1 Introduction

The word order of natural languages has been initially characterized by Behagel's laws[1]. These laws state elements that are related to each other tend to be placed close together in a sentence. Until recently, these laws have been summarized as dependency length minimization[2–4]. Dependency lengths are the distances between linguistic units that are related to each other in a sentence. This theory states that languages had evolved to minimize the sum of these distances so that the users can easily produce and comprehend sentences by reducing cognitive load. It has been shown 37 languages follow this principle[2].

One potential explanation for this phenomenon is that human have limited working memory capacity[5]. This limitation forces languages to evolve by placing related words closer to each other, allowing speakers and listeners could easily integrate words and process semantic information from memory. These ideas are known as Dependency length theory (DLT)[6]. One key prediction of DLT is the locality effects: longer dependencies leads to longer reading times and vice versa, which has been supported by various psycholinguistic studies[7;8].

Recently, information-theoretic approaches has been proposed to formalize the relationship between incremental working memory usages in languages processing and sequential order, known as memory-surprisal tradeoff[9]. This theory has shown the ease of comprehension is determined by the resources allocated to store elements (e.g., words) in working memory, hence there exists a tradeoff between memory usage and comprehension difficulty in a corpora of 54 languages by using surprisal (shannon entropy copnditioned on limited memory system). However, it is still unclear how the architecture of different language model model, tokenization methods and the syntatic structures of different languages affect surpisal measures and thus memory-surprisal tradeoff. Here, we aim to investigate these factors by implementing a naive hidden markov model and a simple LSTM network and different tokenization methods on corpora of 10 chosen languages.

2 Method

2.1 Architecture and the tokenizer of Language Models

Here we implement two different language models: a simple markovian feed-foward neural network language model (NNLM) and a long-short term memory (LSTM) network.

Neural Network Language Model (NNLM)

The NNLM recieves a seqeuence of tokens $x = \{w_0, x_1, \ldots, x_t\}$ with embeddings $e = \{e_0, e_1, \ldots, e_t\}$ as input and compute the proability of the next word given a context window with size n:

$$P(w_t | w_{t-n+1}, \ldots, w_{t-1}) \tag{1}$$

The embedding is then passed to a hidden layer:

$$h_t = tanh(W_{inh} e_t + b_{ih}) \tag{2}$$

The resulting hidden state is then passed to an output layer with softmax activation to compute the probability distribution of the next token:

$$P(w_t | w_{t-n+1}, \ldots, w_{t-1}) = softmax(W_{out} h_t + b_{out}) \tag{3}$$

The tokenizer used for NNLM is a simple whitespace tokenizer that splits the text into words based on spaces. However, this method is problematic for languages without explicit word boundaries (e.g., Chinese, Japanese). **(Answer for Exercise1)** In addition, given that NNLM is a next-token preidiction model, only last few tokens are predicted due to the limited size of the sentence. For example, a test sentence with a fixed length T were passed to the model, only the last $T - N + 1$ tokens were predicted because the first $N - 1$ tokens do not have enough context for a memory window of length N. To solve this problem, we padded the begining of each sentence with $N - 1$ special tokens `<pad>` so that all tokens in the sentence could be predicted. Our modification is maded in the `NgramsLanguageModelDataSet(Dataset)` class in the `dataloader.py` file.

Long Short-Term Memory (LSTM) Network

(Answer for Exercise 3)

A normal Recurrent Neural Network would predict the next word w_t given its preceding context $(w_1, \ldots, w_{t-1})$ by maintaining a hidden state h_t that acts as a memory. However, such a vanilla RNN struggles with long-term dependencies due to instability in the gradient propagation (vanishing/exploding gradients).

So, we rather prefere Long Short-Term Memory networks to address this limitation. LSTM introduces an additional hidden state: a cell state c_t and a system of gates that regulate information flow. This gating mechanism allows LSTMs to selectively retain and discard information, effectively learning long-range dependencies trough the cell state.

In PyTorch, implementing an LSTM-based RNNLM involves an 'nn.Embedding' layer to convert token IDs to dense vectors (x_t), an 'nn.LSTM' module for the recurrent processing (handling h_t and c_t updates), and an 'nn.Linear' layer to project the LSTM's final hidden state to a vocabulary-sized output.

The biggest difference between our RNNLM and the previous NNLM we used lies in the way they handle context and memory. The NNLM uses a fixed-size context window of length n and concatenates the embeddings of the $n-1$ preceding words as input for a single forward pass. It treats each prediction as independent, given its immediate $n-1$ left neighbors in the sequence, without carrying any information beyond this window.

On the other hand, our RNNLM processes the input sequence recurrently, which means that it maintains the dynamic hidden state (h_t) and the cell state (c_t) that act as a "memory" of all preceding tokens in the sequence. This allows the RNNLM to capture much longer dependencies than a fixed window NNLM.

The most important challenges when implementing the RNNLM in PyTorch are :

- Because of their inherent recurrent structure, LSTMs are likely to suffer from gradients issues (exploding or vanishing), breaking calculation during traning with NaN or infinite values. However we haven't faced this issue as we used rather short sequences. In addition,in these condition it can be meaningful to add clipping features when gradients get too big or too small.
- For independent n-grams, as we have been using with our previous model, each n-gram is a fresh sequence. But when we are dealing with continuous sequences, LSTMs have to maintain internal hidden and cell states. Thus, we need to pass these states from the end of one batch as the initial states for the next, which requires subtle handling of the variable sequence lengths and batching.
- These considerations are more real-life issues, but we've been able to run our model as a POC. But because of the recurrent nature of LSTMs, for long sequences, training can be much slower compared to feedforward models.

GPT-2 Model

(Answer for Exercise2) To compare the surprisal measures between different tokenization methods, we also used a simplied GPT-2 model with byte-pair encoding (BPE) tokenizer to control for vocabulary and vocabulary size of the model. The GPT-2 model is a transformer-based language model that uses self-attention mechanisms to capture long-range dependencies in text. It is a compact GPT-2 decoder-only transformer with a 256-dimensional embedding space, 4 transformer blocks and a hidden size of 256. Each block contains multi-head self-attention (implemented via Conv1D projections), a 1024-dimensional feedforward MLP with GELU activation and residual connections with layernorm and droptout. A final projection layer maps the hidden states to the vocabularty.

The BPE tokenizer splits the text into subword units, allowing the model to handle out-of-vocabulary words and capture morphological information.

Randomization Protocol of Syntactic Structures

(breifly describe how we do exercise 5 here)

Choice of corpora and languages

(Answer for Exercise 4) Regarding selection criteria for the corpora, we want to ensure certain things. First, the languages must be present in UD. Second, we want to prioritize languages whose corpora are large enough to estimate language models (though we will be limited by our compute capabilities, so this is more a real life concern). Third, we want to aim for diversity of families and geographic regions to make the results more general.

So we can go for the following UD available languages:

- **English** – typologically fairly fixed SVO order.
- **Spanish** – another SVO, fixed-order example, Romance family.
- **German** – but typologically more flexible (WALS shows it lacks a single dominant SVO vs. SOV) → good "semi-free" testing case.
- **Turkish** – typologically strongly SOV/OV (head-final) and relatively more fixed order (though topic-marking allows variation) → fixed-order side.
- **Hungarian** – typologically discourse-configurational with flexible word order → good free-order candidate.
- **Russian** – Slavic, moderately flexible word order (case marking allows alternations) → another free-order candidate.
- **Japanese** – strongly SOV, fairly fixed in canonical word order → fixed side.
- **Finnish** – Uralic language, fairly flexible word order due to rich morphology → free-order side.
- **Arabic (Modern Standard Arabic)** – typologically VSO or SVO depending on dialect; variation makes it interesting → flexible side.
- **Chinese (Mandarin)** – typologically SVO and relatively fixed in major clauses → fixed side.

Experimental Procedures

(briefly describe how we do exercise 7 here)

3 Results and Discussion

3.1 Effects of tokenization methods on surprisal measures

(Answer for Exercise2 Continued) The GPT-2 use a Byte-Pair Encoding (BPE) tokenizer, which will decompose unknown words into fragments of known tokens, meaning that BPE guarantees any string is tokenizable. In addition, common patterns (e.g., the, ing) become short token sequences but for rare/unseen patterns will stay as small tokens, suggesting that an unkown sequences required longer sequences and therefore giving higher surprisals. For example, if "rareword" is a common pattern in the training corpora, that means this patten can be represented as a single token, hence the surprisal will be $p(rareword|context) = a$. However, if it is less likely, lets say it will be decomposed into $[t_1, t_2, t_3, t_4]$, even the model learns that these tokens are more likely (i.e., $p(t_i) = b > a$), the suprisal will still be larger due to the multiplicative nature of the tokens – $p(rareword|context) = \prod_{i=1}^{n} p(t_i) = \sum_{i=1}^{n} \log p(t_i)$, as shown in Figure 1.

The properties of BPE make it very difficult to compare two different approaches. To compare them, first an untrained GPT-2 model that has be trained on the same copera and maintain the same vocab. To capture the unknown sequences that has not been shown to GPT-2 tokenizer, we have to align and concatenate the tokens so that it matches the unknown words in the context, and sum their surprisals for comparison. Then a statistical test is required to test if the KL-divergence between two distributions of the surprisals does not arise from chance, which tell us whether the two distributions are different from each other (perhaps a permutation test)

To conclude, it is not easy to compare the surprisals between the two distributions as it required sophisticated alignments and design of the GPT-2 and training procedures. From a theoretical point of view, for unknown sequences, most unknown words will recieve a larger surprisal in the case of BPE tokenizer, while the model with a simple `<unk>` exhibits a more narrower range of surprisal due to weaker dependency of the context.

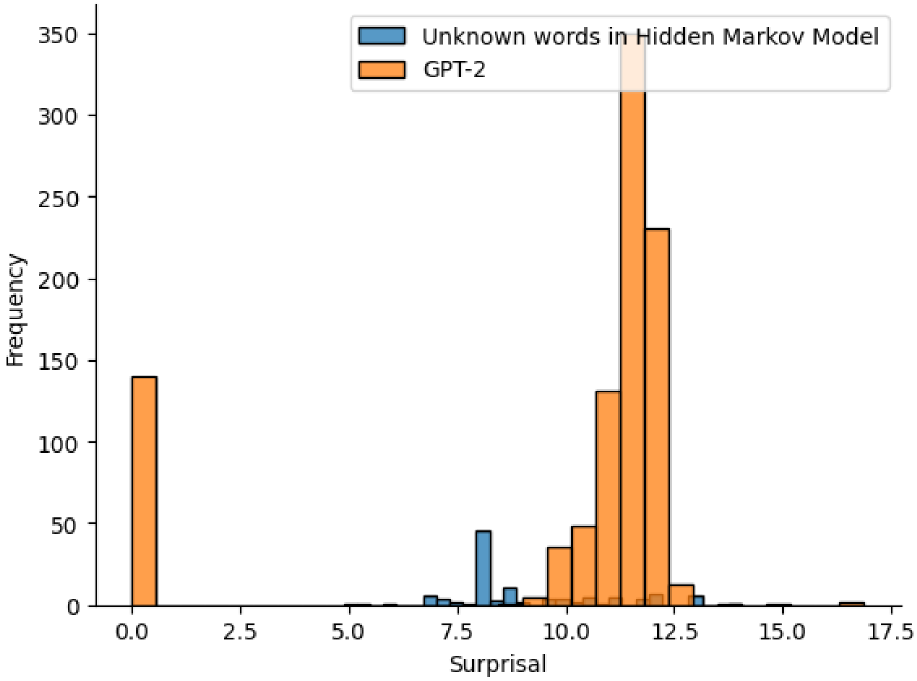

Figure 1: The distribution of suprisal for two different tokenization methods: whitespace tokenizer (blue) and byte-pair encoding tokenizer (orange) on English corpus using NNLM and GPT-2 model. The BPE tokenizer results in a more skewed distribution with a longer tail, indicating that it produces higher surprisal values for certain tokens compared to the whitespace tokenizer. This suggests that the choice of tokenization method can impact the surprisal measures obtained from language models.

4 References

References

[1] Behaghel, O. Beziehungen zwischen Umfang und Reihenfolge von Satzgliedern. *Indogermanische Forschungen* **25**, 110–142 (1909).

[2] Futrell, R., Mahowald, K. & Gibson, E. Large-scale evidence of dependency length minimization in 37 languages. *Proc. Natl. Acad. Sci. USA* **112**, 10336–10341 (2015). doi:10.1073/pnas.1502134112.

[3] Liu, H. Dependency distance as a metric of language comprehension difficulty. *J. Cogn. Sci.* **9**, 159–191 (2008). doi:10.17791/jcs.2008.9.2.159.

[4] Temperley, D. Minimization of dependency length in written English. *Cognition* **105**, 300–333 (2007). doi:10.1016/j.cognition.2006.09.011.

[5] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA* **79**, 2554–2558 (1982). doi:10.1073/pnas.79.8.2554.

[6] Gibson, E. Linguistic complexity: locality of syntactic dependencies. *Cognition* **68**, 1–76 (1998). doi:10.1016/S0010-0277(98)00034-1.

[7] Grodner, D. & Gibson, E. Consequences of the serial nature of linguistic input for sentential complexity. *Cogn. Sci.* **29**, 261–290 (2005). doi:10.1207/s15516709cog0000_7.

[8] Vasishth, S. & Drenhaus, H. Locality in German. *Dialogue & Discourse* **2**, 59–82 (2011). doi:10.5087/dad.2011.104.

[9] Hahn, M., Degen, J. & Futrell, R. Modeling word and morpheme order in natural language as an efficient trade-off of memory and surprisal. *Psychol. Rev.* **128**, 726–756 (2021). doi:10.1037/rev0000269.