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.

Randomization Protocol of Syntactic Structures

(Answer for Exercise 5: Randomization methods)
By studying the provided script dependencies.py we learned its sophisticated approach to preserving dependency structure during randomization. The module offers two methods: shuffle tree() which preserves dependency relationships while randomizing word positions, and shuffle projective() which maintains projectivity constraints. However, for our information locality research, we developed a custom conllu parser.py module to have more direct control over word forms and seamless integration. We implemented two complementary randomization strategies to test distinct linguistic hypotheses.

- The first is complete randomization, where word order is entirely shuffled without preserving any structural information. This serves as the strongest baseline to test Hypothesis H1 (Information Locality): natural language word order is optimized to cluster predictively relevant information locally. If Surprisalnatural $<$ Surprisalrandom, this demonstrates that word order organization reduces prediction uncertainty, as proposed by Hahn et al. (2021).
- The second strategy is dependency-preserving randomization, inspired by dependencies.py's shuffle tree() method. In this randomization method, we shuffle the linear order of the words in a sentence while preserving all original dependency relations. Each token in a dependency tree is associated with two indices: a linear position ID (*tid*) and the position ID of its syntactic head (*head tid*). The algorithm first applies `random.shuffle()` to the list of linear position IDs, producing a randomized target order. A mapping from old to new positions is then constructed, and each token's head index is updated using this mapping so that all governor–dependent pairs are preserved. This procedure maintains the topology of the dependency tree while altering

the surface word order, thereby breaking grammatical word order but retaining the underlying syntactic structure. This dependency-preserving shuffle serves as a controlled baseline for dependency length minimization (DLM) experiments. Compared to fully random shuffling, it provides a more structurally informed contrast condition, as it isolates the effect of linear order independent of the dependency graph itself. This tests Hypothesis H2 (Dependency Length Minimization): the optimization in natural language specifically operates by placing syntactically related words closer together (Futrell et al., 2015).

Experimental Procedures

We trained NNLM and RNNLM models with memory sizes ranging from M=1 to M=16 on three types of corpora: natural (original word order), completely randomized (all structure removed), and dependency-preserved (linear order shuffled but syntactic dependencies maintained). For each language and configuration, we measured average surprisal on a held-out test set (20% of the data). Models were trained for 16–50 epochs (NNLM) and 40 epochs (RNNLM) using the AdamW optimizer with learning rates of 0.005 (NNLM) and 0.003 (RNNLM). The NNLM used a fixed hidden dimension adjusted proportionally to memory size (hidden_dim = 64 + memory_size $\times$ 16), while the RNNLM maintained a constant hidden dimension of 128 across all memory sizes.

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.

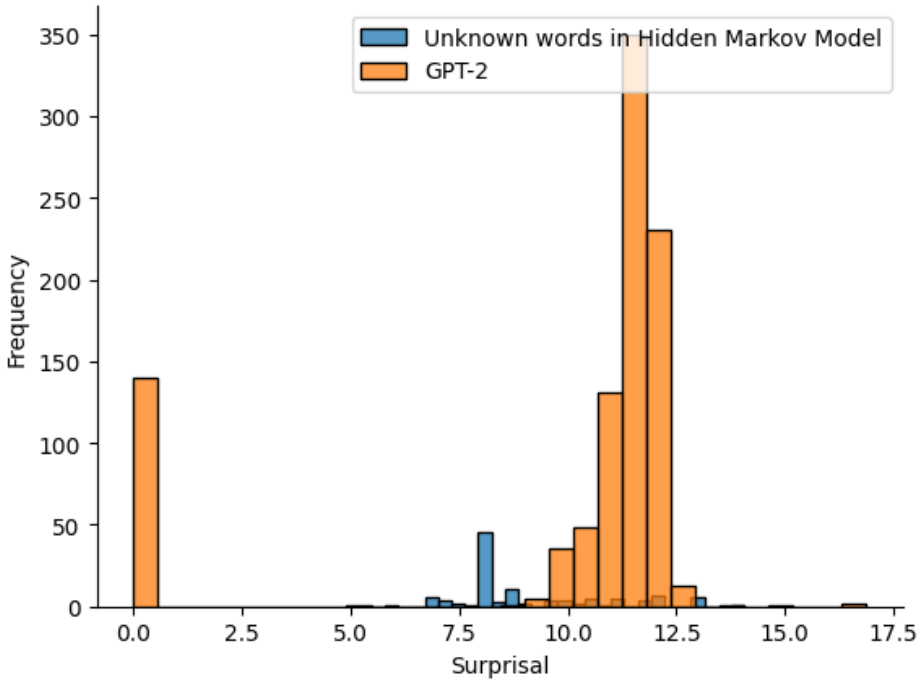


Figure 1: The distribution of suprisal for two different tokenization methods: whitespace tokenizer (blue) and byte-pair encoding tokenizer (orange) on English corpus using NNLM and GPT-2 model. The BPE tokenizer results in a more skewed distribution with a longer tail, indicating that it produces higher 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.

3.2 Observations on artificial corpora randomized

Answer for Exercise 6

For statistical testing, we implemented a permutation test to compare the means of dependency lengths between natural corpora and randomized corpora by shuffling the labels 1000 times. The null hypothesis assumes that the average dependency length in the randomized dataset is not different from that in the natural corpora. After running the permutation test, the null hypothesis yielded a p-value of 0 across 1000 iterations, indicating that we can reject the null hypothesis and accept the alternative hypothesis: the average dependency length in the randomized dataset is significantly different from that in the natural corpora.

There is a clear and consistent difference. As shown in our results, all languages exhibit **longer** average dependency lengths after word order randomization in both methods. For example, in English: 11.01 (natural) $\rightarrow$ 32.83 (complete shuffle) — a 198% increase. This increase is observed **across all ten languages** and **both randomization methods**. Notably, when dependency relations are preserved, the increase in dependency length is **smaller** than in full randomization, which breaks all relations. These results strongly support the hypothesis proposed in the paper: natural language word order is optimized to minimize dependency length, keeping syntactically related words closer together.

Interestingly, free word order languages do **not** systematically exhibit longer dependency lengths. For example, Turkish: 8.83 vs. English: 11.01; Russian: 60.29 vs. German: 60.15. In fact, Turkish (a free word order language) has one of the shortest dependency lengths, while Spanish (fixed SVO) has the longest.

The choice of randomization method is crucial for drawing valid conclusions. Complete randomization breaks all syntactic relations and dependencies without any constraints, which can only show that natural language is better than a completely randomized language but does not allow more nuanced observations. In contrast, dependency-preserving shuffle (shuffle tree) maintains structural constraints

and respects linguistic dependencies. This method provides a suitable baseline for controlled experiments, allowing us, for instance, to systematically manipulate the order of nouns and verbs while preserving dependency relations, isolating the effect of word order on dependency length.

Finally, comparing dependency lengths under complete randomization and dependency-preserved randomization reveals a clear cross-linguistic pattern. When a sentence is fully randomized without preserving any syntactic structure, dependency lengths increase dramatically, as expected in a system with no grammatical constraints. In contrast, shuffling only the linear order while preserving original dependency relations leads to a much smaller increase in dependency length.

This contrast suggests that natural languages follow structural regularities that constrain linearization, with dependency structure being one such principled constraint. This raises a broader question: is this locality effect a universal property of language? More specifically, beyond dependency locality, do other syntactic frameworks—such as phrase-structure grammar and constituent-based models—also exhibit some form of locality?

In summary, natural language universally shows **optimized word order** for dependency length minimization, **regardless** of whether the language is classified as "free" or "fixed" word order. The choice of randomization procedure is essential for isolating and identifying this specific optimization principle.

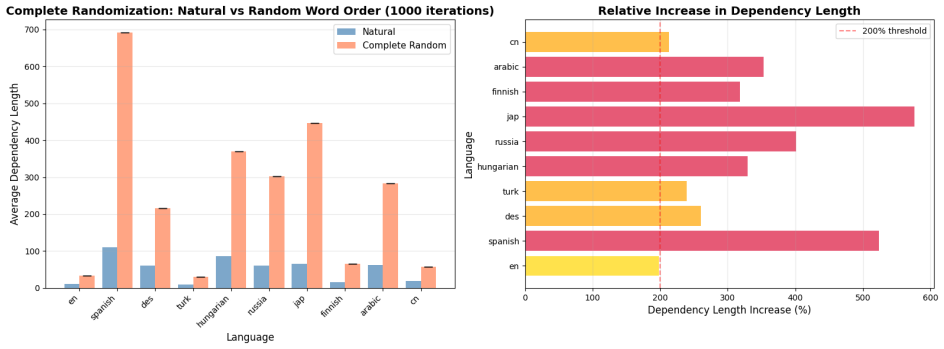


Figure 2: Average increase: 341.4%. Minimum increase: 198.0% (en). Maximum increase: 576.6% (jap). P-value range: 0.000 - 0.000 All p-values ¡ 0.05

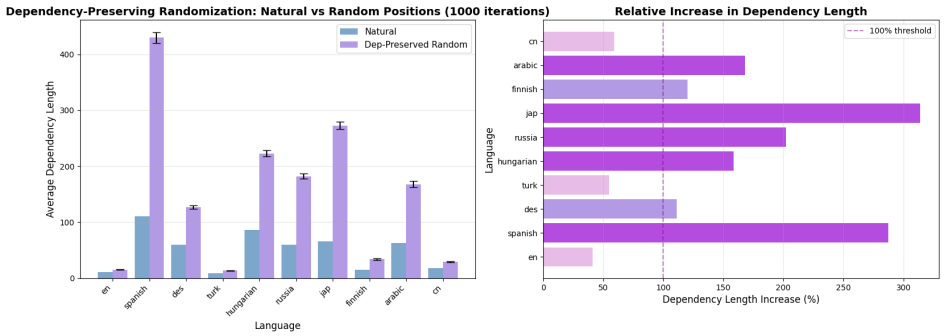


Figure 3: Average increase: 151.7%. Minimum increase: 41.1% (en) Maximum increase: 313.9% (jap). P-value range: 0.000 - 0.000 All p-values ¡ 0.05

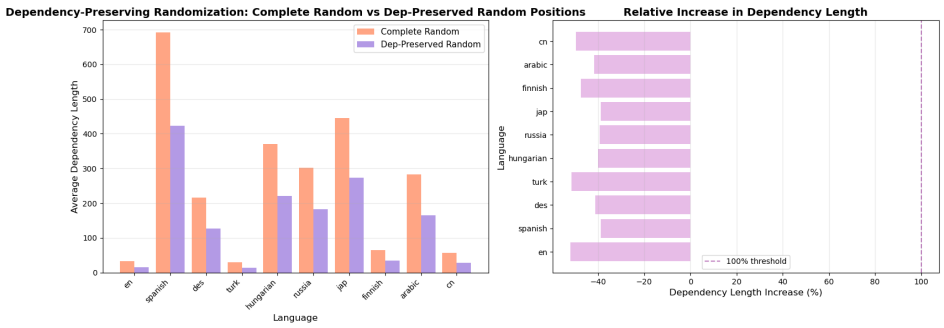


Figure 4: Statistical Summary: Dependency-Preserving Randomization relative to Complete Random. Dep-preserved increase compared to complete random: -44.1%. Minimum increase: -52.0% (en). Maximum increase: -38.7% (jap). All p-values < 0.05.

3.3 Information Locality: Surprisal as a Function of Memory Size

(Answer for Exercise 7)

Information locality tells than in constraint working memory, words close to the next work provide much more useful information than those who are far to the next word. And as memory size increases, the surprisal for the next word should decrase. We trained NNLM and RNNLM with varying memory sizes (M = 1 to 16) on natural, completely randomized, and dependency-preserved corpora across all 10 languages. For each configuration, we measured average surprisal on a held-out test set to evaluate how memory size affects predictive uncertainty.

Observations

Figure bellow presents the relationship between memory size and average surprisal across models and corpus types.

NNLM Results: Contrary to the theoretical expectation that longer memory should reduce surprisal (by providing more context), we observe an **overall increase** in surprisal as memory size grows. Averaging across all 10 languages, the natural corpus shows an increase from 1.29 (M=1) to 1.66 (M=16). This counterintuitive trend holds across all three corpus types (natural, complete random, and dependency-preserved).

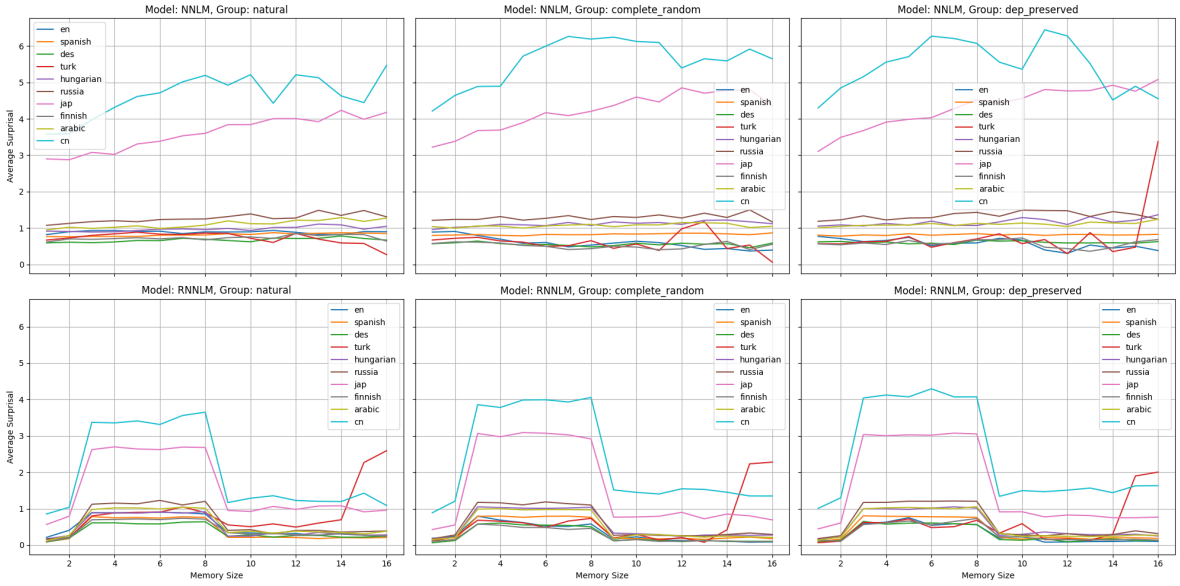
Why does this happen? As memory size increases, the number of possible n-gram combinations grows exponentially (V^n, where $V \approx 3400$ is the vocabulary size), while the training data remains fixed. This creates a severe **data sparsity** problem: the model encounters many unseen long n-grams during testing, leading to high uncertainty and thus high surprisal. Additionally, the NNLM's architecture—which concatenates embeddings of the past n tokens—suffers from a parameter explosion: the input dimension grows as emb_size $\times\, n$, but the hidden layer capacity remains fixed. This bottleneck prevents the model from effectively learning long-range dependencies.

RNNLM Results: The RNNLM, which processes sequences recurrently via an LSTM, exhibits a more nuanced pattern. Averaging across languages, we observe **three distinct phases**:

1. **Phase 1 (M=1 → M=2):** Surprisal rises sharply from 0.25 to 0.37 (+48%). This initial increase may reflect the model's struggle to integrate an additional token into its hidden state effectively.
2. **Phase 2 (M=3 → M=8):** Surprisal plateaus around 1.29–1.33, showing minimal variation. Despite increasing memory size, the model's performance remains stable, suggesting that the LSTM's hidden state saturates and cannot fully exploit the extended context. This may be due to:
    - **Limited hidden capacity:** With a fixed hidden dimension (128), the LSTM struggles to encode information from longer sequences.

- **Gradient issues:** Although LSTMs mitigate vanishing gradients, they can still struggle with very long dependencies, especially with limited training data.

3. **Phase 3 (M=9 → M=16):** Surprisal **declines** from 1.33 (M=8) to 0.66 (M=16).



Comparing Across Languages: Figure shows the percentage change in surprisal across three phases for each language. Most languages exhibit positive change in M1→M2 and M2→M8, but several (e.g., Chinese, Turkish, Japanese) show large increases, suggesting language-specific challenges in modeling long dependencies. In the M8→M16 phase, most languages show modest decreases, though the magnitude varies widely.

This heterogeneity complicates cross-linguistic comparison. Fixed vs. free word order does not clearly predict surprisal patterns. For instance, Turkish (free word order) shows an extreme 1505% increase from M=1 to M=16, while Finnish (also flexible) shows a moderate 155.5% increase. These differences likely reflect:

- **Corpus quality and size:** Smaller or less representative corpora hinder model training.
- **Tokenization mismatch:** Languages with complex morphology or non-Latin scripts suffer more from our simple whitespace tokenizer.
- **Intrinsic complexity:** Some languages may genuinely require more context for accurate prediction, independent of word order flexibility.

Problematics and Limitations

Is the comparison straightforward? No. Several challenges arise:

1. **Data sparsity:** As memory grows, the effective training data per n-gram shrinks exponentially. This confounds the relationship between memory size and surprisal, making it difficult to isolate the linguistic signal from implementation artifacts.
2. **Model capacity constraints:** Both NNLM and RNNLM use fixed hidden dimensions, which may be insufficient for large memory sizes. Ideally, model capacity should scale with memory to ensure fair comparison.
3. **Training regimen:** We used 16–50 epochs for NNLM and 40 for RNNLM. Large memory sizes may require many more epochs to converge, and early stopping may have prevented some models from reaching optimal performance.

4. **Language-specific factors:** Tokenization, corpus size, morphological complexity, and writing system all vary across languages, making it hard to attribute differences to word order properties alone.

Due to time constraints, we did not perform these tests.

Answer for Exercise 8

It's possible, we have pretrained multilingual models like mBERT, XLM-R, mT5, but there have several challenges. Since the models have already learned language-specific patterns, it's hard to separate "inherent language locality" from "learned bias".

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.