Block 4: Designing processes

B.3 CompMech

In this session, you will design your own hidden Markov model (HMM) and reuse code from previous exercises to explore its structure. Not all HMMs are interesting — use the Desiderata Section to guide your design toward processes with meaningful properties. You will then present your process to the class.

1. ***Form into pairs.*** *Find a partner to collaborate with on the design exercise.*
2. ***Design your process (30 minutes).*** *Using the Desiderata Section as your guide, construct an HMM. Use the code from previous exercises to visualise the next-token probability vector and the belief state geometries.*
3. ***Present your process to the class (2 minutes).*** *Describe your process and demonstrate its key features, including the observation probabilities, belief state geometry, and any notable structural properties.*

Notation

(Feel free to skip straight to the Desiderata section and refer back when necessary.)

Here, we consider data generated from a generalised hidden Markov model (GHMM), and introduce the following notation:

- $\mathcal{X}$ – a set corresponding to the dictionary ($|\mathcal{X}|$ = #tokens)
- $(T^{(x)})_{x\in\mathcal{X}}$ – transition matrices (#hidden states x #hidden states)
- $\eta^{(\emptyset)}$ – the initial state row vector (1 x #hidden states)
- ϕ – the final state column vector (#hidden states x 1).

We also introduce the following notation for token sequences, products of transition matrices and sequences of up to length L

$$x_{1:\ell} := x_1 x_2 \cdots x_\ell, \qquad T^{(x_{1:\ell})} := T^{(x_1)} T^{(x_2)} \cdots T^{(x_\ell)}, \qquad \mathcal{X}_{1:L} := \bigcup_{i=1}^{L} \mathcal{X}^i .$$

Given the observed token sequence $x_{1:\ell} \in \mathcal{X}^\ell$, the *belief state* can be expressed in terms of HMM data as

$$\eta^{(x_{1:\ell})} = \eta^{(\emptyset)} T^{(x_{1:\ell})} / \eta^{(\emptyset)} T^{(x_{1:\ell})} \phi \, .$$

Similarly the vector of next token *observation probabilities* is given by

$$p^{(x_{1:\ell})} = (\eta^{(x_{1:\ell})} T^{(x_{\ell+1})} \phi)_{x_{\ell+1} \in \mathcal{X}}.$$

Consulting the previous two expressions, we see that there exists a matrix A that takes beliefs $\eta^{(x_{1:\ell})}$ to observation probabilities $p^{(x_{1:\ell})}$, i.e.

$$p^{(x_{1:\ell})} = \eta^{(x_{1:\ell})} A$$

where A is a matrix whose elements are given by

$$A_{ij} := \sum_k T^{(j)}_{ik} \phi_k.$$

We note that for GHMMs, there always exists this linear map from beliefs to observation probabilities, however, it is generically not the case that this map is invertible.

Desiderata

We now list a series of properties (in decreasing level of importance) that we would like our process to have.

No affine map from observation probabilities to beliefs. One of the central claims of comp-mech applied to neural networks is that beliefs are instrumental to predicting observation probabilities. If there exists an affine map from observation probabilities to beliefs, then, at the level of linear representations, these objects are indistinguishable and the claim of comp-mech is empty. Concretely, the desired property is that the matrix A is not invertible. A nice heuristic for this is that #hidden states > #vocab.

We may also want to require that there exists no linear map from observation probabilities to *approximate* beliefs. One way we might quantify this is to compute the R^2 of the regression from observation probabilities to beliefs, and ensure that it satisfies:

R^2(obs probs -> beliefs) $<$ R^2(activations -> beliefs)

Intuitively, this constraint ensures that the transformer's representation of beliefs cannot be simply explained by the model representing obs probs.

In practice, I like to select processes where # hidden states > # vocab, and where the beliefs occupy a (# hidden states - 1)-dimensional region of the simplex. I also like to indicate, on the belief simplex, the directions in the kernel of the map from beliefs to observation probabilities. This allows one to see which belief states collapse to the same observation probabilities.

Easy to visualise. It is convenient to be able to visualise the ground truth belief states, observation probabilities and log observation probabilities without having to use dimensionality reduction techniques. As such, we have the condition:

hidden states ≤ 4.

As we are also (likely) imposing that # hidden states > # vocab, the dimensions of observation probabilities and log observation probabilities are readily handled. In practice, I like # hidden states = 3 & # vocab = 2.

Striking belief geometry. It is helpful to be able to quickly glance at e.g., the PCA of activations, and be able to identify patterns that correspond to some structure in the belief geometry. As such, it is nice if the belief geometry has some particular structure that distinguishes it from the observation probabilities.

Natural properties. While HMMs lack expressivity of other non-linear models, one can still endow them with properties shared by natural language, for example:

- **Factors.** Natural language often consists of many distinct, independent components that each influence the output distribution. For example, a character's name determines which pronouns appear; the setting (beach vs. courtroom) shapes vocabulary; the genre (horror vs. comedy) affects tone; and the tense constrains verb forms. These components combine to determine what text is likely to follow, yet operate largely independently – knowing a story is set on a beach tells you little about whether the protagonist is named Alice or Bob.
- **Non-ergodicity.** The training corpus of LLMs is a mixture of text from diverse sources (novels, code, lawsuit filings, and more), each with its own distinct statistical structure. This can naturally be characterised as a non-ergodic process that samples from a distribution over a set of component processes (which may themselves be non-ergodic). Each component process corresponds to a broad category of text: one governing how novels are generated, another how code is written, another how legal documents are structured, and so on.
- **Algorithms.** Many simple algorithms appear naturally in language. For example, answering "two days after Saturday is?" requires modular addition over days of the week, while "what is the reverse of 'cat'?" requires a reversal operation. The core logic of such algorithms can be encoded as hidden Markov models (e.g., modular addition for the days-of-the-week example), making HMMs a natural framework for studying how language models learn and represent these computations.