Transformers

STATS305B: Applied Statistics II

Scott Linderman

February 26, 2025

Last Time...

- (Probabilistic) PCA as a (stochastic) linear autoencoder
- Variational Autoencoders (VAEs)
- Review of Gradient-based Variational Inference
- Demo

Today...

Outline:

- Transformers
- Self-Attention
- MLPs
- Other Tricks

Transformers

Transformers are deep neural networks that underly modern large language models like ChatGPT, Gemini, and Claude.

They are also widely used for computer vision tasks by treating each patch of an image as a "token."

The basic building blocks of a transformer include: self-attention, token-wise nonlinear transformations, layer norm, and positional encodings.

We will focus on modeling sequential data. We will follow the presentation of Turner et al. (2023), but we will make some slight modifications to the notation to be consistent with Homework 4.

Preliminaries

Let $\boldsymbol{X}^{(0)} \in \mathbb{R}^{T \times D}$ denote our data matrix with row $\boldsymbol{x}_t^{(0)} \in \mathbb{R}^D$ representing the t-th **token** in the sequence. For example, the tokens could be a vector embedding of a word, sub-word, or character. The embeddings may be fixed or learned as part of the model.

The output of the transformer will be another matrix of the same shape, $\boldsymbol{X}^{(M)} \in \mathbb{R}^{T \times D}$. These output features can be used for downstream tasks like sentiment classification, machine translation, or autoregressive modeling.

The output results from a stack of transformer blocks,

$$\boldsymbol{X}^{(m)} = \texttt{transformer-block}(\boldsymbol{X}^{(m-1)})$$

Each block consists of two stages: one that operates vertically, combining information across the sequence length; another that operates horizontally, combining information across feature dimensions.

Preliminaries

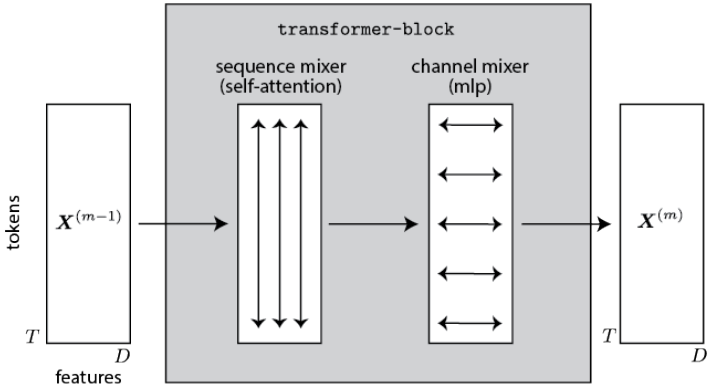

Figure: Transformer Block

Attention

The first stage combines information across sequence length using a mechanism called **attention**.

Mathematically, attention is just a weighted average,

$$\boldsymbol{Y}^{(m)} = \boldsymbol{A}^{(m)} \boldsymbol{X}^{(m-1)},$$

where $\boldsymbol{A}^{(m)} \in \mathbb{R}_{+}^{T \times T}$ is a row-stochastic attention matrix.

That is, $\sum_s A_{ts}^{(m)} = 1$ for all t.

Intuitively, $A_{t,s}^{(m)}$ indicates how much output location t attends to input location s.

Attention

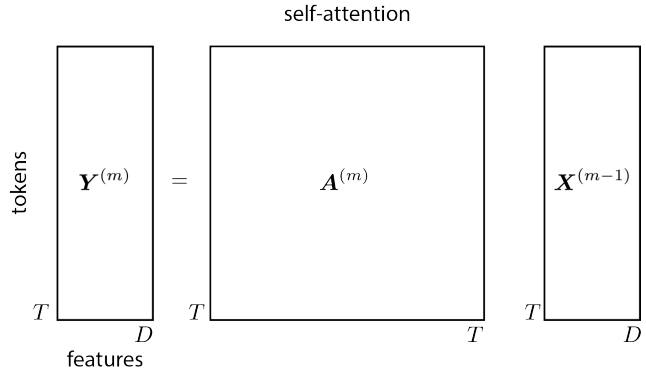

Figure: Self Attention

Self-Attention

Where does the attention matrix come from? In a transformer, the attention weights are determined by the pairwise similarity of tokens in the sequence.

The simplest instantiation of this idea would be something like,

$$A_{t,s} \propto \exp\left\{\boldsymbol{x}_t^\top \boldsymbol{x}_s\right\}$$

Once normalized,

$$A_{t,s} = \frac{\exp\{\boldsymbol{x}_t^\top \boldsymbol{x}_s\}}{\sum_{s'=1}^{T} \exp\{\boldsymbol{x}_t^\top \boldsymbol{x}_{s'}\}}.$$

(Note: we have dropped the superscript $^{(m)}$ for clarity in this section.)

Self-Attention

This approach implies that attention depends equally on all D dimensions of the embedding, but some dimensions may convey different kinds of information that may be of greater or lesser relevance for the attention mechanism.

One way to allow for this is to project the tokens into a feature space before computing the attention weights,

$$A_{t,s} = \frac{\exp\{(\boldsymbol{U}\boldsymbol{x}_t)^\top(\boldsymbol{U}\boldsymbol{x}_s)\}}{\sum_{s'=1}^{T}\exp\{(\boldsymbol{U}\boldsymbol{x}_t)^\top(\boldsymbol{U}\boldsymbol{x}_{s'})\}}$$

where $\boldsymbol{U} \in \mathbb{R}^{K\times D}$ with $K < D$.

Self-Attention

If the attention weight specifies how relevant token $\boldsymbol{x}_s$ is to updating the feature representation for token $\boldsymbol{x}_t$, then directionality may matter. Transformers address this asymmetry with,

$$A_{t,s} = \frac{\exp\{(\boldsymbol{U}_q \boldsymbol{x}_t)^\top (\boldsymbol{U}_k \boldsymbol{x}_s)\}}{\sum_{s'=1}^{T} \exp\{(\boldsymbol{U}_q \boldsymbol{x}_t)^\top (\boldsymbol{U}_k \boldsymbol{x}_{s'})\}},$$

where $\boldsymbol{U}_q \boldsymbol{x}_t \in \mathbb{R}^K$ are the **queries** and $\boldsymbol{U}_k \boldsymbol{x}_s \in \mathbb{R}^K$ are the **keys**.

The parameters $\boldsymbol{U}_q \in \mathbb{R}^{K \times D}$ and $\boldsymbol{U}_k \in \mathbb{R}^{K \times D}$ define the self-attention mechanism.

Self-Attention

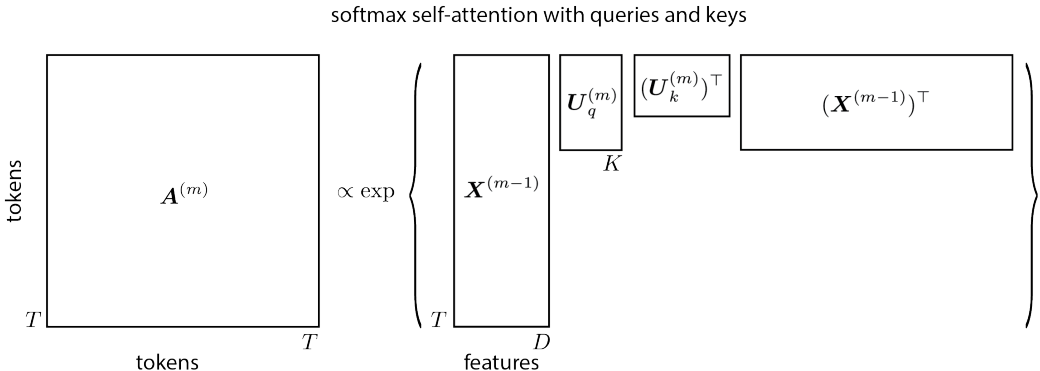

Figure: Self Attention with Queries and Keys

Causal Self-Attention

When we are using transformers for autoregressive sequence modeling, we constrain the attention matrix to be **causal** by requiring $A_{t,s}^{(m)} = 0$ for all $t < s$.

In other words, the matrix is **lower triangular**.

To enforce this constraint, we simply mask the upper triangular part of the attention matrix and normalize the rows appropriately,

$$A_{t,s} = \frac{\exp\{(\boldsymbol{U}_q \boldsymbol{x}_t)^\top (\boldsymbol{U}_k \boldsymbol{x}_s)\}}{\sum_{s'=1}^{t} \exp\{(\boldsymbol{U}_q \boldsymbol{x}_t)^\top (\boldsymbol{U}_k \boldsymbol{x}_{s'})\}} \cdot \mathbb{I}[t \geq s].$$

Comparison to RNNs

Compare the self-attention mechanism to the information processing in an RNN.

Rather than propagating information about past tokens via a hidden state, a transformer with causal attention can directly attend to any one of the past tokens.

Comparison to Convolutional Neural Networks (CNNs)

If the attention weights were only a function of the distance between tokens, $A_{t,s} = a_{t-s}$, then the matrix $\boldsymbol{A}$ would be a **Toeplitz matrix**.

Multiplication by a Toeplitz matrix corresponds to a discrete convolution.

From this perspective, we can think of the attention mechanism in a transformer as a generalization of convolution that allows for input-dependent and time-varying filters.

Multi-Headed Self-Attention

Just as in a CNN each layer performs convolutions with a bank of filters in parallel, in a transformer each block uses a bank of H **attention heads** in parallel.

Let,

$$\boldsymbol{Y}^{(m,h)} = \boldsymbol{A}^{(m,h)} \boldsymbol{X}^{(m-1)} \in \mathbb{R}^{T \times D}$$

where

$$A_{t,s}^{(m,h)} = \frac{\exp\{(\boldsymbol{U}_q^{(m,h)} \boldsymbol{x}_t^{(m-1)})^\top (\boldsymbol{U}_k^{(m,h)} \boldsymbol{x}_s^{(m-1)})\}}{\sum_{s'=1}^{T} \exp\{(\boldsymbol{U}_q^{(m,h)} \boldsymbol{x}_t^{(m-1)})^\top (\boldsymbol{U}_k^{(m,h)} \boldsymbol{x}_{s'}^{(m-1)})\}},$$

is an attention weight at layer m and head h for $h = 1, \ldots, H$.

(Now you see why we dropped superscripts above — the notation is a handful!)

Multi-Headed Self-Attention

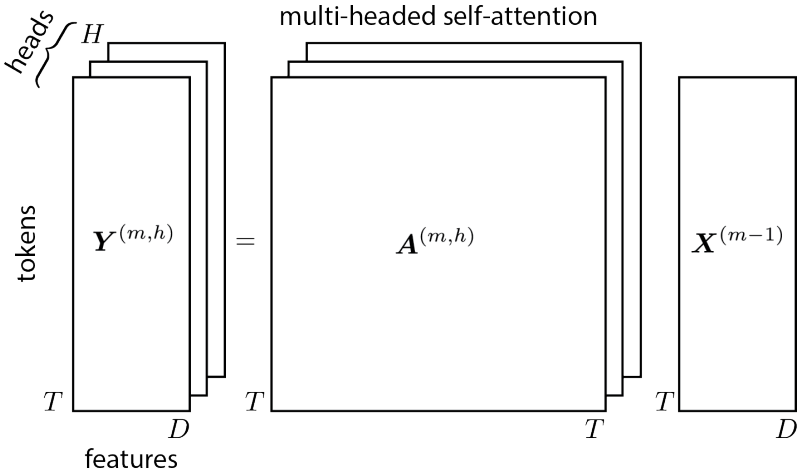

Figure: Multi-Headed Self Attention

Queries, Keys, and Values

The original transformer paper presents the output of a single head as a set of values,

$$\boldsymbol{Y}^{(m,h)} = \boldsymbol{A}^{(m,h)}\big(\boldsymbol{X}^{(m-1)}\big(\boldsymbol{U}_v^{(m,h)}\big)^\top\big) \in \mathbb{R}^{T\times K}$$

where $\boldsymbol{U}_v^{(m,h)} \in \mathbb{R}^{K\times D}$ is a **value matrix**.

Then the attention mechanism can be though of as taking a weighted average of **values** $\boldsymbol{U}_v\boldsymbol{x}_t$ where the weights are determined by an inner product between the queries and keys.

The final output is projected back into the original token dimension and linearly combined,

$$\boldsymbol{Y}^{(m)} = \sum_{h=1}^{H} \boldsymbol{Y}^{(m,h)}\big(\boldsymbol{U}_o^{(m,h)}\big)^\top$$

where $\boldsymbol{U}_o^{(m,h)} \in \mathbb{R}^{D\times K}$ is an **output matrix** that maps from values to new tokens.

This formulation corresponds to a low-rank read-out matrix $\boldsymbol{V} = \boldsymbol{U}_o\boldsymbol{U}_v^\top$, where we have again dropped the superscripts for clarity.

Token-wise Nonlinearity

After applying the multi-headed self-attention to obtain $\boldsymbol{Y}^{(m)}$, the transformer applies a token-wise nonlinear transformation to nonlinearly mix the feature dimensions.

This is done with a simple feedforward neural network, also known as a **multilayer perceptron (MLP)**,

$$\boldsymbol{x}_t^{(m)} = \texttt{mlp}(\boldsymbol{y}_t^{(m)})$$

Note that the same function is applied to all positions t.

Token-wise Nonlinearity

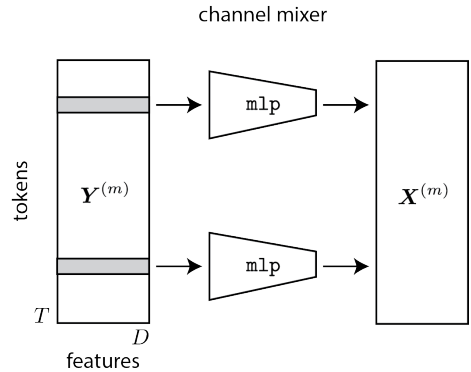

Figure: MLP

Computational Complexity

The cost of self-attention is $\mathcal{O}(T^2K)$. It scales quadratically with the sequence length (or more specifically, the length of the context window).

The MLP typically has hidden dimensions of at least D, so the computational complexity of this step is at least $\mathcal{O}(TD^2)$.

For transformers with very large featured dimensions, the MLP can be the dominant cost.

Residual Connections

Rather than parameterizing $\boldsymbol{X}^{(m)}$ as the output of the MLP, a common trick in deep learning is to use residual connections.

Simple idea: when the input and output are of the same dimensionality, we can let the network learn the **residual**, which is typically smaller in magnitude than the overall function.

Transformers use residual connections for both the multi-headed self-attention step and the MLP. So,

$$\boldsymbol{Y}^{(m)} = \boldsymbol{X}^{(m-1)} + \texttt{mhsa}(\boldsymbol{X}^{(m-1)})$$
$$\boldsymbol{X}^{(m)} = \boldsymbol{Y}^{(m)} + \texttt{mlp}(\boldsymbol{Y}^{(m)})$$

Residual Connections

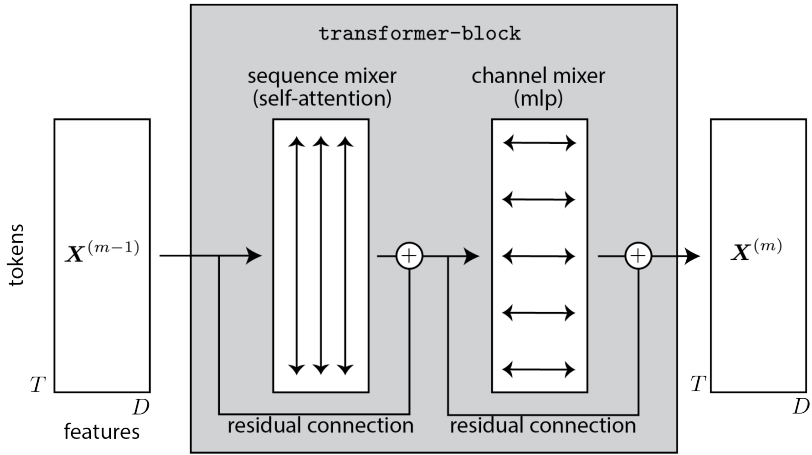

Figure: Residual Connections

Layer Norm

Finally, it is important to use other deep learning "tricks" like LayerNorm to stabilize training. In a transformer, LayerNorm amounts to z-scoring each token $\boldsymbol{x}_t$ and then shifting and scaling each feature dimension,

$$\texttt{layer-norm}(\boldsymbol{x}_t) = \boldsymbol{\beta} + \boldsymbol{\gamma} \odot \left(\frac{\boldsymbol{x}_t - \texttt{mean}(\boldsymbol{x}_t)}{\texttt{std}(\boldsymbol{x}_t)} \right)$$

where

$$\texttt{mean}(\boldsymbol{x}_t) = \frac{1}{D} \sum_{d=1}^{D} x_{t,d}$$

$$\texttt{std}(\boldsymbol{x}_t) = \left(\frac{1}{D} \sum_{d=1}^{D} (x_{t,d} - \texttt{mean}(\boldsymbol{x}_t))^2 \right)^{\frac{1}{2}}$$

and $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^D$ are learned parameters.

Layer Norm

LayerNorm is typically applied before the multi-headed self-attention and MLP steps,

$$
\begin{aligned}
\overline{\boldsymbol{X}}^{(m-1)} &= \texttt{layer-norm}(\boldsymbol{X}^{(m-1)}) \\
\boldsymbol{Y}^{(m)} &= \overline{\boldsymbol{X}}^{(m-1)} + \texttt{mhsa}(\overline{\boldsymbol{X}}^{(m-1)}) \\
\overline{\boldsymbol{Y}}^{(m)} &= \texttt{layer-norm}(\boldsymbol{Y}^{(m)}) \\
\boldsymbol{X}^{(m)} &= \overline{\boldsymbol{Y}}^{(m)} + \texttt{mlp}(\overline{\boldsymbol{Y}}^{(m)})
\end{aligned}
$$

This defines one `transformer-block`.

A transformer stacks M `transformer-blocks` on top of one another to produce a deep sequence-to-sequence model.

Positional Encodings

Except for the lower triangular constraint on the attention matrices, the transformer architecture knows nothing about the relative positions of the tokens.

Absent this constraint, the transformer essentially treats the data as an **unordered set** of tokens.

This can actually be a feature! It allows the transformer to act on a wide range of datasets aside from just sequences.

However, when the data posses some spatial or temporal structure, it is helpful to include that information in the embedding. A simple way to do so is to add position and content in the token,

$$\boldsymbol{x}_t^{(0)} = \boldsymbol{c}_t + \boldsymbol{p}_t,$$

where $\boldsymbol{c}_t \in \mathbb{R}^D$ is an embedding of the content and $\boldsymbol{p}_t \in \mathbb{R}^D$ encodes the position (e.g., with a set of sinusoidal basis functions).

Autoregressive Modeling

To use a transformer for autoregressive modeling, we need to make predictions from the final layer's representations.

If the goal is to predict the next word label $\ell_{t+1} \in \{1, \ldots, V\}$ based on encodings of past words $\boldsymbol{x}_{1:t}^{(0)}$, where V is the vocabulary size, then we could use a categorical distribution,

$$\ell_{t+1} \sim \mathrm{Cat}(\mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t^{(M)}))$$

where $\boldsymbol{W} \in \mathbb{R}^{V \times D}$.

Like the hidden states in an RNN, the final layer's representations $\boldsymbol{x}_t^{(M)}$ combine information from all tokens up to and including index t.

Training

Training deep neural networks is somewhat of a dark art.

Standard practice is to use the Adam optimizer with a bag of tricks including gradient clipping, learning rate annealing schedules, increasing minibatch sizes, dropout, etc.

Generally, treat these algorithmic decisions as hyperparameters to be tuned.

We won't try to put any details in writing lest you overfit to them.

Conclusion

Transformers are a workhorse of modern machine learning and key to many of the impressive advances over recent years.

However, there are still areas for improvement. For example, the computational cost of attention is $\mathcal{O}(T^2)$, and a lot of work has gone into cutting that down.

Likewise, while the transformer allows for predictions to be made in parallel across an entire sequence, sampling from the learned autoregressive model is still takes linear time.

In the lectures ahead, we'll discuss other recent architectures that address some of these concerns.