Evaluating and Interpreting Transformers for Sequence Alignment

Candidate number: 1091469
Word count: 7242
MSc Mathematical Sciences

Trinity Term, 2025

Abstract

This work evaluates and interprets an encoder-decoder transformer model for pairwise DNA sequence alignment, using synthetic data generated from a pair Hidden Markov Model (pair HMM) as ground truth to better understand the black-box nature of deep learning approaches to alignment. We assess the transformer model's ability to replicate optimal alignments and probability distributions defined by the pair-HMM, employing constrained beam search and comparing output distributions. Mechanistic interpretability techniques, including linear probing and LogitLens, are used to analyse internal computations including inference of evolutionary parameters. Results indicate the transformer effectively approximates pair-HMM likelihoods and distributions on synthetic data, with linear probes confirming its capacity to implicitly learn evolutionary parameters almost as accurately as is theoretically possible. This interpretability analysis offers insight into the model's computational mechanism, enhancing confidence in the models probability distribution over alignments but casting doubt on the efficacy of beam search.

Contents

# Chapter 1

Introduction

1.1 Motivation

Sequences are fundamental to life on Earth. DNA sequences are transcribed into proteins, which can be described in turn as sequences of amino acids. These sequences evolve in three main ways: insertions, deletions, and substitutions of sequence positions. In closely related organisms, sequences share many homologous positions (positions that are retained from a common ancestor). Identifying these homologous positions in sequences and tracking how sequences have changed is crucial in biological research. This can be done by *aligning* sequences: strategically adding gaps so that the homologous sites align.

For example, for the DNA sequences $ATGC$ and AGC, a plausible alignment is, suggesting that the T was either inserted into the first sequence or deleted from the second sequence:

$$\begin{array}{cccc} A & T & G & C \\ A & - & G & C \end{array}$$

Established algorithms exist both for computing alignments that are optimal by some measure (Needleman and Wunsch, 1970) and for performing statistical inference given sequences and a model of how sequences evolve (Thorne et al., 1991; Holmes and Bruno, 2001; Hein, 2001). However, these require knowledge of parameters such as the rate of substitutions of nucleotides or amino acids and these parameters can vary between different phylogenetic groups. This challenge has a parallel in the field of language modelling, where context (such as the language of a text) determines word probability distributions. In natural language processing, transformer models (Vaswani et al., 2017) have been found to be effective at implicitly inferring the context of an input text and generating likely continuations conditional on the context.

Dotan et al. (2025) train a model they name BetaAlign with the encoder-decoder transformer architecture (Vaswani et al., 2017) to autoregressively generate a multi-sequence alignment conditional on the non-aligned sequences. Surprisingly, they find that this model matches and sometimes exceeds the performance of existing approaches to alignment!

Despite BetaAlign's impressive performance, the intermediate computations performed remain opaque. This makes it difficult to rely on the output of the model. This limitation of transformer-based approaches is especially pressing because similar approaches are being applied to many types of biological data; see, e.g., Greenshields-Watson et al. (2025).

In order to address this issue, this work evaluates the model when trained on synthetic data generated from a pair Hidden Markov Model (pair-HMM), and applies mechanistic interpretability techniques to analyse the model's internal computation. We also evaluate how the probability distribution over alignments that the model predicts compares to the true distribution (which can be tractably computed if the data-generating process is a pair-HMM).

This work focuses exclusively on pairwise alignment, the case where there are only two input sequences. While typical downstream tasks require aligning many sequences (multiple sequence alignments, or MSAs), the conceptual investigation is more easily performed in the pairwise case and many ideas straightforwardly generalise to more sequences. We also focus on DNA sequences rather than proteins, because this means we need only deal with four base pairs rather than 20 amino acids.

1.2 Related work

There is an extensive literature on each of sequence alignment and neural network interpretability. This section surveys and contrasts the most relevant previous work.

1.2.1 Sequence alignment

Sequence alignment began to be computationally tractable when Needleman and Wunsch (1970) introduced the Needleman-Wunsch algorithm (discovered several times independently in various applications), which determines the optimal alignment according to an objective function that assigns a cost to each column of the alignment. It does this with a dynamic programming algorithm that avoids the prohibitive cost of evaluating all possible alignments of sequences, which would be super-polynomial in time complexity.

A shortcoming of the Needleman-Wunsch algorithm and its variants is that it yields only a single alignment, rather than expressing uncertainty over the set of possible alignments. Additionally, its parameters (i.e., the column costs) do not straightforwardly relate to specific evolutionary processes.

The field of statistical alignment seeks to remedy this by introducing probabilistic models of sequence evolution. A crucial insight is that to determine the "true" alignment between two sequences, it suffices to know what the most recent common ancestor of the two sequences was, and what sequence of insertion, deletion, and substitution events occurred between the ancestor and each of the two descendants. Because the parameters of the models now correspond to the rates of evolutionary events, they can be set in a more principled manner.

A foundational model in this statistical alignment was introduced by Thorne et al. (1991), often referred to as the Thorne-Kishino-Felsenstein (TKF91) model. The model describes sequence evolution as a continuous-time Markov process: it describes how a sequence is likely to change in the future given its current state and some evolutionary parameters such as the rate of substitutions between base pairs and the rate of insertion and deletion events. A feature of the model is time reversibility.

To perform inference using such evolutionary models, a standard framework is the *Pair Hidden Markov Model (pair HMM)* (Durbin et al., 1998). Pair HMMs directly model the probability of a pair of sequences evolving with a specific alignment. The parameters of a pair HMM can be derived directly from the evolutionary rates specified by models like TKF91; and related computations were substantially accelerated by Hein et al. (2000). Pair HMMs are used to generate the synthetic dataset of pairs of aligned sequences, and their key benefit is that they enable the following computations:

- The likelihood of two sequences occurring with a specific true alignment, via a formula (derived in e.g., the introduction of Hein (2001));
- The maximum-likelihood alignment given a pair of sequences, via the Viterbi algorithm (invented independently many times, and analogous to Needleman-Wunsch algorithm);
- The likelihood of two sequences occurring, via the forward algorithm.

The established software packages for sequence alignment by biologists are typically based on a mixture of dynamic programming algorithms and heuristics. Prominent examples include Clustal (Chenna et al., 2003) and MUSCLE (Edgar, 2004). The traditional approaches used by these tools differ sharply from BetaAlign in design, in that their heuristics were hand-crafted whereas BetaAligns' are learnt from data.

Other alignment approaches integrate machine learning more directly with classical algorithms, representing a middle ground. For example, Petti et al. (2022) use a learnt convolution to determine how similar sites in the sequences are, and then performs the alignment using a differentiable version of dynamic programming.

1.2.2 Transformer architecture

The transformer (Vaswani et al., 2017) is a neural-network architecture, variants of which have found success in many domains including natural language processing and computer vision (Dosovitskiy et al., 2021; Radford et al., 2019). A major innovation is to use attention mechanisms (Bahdanau et al., 2015) to control the mixing of information between different locations in the input (e.g. between different regions of an image or between different tokens in a sequence).

Dotan et al. (2024) introduce a framing of the multi-sequence alignment (MSA) problem as a sequence-to-sequence task, enabling the use of the encoder-decoder transformer architecture of Vaswani et al. (2017), which implements a sequence-to-sequence mapping (originally for machine translation).

1.2.3 Neural network interpretability

It is not obvious from cursory inspection of the parameters of a trained deep neural network how it performs tasks. The intermediate representations input are not human-readable, nor are the processes used to progressively transform the input to the output transparent. The field of mechanistic interpretability seeks to remedy this, by providing an explanation of what *features* of the input are computed, and what *circuits* are used to compute more abstract features from lower-level features (Rai et al., 2024). Initially, this paradigm was used to study convolutional neural networks used for image classification (Olah et al., 2020), but it has since been applied also to language models (Elhage et al., 2021, e.g.)

To find features used by models, both supervised and unsupervised approaches are used. Supervised approaches include linear probes (Alain and Bengio, 2017), which use a dataset of inputs that do and not contain the feature to train a second model (the probe) to predict the feature from the intermediate representations of the original model. Unsupervised approaches include training sparse autoencoders on the intermediate representations (Huben et al., 2024). The autoencoder loss function has a term added to encourage the features to be sparse, in the hope that the sparsity means that the learnt features will be more interpretable.

Neural network interpretability has been applied to models trained to perform tasks on nucleotide sequences. For example, Deng et al. (2025) train sparse autoencoders on the intermediate representations of Evo 2, a transformer a trained to model DNA sequences. This method discovers features including which positions are intron-exon boundaries. Our interpretability approach differs in that it uses a supervised approach (linear probes) rather than unsupervised (autoencoders). This is possible because we know in advance which features we are interested in (inference of evolutionary parameters), whereas Deng et al. (2025) seek to find features that are not anticipated in advance. It has been shown (Kantamneni et al., 2025) that linear probes are often superior to sparse autoencoders at finding features that generalise well.

Chapter 2

Prerequisites

2.1 Biological concepts

Deoxyribonucleic acid (DNA) is a polymer composed of molecules called nucleotides. Any given nucleotide contains one of cytosine, guanine, adenine, or thymine; these types of nucleotides are denoted by C, G, A, and T, respectively. Thus, a section of DNA can be described using a sequence of these characters.

An similar description holds for proteins, which are formed of sequences of amino acids. Some sections of DNA are used to determine the ordering of amino acids in proteins, with three consecutive nucleotides coding for an amino acid. DNA therefore holds the information that is central to the functioning of organisms.

Over time, DNA sequences mutate. This occurs in three main ways: insertions of new nucleotides into the sequence, deletions of nucleotides, and substitution of one nucleotide for another. The rates of these events differ between species and locations in the genome (Chen et al., 2009, e.g.) These mutations affect the function of organisms, and evolutionary pressures can promote certain variants in populations. Therefore, DNA sequences differ between species and individuals in ways that explain their different traits.

2.2 Nomenclature and notation

Note that the word "sequence" is used both for sequences of tokens being processed by a transformer and for DNA sequences. When it is necessary to disambiguate, the former will be referred to as a "token sequence" and the latter as a "nucleotide sequence".

Nucleotide sequences are denoted as $x = x_1, \ldots, x_{\ell(x)}$.

2.3 Statistical alignment

Statistical alignment contrasts with optimisation alignment: instead of finding a single alignment that is optimal according to some metric, we instead use a statistical model of sequence evolution to compute a distribution over all possible alignments. We use sequences and alignments sampled from such a model to train our transformer model, and then evaluate its performance in reference to the statistical model.

The sequence evolution model we use was introduced by Thorne et al. (1991), and is referred to as the TKF91 model. This models sequence evolution as a continuous-time Markov process, meaning that a sequence's future evolution is conditionally independent of the past trajectory given the current state. The parameters are the insertion rate coefficient λ, deletion rate coefficient μ, and equilibrium probabilities $\pi = (\pi_A, \pi_T, \pi_C, \pi_G)$ for each nucleotide.

The equilibrium distribution determines the probability of nucleotide a being substituted for nucleotide b over time t as follows:

$$s_t(a, b) = \pi_a(1 - e^{-t}) + \mathbb{1}_{\{a=b\}}e^{-t} \tag{2.1}$$

which uses a substitution rate of 1 without loss of generality, because the time units can be defined in terms of the substitution rate.

We define our model probability $p_t(x, y, \alpha)$ as the probability that sequences x and y evolved from an ancestral sequence over a duration t of evolutionary time such that the homologous positions are given by alignment α. A key simplification arises if our model is time-reversible: the joint probability can be rewritten as

$$p_t(x, y, \alpha) = p_\infty(x)p_{2t}(x \to y, \alpha) \tag{2.2}$$

where $p_\infty(x)$ is the equilibrium probability of sequence x occurring, and $p_{2t}(x \to y, \alpha)$ is the probability of sequence x evolving to sequence y over time $2t$ with homology given by alignment α. This insight allows for simplification; we can now think only about modelling evolution from sequence x to sequence y over twice as long a duration, without having to explicitly consider the possible ancestral sequences.

2.3.1 Pair hidden Markov model

A naive way to use the TKF91 model to generate pairs of sequences (along with alignments) that diverged from a common ancestor in time t is as follows:

- Sample an ancestor sequence from the stationary distribution of the model;
- Simulate the evolution of the ancestor sequence for a time duration t twice independently to generate two descendant sequences, while keeping track of which positions in the descendants are homologous;
- Return the sequences along with an alignment that aligns the homologous positions.

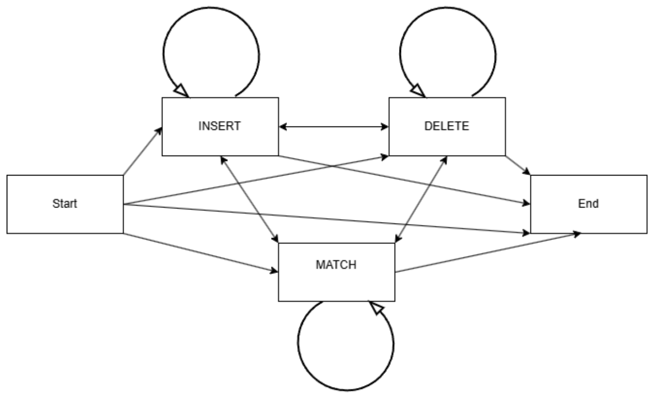


Figure 2.1: A diagram showing the states of the pair HMM and the possible transitions between them.

Fortunately, there is a more straightforward way to sample from this distribution! The probability distribution over alignments that results from the above procedure can be specified directly by a pair Hidden Markov Model (pair HMM) (Krogh and Brown, 1994; Durbin et al., 1998; Holmes and Bruno, 2001).

A general definition of the pair HMM model is beyond the scope of this report; for an introduction see Durbin et al. (1998). Instead we define pair HMMs in the context of pairwise alignment.

We start with a change of perspective on alignments. Instead of thinking of an alignment as inserting gaps into a sequence of nucleotides, we notice that the information is encoded equally well by simply recording for each column in the alignment whether it is of the form $\substack{a\\b}$, $\substack{a\\-}$, or $\substack{-\\a}$ (where a, b denote arbitrary nucleotides). Taking the perspective of Equation 2.2 where we think of the evolution happening from sequence x to y, we can think of these types of column as retentions or substitutions, deletions, and insertions respectively.

They key insight of the pair HMM model is that these column types form a Markov chain: under the TKF91 model, the probability of a specific column type occurring depends only on the type of the immediately preceding column. However, we can't directly observe the states of the Markov chain; instead we observe the two sequences, which are compatible with many possible sequences of states (i.e., alignments). This is what is meant by a *pair hidden* Markov model.

This leads us to the procedure for sampling from the pair HMM. $Q = \{\text{START}, \text{INSERT}, \text{DELETE}, \text{MATCH}, \text{END}\}$ is the set of states the pair HMM. To sample a pair of aligned sequences x and y, we start in the state START, and then transition to state s with probability given by state transition function $T(\text{START}, s)$. The transition probabilities between the states are determined by the parameters of the TKF91 model, and are shown in Table 2.1; for a derivation see Holmes and Bruno (2001) and Hein (2001). In each state, we perform the following action, and keep transitioning between states until END is reached:

- START corresponds to the start of the alignment (no action).
- INSERT corresponds to a column that matches a gap in sequence x to a nucleotide in sequence y. Therefore, we add to sequence y a nucleotide sampled from π.
- DELETE corresponds to a column that matches a nucleotide in sequence x to a gap in sequence y. Therefore, we add to sequence x a nucleotide sampled from π.
- MATCH corresponds to a column where a nucleotide is matched to a (potentially different) nucleotide. Therefore, we add a nucleotide $a \sim \pi$ to x and a nucleotide sampled from $s_t(a, \cdot)$ to y.

Note that the alignment is determined by the path taken through the pair HMM's states; each state except for START and END determines one column in the alignment sequentially.

From\To	END	MATCH	INSERT	DELETE
START	$(1-\frac{\lambda}{\mu})(1-\beta)$	$\frac{\lambda}{\mu}(1-\beta)\alpha$	β	$\frac{\lambda}{\mu}(1-\beta)(1-\alpha)$
INSERT	$(1-\frac{\lambda}{\mu})(1-\beta)$	$\frac{\lambda}{\mu}(1-\beta)\alpha$	β	$\frac{\lambda}{\mu}(1-\beta)(1-\alpha)$
MATCH	$(1-\frac{\lambda}{\mu})(1-\beta)$	$\frac{\lambda}{\mu}(1-\beta)\alpha$	β	$\frac{\lambda}{\mu}(1-\beta)(1-\alpha)$
DELETE	$(1-\frac{\lambda}{\mu})\cdot\frac{\mu}{\lambda}\cdot\beta\cdot\frac{1}{1-\alpha}$	$\frac{\beta\alpha}{1-\alpha}$	γ	β

Table 2.1: Transition probabilities (the function $T(\cdot,\cdot)$) for the pair HMM. Note that the probability of transitioning to the START state is always 0. The expressions use intermediate quantities that depend on the TKF91 parameters and the divergence time t: $\alpha = e^{-\mu t}, \beta = \frac{\lambda - \lambda e^{(\lambda-\mu)t}}{\mu - \lambda e^{(\lambda-\mu)t}}, \gamma = 1 - \frac{\mu - \mu e^{(\lambda-\mu)t}}{(1-\alpha)(\mu - \lambda e^{(\lambda-\mu)t})}$.

2.3.2 Probability of aligned sequences

To illustrate the pair HMM, we show how to compute the probability of a pair of sequences being generated with a specific alignment. Let the aligned sequences be $\begin{array}{cccc} x: & A & T & - \\ y: & A & - & C \end{array}$. The probability is given by the following product:

$$\begin{aligned} &T(\text{START}, \text{MATCH})\pi_A s_t(A, A) \\ \times &T(\text{MATCH}, \text{DELETE})\pi_T \\ \times &T(\text{DELETE}, \text{INSERT})\pi_C T(\text{INSERT}, \text{END}) \end{aligned}$$

where each line corresponds to a column in the alignment.

2.3.3 Viterbi algorithm for maximum-probability alignment given pair HMM parameters

The Viterbi algorithm computes the alignment with maximum probability given a pair of sequences (and also returns its probability). The Markov property means that we have a recurrence that enables us to avoid considering every possible alignment, which would not be feasible for long sequences. The algorithm, containing the recurrence, is shown in Algorithm 1.

Algorithm 1 Viterbi algorithm

```
Require: sequences $x = x_1, \dots, x_n, y = y_1, \dots, y_m$
  Initialise $v^I, v^D, v^M = 0 \in \mathbb{R}^{(n+1)\times(m+1)}$    ▷ for insert, delete, match states
  Set $V^M_{0,0} = 1$

  for $i = 1, \dots, n$ do    ▷ Account for deletions from start of $x$
    $v^D_{i,0} = \pi_{x_i} \cdot \max \begin{cases} v^M_{i-1,0} \cdot T(\text{MATCH}, \text{DELETE}) \\ v^I_{i-1,0} \cdot T(\text{INSERT}, \text{DELETE}) \\ v^D_{i-1,0} \cdot T(\text{DELETE}, \text{DELETE}) \end{cases}$
  end for
  for $j = 1, \dots, m$ do    ▷ Account for insertions to start of $y$
    $v^I_{0,j} = \pi_{y_j} \cdot \max \begin{cases} v^M_{0,j-1} \cdot T(\text{MATCH}, \text{INSERT}) \\ v^I_{0,j-1} \cdot T(\text{INSERT}, \text{INSERT}) \\ v^D_{0,j-1} \cdot T(\text{DELETE}, \text{INSERT}) \end{cases}$
  end for
  for $i = 1, \dots, n$ do
    for $j = 1, \dots, m$ do
      $v^M_{i,j} = \pi_{x_i} s_t(x_i, y_j) \cdot \max \begin{cases} v^M_{i-1,j-1} \cdot T(\text{MATCH}, \text{MATCH}) \\ v^I_{i-1,j-1} \cdot T(\text{INSERT}, \text{MATCH}) \\ v^D_{i-1,j-1} \cdot T(\text{DELETE}, \text{MATCH}) \end{cases}$
      $v^I_{i,j} = \pi_{y_j} \cdot \max \begin{cases} v^M_{i,j-1} \cdot T(\text{MATCH}, \text{INSERT}) \\ v^I_{i,j-1} \cdot T(\text{INSERT}, \text{INSERT}) \\ v^D_{i,j-1} \cdot T(\text{DELETE}, \text{INSERT}) \end{cases}$
      $v^D_{i,j} = \pi_{x_i} \cdot \max \begin{cases} v^M_{i-1,j} \cdot T(\text{MATCH}, \text{DELETE}) \\ v^I_{i-1,j} \cdot T(\text{INSERT}, \text{DELETE}) \\ v^D_{i-1,j} \cdot T(\text{DELETE}, \text{DELETE}) \end{cases}$
    end for
  end for
  Return $\max\{v^M_{n,m}, v^I_{n,m}, v^D_{n,m}\}$
```

2.3.4 Forward algorithm for sequence likelihood

The forward algorithm computes $\sum_\alpha p(x, y, \alpha)$, the probability of x, y sequences evolving from some common ancestor (in any alignment. It is entirely analogous to the Viterbi algorithm, replacing every instance of a maximum with a sum.

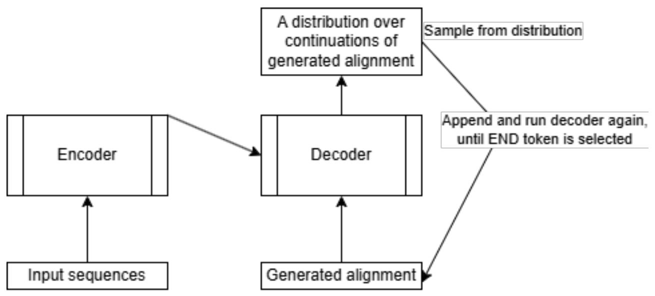

A distribution over continuations of generated alignment
Sample from distribution
Encoder
Decoder
Append and run decoder again, until END token is selected
Input sequences
Generated alignment

Figure 2.2: A schematic diagram showing how the encoder and decoder process information.

2.4 Transformer architecture

This section is an overview of the transformer architecture. The exposition that follows differs from that of Vaswani et al. (2017) (which introduced the transformer) and instead uses concepts introduced in Elhage et al. (2021).

The transformer architecture was first used for machine translation, which is a task that takes as input a sequence of tokens (the original text) and generates a sequence of tokens (the translation into the target language). We can view pairwise alignment as an analogous task; we map from a pair of DNA sequences to an alignment of them (both of which can be represented as a sequence of tokens). The transformer achieves this with two components: an encoder and a decoder.

The **encoder** maps the input sequence $x_1, \ldots, x_n$ to a high-dimensional representation $z_1, \ldots, z_n$ where $z_i \in \mathbb{R}^d$. The representation captures the meaning of the input for use by the decoder.

The **decoder** then takes the representations $z_1, \ldots, z_n$ and, step-by-step, generates the output sequence. It works in an 'autoregressive' manner, meaning that to generate the next token y_{t+1}, it considers the encoder's output $z_1, \ldots, z_n$ and all the tokens it has already generated $y_1, \ldots, y_t$. At each step, it outputs a probability distribution over the possible continuations of the sequence generated so far. Figure 2.2 illustrates this, using DNA sequence alignment as an example task.

The internal structure of the encoder is very similar to that of the decoder, so they are treated together.

2.4.1 Token embedding

The processing begins by embedding each token of the token sequence $t_1, \ldots, t_m$ being processed (the input sequences in the case of the encoder, and the sequence generated so far in the case of the decoder) as a high-dimensional vector in $\mathbb{R}^d$. d is the embedding dimension, a hyperparameter of the model. The initial embedding vector for each token t_j is created by adding together two vectors:

1. **Token Content Embedding:** A vector representing the identity of the token itself (e.g., the nucleotide C). These embeddings are learned from data during the training.

2. **Positional Encoding:** A vector representing the token's ordinal position within the sequence. Otherwise, the transformer mechanism would not know the order of the tokens. Unlike the content embedding, this is calculated using a fixed formula (Vaswani et al., 2017, Section 3.5).

We denote the initial embedding vector by $x_1^{(0)}, \ldots, x_n^{(0)}$, and these are the starting point for further processing. They pass through a series of stages, each of which stage takes the output from the previous stage, performs some computation, and adds its result back to the input it received. The concept of additive updates to the representation is called a 'residual connection', and the sequence of evolving vectors $x_1^{(i)}, \ldots, x_n^{(i)}$ at each stage is referred to as the **residual stream**. We can describe the residual stream as a matrix with the vectors for each token as its columns: $x^{(i)} = \left(x_1^{(i)} \ldots x_n^{(i)}\right)$

The processing stages alternate between a multi-head attention and multi-layer perceptrons (MLP). A pair of multi-head attention followed by an MLP is referred to as a **layer**, and the full model contains several layers applied sequentially.

2.4.2 Multi-head attention

This is a central innovation of the transformer. At this stage, the representation for each token $x_j^{(i)}$ is updated based on information from *all* other tokens in the sequence.

Multi-head attention means that there are $h \geq 1$ attention heads that act independently, each outputting a vector that gets added to the residual stream. That is to say, multiheaded attention updates the residual stream as follows:

$$x^{(i+1)} = x^{(i)} + \sum_{j=1}^{h} H_j(x^{(i)})$$

where H_j is a function denoting the output of the j^{th} attention head. Now it remains to describe the behaviour of a single attention head on a residual stream sequence $x \in \mathbb{R}^{d \times n}$.

The first operation performed as an attention head can be thought of as "reading" some information from the residual stream, multiplying each token's vector by a matrix $W_V \in \mathbb{R}^{d_h \times d}$, to obtain $W_V x$ (here, W_V is a parameter learnt during training, $d_h < d$ is the head dimension). The columns of $W_V x$ are referred to as the value vectors, and the next step in the attention head creates a sequence of linear combinations of value vectors: $r_i = \sum_{j=1}^{n} A_{i,j}(W_V x)_j$. The coefficient $A_{i,j}$ is thus the amount of information transferred from position j in the residual stream to position i. In the decoder, any value $A_{i,j}$ with $j > i$ is set to 0, so that each position learns to predict what comes next in the sequence without "cheating" by looking ahead; this is referred to as *masked* attention.

What determines the coefficients $A_{i,j}$? By analogy to databases, the transfer strength by calculating a "query" vector for each position each and a "key" vector for each position. This is done by learnt linear maps: $k_i = W_K x_i$, $q_i = W_Q x_i$. Then, $A'_{i,j} = \langle q_i, k_j \rangle$ – the dot product. To obtain A, we take the softmax of each row of A', so that the rows sum to 1 (and before taking the dot product, scale by the dimension d_k of the key/query vectors for stability). Then, the final output of the transformer is obtained by multiplying each linear combination r_i by the learnt matrix W_O that maps the vectors back to the dimension of the residual stream. Putting this together in matrix form, we get a version of the formula from Vaswani et al. (2017):

$$H(x) = W_O \left(\text{softmax} \left(\frac{(W_Q x)(W_K x)^\top}{\sqrt{d_k}} \right) W_V x \right)$$

In the decoder, this attention mechanism is followed by another attention mechanism that transfers across information from the encoder's output.

2.4.3 Multi-layer perceptron

Unlike attention, the MLP stages do not mix information between positions in the residual stream. Instead, they use a fully-connected two-layer neural network with ReLU activation to process each position individually.

2.4.4 Layer normalisation

After the attention and MLP steps, the residual stream is normalised, as described by L. J. Ba et al., 2016. This step modifies each position x_i in the residual stream as follows:

$$x_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

where μ and σ are the mean and standard deviation, respectively, of the vector x_i. ε is a small constant used to avoid dividing by zero. This is followed by an affine transformation with learnt parameters that allows the model to learn to undo the normalisation if necessary.

2.4.5 Output

Once we have reached the final layer of the decoder, we seek to transform the residual stream into a probability distribution over each possible continuation of the decoder's input sequence (i.e., the generated sequence so far). This is done by means of a linear transformation to the correct dimension; often this is done by the transpose of the linear transformation that performs the content embedding. Once we have a vector that is of the correct dimension to represent a distribution over next tokens, we apply the softmax function to convert it to a valid distribution (i.e., with positive entries summing to 1). The values of the vector before applying softmax are referred to as logits.

Chapter 3

Training and Inference

This chapter outlines how we train a BetaAlign-like transformer to perform sequence alignment, and improved approaches for inference. We address the following issues with the inference procedures described by Dotan et al. (2025):

- The procedure often fails to output a valid alignment, and therefore requires the model to be run multiple times to ensure validity.
- The procedure does not give a probability distribution over alignments, instead only giving a single alignment with a certainty score.

In the final two sections of this chapter we present a remedy for the first and second problems respectively.

3.1 Training

Data

The synthetic dataset consists of a pair of sequences generated by a pair HMM with parameters selected uniformly at random from one of two options, a true alignment between the sequences aligning positions that are homologous in the simulated evolution, and a record of which set of pair HMM parameters (each of which will be referred to hereafter as "a pair HMM") was used. The idea of using two different pair HMMs is to simulate the setup where the model must infer the evolutionary parameters in order to compute the alignment.

The two sets of pair HMM parameters differ in the equilibrium probability of nucleotides, with the first pair HMM having uniform distribution but the second being $(\pi_A = 0.4, \pi_T = 0.1, \pi_c = 0.25, \pi_G = 0.25)$.

Alignment as a sequence-to-sequence task

In order to train the transformer, we must frame pairwise alignment as a task that generates an output sequence given an input sequence. Therefore, we must encode the pairs of input DNA sequences and the alignments as sequences of tokens. To do this, we use a token for each nucleotide and some special tokens, following the encoding found by Dotan et al. (2024) to lead to the best results.

To encode the input DNA sequences as a token sequence, we use a special seperator token, denoted "|". Therefore, the sequence pair CGT, CT would be denoted as the following token sequence $CGT|CT$.

In training, we add to the start of each alignment a "START" token (denoted "\$") and to the end of each alignment an "END" token (denoted "#"), so the transformer can learn to predict the first token and to predict the end of the alignment. The rows of the alignment are interleaved, thus the alignment $\begin{matrix} C & G & T \\ C & - & T \end{matrix}$ would be encoded as $\$CCG-TT\#$.

Partitions

The data is partitioned into a train set, a validation set, and multiple test sets. The dataset is balanced, so that the number of sequences generated by each pair HMM is equal. We also filter out all datapoints where the alignment is longer than the maximum sequence length that the transformer can handle.

Note that there could be overlap between these partitions in multiple ways; an identical pair of sequences with the same alignment could occur in two partitions, or an identical pair of sequences could be shared between partitions but with differing alignments in each. Since we seek to evaluate the parameter inference performance (a task that depends only on the sequences), we avoid both types of overlap by partitioning the dataset based on the numerical MD5 hash (Rivest, 1992) of the sequences reduced modulo an integer.

3.1.1 Architecture

We scale down the architecture described by Dotan et al. (2025) for faster experimentation with limited computational resources and ease of conceptual investigation. The changes are: processing sequences of length 32 instead of 1024, an embedding dimension of 128 instead of 1024, and four layers instead of six.

3.1.2 Optimisation

To train the model, we use the Adam optimiser (Kingma and J. Ba, 2015) with a two-phase learning schedule. The first phase is *warmup* and lasts for 30% of the total number of training epochs. In the warmup phase, the learning rate begins at zero and is linearly increased to a maximum of 0.0001. The second phase lasts for the remaining 70% of epochs, linearly reducing the learning rate from 0.0001 down to zero. Warmup appears necessary for training models of this kind, as analysed by e.g., Kosson et al. (2024).

Figure 3.1 shows that this results in a reduction of the training loss over the course of training; the loss on a validation dataset declines similarly.

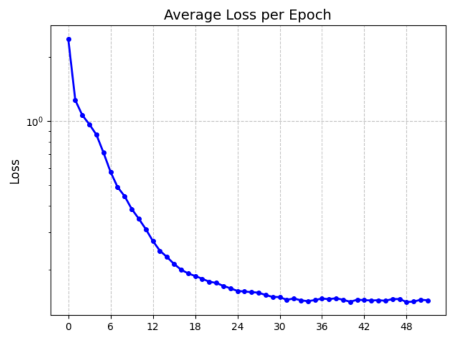


Figure 3.1: Average loss on the training data over training

3.2 Constrained beam search

In personal communication, the authors of Dotan et al. (2025) confirmed that they use beam search to generate alignments. Constrained beam search refers to variants of beam search that enforce constraints. Below, is a variant of beam search constraining the output to be a valid alignment of n sequences.

When generating an alignment, we know precisely which tokens are valid at any given generation step:

- The sequence must begin with the START token
- The alignment can either use the next nucleotide from a sequence, or contain a gap *unless* we are currently at the end of a column that only contains gaps so far (a column cannot contain only gaps)
- Once all nucleotides have been used and a column is completed, the sequence must end with the END token

Algorithm 2 k-beam search, constrained to valid alignments

```
Require: $k \geq 1, n \geq 2$, sequences $x_1, \ldots, x_n$
  Set beams = $\varnothing$
  while there are non-terminated beams do
      for each non-terminated beam $b$ do
          for each valid extension $b'$ of $b$ do
              Set beams = beams $\cup \{b'\}$
          end for
          Set beams to contain only those with top-$k$ likelihood
      end for
  end while
```

3.3 Distribution over alignments

The conventional way to compute a probability distribution over strings generated by a transformer is via the following formula:

$$q(y_1, \ldots, y_l|x) = \exp\left(\sum_{i=1}^{l} \log q(y_i|x, y_1, \ldots, y_{i-1})\right) \tag{3.1}$$

This can be computed using a single forward pass.

However, a straightforward application of this formula here could result in a probability distribution that is supported on invalid alignments if the model assigns a positive probability to invalid tokens (and this happens in practice). Therefore, we require a modified version of the formula. We propose reallocating the probability mass to the valid tokens only at each generation step. This can be done by using a modified version of Equation 3.2:

$$\tilde{q}(y_1, \ldots, y_l|x) = \exp\left(\sum_{i=1}^{l} \log \tilde{q}_i(y_i|x, y_1, \ldots, y_{i-1})\right) \tag{3.2}$$

where

$$\tilde{q}_i(y_i|x, y_1, \ldots, y_{i-1}) = \frac{1}{\sum_{y \in \text{validTokens}} q(y|x, y_1, \ldots, y_{i-1})} q(y_i|x, y_1, \ldots, y_{i-1})$$

which is redistributes the probability mass to valid tokens and renormalises to ensure a valid distribution. Figure 4.2 shows how this gives rise to a distribution over possible alignments for short sequences where there are few valid alignments that can be tabulated.

Chapter 4

Evaluation

We perform multiple evaluations of our BetaAlign-like transformer model, comparing the behaviour of the transformer model to the pair HMM models that generated the training data. For these evaluations, we can look at how point-estimates of the optimal alignment (computed using constrained beam search) compare to the optimal alignment (computed using the Viterbi algorithm), and also at how the distribution over alignments predicted by the transformer model differs from the true distribution according to the pair HMM models.

4.1 Comparing point estimates

First, we evaluate how closely the model's prediction of the optimal alignment is to the true optimal alignment. This is done using sequences where the total length of the alignment is at least 20 selected from the test dataset, thus testing the model on the hardest sequences. The model's true alignment is generated using constrained beam search (Algorithm 2), and the optimal alignment is given by the Viterbi algorithm. To evaluate how close the model's prediction is to the optimal alignment, we measure the ratio of their probabilities according to the pair HMM.

Figure 4.1 shows that the vast majority of the time, model's prediction attains the maximum possible likelihood (i.e., the ratio between the likelihood of the model's prediction and the likelihood of the Viterbi algorithm's output is 1, up to numerical precision issues). This provides reassurance that the transformer model is expressive enough to approximate the pair HMM (at least for alignments of length 20 to 28). The reason that it does not monotonically increase up to 1 is that in the pair HMMs it is often the case that there is a gap between the joint-top likely alignments and the next most likely ones.

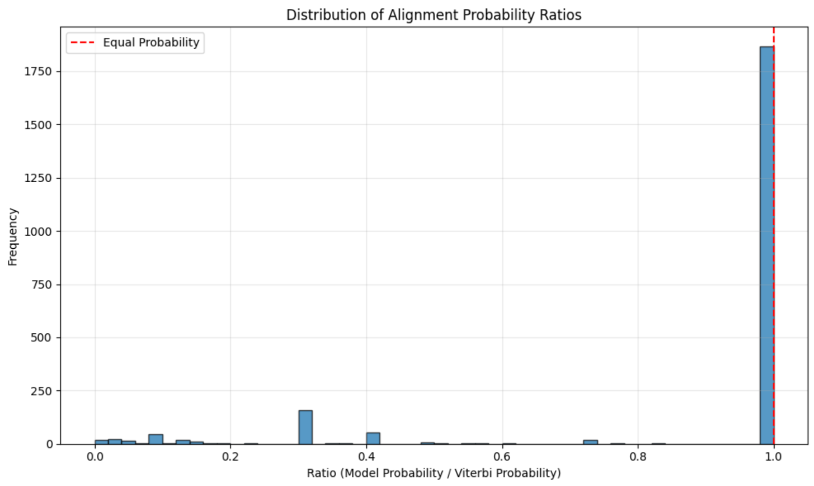


Figure 4.1: A histogram showing the ratio of the pair-HMM probability of the alignment generated by the transformer to the probability of the optimal alignment generated by the Viterbi algorithm, for sequences where the true alignment length is at least 20. A value of 1 shows that the transformer's output attains the optimal probability.

4.2 Comparing distributions

For sequences of length at most 8, it is feasible to compute the model's distribution over all possible alignments. This can be evaluated by comparing it to the "ground truth" distribution given by the pair HMM. For these tests, we use sequences that were not in the training dataset, to test the ability of the model to generalise: for short sequences, many of the possible alignments may have occurred in the training dataset.

Figure 4.2 shows the transformer's versus the pair HMM's distribution over the top-20 most probable alignments (according to the transformer). We observe that the transformer assigns similar probabilities to the pair HMM models (which in this case agree). The most notable difference is that the transformer underestimates the probability of some of the 12th to 16th most probable sequences by an order of magnitude.

For a more systematic comparison, we produce similar plots averaged over many short sequences showing how the probabilities assigned to the most likely sequences compare to the probability according to the pair HMM that generated each sequence.

Figure 4.3 shows the result of this experiment. Again we see that the transformer's distribution underestimates the probability of sequences of probability rank around 15, and that the transformer assigns more probability to the rest of the sequences. Overall, the distributions are closely matching.

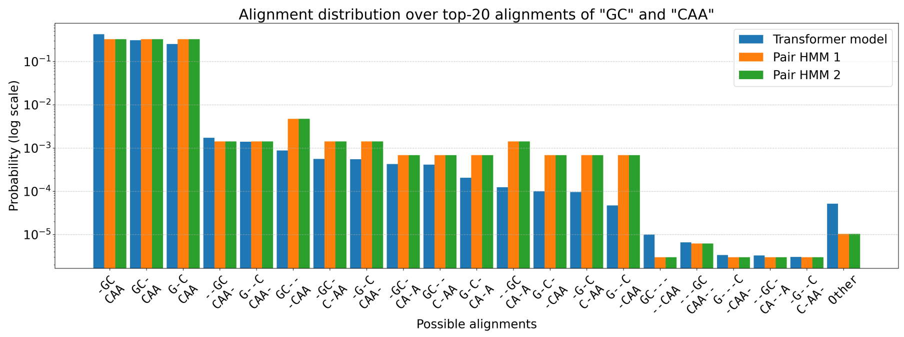


Figure 4.2: A bar chart showing how the probability distribution over alignments given by the transformer model compares to the pair HMM's distribution conditional on the pair of sequences GC and CAA (from a dataset partition that the model was not trained on). We see that the transformer probabilities are broadly in line with those of the pair HMM but for some sequences underestimates the probability by an order of magnitude.

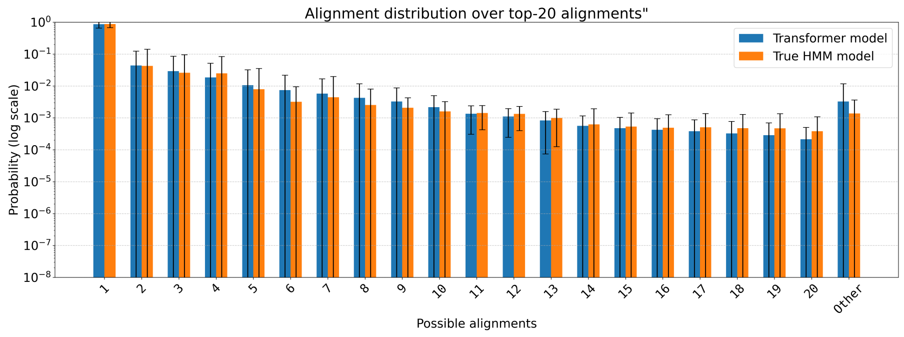


Figure 4.3: A plot analogous to Figure 4.2, but averaged over 100 sequences and showing the rank of the alignments order by the transformer's probability. The pair HMM probability compared is calculated using the pair HMM that generated each sequence (the "true" model). The error bars show one standard deviation either side (calculated for the original values and shown in log-scale, hence the asymmetry).

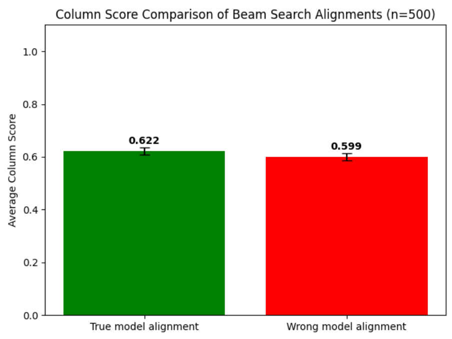


Figure 4.4: Average column scores showing how closely the alignment outputted by constrained beam search matches the Viterbi algorithm's alignment for the true generator model and the other generator model, over 500 sequences. The error bars show the standard error of the mean.

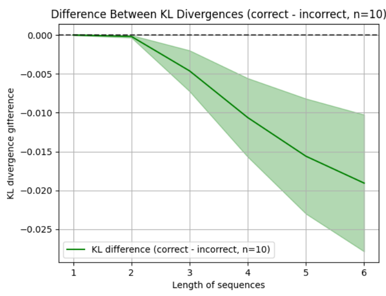
4.3 Parameter inference

The final evaluation we perform tests how well the transformer can internally infer which of the two pair HMMs was used to generate a given sequence. This is a simple test of inference of evolutionary parameters that must be performed to align real sequences. We evaluate this in two ways: how well the point-estimate of the optimal alignment using constrained beam search can distinguish between alignments in the case where this differs between the two pair HMMs, and how closely the probability distributions align to the true pair HMM's distribution compared to the other pair HMM. This is done using sequences in the test set.

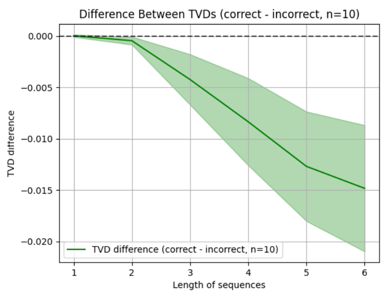
4.3.1 Point estimates

First, we identify sequences where the alignment returned by the Viterbi algorithm differs between the pair HMMs, as a proxy for sequences where it is important to infer which generator model was used in order to align correctly. This occurs for approximately 10% of the sequences in the database. Then, we use constrained beam search to align those sequences, measuring how well they match the alignment of the true model. We use the column score to measure how well the transformer's alignment matches the optimal (Viterbi) alignment for each pair HMM. The column score, introduced by Thompson et al. (1999), is the fraction of columns in the transformer's alignment that contain two sequence positions that are also aligned to each other in the optimal alignment.

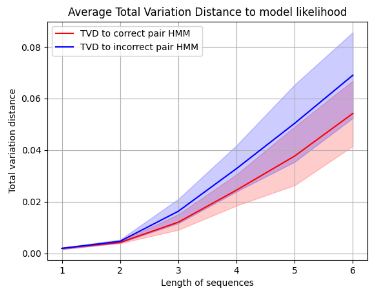
The alignment returned by constrained beam search does match slightly more closely to that of the true pair HMM that generated the sequences over the other pair HMM. This is shown in Figure 4.4, where the column score is only slightly higher for the true pair HMM, and the difference seems to be small and potentially not statistically significant.



(a) KL divergence



(b) Total variation distance



(c) Total variation distance

Figure 4.5: Measuring how closely the transformer's distribution over possible alignments matches the pair HMM that generated the sequence pair (the "true" model) over the other pair HMM. This is measured using the difference between measures of difference between the probability distributions; thus a negative value indicates correct inference of which pair HMM generated the sequences because the distribution. The band shows the standard error of the mean.

Note that this evaluation is flawed in that the optimal alignment according to a pair HMM is not always unique. Therefore, when the transformer model's alignment does not match the result of the Viterbi algorithm, this is not necessarily a failing; perhaps it simply found another equally likely alignment. To overcome this failing, we also compare the likelihood of the model's point estimate according to each of the pair HMMs, and find no significant difference.

4.3.2 Distributions

To evaluate this, we measure how much the transformer's probability diverges from that of each of the two pair HMM models using the KL-divergence $D_{\mathrm{KL}}(P_{\text{pair HMM}}||P_{\text{transformer}})$ and the total variation distance (which is symmetric).

Figure 4.5 shows the result of this experiment. As the transformer sees longer extracts of the sequence, its distribution over alignments becomes more decisively similar to that of the true pair HMM over the other pair HMM. However, this difference is relatively small in magnitude compared to the KL divergence or total variation distance between the distributions; Figure 4.5c shows the raw total variation distances to show the difference in magnitude.

In conclusion, the transformer often correctly infers which set of parameters are responsible for its input sequences, and that this correctly impacts its predicted distribution over alignments. However, constrained beam search is not a sufficiently good approximation of the optimal alignment for it to distinguish between alignments that are more likely according to the true pair HMM as opposed to the other pair HMM. Alternatively, the transformer's distribution might match the true pair HMM's probabilities mainly for less probable alignments, but not for the top candidates selected by constrained beam search.

Chapter 5

Model representations

In order to gain insight into the computation performed by a deep learning system, a first step can be to understand which intermediate representations it computes. The linear representation hypothesis (Park et al., 2024) posits that deep learning models learn *linear* features: ones that can be detected by linear classifiers on the activations.

5.1 Probing evolutionary parameters

One such intermediate quantity that alignment models are likely to learn to represent is inference of the evolutionary dynamics that the input sequences have been subject to, because this determines the probability distribution over possible alignments.

The technique we use to evaluate how well the model has learns to represent the evolutionary parameters is linear probing (Alain and Bengio, 2017). This trains a linear classifier (with parameters independent of the transformer model) that is trained to predict features of the input based on a layer of the residual stream.

Denote the l^{th} layer of the residual stream of the encoder on input x as $h^l(x) \in \mathbb{R}^{nd}$, concatenating the n token positions that are each of dimension d. Then, to probe layer l of the encoder, we seek a classifier vector $w \in \mathbb{R}^{nd}$ such that $\langle w, h^l(x) \rangle \geq 0$ if x has the feature being probed, and $\langle w, h^l(x) \rangle < 0$ otherwise.

5.1.1 Probe training

We obtain this classifier vector by gradient descent on a dataset of examples from each of the generator models, with the cross-entropy loss training the probe to predict which of the pair HMMs generated the sequence. The training uses the Adam optimiser Kingma and J. Ba, 2015 with weight decay (with strength 0.000001) and learning rate 0.0001.

5.1.2 Probe evaluation

This section evaluates the performance of the trained probe. We can explicitly calculate the theoretical optimum for classifying generator models; this is the Bayes classifier which has the least misclassification probability out of all classifiers. We then compare the performance of the probe to the optimum. Since we have a uniform prior over the generator models, the Bayes classifier is the maximum-likelihood classifier. A formal statement and proof of this fact is provided below.

Theorem 1. *Let S be a discrete random variable with uniform distribution over $\{1, 2\}$ (i.e., $P(S = 1) = P(S = 2) = \frac{1}{2}$). Let X be a discrete random variable with conditional distribution $P(X = x|S = i) = p_i(x), i \in \{1, 2\}$. Then, the Bayes classifier (the classifier with the smallest misclassification probability) for predicting S based on X is $\hat{S}(x) = \arg\max_{i \in \{1,2\}} p_i(x)$.*

Proof. The Bayes classifier is the classifier that maximises the posterior probability $P(S = \hat{S}(x)|X = x)$. Applying Bayes' theorem and using the fact that the prior $P(S = i)$ is uniform, we have:

$$P(S = i|X = x) = \frac{P(X = x|S = i)P(S = i)}{P(X = x)} \propto P(X = x|S = i) = p_i(x)$$

so maximising $p_i(x)$ is equivalent to maximising the posterior probability. □

We can use the Bayes' classifier to evaluate the performance of the linear probe. Figure 5.1 shows that the linear probe has accuracy that increases as the length of the aligned sequences increases. This is in line with expectations: having longer sequences means that there is more information present for the model to use to infer which model generated the sequences. We also see that the accuracy of the linear probe is close to the performance of the Bayes classifier. The overall accuracy of the probe is 75%, comparing well to the Bayes classifier's accuracy of 78%.

A possible concern with this methodology is that the work of the classification here is being done entirely by the linear classifier and not by the transformer. To allay this concern, we also attempt to train a linear probe on the embeddings, before they have been processed by any layers. This does not perform well, suggesting that the model itself is responsible for converting the data to a format that is amenable to linear prediction.

5.2 Causal intervention

So far we have identified components of the model which compute features from which the generator model can be predicted by a linear predictor, providing evidence that the model learns representations of the input sequences amenable to this prediction. However, this does not imply that these features are in fact used in later computations.

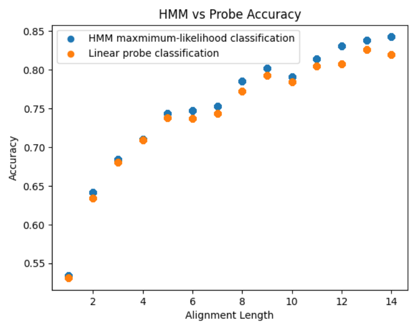


Figure 5.1: The accuracy of a linear probe at classifying which model was used to generate sequences of each length from the activations in layer 2 of the encoder, on data that used neither in the training of the model nor in the training of the probe. This is compared to maximum-likelihood classification using the models that generated the sequences, which is the Bayes classifier.

Nanda (2023) proposes that a way to determine if a linearly-represented feature is used in later model computations is to investigate if linearly modifying the residual stream in a way that effects only the linear direction of the feature can change the model's output in the expected way. In our case, this would mean that if we modify the output of the layer that the probe was trained on, we would change the downstream behaviour of the model to be consistent with changed parameter estimation. This line of work has been

The intervention we investigate is adding a multiple of the probe vector to the residual stream:

$$h^l(x) \leftarrow h^l(x) + cw$$

.

The hope is that when $c > 0$, this would modify the transformer's distribution to be closer to that of the second pair HMM, and vice versa for $c < 0$. We apply the steering to the first layer of the encoder, because earlier layers have been shown to be best for linear interventions (Canby et al., 2025).

Figure 5.2 shows the total variation distance with the first pair HMM with no steering ($c = 0$), positive steering ($c = 30$) and negative steering ($c = -30$). If the steering was successful we would expect the two steered model distributions to be either side of the unsteered model in terms of total variation distance. We hypothesise that this is because the probe has not identified the single feature used to infer which pair HMM generated the input sequences but rather has identified some of many features that are correlated and are used in the decision. Lending support to this hypothesis, we train multiple probes from different random initialisations of the vector w and apply principal components analysis, finding that the probe vectors do not lie on a low-dimensional subspace. Therefore, it would appear that there are many redundant directions in the residual stream that could be used by the model to decide which set of parameters

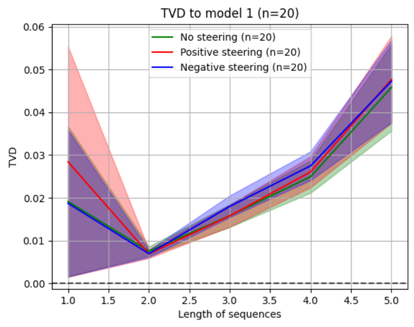


Figure 5.2: Total variation distance of the transformers distribution from that of the first pair HMM, with positive, negative, and no steering.

to use. This explains why steering one of these directions would not be effective.

5.3 Intermediate representations via LogitLens

LogitLens (nostalgebraist, 2020) is an approach to determining where in the model computations are performed. It works by applying the final linear output map O to early layers of the model, in order to obtain a distribution over continuation tokens based on earlier layers (e.g. $Oh^i(x)$ at layer i). There is no rigorous reason for this to work (for example, the transformer could well rotate the residual stream between layers), but in practice it has been found to often be successful at showing how the distribution over next tokens evolves over the layers.

Figure 5.3 shows the margin (in logits) of the token that is ultimately predicted to be most probable compared to the second most probable token predicted by LogitLens at each layer. A positive value means that the ultimately-predicted token is already predicted most likely at the layer. We see that by layer 2, the margin is positive. If we focus on cases where the ultimately-predicted token is the gap symbol (Figure 5.4), we see that only in the final layer is the margin positive. This suggests that most of the mechanisms responsible for selecting whether to choose the next nucleotide or to add a gap symbol are in the final decoder layer.

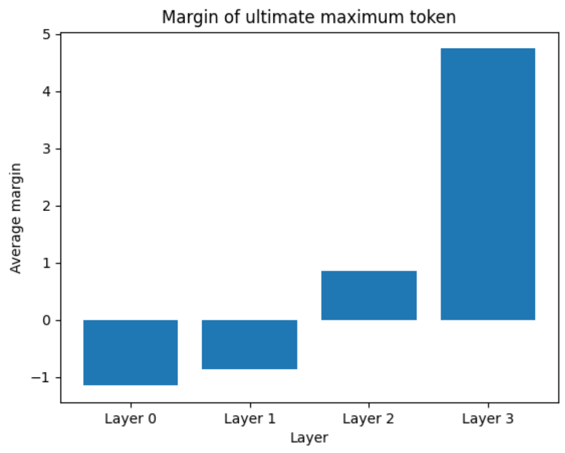


Figure 5.3: The margin of the most likely token over other tokens evolving over layers.

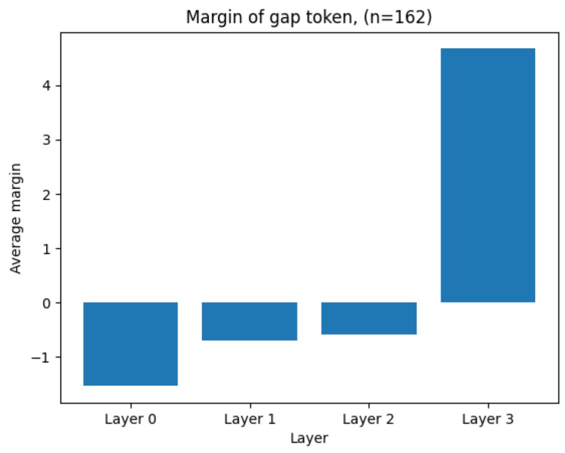


Figure 5.4: The margin of the most likely token over other tokens, when the most likely token is the gap symbol.

Chapter 6

Conclusion

6.1 Summary of results

This work investigated the parameter inference abilities of transformers trained on synthetic DNA sequences in a simplified setup, hoping to shed light on what internal computations the model performs. We have shown:

1. That a transformer trained on sequence alignment can not only make good point-estimates of an optimal alignment but can also approximate the entire distribution over alignments well;
2. That the distribution over alignments adapts to inferred evolutionary parameters;
3. That the encoder learns features that allow linear prediction of the generating model of the current input sequence with accuracy close to the best that is theoretically possible;
4. That causal intervention on these features does not straightforwardly change the model's implicit prediction of the evolutionary parameters, suggesting that perhaps many model components are responsible for estimating evolutionary parameters.

We also have initial results showing how LogitLens can be used to suggest where in the transformer the decision to insert a gap character is made.

6.2 Limitations

One limitation is that we do not investigate the full-scale BetaAlign model as introduced in Dotan et al. (2024), because of computational constraints. It is possible that our scaled-down version has different properties, perhaps due to having less computational power. Additionally, in the case of pairwise alignment, we cannot implement the majority-vote algorithm used for inference on BetaAlign, because it relies on there being many ways to permute the sequences to have many different alignments.

Our toy-model setting of evolutionary parameter inference is highly idealised, involving only a binary decision between two alternative sets of parameters. The linear-probe methodology may not perform as well in realistic settings where we need to do regression instead of classification in order to predict continuously-varying parameters such as substitution rates.

6.3 Future work

Some future directions to extend this work are listed below.

- Finding the circuit (Rai et al., 2024), responsible for deciding if the next token in the alignment should be the gap character or a nucleotide. This would contribute to a better understanding of how exactly the model aligns sequences. A starting point could be the LogitLens investigation, which identifies parts of the model likely to contain such circuits.
- We found that beam search was not effective at identifying the highest-likelihood alignments. This matches the results of Dotan et al. (2025), who had to improve upon beam search by aligning the input sequences multiple times in different permutations. Future work could try to improve on beam search for this purpose, perhaps using known structure about alignments analogously to Hein et al. (2000), who use heuristics to avoid considering alignments that are a priori unlikely.
- Extending the analysis to the full-scale BetaAlign model with multiple sequences and more complex statistical models of evolution. This is the scenario where the transformer approach compares most favourably to existing approaches, so is of the greatest interest.

Bibliography

Alain, Guillaume and Yoshua Bengio (2017). “Understanding intermediate layers using linear classifier probes”. In: *ICLR (Workshop)*. OpenReview.net.

Ba, Lei Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). “Layer Normalization”. In: *CoRR* abs/1607.06450.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ICLR*.

Canby, Marc, Adam Davies, Chirag Rastogi, and Julia Hockenmaier (2025). *How Reliable are Causal Probing Interventions?*

Chen, Jian-Qun, Ying Wu, Haiwang Yang, Joy Bergelson, Martin Kreitman, and Dacheng Tian (Mar. 2009). “Variation in the Ratio of Nucleotide Substitution and Indel Rates across Genomes in Mammals and Bacteria”. In: *Molecular Biology and Evolution* 26.7, pp. 1523–1531.

Chenna, Ramu, Hideaki Sugawara, Tadashi Koike, Rodrigo Lopez, Toby J Gibson, Desmond G Higgins, and Julie D Thompson (July 2003). “Multiple sequence alignment with the Clustal series of programs”. en. In: *Nucleic Acids Res* 31.13, pp. 3497–3500.

Deng, M., D. Balsam, L. Gorton, N. Wang, N. Nguyen, E. Ho, and T. McGrath (2025). *Interpreting Evo 2: Arc Institute’s Next-Generation Genomic Foundation Model*.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *ICLR*. OpenReview.net.

Dotan, Edo, Elya Wygoda, Noa Ecker, Michael Alburquerque, Oren Avram, Yonatan Belinkov, and Tal Pupko (2024). “BetaAlign: a deep learning approach for multiple sequence alignment”. In: *Bioinformatics* 41.1.

— (Jan. 2025). “BetaAlign: a deep learning approach for multiple sequence alignment”. In: *Bioinformatics* 41.1, btaf009.

Durbin, Richard, Sean R. Eddy, Anders Krogh, and Graeme Mitchison (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Edgar, Robert C. (Mar. 2004). “MUSCLE: multiple sequence alignment with high accuracy and high throughput”. In: *Nucleic Acids Research* 32.5, pp. 1792–1797.

Elhage, Nelson, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah (2021). “A Mathematical Framework for Transformer Circuits”. In: *Transformer Circuits Thread*. https://transformer-circuits.pub/2021/framework/index.html.

Greenshields-Watson, Alexander, Parth Agarwal, Sarah A. Robinson, Benjamin Heathcote Williams, Gemma L. Gordon, Henriette L. Capel, Yushi Li, Fabian C. Spoendlin, Broncio Aguilar-Sanjuan, Fergus Boyles, and Charlotte M. Deane (2025). “ANARCII: A Generalised Language Model for Antigen Receptor Numbering”. In: *bioRxiv*.

Hein, J. (2001). “An algorithm for statistical alignment of sequences related by a binary tree”. In: *Pacific Symposium on Biocomputing*, pp. 179–190.

Hein, J., C. Wiuf, B. Knudsen, M.B. Møller, and G. Wibling (2000). “Statistical alignment: computational properties, homology testing and goodness-of-fit11Edited by J. Karn”. In: *Journal of Molecular Biology* 302.1, pp. 265–279.

Holmes, Ian and William J. Bruno (Sept. 2001). “Evolutionary HMMs: a Bayesian approach to multiple alignment”. In: *Bioinformatics* 17.9, pp. 803–820.

Huben, Robert, Hoagy Cunningham, Logan Riggs, Aidan Ewart, and Lee Sharkey (2024). “Sparse Autoencoders Find Highly Interpretable Features in Language Models”. In: *ICLR*. OpenReview.net.

Kantamneni, Subhash, Joshua Engels, Senthooran Rajamanoharan, Max Tegmark, and Neel Nanda (2025). “Are Sparse Autoencoders Useful? A Case Study in Sparse Probing”. In: *CoRR* abs/2502.16681.

Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *ICLR (Poster)*.

Kosson, Atli, Bettina Messmer, and Martin Jaggi (2024). “Analyzing & Reducing the Need for Learning Rate Warmup in GPT Training”. In: *NeurIPS*.

Krogh, Anders and I Brown (1994). “Hidden Markov Models in Computational Biology”. In: *J. Mol. Bioi* 235, pp. 1501–1531.

Nanda, Neel (Mar. 2023). *Actually, Othello-GPT Has A Linear Emergent World Model*.

Needleman, Saul B. and Christian D. Wunsch (1970). “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of Molecular Biology* 48.3, pp. 443–453.

nostalgebraist (2020). “Interpreting GPT: the logit lens”. In: *LessWrong*.

Olah, Chris, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter (2020). “Zoom In: An Introduction to Circuits”. In: *Distill*. https://distill.pub/2020/circuits/zoom-in.

Park, Kiho, Yo Joong Choe, and Victor Veitch (2024). “The Linear Representation Hypothesis and the Geometry of Large Language Models”. In: *ICML*. OpenReview.net.

Petti, Samantha, Nicholas Bhattacharya, Roshan Rao, Justas Dauparas, Neil Thomas, Juannan Zhou, Alexander M Rush, Peter Koo, and Sergey Ovchinnikov (Nov. 2022). “End-to-end learning of multiple sequence alignments with differentiable Smith–Waterman”. In: *Bioinformatics* 39.1, btac724.

Radford, Alec, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). “Language Models are Unsupervised Multitask Learners”. In.

Rai, Daking, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao (2024). “A Practical Review of Mechanistic Interpretability for Transformer-Based Language Models”. In: *CoRR* abs/2407.02646.

Rivest, Ronald L. (1992). “The MD5 Message-Digest Algorithm”. In: *RFC* 1321, pp. 1–21.

Thompson, Julie D., Frédéric Plewniak, and Olivier Poch (July 1999). “A comprehensive comparison of multiple sequence alignment programs”. In: *Nucleic Acids Research* 27.13, pp. 2682–2690.

Thorne, JL, H Kishino, and J Felsenstein (1991). “An evolutionary model for maximum likelihood alignment of DNA sequences”. In: *Journal of Molecular Evolution* 33.2, pp. 114–124.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). “Attention is All you Need”. In: *NIPS*, pp. 5998–6008.

Appendix A

Key code extracts

Single lines to long to fit on the page are broken and denoted with "...".

A.1 Constrained beams

This code implements the checking of the valid ways that a beam can be extended.

```
def valid_next_tokens(self, input_seqs):
    """
    Output the tokens that are valid given the input sequences and the tokens
    that have been included in the beam so far.
    """
    valid_next_tokens = []

    remaining = False
    for i in range(len(input_seqs)):
        if self.seq_pointers[i] < len(input_seqs[i]):
            remaining = True
            break
    if remaining:
        #no gaps after consuming all positions in the input sequences,
        #except in last column
        valid_next_tokens.append("-")
    else:
        valid_next_tokens.append("#")
        #if we are currently in the final column, and there is
        #at least one non-gap token in the column, we can allow gaps
        if self.column_entry < len(input_seqs):
            valid_next_tokens.append("-")

    #if we're at the end of the column, ensure that there is
    #a non-gap token in the column before allowing gaps
    if self.seq_pointers[self.column_entry] == len(input_seqs[self.column_entry]) - 1:
        #loop over column entries
        non_gap_found = False
        for i in range(self.n_input_seqs):
            if self.seq_pointers[i] < len(input_seqs[i])
                ...and input_seqs[i][self.seq_pointers[i]] != "-":
                non_gap_found = True
                break
        if not non_gap_found:
            valid_next_tokens = ["#"]
    try:
        valid_next_tokens.append(input_seqs[self.column_entry]
                ...[self.seq_pointers[self.column_entry]])
    except IndexError:
        pass
    return valid_next_tokens
```

A.2 Generating aligned sequences from pair HMM

This code is used in dataset generation.

```
def generate(self):
    """Generate a pair of sequences with an alignment from the pair HMM model."""
    sequence1 = ""
    sequence2 = ""
    alignment1 = ""
    alignment2 = ""
    current_state = START

    while current_state != END:
        #Sample next state based on transition probabilities
        transitions = self.transition_probs[current_state]
        states = list(transitions.keys())
        probs = [transitions[s] for s in states]
        next_state = np.random.choice(states, p=probs)

        #Sample emissions based on state
        if next_state == MATCH:
            #Sample first nucleotide from equilibrium
            residue1 = np.random.choice(list(
                ...self.equilibrium_dist.keys()),
                                      p=list(self.equilibrium_dist.values()))
            #Sample second nucleotide using substitution probability
            probs = [self.substition_prob(
                ...residue1, r) for r in self.equilibrium_dist.keys()]
            residue2 = np.random.choice(list(self.equilibrium_dist.keys()), p=probs)
            sequence1 += residue1
            sequence2 += residue2
            alignment1 += residue1
            alignment2 += residue2
        elif next_state == INSERT:
            residue = np.random.choice(list(self.equilibrium_dist.keys()),
                                       p=list(self.equilibrium_dist.values()))
            sequence2 += residue
            alignment1 += "-"
            alignment2 += residue
        elif next_state == DELETE:
            residue = np.random.choice(list(self.equilibrium_dist.keys()),
                                       p=list(self.equilibrium_dist.values()))
            sequence1 += residue
            alignment1 += residue
            alignment2 += "-"

        current_state = next_state

    return sequence1, sequence2, alignment1, alignment2
```

A.3 LogitLens

This code is used for the LogitLens analysis.

```
def register_hooks(self):
    """Register hooks for all specified decoder layers to apply LogitLens."""
    self.hooks = []

    #Register hooks for each specified decoder layer
    for layer_idx in self.layers_to_analyze:
        layer = self.model.transformer.decoder.layers[layer_idx]

        #Hook for output of the specified decoder layer
        hook = layer.register_forward_hook(
            lambda layer, input, output, idx=layer_idx:
                self.layer_outputs.update({f"decoder_layer_{idx}": output.detach()})
        )
        self.hooks.append(hook)
```