attention Score (a_ℓ**):** scaled dot-product to measure similarity by direction

$$a_\ell(q, k) = \text{softmax}\left(\frac{q^T k}{\sqrt{d_k}}\right)$$

Measures similarity between the query and key vectors

Appendix A: Extra Proofs and Results

More appendix content here.

Appendix B: Supplementary Figures

More appendix content here.