Autoregressive Models: Inductive Biases and Projections

Abstract

This paper explores the use of inductive biases and projection functions in autoregressive (AR) models to enhance out-of-distribution (OOD) generalization. We revisit the concept of infini-grams, which leverage suffix arrays to manage arbitrary input (context) lengths efficiently. This approach is compared to traditional n-gram models, highlighting its advantages in sample efficiency and computational scalability. We delve into various inductive biases, such as the recency bias, shortest edit distance, and semantic similarity, illustrating their impact on AR model performance. By framing OOD generalization as a projection problem, we propose strategies to optimize these projections through meta-learning and nested optimization. Furthermore, we discuss the integration of classical information retrieval techniques and pre-trained language model embeddings to enhance the semantic relevance of projections. Our findings suggest that combining symbolic AI methods with deep learning representations can yield more interpretable and sample-efficient AR models, with broad applications in natural language processing, code generation, and scientific discovery.

Introduction

The infini-gram model is an autoregressive (AR) model that predicts the next token based on the longest suffix in the training data that matches the input. Essentially, they are finding some *projection* of the input to the training data to allow the AR model to generate coherent text continuations from inputs it has never seen before. This is known as out-of-distribution (OOD) generalization, where we are trying to generalize to tasks (like predict continuations of an input never seen before) that is not in the training data.

Since the model converges in distribution to the data generating process (DGP) as the sample size goes to infinity, the key challenge is to find sample-efficient inductive biases that provide the model with more information about the task or the DGP, allowing it to generalize to OOD data more effectively and with fewer samples.

In this paper, we seek to formalize a class of inductive biases as *projections* of the input onto the training data.

AR Models

AR models form a cornerstone in natural language processing, predicting the probability of a word w_t given all preceding words $w_{<t}$ and the training data D:

$$\Pr_D\{w_t \mid w_{<t}\}.$$

Historically, the prefix $w_{<t}$ is limited to a fixed length n,

$$\Pr_D\{w_t \mid w_{t-n:t}\},$$

where $a : b$ denotes the range $a, a+1, \ldots, b-1$.

Infini-gram models dynamically adjust the order of the n-gram based on the longest suffix in the training data that matches the input:

$$\Pr_D\{w_t \mid \text{longest\_suffix}_D(w_{<t})\},$$

where longest_suffix_D finds the longest suffix of the context $w_{<t}$ in the training data D.

For AR models to generate continuations of the input, longest_suffix makes a lot of sense. It allows the model to find training data that is both similiar to the input and relevant to the task of predicting the next token from previous tokens.

Let's be a bit formal about what longest_suffix_D represents: it is a kind of *projection* of the input onto the training data D, which is an i.i.d. sample from some (unknown) data generating process (DGP). Let us denote the probability distribution of the DGP as $\Pr_\theta$, where θ are unknown parameters, and the probability distribution of the AR model as $\Pr_{\hat{\theta}}$, where $\hat{\theta}$ are the estimated parameters of the AR model based on the training data D.

The goal of the AR model is to estimate θ from the training data, which will allow it to generalize to new data that the DGP would plausibly produce. A paricularly useful task is to predict what the DGP would plausibly produce *given* some input $w_{<t}$, where $w_{<t}$ is a sequence of tokens that the DGP has produced so far and may represent some task of interest, like "What is the solution to <math problem>?"

The distribution of $w_{t:t+k}$ conditioned on $w_{<t}$ is given by

$$\Pr_\theta\{w_{t:t+k} \mid w_{<t}\} = \frac{\Pr_\theta\{w_{1:(t+k)}\}}{\Pr_\theta\{w_{1:t}\}},$$

where $w_{a:b}$ is a sub-sequence of tokens produced by the DGP from time a to time b (time is a *logical time* that just implies some ordering). The primary task is often to *generate* plausible continuations of the input, for which there are many possible *sampling* strategies to do this, like beam search, top-k sampling, and nucleus sampling, all of which use the conditional probability distribution to generate continuations one token at a time. This approach is justfied by the chain rule of probability:

$$\Pr_\theta\{w_t \mid w_{<t}\} = \prod_{i=1}^{t} \Pr_\theta\{w_i \mid w_{<i}\}.$$

Notice that when we generate continuations of the input, we are not trying to find a sequence that *maximizes* the conditional probability:

$$w^*_{t:(t+k)} = \arg\max_{w_{t:(t+k)}} \Pr_\theta\{w_{t:(t+k)} \mid w_{<t}\},$$

but rather we are *sampling* from the distribution. We identify a few justifications for doing this:

1. The DGP $\Pr_\theta$ is often stochastic and we capture this stochasticity in our predictions or continuations. However, even if the DGP is not stochastic, we only have an uncertain estimate $\Pr_{\hat{\theta}}$ conditioned on data D randomly sampled from data by the DGP. So, sampling from it is a way of generating continuations that reflect the uncertainty. See Appendix F: Bootstrapping the Sampling Distribution for a more rigorous way to estimate uncertainty in the model as opposed to the DGP.
2. There is a trade-off between exploration and exploitation, where the model needs to balance between generating plausible continuations and exploring new possibilities.
3. Finding the most likely sequence of tokens is NP-hard, so we often resort to approximate methods like greedily sampling from the conditional distribution one token at at time, or using more accurate but computationally expensive methods like beam search to find more likely sequences of tokens.

Since we do know know the DGP $\Pr_\theta$, we replace it with our AR model based on a training data D, $\Pr_{\hat{\theta}}$, and use the AR model to approximate the DGP. As the sample size goes to infinity, by the law of large numbers, the empirical distribution of the training data will converge to the true distribution of the DGP:

$$\Pr_{\hat{\theta}}\{w_t \mid w_{<t}\} \to_d \Pr_\theta\{w_t \mid w_{<t}\}.$$

The Infini-gram model converges in distribution to the DGP, but we do not have *infinite* data. Thus, since virtually all inputs have never been senn before, we are interested in finding ways to allow the model to generalize *out-of-distribution* (OOD). On the task of next-token prediction, this means generating continuations of the input that the DGP would plausibly produce but are not in the training data.

This is a key challenge in machine learning. Ideally, we want the AR model to generate plausible continuations of any input from very small amounts of

training data D. A primary way to do this is to *constrain* or *bias*, which we call an *inductive bias*.

The projection function longest_suffix$_D$ is an example of an inductive bias. It is a way for the model to find the most relevant part of the training data to the input to give it some ability to generalize OOD on the task of generating plausible continuations of the input.

We formalize this idea of projection as an inductive bias and discuss how it can be used to improve the sample efficiency of both n-gram models and AR models, like transformers, LSTMs, and RNNs.

Inductive Biases

Given two learning algorithms, A and B, if A requires fewer samples to do well on a task than B, then A is more sample-efficient than B on that task. In the context of n-gram models, the task is to predict the next token given a sequence of previous tokens. One way to improve sample efficiency is to choose an inductive bias that provides the model with more information about the task or the DGP, allowing it to generalize to OOD data more effectively and with fewer samples.

The longest_suffix$_D$ projection is an inductive bias that we might label the *recency bias*. The recency bias has some advantages:

1. It is computationally efficient, as shown by the suffix array data structure used in the infini-gram model. It only requires a linear scan of the training data to find the longest suffix. This scalability is crucial for training on large datasets, as the time complexity of the recency bias is $O(n)$, where n is the length of the context.
2. It corresponds to a simple inductive bias that is easy to understand, implement, and justify. If the future is like the past, then the most recent past is often the most relevant data point. This is particularly relevant for tasks like language modeling, where the context is often a sequence of words that are related to each other in a temporal order and in which the most recent words are often the most relevant for predicting the next word.

The recency bias may not always help to find the most relevant context in the training data, e.g., the most relevant context may be at the start of a document. However, even when the most relevnat context is the most recent, the longest_suffix may fail to properly use it. For example, if the context is `the dog ran after the` and we ask it to predict the next word, but the training data only contains `the dog chased the cat`, the longest suffix is the highly uninformative word `the`. We see that the naive longest suffix match fails to take into account slight variations, even if those slight variations have essentially identical meanings.

These challenges suggest some possible inductive biases that can be used to improve the OOD generalization of $\Pr_{\hat{\theta}}$. We consider the set of inductive biases that can be formulated as *projections* of the input (context) onto the training data. Let us formally write down the problem of OOD generalization in the context of AR models as a projection problem:

$$\Pr_D\{w_t \mid \text{proj}_D(w_{<t})\},$$

where proj_D is a function that maps the input $w_{<t}$ to a subset of the training data D that is most relevant for producing continuations of the $w_{<t}$ that the DGP would *likely* produce.

Learing the Projection Function

Let us parameterize the projection function as

$$\mathcal{F} = \{\text{proj}_D(x; \beta) \mid \beta \in \mathcal{B}\},$$

where β is an index or label that specifies the projection. For example, $\beta = 1$ could be a label for longest_suffix_D, or it could be something more complicated based on the space of possible projections $\mathcal{F}$.

We can choose a projection function from $\mathcal{F}$ by choosing a β in $\mathcal{B}$, which is frequently a discrete set of possible projections.

We choose the projection function in one or two ways:

- Utilize domain-knowledge expertise (hand-crafted feature engineering). By lessons of the bitter kind, we observe that this appraoch often does not scale with increasing compute and data, as it requires human expertise that is often scarce and limited.
- Treat it as an optimization (search or learning) problem, where β is a tunable parameter of the model.

The second approach is more general and can be used to optimize the projection function based on the data D and the task we are measuring performance on. Note that because the projection function $\text{proj}_D(\cdot; \beta)$ is intended to improve OOD generalization performance, we do not optimize it on the training data D but on a held-out test data D'.

The optimization problem is then conceptualized as an iterated two-stage process.

1. **Initialize:** Set i to 1 and choose a β_0 based on prior knowledge.
2. **Stage 1:** Optimize the parameters of the AR model θ on the training data D using $\text{proj}_D(\cdot; \beta_{i-1})$:

$$\hat{\theta}_i = \arg\max_{\theta} \prod_{t=1}^{T} \Pr_{\theta}(w_t \mid \text{proj}_D(w_{<t}; \beta_{i-1})).$$

3. **Stage 2:** Optimize the parameters of the projection function indexed by β_i on the test data D' using the AR model indexed by $\hat{\theta}_i$:

$$\hat{\beta}_i = \arg\max_{\beta} \prod_{t=1}^{T} \Pr_{\hat{\theta}_i}(w_t \mid \text{proj}_{D'}(w_{<t}; \hat{\beta}_i)),$$

   where D' is test data D' (e.g., held-out test data) used to estimate the quality of the projection function.

4. **Convergence Test:** If the parameters $\hat{\theta}_i$ and $\hat{\beta}_i$ have converged, stop. Otherwise, set $i = i + 1$ and go to step 2 (Stage 1).

To mitigate overfitting on the test data, we can use strategies like early stopping, where we stop the optimization process before convergence, or choose different test data at each iteration.

If the set of projection functions do not affect the performance of the AR model on the training data D, then convergence is obtained after one iteration. Since the parameters β and θ are usually disjoint (they do no share parameters), the primary way in which a projection function can affect the performance of the AR model on the training data is by changing the distribution of the training data that the AR model sees. For instance, β may include a parameter that limits the maximum length of the context, which can change the parameters of the AR model that are estimated from the data.

Next, we consider the space of possible projection functions $\mathcal{F}$.

Hypothesis Space of Projection Functions (Inductive Biases)

To formalize notation, we denote the space of projection functions $\mathcal{F}$ with the type

$$\mathcal{T}^* \mapsto \mathcal{T}^*,$$

where $\mathcal{T}$ are the set of *tokens* (words, characters, etc.) and $\mathcal{T}^*$ is the set of sequences of tokens. The projection function proj_D maps a sequence of tokens to another sequence of tokens, which is (ideally) a subset of the training data D such that the Infini-gram model can generate plausible continuations of the input based on suffix matches in the training data.

This space is of course too large to search over, so we need to make some assumptions about the structure of the space of projection functions. We can

consider a few simple projection functions that can be used to improve the sample efficiency of AR models:

1. **Recency Bias:** The recency bias is a simple projection function that finds the longest suffix of the input in the training data. It is a kind of *greedy* projection that assumes the most recent tokens are the most relevant for predicting the next token. The recency bias is a simple and computationally efficient inductive bias that can be used to improve the sample efficiency of AR models. This is the *default* behavior of the Infini-gram model, and by construction all other projection functions incorporate the recency bias.
2. **Similarity Bias:** The similarity bias is a more complex projection function that finds the most similar sequence in the training data to the input. Because this could distort the input too much, we constrain the similarity bias to only apply so-called suffix extensions to the left.

We can draw inspiration from techniques developed in information retrieval (IR), natural language processing (NLP), and classical AI informed search strategies to design projections (inductive biases) that yield more sample efficient algorithms that improve OOD generalization. It is worth pointing out that in high-dimensional spaces, essentially every input is OOD, so designing effective inductive biases (projections) is crucial for generalization.

Since the longest suffix projection function is already given, in the next section we consider ways to extend approximations of the input suffix to find longer and potentially more relevant context in the training data.

Extending The Suffix

When we project the input onto the training data and obtain the longest matching suffix, we necessarily lose information about the rest of the input.

We have a predictive model, the Infini-gram model itself, that can be used to go extend the suffix in a way that projects onto the training data.

Let us formalize this. We have an input $w_{<t}$ and a training data D. We project the input onto the training data to find the longest matching suffix $w_{t':t}$ in the training data, where $t' \leq t$.

We know that $w_{t'-1:t}$ does not match the training data D but the suffix $w_{t':t}$ does. Thus, $w_{t'-1}$ needs to be substituted for a different token for the suffix to have a chance at finding a match in the training data.

Let us denote this $t'-1$-th token as $w'_{t'-1}$. It is a random variable that we can sample from the AR model. That is, we can use the AR model to compute the conditional probability of $w'_{t'-1}$ given $w_{t':t}$ as a way of sampling extensions of the suffix to the left:

$$\Pr_{\hat{\theta}}\{w_{t'-1} \mid w_{t':t}\} = \frac{\Pr_{\hat{\theta}}\{w_{t'-1:t}\}}{\Pr_{\hat{\theta}}\{w_{t':t}\}}.$$

We can compute joint probabilities using the AR model, and thus we can use the AR model to cosnider realizations of $w'_{t'-1}$ given $w_{t':t}$ that the model (training data) would likely produce.

This is mostly a *computational* trick, since we do not want to randomly sample tokens that are unlikely to be produced by the DGP (and thus unlikely to project onto the training data).

We may rewrite the conditional probability as:

$$\Pr_{\hat{\theta}}\{w'_{t'-1} \mid w_{t':t}\} \propto \Pr_{\hat{\theta}}\{w_{t'-1}\} \prod_{i=t'}^{t} \Pr_{\hat{\theta}}\{w_i \mid w_{t'-1:i}\},$$

which is something that the Infini-gram model can compute very efficiently. Thus, we can generate the conditional distribution of $w_{t'-1}$ given $w_{t':t}$ and sample from this distribution to consider suffix extensions.

However, we have to have some similarity measure to determine when to stop extending the suffix, as we may end up with a very long suffix that is not very relevant to the input. We can use the earlier similarity measures discussed.

We can sample multiple left-extensions of the suffix, compute the similarity of each extension to the input, and use some strategy to either stop extending the suffix or to accept an extension based on the similarity to the input, such as $\arg\max$ or sampling based on the similarity.

Challenges

Learning sample efficient representations of the data is the primary driver of OOD generalization. *Deep Learning* is about learning these representations from the data. We can use pre-trained models like BERT and GPT to learn representations of the data (sequences of tokens), also known as embeddings, that are a more sample-efficient representation than the token sequences in our Infini-gram model.

In particular, these embeddings can be used to compute the similarity between tokens. A canonical example is `word2vec`, which learns an embedding of words that allows a kind of semantic algebra on words such linear combinations of embeddings often result in meaningful embeddings. The canoncial example is:

$$\text{embed}(\text{king}) - \text{embed}(\text{man}) + \text{embed}(\text{woman}) \approx \text{embed}(\text{queen}).$$

We can use these embeddings to compute the semnatic similarity between tokens, and thus try to find suffix extensions of the input that oth retain the meaning of the input and project onto the training data.

Sequence Embeddings Suffix extensions using token embeddings like `word2vec` may be too simplistic, as they operate at the level of atomic tokens. Most of the *meaning* of a sequence of tokens is in the relationships and order of the tokens, not just the tokens themselves. This is a well-studied problem in NLP, and there are many models that model the semantics of a language, from classical models

Computational Complexity If we use LLM embeddings, it may be costly to compute the similarity between the input and all segments in the training data. We could, however, take the training data and compute embeddings for each segment and store them in a vector storage database for fast retrieval:

$$\text{proj}_D(x;\beta) = \arg \max_{y \in \text{segments}_\beta(D)} \text{similarity}_\beta(\text{embed}(x), \text{embed}(y)),$$

where embed is a function that maps tokens or sequences to embeddings. Since we have all of the embeddings in a vector storage database, the above arg max operation can be computed very efficiently at the cost of precomputing and storing the embeddings.

Uncertainty Estimation

While infini-gram models provide point estimates for token probabilities, understanding the uncertainty in these estimates is crucial for robust decision-making and for gaining insights into model confidence.

We can apply bootstrapping to Infini-gram models by repeatedly sampling with replacement from the training data to create multiple bootstrap samples. For each sample, we train an Infini-gram model and use it to produce next-token probabilities for a given input. This process allows us to construct confidence intervals for our probability estimates.

More precisely, we estimate the *sampling* distribution of the model estimate by resampling from the data D. The bootstrapped sampling distribution is given by

$$\{\hat{\theta}^j\}_{j=1}^R,$$

where R is the number of resamples (with replacement) from D and

$$\hat{\theta}_b^j = \arg\max_\theta \prod_{t=1}^T \Pr_{D^j}\{w_t \mid w_{<t}\},$$

is the j-th estimate based on the resampled data and D^j is the j-th resample of the data D. Since $\widehat{\theta}^j$ is an estimate of the model parameters based on the resampled data D^j, by the plug-in principle, we can estimate the sampling distribution of the model as

$$\left\{\Pr_{\hat{\theta}^j}\right\}_{j=1}^{R}.$$

Confidence Intervals

To generate confidence intervals of, say, the predictive distribution of the model, we can sample a model from the sampling distribution and provide the input to the model to produce the set of next-token probabilities. We can do this B times to get a set of B next-token probabilities, and thus for each next-token, we can generate a confidence interval for its probability.

This is not easy to do with the neural language models because they are computationally expensive to train. For the Infini-gram model, we can just resample the documents in the training data D.

Implications for LLMs

Interestingly, the confidence intervals derived from bootstrapped infini-gram models may provide valuable insights into the uncertainty of larger language models (LLMs) trained on the same data. While LLMs are more complex and capture higher-order dependencies, the fundamental uncertainties present in the training data should affect both types of models. For instance, if an infini-gram model shows wide confidence intervals for certain contexts or token predictions, it suggests high variability or insufficient data in those areas. An LLM trained on the same data might also struggle with these contexts, even if it doesn't explicitly compute confidence intervals. This connection opens up possibilities for using simpler, more interpretable models like infini-grams as proxies for understanding the uncertainties in more complex models. It could provide a computationally efficient way to estimate when an LLM might be less confident, without needing to compute expensive uncertainty estimates directly on the LLM itself.

Consider a scenario where both an infini-gram model and an LLM are trained on a corpus of scientific papers. If the infini-gram model shows wide confidence intervals when predicting terms in a specific scientific domain, it might indicate that the LLM should also be less confident when generating content in that domain, even if the LLM doesn't explicitly calculate confidence intervals.

While this approach shows promise, it's important to note that the relationship between infini-gram uncertainty and LLM uncertainty is not guaranteed to be straightforward. Factors such as the LLM's ability to leverage long-range dependencies and its more complex training process may lead to divergences. Future work could involve empirically studying the correlation between infini-gram confidence intervals and LLM performance or uncertainty estimates derived through

other means.

Experimental Results

We can compare the performance of the recency bias, shortest edit distance, and semantic similarity bias on a language modeling task. We can use perplexity as a measure of the model's performance on the task, where lower perplexity indicates better performance.

Python Code

```
# code here
```

For more details, see the GitHub repository for this project.

Conclusion

We have reframed of OOD generalization in the context of AR models as a context reduction and matching problem and explored various inductive biases to improve sample efficiency.

Even hand-crafted inductive biases like the recency bias and similarity bias can significantly enhance AR models' performance, but utilizing learned embeddings from pre-trained models like BERT and GPT can likely yield more effective results.

In either case, we see that the goal is to reduce or rewrite the context to increase the probability of finding a match in the training data. This is a discrete optimization problem that can be solved using techniques from information retrieval and natural language processing, and so we can leverage these techniques to design more sample-efficient learning algorithms that search over the space of possible context reductions and rewrites to find the most relevant training data to facilitate OOD generalization.

Even if an exact match is found in the training data, we often still want to explore a larger space of possibilities to make new discoveries and generate novel and creative outputs.

Further research into optimizing these techniques and seamlessly integrating them into AR frameworks promises to advance natural language processing, driving innovation in computational linguistics and machine learning, and help facilitate the development of more intelligent and creative AI systems that can be more easily explained and understood than current neural models, which are often seen as black boxes with inscrutable decision-making processes.

By leveraging a LLMs embeddings for sample efficient representatations and classical symbolic AI techniques for context reduction and matching, we can build more interpretable and efficient AR models that can be used in a wide

range of applications, from chatbots to code generation to scientific discovery and beyond.

Appendices

A: Reward Functions

Previously, we discussed the idea of projecting the input onto the training data to find the most relevant context for predicting the next token that the DGP is likely to produce.

However, we are normally not interesed in predicting the next token, but *biasing* the model to generate outputs that are more likely to score well on some task. For example, if the task is to generate a coherent text continuation, we want to bias the model to generate text continuations that are coherent and correct, even if the the DGP, given the input, is more likely to generate very different continuations (e.g., toxic, incoherent, or incorrect).

One way to fine-tune the model to generate more effective outputs is to use a reward function that scores the outputs based on some task-specific criteria. A nearly universal approach is to take your reward function and produce k samples from the model, then score each sample using the reward function, and then sample these outputs based on their scores, e.g., $\Pr_{\hat{\theta}}\{w_{t:(t+k)} \mid w_{<t}\} \propto \exp\{R(w_{1:(t+k)})\}$.

However, what if we want to go beyond predicting the DGP's next token and instead generate outputs that are more effective at solving a particular task?

B: Data Transformation

The training data D is the primary source of information we have about the DGP. However, the training data may not be in the optimal form for solving the task we have in mind.

So, a final inductive bias we can consider is data transformation. We can transform the training data into a more suitable form for solving the task at hand. One way which can be particularly effective at improving the sample efficiency of the model is to transform the training data into a more abstract representation space.

There are a lot of fancy things you can do, but in interest of transparency and interpretability, we will consider a simple transformation: stemming or lemmatization.

Both of these are computationally efficient techniques that reduce the vocabulary size and thus increase the probability of finding relevant projections of the input onto the training data. They are also simple to implement and understand, making them a good choice for a first pass at data transformation.

Essentially, this transformation allows for "reasoning" over more abstract representations of the DGP, facilitating OOD generalization but at a loss of some information and expressiveness.

> Predictive modeling now takes place over this more abstract representation space, which can be more sample efficient. However, when we generate sequences, if the end product is, say, high-quality text, we may have to decode the stemmed or lemmatized representations back to unstemmed or unlemmatized forms, which can be a challenge.

C: Other Kinds of Inductive Biases

We formalized most of our inductive biases as projections of the input onto the training data. However, we can consider other kinds of inductive biases that can be used to improve the sample efficiency of AR models that directly affect the AR model's probabilty distribution.

In particular, we can also consider more complex inductive biases that involve pattern matching and context-free grammars. For example, we can use a context-free grammar to define the space of possible continuations of the input.

In theory, we could have a number of production rules on the input that restrict the set of possible continuations to some CFG. This is a more hand-crafted inductive bias that requires more domain knowledge and human expertise to implement, but it may be appropriate in some circumstances, such as when the task requires generating text that follows a specific structure or format, like JSON or Python code.

D: Similarity Bias: Shortest Edit Distance

Shortest edit distance finds the shortest sequence of operations (e.g., swaps, insertions, deletions, and substitutions) transforming the current context $w_{<t}$ into a sequence in D.

The recency bias can be seen as a special case of the similarity bias where we only allow deletions from the end of the context until a match is found.

A justification for using shortest edit distance is based on the idea of similarity: if two sequences are similar, they are more likely to have similar continuations. Edit distance is a way to measure the similarity between two sequences based on the minimum number of operations needed to transform one into the other.

Example Let's use the earlier example, where we have as input "the dog ran after the" and the training data D contains the following:

1. "a dog chased the cat in the garden"
2. "the dog chased the cat, but the cat climbed a tree and got away"

3. "the mouse ran from the cheese trap after setting it off"

The longest suffix is "the". What is shortest edit distance to find a match in the training data? We can perform the following two edits: substitute "ran" with "chased" and delete "after", resulting in "the dog chased the" which has a longest suffix match equal to "the dog chased the" (2), and so the next token predicted is "cat". Thus, a completion by the model might be: "the dog ran after the cat, but the cat climbed a tree and got away". The training data does not contain this completion, but the shortest edit distance found a *relevant* projection to the training data.

What else could we have found within two edits? We could have substituted "dog" with "mouse" and "after" with "from", resulting in the input "the mouse ran from the" and the completion "the dog ran after the cheese trap after setting it off". This is a less plausible completion for the DGP, and things could go much worse, but we see that just counting the number of edits can lead to a poor projection onto the training data.

Challenges As the example demonstrated, a primary challenge with shortest edit distance is that it treats all single edits as having a uniform cost. Ideally, when we edit the input, we want to preserve the "meaning" of the context. Thus, some edits should be more costly than others, based on how much they change the meaning of the input.

Also, the shortest (least-cost) edit distance, particularly when we combine it with non-uniform costs, is more computationally intensive than the longest prefix projection. However, it is tractable, e.g., graph search in GOFAI. Approximate methods like Monte Carlo Tree Search (MCTS) can also be used to find approximate solutions.

E: Semantic Similarity: Least-Cost Edit Distance

A significant issue with *shortest edit distance* is that it treats each edit as having a uniform cost (a kind of uninformed search). A simple extension is to add a cost to each edit based on some measure of semantic similarity between tokens or sequences.

Once we have a cost in place, we can use classical search techniques, like A* search, to find relevant sequences in the training data to the current context.

Classical IR (Information Retrieval) techniques like BM25, query expansion, and semantic similarity measures can be used to assign costs to edits.

1. Input expansion (query expansion): Expand the input to multiple possible sequences that are similar to the input.
2. Treat the input like a search query in IR and use BM25 or other similarity measures to find the most similar sequences in the training data. We then define the projection function as:

$$\text{proj}_D(x;\beta) = \arg \max_{y \in \text{segments}_\beta(D)} \text{similarity}_\beta(x, y),$$

where $\text{segments}_\beta(D)$ is a segmentation strategy of the data D into segments (e.g., sentences paragraphs) and similarity_β is a similarity measure between the input x and the segment y, e.g., cosine similarity or Euclidean distance between embeddings or some tf-idf measure, like BM25.

We can use the inferred β to find the most relevant segments in the training data to the input, and then use the AR model to generate continuations of the input based on these segments.