A Mathematical Construction of a Transformer for Bayesian Coin Flip Prediction

Gemini

July 31, 2025

1 Problem Statement

We construct a single-layer, single-head transformer designed to perform next-token prediction on a sequence of coin flips. The sequence consists of tokens representing heads (1), tails (0), and a beginning-of-sequence token (2). The underlying generative process is a coin with a random probability of heads, $p \sim \text{Beta}(1, 1)$ (a uniform prior).

The transformer's task is to output logits at the final sequence position N that correspond to the posterior mean of the Bernoulli distribution for the $(N + 1)$-th flip. Given H heads and T tails in $N = H + T$ trials, the posterior distribution for the coin's bias is $p|\text{data} \sim \text{Beta}(1 + H, 1 + T)$. The posterior predictive probability for the next flip being heads is:

$$P(\text{Heads}|\text{data}) = \frac{1 + H}{2 + N} \tag{1}$$

The target log-odds for the next token are therefore:

$$\log\left(\frac{P(\text{Heads})}{P(\text{Tails})}\right) = \log\left(\frac{1 + H}{1 + T}\right) \tag{2}$$

The transformer must compute the sufficient statistics (H, T) from the input sequence and use them to calculate these log-odds.

2 Model Setup and Notation

We define a minimal transformer architecture.

- **Vocabulary:** $\mathcal{V} = \{0 \text{ (Tails)}, 1 \text{ (Heads)}, 2 \text{ (BOS)}\}$
- **Dimensions:**
  - Model dimension: $d_{\text{model}} = 4$
  - Head dimension: $d_{\text{head}} = 2$
- **Basis Vectors:** We define an orthonormal basis for $\mathbb{R}^{d_{\text{model}}}$:
  - $\mathbf{e}_T = [1, 0, 0, 0]^T$: Basis for Tails token.
  - $\mathbf{e}_H = [0, 1, 0, 0]^T$: Basis for Heads token.
  - $\mathbf{e}_{\text{BOS}} = [0, 0, 1, 0]^T$: Basis for BOS token.
  - $\mathbf{e}_N = [0, 0, 0, 1]^T$: Basis for positional information.

3 Embeddings

3.1 Token Embedding

The token embedding matrix, $W_E \in \mathbb{R}^{|\mathcal{V}| \times d_{\text{model}}}$, maps token IDs to their vector representations.

$$W_E = \begin{bmatrix} \mathbf{e}_T^T \\ \mathbf{e}_H^T \\ \mathbf{e}_{\text{BOS}}^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3}$$

3.2 Positional Embedding

The positional embedding for position i is defined as $\mathbf{p}_i = i \cdot \mathbf{e}_N$. The input to the transformer at sequence position i for token t_i is the sum of its token and positional embeddings:

$$\mathbf{x}_i = W_E[t_i] + \mathbf{p}_i \tag{4}$$

4 Attention Head Construction

The attention head must compute the sufficient statistics H and T. This is achieved by designing the Query-Key (QK) and Output-Value (OV) circuits.

4.1 QK Circuit: Uniform Attention

To act as an unbiased counter over the entire sequence, the attention head must attend uniformly to all past tokens. We achieve this by ensuring the attention scores are constant. Let the query and key weight matrices for the single head ($h = 0$) be zero matrices:

$$W_Q^{(0)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}} = \mathbf{0}, \quad W_K^{(0)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}} = \mathbf{0} \tag{5}$$

The query and key vectors for any input $\mathbf{x}_i$ are then zero vectors:

$$\mathbf{q}_i = W_Q^{(0)T}\mathbf{x}_i = \mathbf{0}, \quad \mathbf{k}_j = W_K^{(0)T}\mathbf{x}_j = \mathbf{0} \tag{6}$$

The attention scores are all zero, $A_{i,j} = \mathbf{q}_i^T\mathbf{k}_j/\sqrt{d_{\text{head}}} = 0$. The softmax function then yields a uniform attention pattern over all source tokens $j \le i$:

$$\alpha_{i,j} = \frac{\exp(0)}{\sum_{k=0}^{i}\exp(0)} = \frac{1}{i+1} \tag{7}$$

4.2 OV Circuit: Counting Heads and Tails

The OV circuit's role is to count the occurrences of Head and Tail tokens. The value vector $\mathbf{v}_j$ is computed as $\mathbf{v}_j = W_V^{(0)T}\mathbf{x}_j$. We define $W_V^{(0)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ to isolate the token information. The first dimension of d_{head} will count heads, and the second will count tails.

$$W_V^{(0)} = \begin{bmatrix}\mathbf{e}_T & \mathbf{e}_H\end{bmatrix}^T W_V' = \begin{bmatrix}0 & 1\\ 1 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix} \tag{8}$$

This matrix projects the $\mathbf{e}_H$ component of the input to $[1, 0]^T$ and the $\mathbf{e}_T$ component to $[0, 1]^T$, while ignoring the BOS and positional components. The mixed value vector $\mathbf{z}_N$ at the final position N is the weighted sum of value vectors:

$$\mathbf{z}_N = \sum_{j=0}^{N}\alpha_{N,j}\mathbf{v}_j = \frac{1}{N+1}\sum_{j=0}^{N}W_V^{(0)T}\mathbf{x}_j \tag{9}$$

$$= \frac{1}{N+1}\sum_{j=1}^{N}W_V^{(0)T}W_E[t_j] \quad (\text{since } W_V^{(0)T}\mathbf{e}_{\text{BOS}} = \mathbf{0} \text{ and } W_V^{(0)T}\mathbf{e}_N = \mathbf{0}) \tag{10}$$

$$= \frac{1}{N+1}\begin{bmatrix}H\\ T\end{bmatrix} \tag{11}$$

where H is the total number of heads and T is the total number of tails in the sequence $t_1, \ldots, t_N$.

The head's output is $\mathbf{attn\_out}_N = W_O^{(0)T}\mathbf{z}_N$. We define $W_O^{(0)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ to project these counts back into the residual stream, into the $\mathbf{e}_H$ and $\mathbf{e}_T$ directions respectively.

$$W_O^{(0)} = \begin{bmatrix}\mathbf{e}_H & \mathbf{e}_T\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}^T \tag{12}$$

Thus, the final output of the attention head is:

$$\mathbf{attn\_out}_N = \frac{1}{N+1}(H \cdot \mathbf{e}_H + T \cdot \mathbf{e}_T) \tag{13}$$

5 MLP Input

The input to the MLP at position N is the state of the residual stream after the attention block:

$$\begin{aligned} \mathbf{resid\_mid}_N &= \mathbf{x}_N + \mathbf{attn\_out}_N && (14) \\ &= (W_E[t_N] + \mathbf{p}_N) + \frac{1}{N+1}(H \cdot \mathbf{e}_H + T \cdot \mathbf{e}_T) && (15) \end{aligned}$$

This vector contains all the necessary information for the MLP:

- N is encoded in the $\mathbf{e}_N$ component from $\mathbf{p}_N$.
- $H/(N+1)$ is encoded in the $\mathbf{e}_H$ component.
- $T/(N+1)$ is encoded in the $\mathbf{e}_T$ component.

6 MLP's Task (Conceptual)

The MLP would be constructed to perform the required non-linear computation. Conceptually, it would implement the following algorithm:

1. **Extract Statistics:**

$$\begin{aligned} N_{\text{extracted}} &= \mathbf{resid\_mid}_N^T \mathbf{e}_N \\ H_{\text{scaled}} &= \mathbf{resid\_mid}_N^T \mathbf{e}_H \quad \implies H = H_{\text{scaled}} \cdot (N_{\text{extracted}} + 1) \\ T_{\text{scaled}} &= \mathbf{resid\_mid}_N^T \mathbf{e}_T \quad \implies T = T_{\text{scaled}} \cdot (N_{\text{extracted}} + 1) \end{aligned}$$

2. **Compute Log-Odds:** Calculate $l = \log(1+H) - \log(1+T)$.
3. **Produce Output:** The MLP's output, $\mathbf{mlp\_out}_N$, would be constructed to influence the final logits. A simple choice is $\mathbf{mlp\_out}_N = l \cdot (\mathbf{e}_H - \mathbf{e}_T)$. The final residual stream state $\mathbf{resid\_post}_N = \mathbf{resid\_mid}_N + \mathbf{mlp\_out}_N$ is then passed to the unembedding layer. With an unembedding matrix $W_U = W_E^T$, the logit for Heads would be greater than the logit for Tails by a factor proportional to l.

7 MLP Mechanism

The MLP block receives the sufficient statistics (H, T, N) encoded in the residual stream vector $\mathbf{resid\_mid}_N$. We can simplify its input to be $\mathbf{x}_{\text{in}} = H \cdot \mathbf{e}_H + T \cdot \mathbf{e}_T + N \cdot \mathbf{e}_N$. The MLP's objective is to compute the logit vector $[\log(1+T), \log(1+H), -\infty]^T$. We outline two potential mechanisms.

7.1 Method 1: MLP as a Lookup Table

For a discrete and finite input domain ($H, T \in \{0, 1, \ldots, n_{\text{ctx}}\}$), the MLP can implement a lookup table. This is achieved in two stages within the MLP's single hidden layer.

7.1.1 Stage 1: One-Hot Encoding with Indicator Neurons

The hidden layer generates a one-hot representation of the inputs H and T. For each integer value $h \in [0, n_{\text{ctx}}]$, a pair of neurons can create an indicator function $\mathbb{I}(H = h)$. Let neuron $N_h^{\geq}$ have pre-activation $H - h + 0.5$, and neuron $N_{h+1}^{\geq}$ have pre-activation $H - h - 0.5$. The activity $\text{ReLU}(H - h + 0.5) - \text{ReLU}(H - h - 0.5)$ is approximately 1 if $H = h$ and 0 otherwise for integer H. Let the hidden activation vector $\mathbf{h}_{\text{mlp}} \in \mathbb{R}^{d_{\text{mlp}}}$ contain sub-vectors that are one-hot encodings for H and T.

7.1.2 Stage 2: Storing Log Values in Output Weights

The output weights $W_{\text{out}} \in \mathbb{R}^{d_{\text{mlp}} \times d_{\text{vocab\_out}}}$ store the pre-computed log values. Let the indicator neuron for $H = h$ be N_h^H. Its corresponding row in the output weight matrix is:

$$W_{\text{out}}[N_h^H, :] = [0, \log(1 + h), -C] \tag{16}$$

Similarly, for the indicator neuron N_t^T for $T = t$:

$$W_{\text{out}}[N_t^T, :] = [\log(1 + t), 0, -C] \tag{17}$$

where C is a large positive constant. When the input is (H, T), only neurons N_H^H and N_T^T are active. The final logits are a linear combination of the rows of W_{out}, which effectively selects and sums the appropriate log values, yielding the desired output $[\log(1 + T), \log(1 + H), -2C]$.

7.2 Method 2: MLP as a Function Approximator

Alternatively, the MLP can learn a piecewise linear approximation of the function $f(x) = \log(1 + x)$.

7.2.1 Stage 1: Basis of Ramp Functions

A set of neurons in the hidden layer can be dedicated to approximating $\log(1+H)$. Their input weights in W_{in} would be aligned with $\mathbf{e}_H$. Their biases b_i would be spaced at negative values (e.g., $-0.5, -1.5, \dots$). The activation of these neurons, $\text{ReLU}(H + b_i)$, forms a basis of ramp functions, each starting at a different integer value of H. A similar set of neurons would be dedicated to T.

7.2.2 Stage 2: Learning Coefficients

The output weights in W_{out} serve as the coefficients for the linear combination of these ramp functions. Through training, the model learns the optimal coefficients to approximate $\log(1 + H)$ and $\log(1 + T)$ and places these values in the corresponding `logit_H` and `logit_T` output dimensions. This method is more parameter-efficient and offers the potential for generalization beyond the training context length, at the cost of being an approximation.

8 Conclusion

This construction demonstrates how a simple transformer with a single attention head can be designed to implement a Bayesian update rule. The QK circuit creates a uniform attention distribution for unbiased aggregation, while the OV circuit is engineered to count the occurrences of specific tokens, thereby computing the sufficient statistics required for the task. The MLP then uses this information to compute the final posterior predictive logits.