Exploring Transformer-Based Models for Cascading Extremes

Clemente Ferrer

clemente.ferrer@usm.cl

Department of Mathematics
Universidad Técnica Federico Santa María

Joint work with **Miguel de Carvalho** & **Ronny Vallejos**

Workshop on Generative AI for Extreme Events

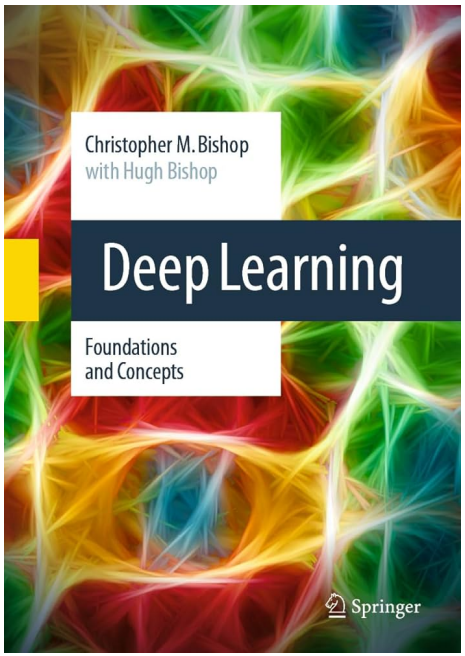
Edinburgh, UK
June 13, 2025

Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).

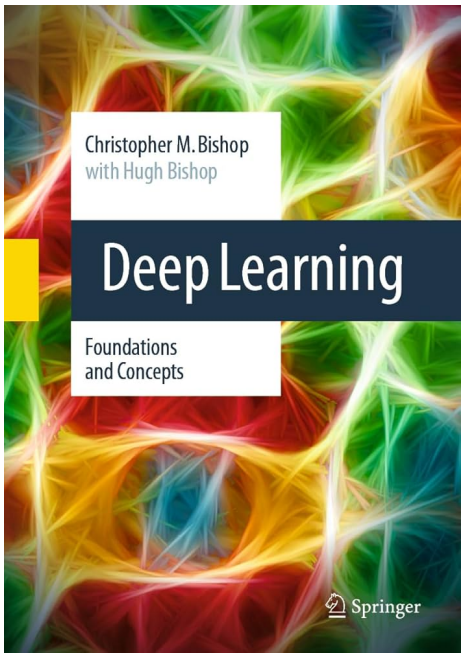


Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).

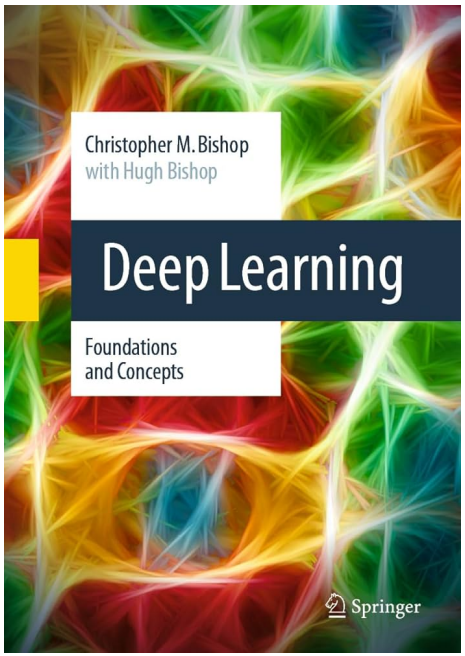


Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).

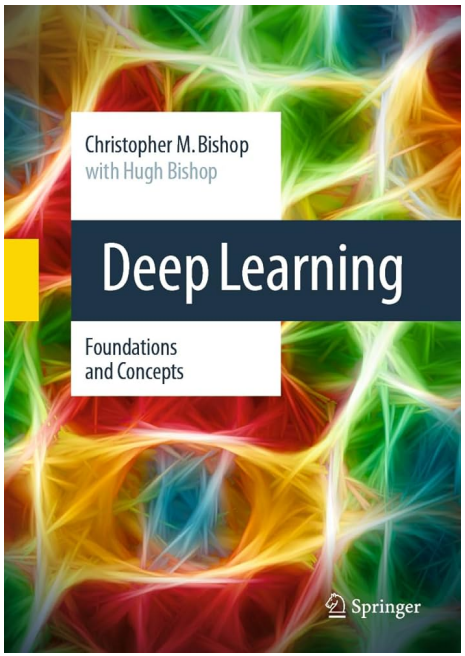


Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).

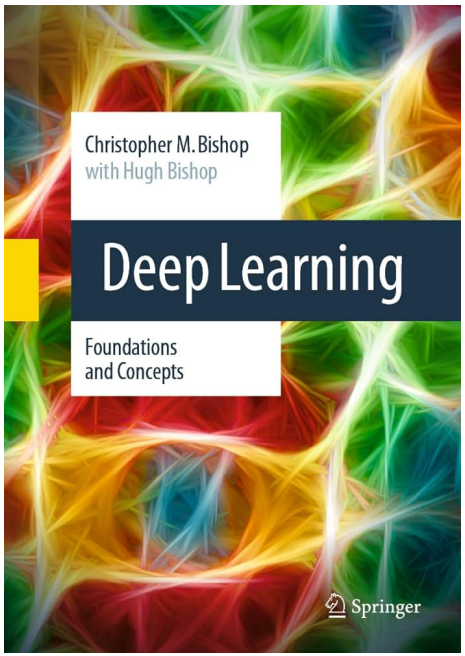


Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).

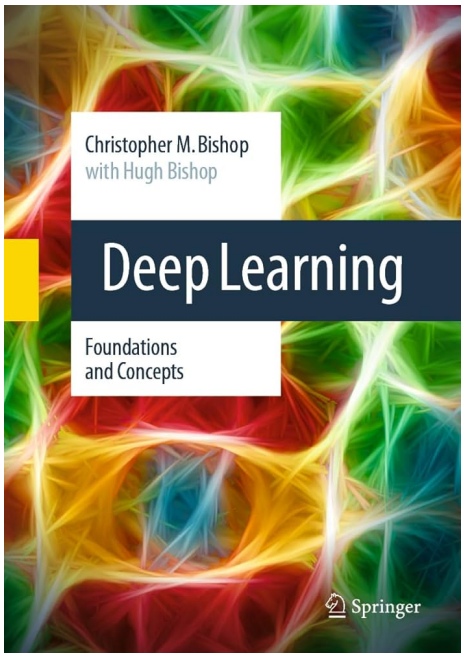


Literature Review

During this talk I will cover the following topics:

- Why use Transformers?
- A Literature Review on Transformers.
- Transformer in Statistics.
- Large Language Models and Generative Pre-trained Transformers (GPT).
- Ideas on Transformers for Chained Extremes.

Most of the figures and ideas in this talk were taken from the book by Bishop and Bishop (2024).



Why use Transformers?

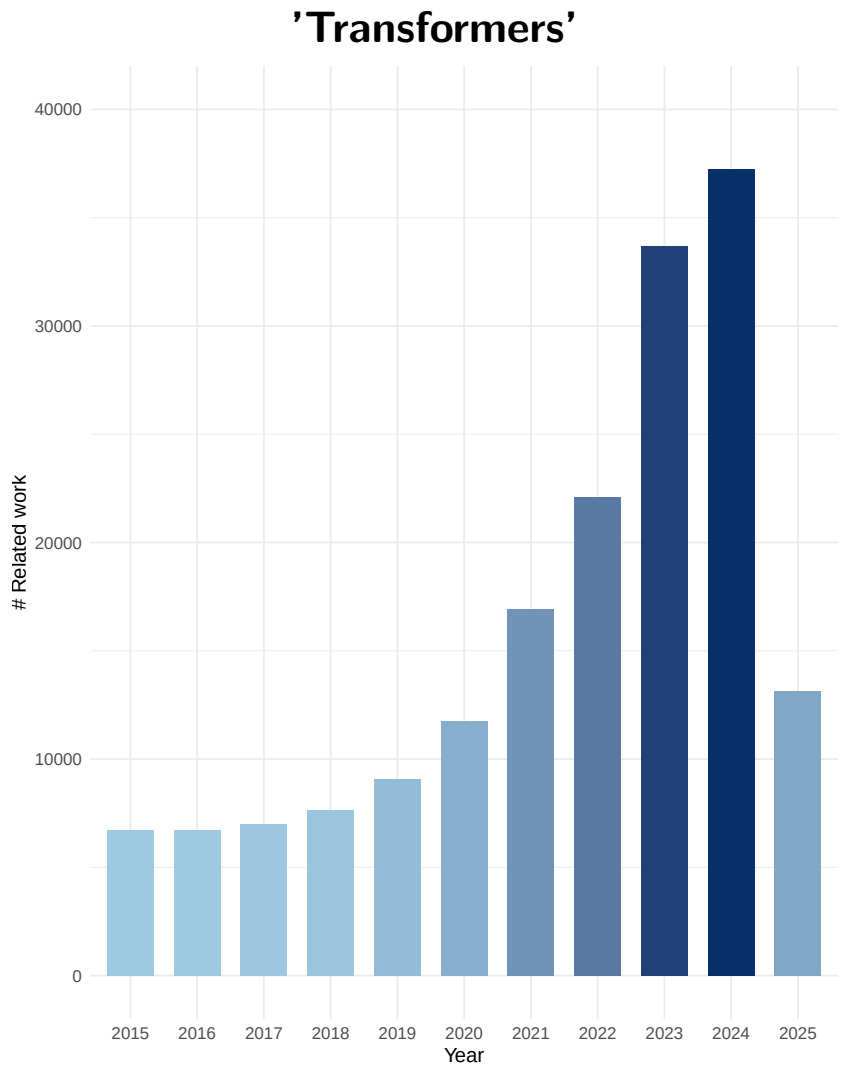


Figure: Number of works (articles, preprints, book chapter) referring to 'Transformers' since 2015 (Source: OpenAlex.org).

Literature Review

Suppose N tokens

$$\mathbf{x}_1, \ldots, \mathbf{x}_N \quad \leftarrow \quad \text{Input data.}$$

where $\mathbf{x}_n = (x_{n1}, \ldots, x_{nD})^{\mathrm{T}} \in \mathbb{R}^D$, $n = 1, \ldots, N$. The elements of the tokens are called features.

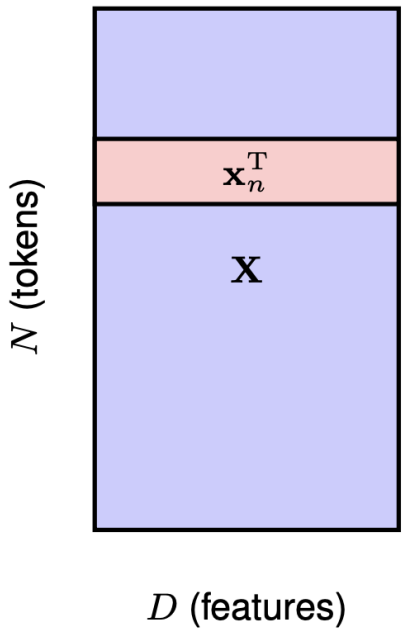


Figure: The structure of the data matrix $\mathbf{X}$, of dimension $N \times D$, in which row n represents the transposed data vector $\mathbf{x}_n^{\mathrm{T}}$.

Literature Review

Suppose N tokens

$$\mathbf{x}_1, \ldots, \mathbf{x}_N \quad \leftarrow \quad \text{Input data.}$$

where $\mathbf{x}_n = (x_{n1}, \ldots, x_{nD})^{\mathrm{T}} \in \mathbb{R}^D$, $n = 1, \ldots, N$. The elements of the tokens are called features.

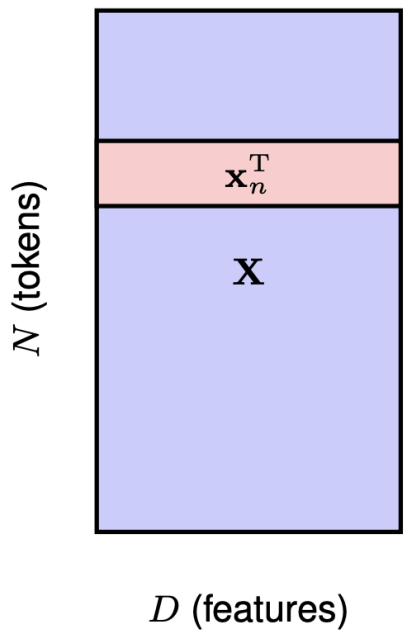


Figure: The structure of the data matrix $\mathbf{X}$, of dimension $N \times D$, in which row n represents the transposed data vector $\mathbf{x}_n^{\mathrm{T}}$.

Literature Review

Goal

Define a transformation from tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to output embeddings $\mathbf{y}_1, \ldots, \mathbf{y}_N$, where each $\mathbf{y}_n$ is a weighted combination of **ALL** $\mathbf{x}_m$, with weights reflecting the relevance of $\mathbf{x}_m$ to $\mathbf{y}_n$.

Proposal

$$\mathbf{y}_n = \sum_{m=1}^{N} \alpha_{nm} \mathbf{x}_m,$$

where $\alpha_{nm} \geq 0$ are attention weights such that $\sum_{m=1}^{N} \alpha_{nm} = 1$.

First approach

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)}{\sum_{m'=1}^{N} \exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_{m'})} \quad \leftarrow \quad \text{Softmax}.$$

Using matrix notation

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{X} \quad \leftarrow \quad \textbf{Self-Attention}.$$

Literature Review

Goal

Define a transformation from tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to output embeddings $\mathbf{y}_1, \ldots, \mathbf{y}_N$, where each $\mathbf{y}_n$ is a weighted combination of **ALL** $\mathbf{x}_m$, with weights reflecting the relevance of $\mathbf{x}_m$ to $\mathbf{y}_n$.

Proposal

$$\mathbf{y}_n = \sum_{m=1}^{N} \alpha_{nm} \mathbf{x}_m,$$

where $\alpha_{nm} \geq 0$ are attention weights such that $\sum_{m=1}^{N} \alpha_{nm} = 1$.

First approach

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)}{\sum_{m'=1}^{N} \exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_{m'})} \quad \leftarrow \quad \text{Softmax}.$$

Using matrix notation

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{X} \quad \leftarrow \quad \textbf{Self-Attention}.$$

Goal

Define a transformation from tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to output embeddings $\mathbf{y}_1, \ldots, \mathbf{y}_N$, where each $\mathbf{y}_n$ is a weighted combination of **ALL** $\mathbf{x}_m$, with weights reflecting the relevance of $\mathbf{x}_m$ to $\mathbf{y}_n$.

Proposal

$$\mathbf{y}_n = \sum_{m=1}^{N} \alpha_{nm} \mathbf{x}_m,$$

where $\alpha_{nm} \geq 0$ are attention weights such that $\sum_{m=1}^{N} \alpha_{nm} = 1$.

First approach

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)}{\sum_{m'=1}^{N} \exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_{m'})} \quad \leftarrow \quad \text{Softmax.}$$

Using matrix notation

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{X} \quad \leftarrow \quad \textbf{Self-Attention.}$$

Goal

Define a transformation from tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to output embeddings $\mathbf{y}_1, \ldots, \mathbf{y}_N$, where each $\mathbf{y}_n$ is a weighted combination of **ALL** $\mathbf{x}_m$, with weights reflecting the relevance of $\mathbf{x}_m$ to $\mathbf{y}_n$.

Proposal

$$\mathbf{y}_n = \sum_{m=1}^{N} \alpha_{nm} \mathbf{x}_m,$$

where $\alpha_{nm} \geq 0$ are attention weights such that $\sum_{m=1}^{N} \alpha_{nm} = 1$.

First approach

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)}{\sum_{m'=1}^{N} \exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_{m'})} \quad \leftarrow \quad \text{Softmax}.$$

Using matrix notation

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{X} \quad \leftarrow \quad \textbf{Self-Attention}.$$

Goal

Define a transformation from tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to output embeddings $\mathbf{y}_1, \ldots, \mathbf{y}_N$, where each $\mathbf{y}_n$ is a weighted combination of **ALL** $\mathbf{x}_m$, with weights reflecting the relevance of $\mathbf{x}_m$ to $\mathbf{y}_n$.

Proposal

$$\mathbf{y}_n = \sum_{m=1}^{N} \alpha_{nm} \mathbf{x}_m,$$

where $\alpha_{nm} \geq 0$ are attention weights such that $\sum_{m=1}^{N} \alpha_{nm} = 1$.

First approach

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)}{\sum_{m'=1}^{N} \exp(\mathbf{x}_n^{\mathrm{T}} \mathbf{x}_{m'})} \quad \leftarrow \quad \text{Softmax}.$$

Using matrix notation

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{X} \quad \leftarrow \quad \textbf{Self-Attention}.$$

Literature Review

Limitation

The mapping $\{\mathbf{x}_n\} \mapsto \{\mathbf{y}_n\}$ **lacks learnable parameters** and thus cannot adapt to data. Additionally, all features in each $\mathbf{x}_n$ contribute equally to attention weights, limiting the model's ability to focus selectively on informative features.

Partial solution

Define

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{U}, \quad \mathbf{U} \in \mathbb{R}^{D \times D},$$

where $\mathbf{U}$ is learnable, analogous to a 'layer' in standard neural networks. Then

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})\mathbf{X}\mathbf{U}.$$

Still a problem

Although this has much more flexibility, the matrix

$$\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}$$

is symmetric, whereas we would like the **attention mechanism** to support **asymmetry**.

Literature Review

Limitation

The mapping $\{\mathbf{x}_n\} \mapsto \{\mathbf{y}_n\}$ **lacks learnable parameters** and thus cannot adapt to data. Additionally, all features in each $\mathbf{x}_n$ contribute equally to attention weights, limiting the model's ability to focus selectively on informative features.

Partial solution

Define

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{U}, \quad \mathbf{U} \in \mathbb{R}^{D \times D},$$

where $\mathbf{U}$ is learnable, analogous to a 'layer' in standard neural networks. Then

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})\mathbf{X}\mathbf{U}.$$

Still a problem

Although this has much more flexibility, the matrix

$$\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}$$

is symmetric, whereas we would like the **attention mechanism** to support **asymmetry**.

Literature Review

Limitation

The mapping $\{\mathbf{x}_n\} \mapsto \{\mathbf{y}_n\}$ **lacks learnable parameters** and thus cannot adapt to data. Additionally, all features in each $\mathbf{x}_n$ contribute equally to attention weights, limiting the model's ability to focus selectively on informative features.

Partial solution

Define

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{U}, \quad \mathbf{U} \in \mathbb{R}^{D \times D},$$

where $\mathbf{U}$ is learnable, analogous to a 'layer' in standard neural networks. Then

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})\mathbf{X}\mathbf{U}.$$

Still a problem

Although this has much more flexibility, the matrix

$$\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}$$

is symmetric, whereas we would like the **attention mechanism** to support **asymmetry**.

Literature Review

Limitation

The mapping $\{\mathbf{x}_n\} \mapsto \{\mathbf{y}_n\}$ **lacks learnable parameters** and thus cannot adapt to data. Additionally, all features in each $\mathbf{x}_n$ contribute equally to attention weights, limiting the model's ability to focus selectively on informative features.

Partial solution

Define

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{U}, \quad \mathbf{U} \in \mathbb{R}^{D \times D},$$

where $\mathbf{U}$ is learnable, analogous to a 'layer' in standard neural networks. Then

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})\mathbf{X}\mathbf{U}.$$

Still a problem

Although this has much more flexibility, the matrix

$$\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}$$

is symmetric, whereas we would like the **attention mechanism** to support **asymmetry**.

Solution

Vaswani et al. (2017) proposed defining separate query, key, and value matrices each having their own independent linear transformations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{(\mathrm{q})}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{(\mathrm{k})}, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{(\mathrm{v})},$$

where $\mathbf{W}^{(\mathrm{q})}$, $\mathbf{W}^{(\mathrm{k})} \in \mathbb{R}^{D \times D_{\mathrm{k}}}$ and $\mathbf{W}^{(\mathrm{v})} \in \mathbb{R}^{D \times D_{\mathrm{v}}}$, where D_{v} governs the dimensionality of the output. Therefore

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{Q}\mathbf{K}^{\mathrm{T}})\mathbf{V} \quad \leftarrow \quad \mathbf{Q}\mathbf{K}^{\mathrm{T}} \text{are the attention weights.}$$

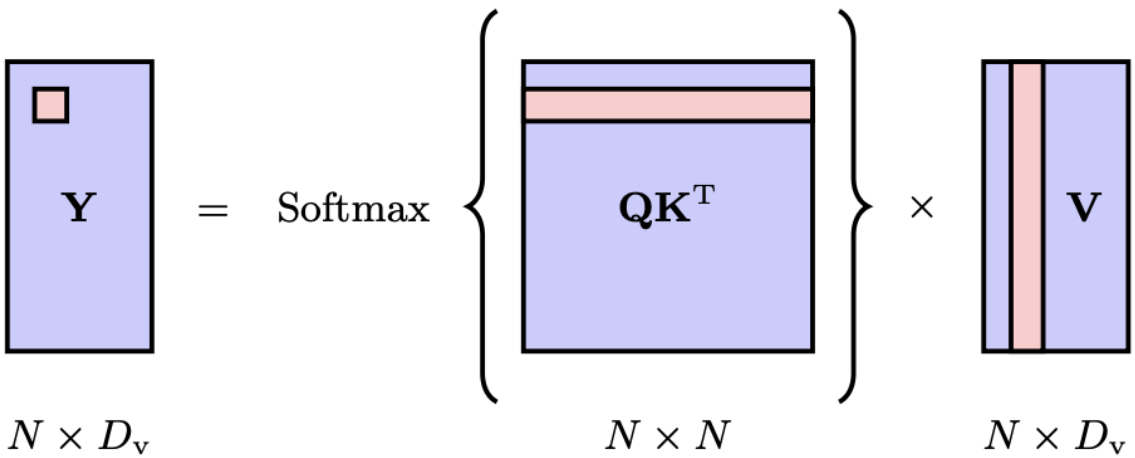


Figure: Illustration of the evaluation of the output from an attention layer given the query, key, and value matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively.

Solution

Vaswani et al. (2017) proposed defining separate query, key, and value matrices each having their own independent linear transformations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{(\mathrm{q})}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{(\mathrm{k})}, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{(\mathrm{v})},$$

where $\mathbf{W}^{(\mathrm{q})}$, $\mathbf{W}^{(\mathrm{k})} \in \mathbb{R}^{D \times D_{\mathrm{k}}}$ and $\mathbf{W}^{(\mathrm{v})} \in \mathbb{R}^{D \times D_{\mathrm{v}}}$, where D_{v} governs the dimensionality of the output. Therefore

$$\mathbf{Y} = \mathrm{Softmax}(\mathbf{Q}\mathbf{K}^{\mathrm{T}})\mathbf{V} \quad \leftarrow \quad \mathbf{Q}\mathbf{K}^{\mathrm{T}} \text{ are the attention weights.}$$

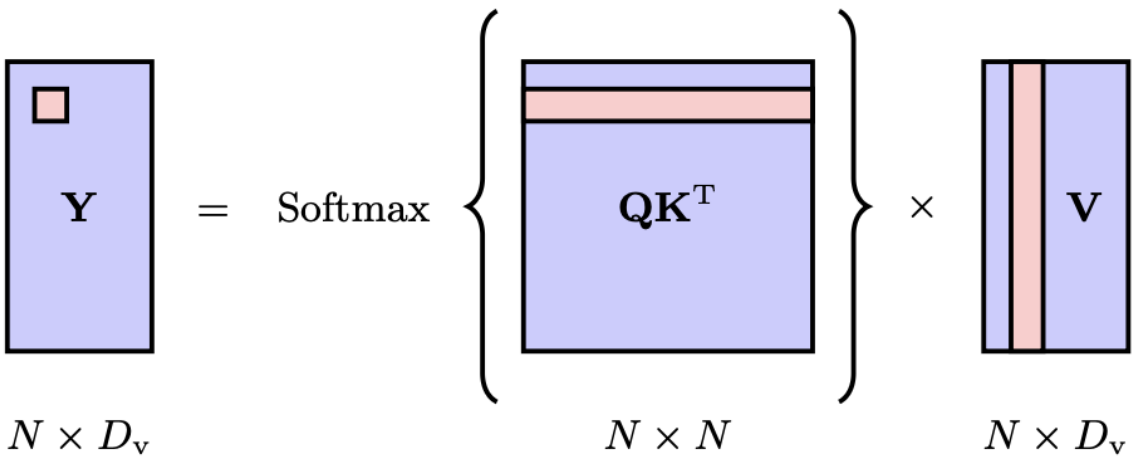


Figure: Illustration of the evaluation of the output from an attention layer given the query, key, and value matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively.

Final adjustment

To avoid vanishing gradients in softmax, scale the dot product of query and key vectors by $\sqrt{D_{\mathrm{k}}}$, yielding the output of the attention layer as:

$$\mathbf{Y} = \mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \equiv \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{D_{\mathrm{k}}}}\right)\mathbf{V}$$

Scaled dot-product self-attention

This structure constitutes a **single attention head**.

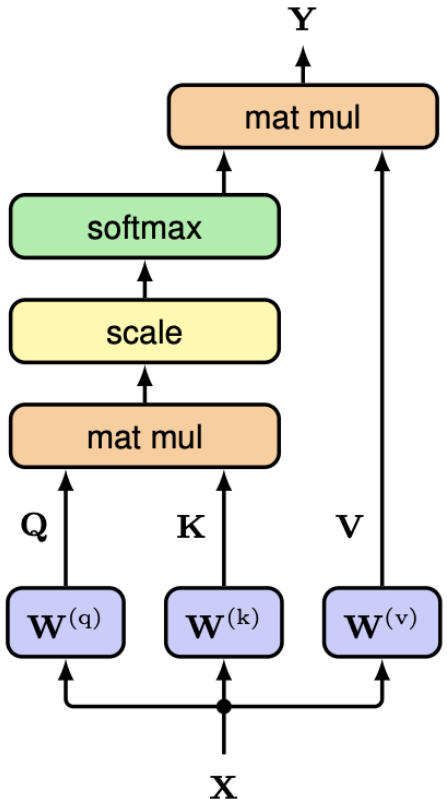


Figure: Information flow. Here 'mat mul' denotes matrix multiplication, and 'scale' refers to the normalization of the argument to the softmax using $\sqrt{D_{\mathrm{k}}}$.

Modeling multiple dependency patterns

To capture different types of relationships in the data simultaneously we replicate the attention mechanism into multiple heads.

Suppose we have H attention heads indexed by $h = 1, \ldots, H$ of the form

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h), \quad \mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^{(\text{q})}, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^{(\text{k})}, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^{(\text{v})}.$$

The heads are first concatenated into a single matrix, and the result is then linearly transformed using a matrix $\mathbf{W}^{(\circ)}$ to give a combined output in the form

$$\mathbf{Y}(\mathbf{X}) = \text{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_H)\mathbf{W}^{(\circ)}, \in \mathbb{R}^{N \times D_\text{v}}.$$

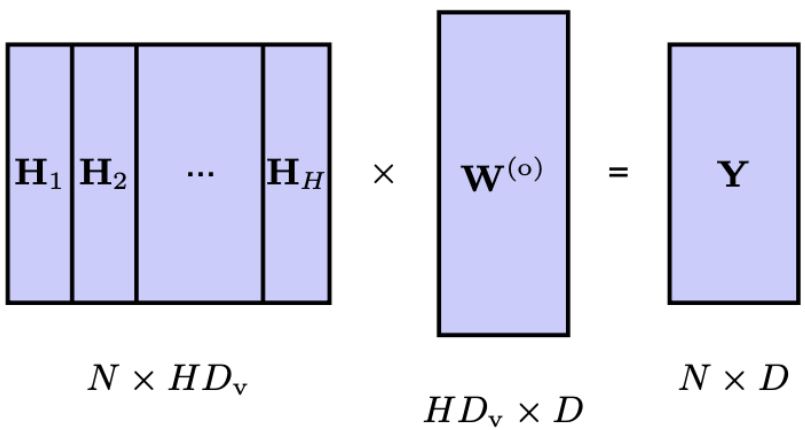


Figure: Network architecture for multi-head attention.

Modeling multiple dependency patterns

To capture different types of relationships in the data simultaneously we replicate the attention mechanism into multiple heads.

Suppose we have H attention heads indexed by $h = 1, \ldots, H$ of the form

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h), \quad \mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^{(\text{q})}, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^{(\text{k})}, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^{(\text{v})}.$$

The heads are first concatenated into a single matrix, and the result is then linearly transformed using a matrix $\mathbf{W}^{(\text{o})}$ to give a combined output in the form

$$\mathbf{Y}(\mathbf{X}) = \text{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_H)\mathbf{W}^{(\text{o})}, \in \mathbb{R}^{N \times D_\text{v}}.$$

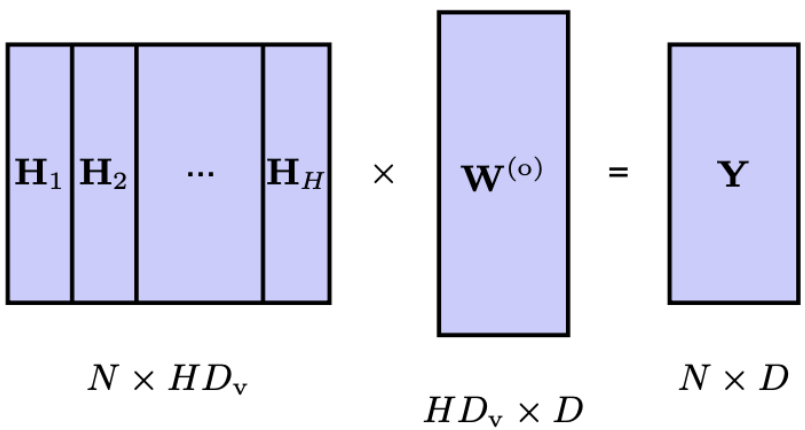


Figure: Network architecture for multi-head attention.

Modeling multiple dependency patterns

To capture different types of relationships in the data simultaneously we replicate the attention mechanism into multiple heads.

Suppose we have H attention heads indexed by $h = 1, \ldots, H$ of the form

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h), \quad \mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^{(\text{q})}, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^{(\text{k})}, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^{(\text{v})}.$$

The heads are first concatenated into a single matrix, and the result is then linearly transformed using a matrix $\mathbf{W}^{(\text{o})}$ to give a combined output in the form

$$\mathbf{Y}(\mathbf{X}) = \text{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_H)\mathbf{W}^{(\text{o})}, \in \mathbb{R}^{N \times D_\text{v}}.$$

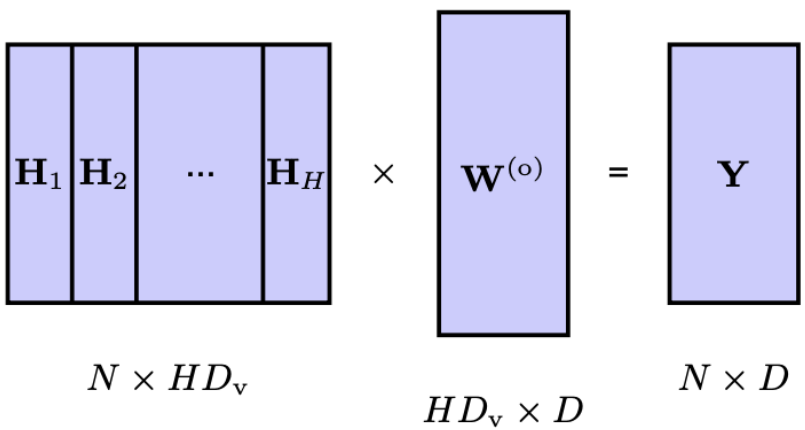


Figure: Network architecture for multi-head attention.

Modeling multiple dependency patterns

To capture different types of relationships in the data simultaneously we replicate the attention mechanism into multiple heads.

Suppose we have H attention heads indexed by $h = 1, \ldots, H$ of the form

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h), \quad \mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^{(\text{q})}, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^{(\text{k})}, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^{(\text{v})}.$$

The heads are first concatenated into a single matrix, and the result is then linearly transformed using a matrix $\mathbf{W}^{(\circ)}$ to give a combined output in the form

$$\mathbf{Y}(\mathbf{X}) = \text{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_H)\mathbf{W}^{(\circ)}, \in \mathbb{R}^{N \times D_{\text{v}}}.$$

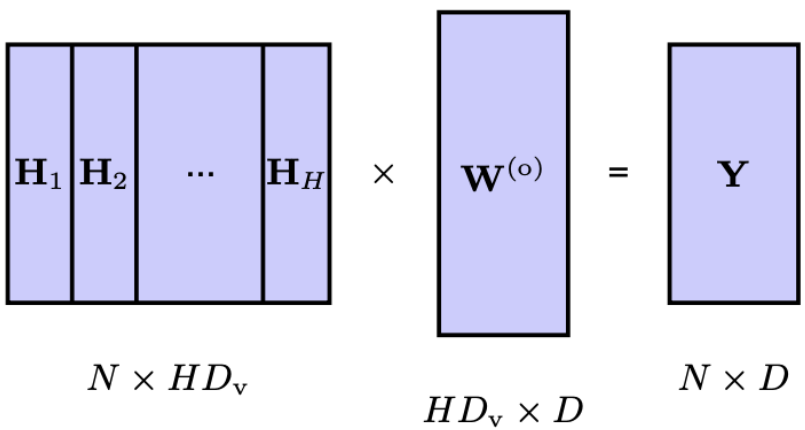


Figure: Network architecture for multi-head attention.

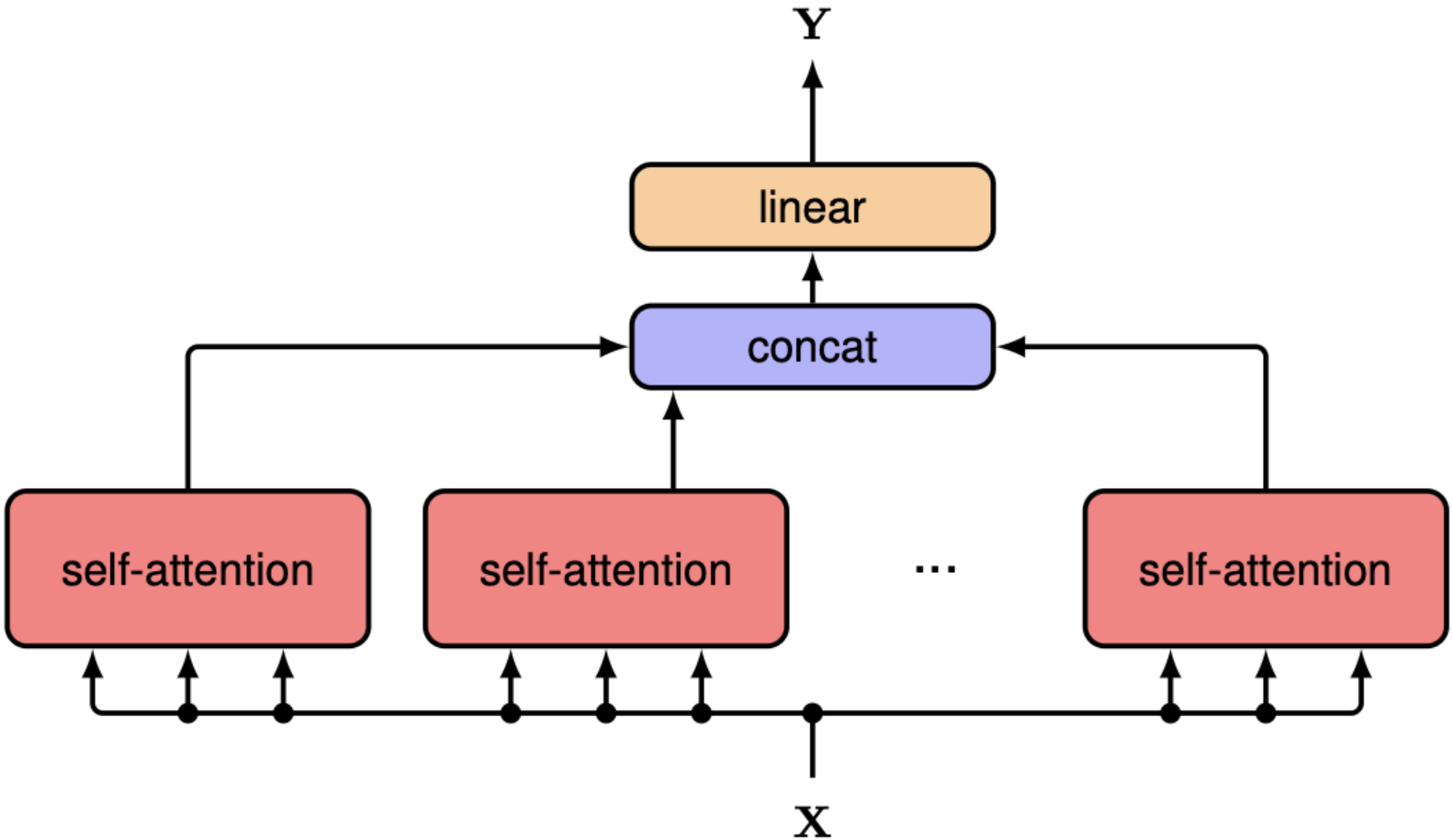


Figure: Information flow in a multi-head attention layer.

Literature Review

Training improvements

To improve training efficiency, the output is followed by a *residual connection* and a *layer normalization* (Ba, Kiros, and Hinton, 2016)

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y}(\mathbf{X}) + \mathbf{X}).$$

Still work to do

An MLP is added after the attention layer to break the linearity of the output and increase the expressive capabilities of the attention layer:

$$\widetilde{\mathbf{X}} = \text{LayerNorm}(\text{MLP}(\mathbf{Z}) + \mathbf{Z}).$$

Again, this neural network layer can be improved by using a *residual connection* and *layer normalization*. A short notation:

$$\boxed{\widetilde{\mathbf{X}} = \text{TF}_{\boldsymbol{\theta}}(\mathbf{X})}$$

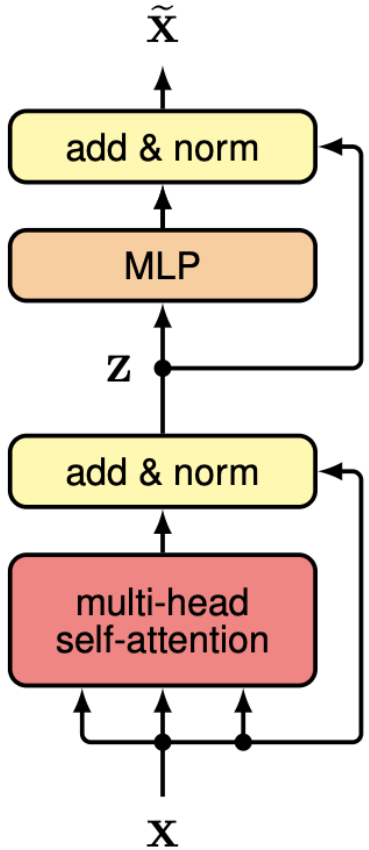


Figure: One layer of the transformer architecture. Here, 'add and norm' denotes a residual connection followed by layer normalization.

Training improvements

To improve training efficiency, the output is followed by a *residual connection* and a *layer normalization* (Ba, Kiros, and Hinton, 2016)

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y}(\mathbf{X}) + \mathbf{X}).$$

Still work to do

An MLP is added after the attention layer to break the linearity of the output and increase the expressive capabilities of the attention layer:

$$\widetilde{\mathbf{X}} = \text{LayerNorm}(\text{MLP}(\mathbf{Z}) + \mathbf{Z}).$$

Again, this neural network layer can be improved by using a *residual connection* and *layer normalization*. A short notation:

$$\boxed{\widetilde{\mathbf{X}} = \text{TF}_{\theta}(\mathbf{X})}$$

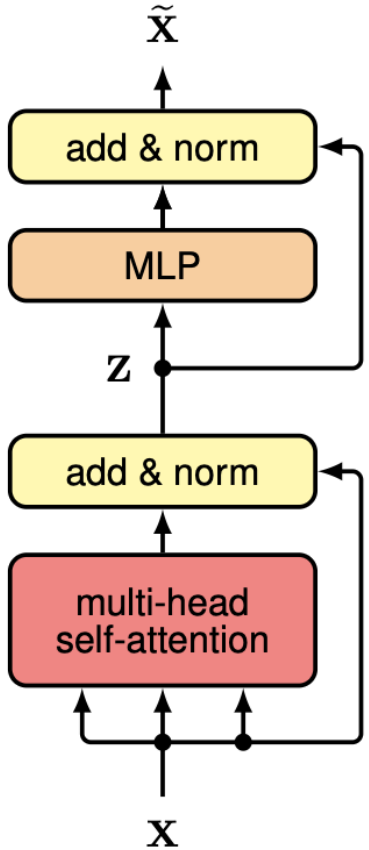


Figure: One layer of the transformer architecture. Here, 'add and norm' denotes a residual connection followed by layer normalization.

Training improvements

To improve training efficiency, the output is followed by a *residual connection* and a *layer normalization* (Ba, Kiros, and Hinton, 2016)

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y}(\mathbf{X}) + \mathbf{X}).$$

Still work to do

An MLP is added after the attention layer to break the linearity of the output and increase the expressive capabilities of the attention layer:

$$\widetilde{\mathbf{X}} = \text{LayerNorm}(\text{MLP}(\mathbf{Z}) + \mathbf{Z}).$$

Again, this neural network layer can be improved by using a *residual connection* and *layer normalization*. A short notation:

$$\boxed{\widetilde{\mathbf{X}} = \text{TF}_{\theta}(\mathbf{X})}$$

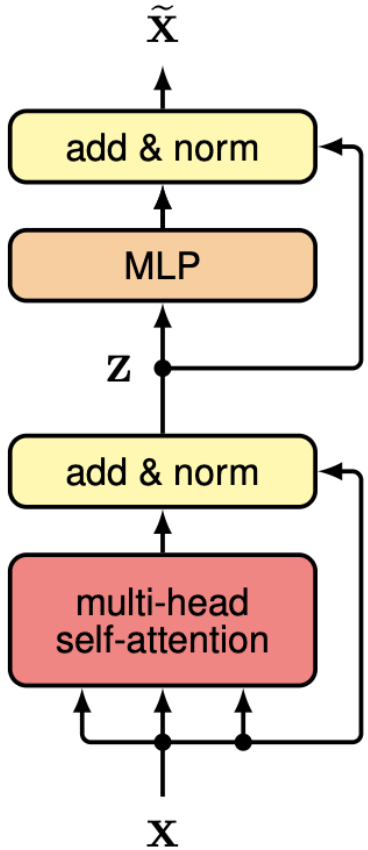


Figure: One layer of the transformer architecture. Here, 'add and norm' denotes a residual connection followed by layer normalization.

Training improvements

To improve training efficiency, the output is followed by a *residual connection* and a *layer normalization* (Ba, Kiros, and Hinton, 2016)

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y}(\mathbf{X}) + \mathbf{X}).$$

Still work to do

An MLP is added after the attention layer to break the linearity of the output and increase the expressive capabilities of the attention layer:

$$\widetilde{\mathbf{X}} = \text{LayerNorm}(\text{MLP}(\mathbf{Z}) + \mathbf{Z}).$$

Again, this neural network layer can be improved by using a *residual connection* and *layer normalization*. A short notation:

$$\widetilde{\mathbf{X}} = \text{TF}_\theta(\mathbf{X})$$

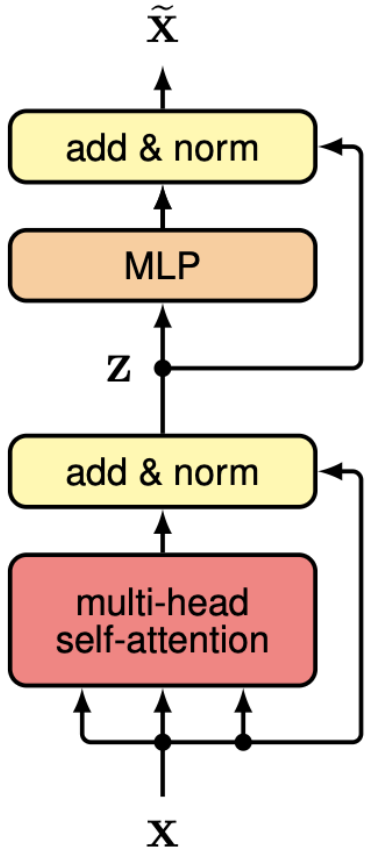


Figure: One layer of the transformer architecture. Here, 'add and norm' denotes a residual connection followed by layer normalization.

Training improvements

To improve training efficiency, the output is followed by a *residual connection* and a *layer normalization* (Ba, Kiros, and Hinton, 2016)

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y}(\mathbf{X}) + \mathbf{X}).$$

Still work to do

An MLP is added after the attention layer to break the linearity of the output and increase the expressive capabilities of the attention layer:

$$\widetilde{\mathbf{X}} = \text{LayerNorm}(\text{MLP}(\mathbf{Z}) + \mathbf{Z}).$$

Again, this neural network layer can be improved by using a *residual connection* and *layer normalization*. A short notation:

$$\boxed{\widetilde{\mathbf{X}} = \text{TF}_{\boldsymbol{\theta}}(\mathbf{X})}$$

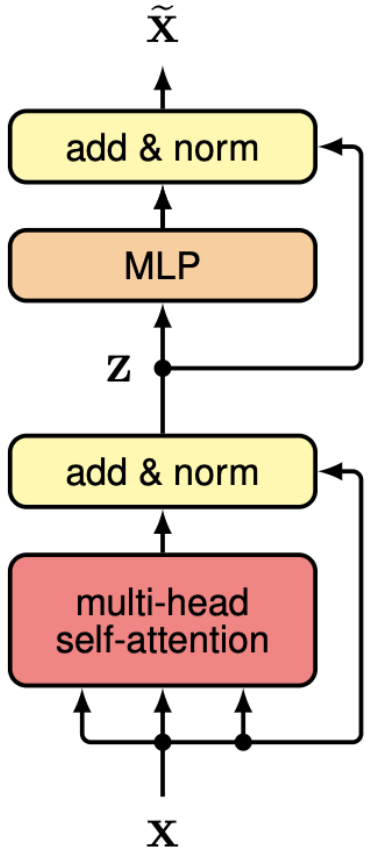


Figure: One layer of the transformer architecture. Here, 'add and norm' denotes a residual connection followed by layer normalization.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathrm{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

   such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.
4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathrm{P}_{\mathcal{X}} \\ f \sim \mathrm{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

   where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathrm{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

   such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.
4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathrm{P}_{\mathcal{X}} \\ f \sim \mathrm{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

   where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathrm{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

   such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.
4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathrm{P}_{\mathcal{X}} \\ f \sim \mathrm{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

   where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathsf{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathsf{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.

4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathsf{P}_{\mathcal{X}} \\ f \sim \mathsf{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathsf{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathsf{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.

4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathsf{P}_{\mathcal{X}} \\ f \sim \mathsf{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathsf{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathsf{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

   such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.
4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathsf{P}_{\mathcal{X}} \\ f \sim \mathsf{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

   where ℓ is an appropriate loss function.

Transformers in Statistics

Garg et al. (2022) show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions.

Notation

Let $\mathcal{X}$ be the input feature space, and $\mathcal{Y}$ be the output/label space, such that $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is class of functions between this spaces.

In-context learning setting

1) Sample $f \sim \mathsf{P}_{\mathcal{F}}$ and N i.i.d. inputs tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathsf{P}_{\mathcal{X}}$.
2) Generate labels $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathcal{Y}$ for $i = 1, \ldots, N$, obtaining $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$.
3) Define a length-i prompt as

$$\mathbf{X}_{\text{prompt}}^{(i)} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i)^{\mathrm{T}},$$

   such that the model prediction is $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}_{\text{prompt}}^{(i)}) \in \mathcal{Y}$, $i = 1, \ldots, N$.
4) Training by minimizing the expected loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathsf{P}_{\mathcal{X}} \\ f \sim \mathsf{P}_{\mathcal{F}}}} \left[\frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \right],$$

   where ℓ is an appropriate loss function.

Example (Linear functions)

Consider the class of linear functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x},\ \mathbf{w} \in \mathbb{R}^d\}.$$

Sample

$$\mathbf{w} \sim \mathrm{P}_{\mathcal{F}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{x}_i \overset{\text{i.i.d.}}{\sim} \mathrm{P}_{\mathcal{X}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad i = 1, \ldots, N,$$

independently. For each f determined by $\mathbf{w}$, generate training pairs

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i.$$

Then, train the Transformer to predict $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}^{(i)}_{\text{prompt}})$ where

$$\mathbf{X}^{(i)}_{\text{prompt}} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i),$$

by minimizing the expected mean squared error loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d) \\ f \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d)}} \left[\frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \right].$$

Example (Linear functions)

Consider the class of linear functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x},\ \mathbf{w} \in \mathbb{R}^d\}.$$

Sample

$$\mathbf{w} \sim \mathrm{P}_{\mathcal{F}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{x}_i \overset{\text{i.i.d.}}{\sim} \mathrm{P}_{\mathcal{X}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad i = 1, \ldots, N,$$

independently. For each f determined by $\mathbf{w}$, generate training pairs

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i.$$

Then, train the Transformer to predict $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}^{(i)}_{\text{prompt}})$ where

$$\mathbf{X}^{(i)}_{\text{prompt}} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i),$$

by minimizing the expected mean squared error loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d) \\ f \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d)}} \left[\frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \right].$$

Example (Linear functions)

Consider the class of linear functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x},\ \mathbf{w} \in \mathbb{R}^d\}.$$

Sample

$$\mathbf{w} \sim \mathrm{P}_{\mathcal{F}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{x}_i \overset{\text{i.i.d.}}{\sim} \mathrm{P}_{\mathcal{X}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad i = 1, \ldots, N,$$

independently. For each f determined by $\mathbf{w}$, generate training pairs

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i.$$

Then, train the Transformer to predict $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}^{(i)}_{\text{prompt}})$ where

$$\mathbf{X}^{(i)}_{\text{prompt}} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i),$$

by minimizing the expected mean squared error loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d) \\ f \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d)}} \left[\frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \right].$$

Example (Linear functions)

Consider the class of linear functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x},\ \mathbf{w} \in \mathbb{R}^d\}.$$

Sample

$$\mathbf{w} \sim \mathrm{P}_{\mathcal{F}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{x}_i \overset{\text{i.i.d.}}{\sim} \mathrm{P}_{\mathcal{X}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad i = 1, \ldots, N,$$

independently. For each f determined by $\mathbf{w}$, generate training pairs

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i.$$

Then, train the Transformer to predict $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}^{(i)}_{\text{prompt}})$ where

$$\mathbf{X}^{(i)}_{\text{prompt}} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i),$$

by minimizing the expected mean squared error loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d) \\ f \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d)}} \left[\frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \right].$$

Example (Linear functions)

Consider the class of linear functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x},\ \mathbf{w} \in \mathbb{R}^d\}.$$

Sample

$$\mathbf{w} \sim \mathrm{P}_{\mathcal{F}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{x}_i \overset{\text{i.i.d.}}{\sim} \mathrm{P}_{\mathcal{X}} := \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d), \quad i = 1, \ldots, N,$$

independently. For each f determined by $\mathbf{w}$, generate training pairs

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i.$$

Then, train the Transformer to predict $\hat{\mathbf{y}}_i = \mathbf{W}\,\mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{X}^{(i)}_{\text{prompt}})$ where

$$\mathbf{X}^{(i)}_{\text{prompt}} = (\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{x}_i),$$

by minimizing the expected mean squared error loss

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_1, \ldots, \mathbf{x}_N \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d) \\ f \sim \mathrm{N}_d(\mathbf{0}, \mathbf{I}_d)}} \left[\frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \right].$$

Transformers in Statistics

In this setting, the Transformer learns to approximate the least squares estimator.

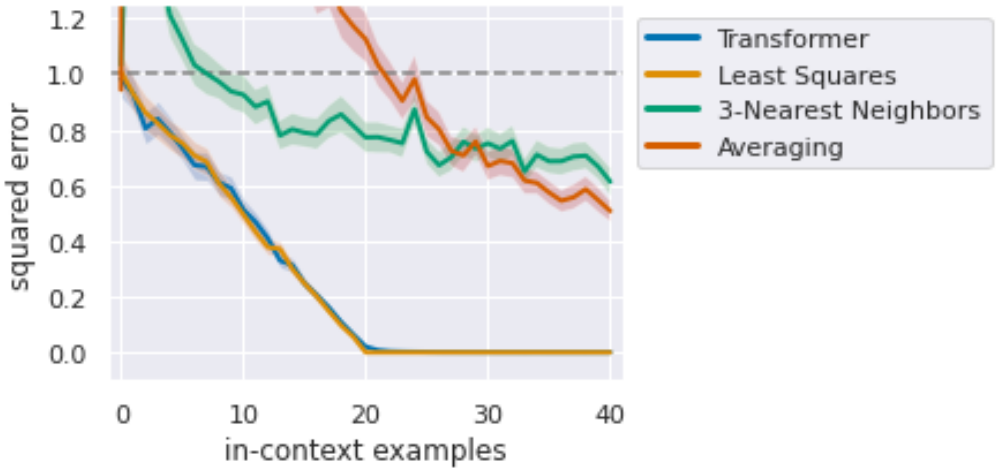


Figure: Normalized squared error of the Transformer as a function of the number of in-context examples (Garg et al., 2022).

Bai et al. (2023) show[1] that Transformers can implement a broad class of standard ML algorithms in context, such as least squares, Ridge regression, Lasso, convex risk minimization for GLMs.

[1] Possibly the most mathematically formal paper on this topic.

Transformers in Statistics

In this setting, the Transformer learns to approximate the least squares estimator.

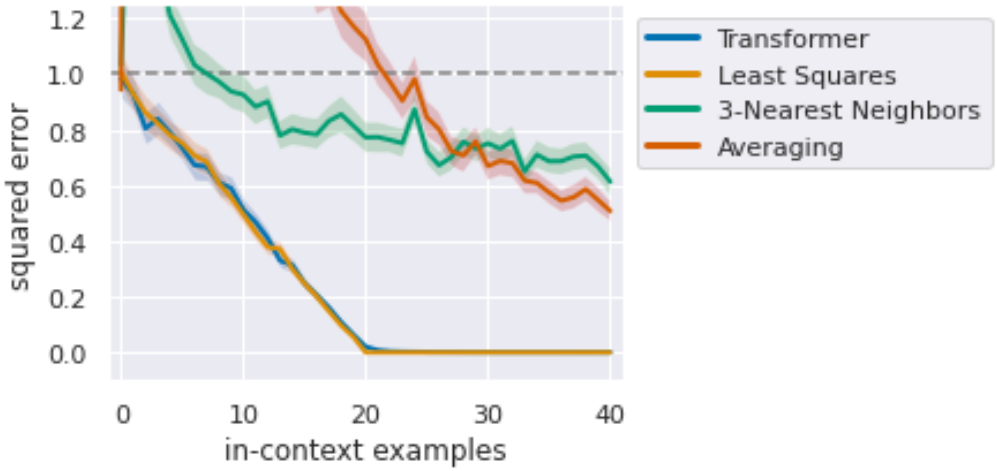


Figure: Normalized squared error of the Transformer as a function of the number of in-context examples (Garg et al., 2022).

Bai et al. (2023) show[1] that Transformers can implement a broad class of standard ML algorithms in context, such as least squares, Ridge regression, Lasso, convex risk minimization for GLMs.

[1] Possibly the most mathematically formal paper on this topic.

Large Language Models

Decoder transformers

These can be used as generative models that create output sequences of tokens. we will focus on a class of models called GPT which stands for generative pre-trained transformer.

Goal

Construct an autoregressive model of the form defined by

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1}), \quad \mathbf{x}_{1:n-1} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}),$$

where conditional distributions $p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1})$ are expressed using a Transformer neural network that is learned from data.

Idea

$$\mathbf{x}_1, \ldots, \mathbf{x}_N \quad \overset{\text{Transformer layers}}{\Rightarrow} \quad \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_N,$$

Each output needs to represent a probability distribution over the class of tokens:

$$\mathbf{Y} = \mathrm{Softmax}(\widetilde{\mathbf{X}}\mathbf{W}^{(\mathrm{p})}), \quad \mathbf{W}^{(\mathrm{p})} \in \mathbb{R}^{D \times K}.$$

Large Language Models

Decoder transformers

These can be used as generative models that create output sequences of tokens. we will focus on a class of models called GPT which stands for generative pre-trained transformer.

Goal

Construct an autoregressive model of the form defined by

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1}), \quad \mathbf{x}_{1:n-1} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}),$$

where conditional distributions $p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1})$ are expressed using a Transformer neural network that is learned from data.

Idea

$$\mathbf{x}_1, \ldots, \mathbf{x}_N \quad \overset{\text{Transformer layers}}{\Rightarrow} \quad \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_N,$$

Each output needs to represent a probability distribution over the class of tokens:

$$\mathbf{Y} = \text{Softmax}(\widetilde{\mathbf{X}}\mathbf{W}^{(\text{p})}), \quad \mathbf{W}^{(\text{p})} \in \mathbb{R}^{D \times K}.$$

Large Language Models

Decoder transformers

These can be used as generative models that create output sequences of tokens. we will focus on a class of models called GPT which stands for generative pre-trained transformer.

Goal

Construct an autoregressive model of the form defined by

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1}), \quad \mathbf{x}_{1:n-1} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}),$$

where conditional distributions $p(\mathbf{x}_n \mid \mathbf{x}_{1:n-1})$ are expressed using a Transformer neural network that is learned from data.

Idea

$$\mathbf{x}_1, \ldots, \mathbf{x}_N \quad \overset{\text{Transformer layers}}{\Rightarrow} \quad \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_N,$$

Each output needs to represent a probability distribution over the class of tokens:

$$\mathbf{Y} = \mathrm{Softmax}(\widetilde{\mathbf{X}}\mathbf{W}^{(\mathrm{p})}), \quad \mathbf{W}^{(\mathrm{p})} \in \mathbb{R}^{D \times K}.$$

GPT Architecture

Embedding and positional encoding

We begin by defining a dictionary

$$\mathcal{V} = \{w_1, w_2, \ldots, w_K\},$$

which is a fixed, finite set of words indexed from 1 to K. A word $\mathbf{v}_i$ is mapped to an index in the dictionary using a tokenization function $\tau : \mathcal{V} \to \{1, \ldots, K\}$, such that

$$\tau(\mathbf{v}_i) = k_i \quad \text{with} \quad \mathbf{v}_i = w_{k_i}.$$

This index is then used to construct a **one-hot encoded vector** $\mathbf{o}_i \in \{0, 1\}^K$, where the k_i-th component is set to 1 and all others are 0.

This one-hot vector is then projected into a dense, learnable embedding space using an embedding matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, and positional information (Vaswani et al. 2017) is added

$$\mathbf{x}_i = \mathbf{E}\mathbf{o}_i + \mathbf{r}_i.$$

GPT Architecture

Embedding and positional encoding

We begin by defining a dictionary

$$\mathcal{V} = \{w_1, w_2, \ldots, w_K\},$$

which is a fixed, finite set of words indexed from 1 to K. A word $\mathbf{v}_i$ is mapped to an index in the dictionary using a tokenization function $\tau : \mathcal{V} \to \{1, \ldots, K\}$, such that

$$\tau(\mathbf{v}_i) = k_i \quad \text{with} \quad \mathbf{v}_i = w_{k_i}.$$

This index is then used to construct a **one-hot encoded vector** $\mathbf{o}_i \in \{0, 1\}^K$, where the k_i-th component is set to 1 and all others are 0.

This one-hot vector is then projected into a dense, learnable embedding space using an embedding matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, and positional information (Vaswani et al. 2017) is added

$$\mathbf{x}_i = \mathbf{E}\mathbf{o}_i + \mathbf{r}_i.$$

GPT Architecture

Embedding and positional encoding

We begin by defining a dictionary

$$\mathcal{V} = \{w_1, w_2, \ldots, w_K\},$$

which is a fixed, finite set of words indexed from 1 to K. A word $\mathbf{v}_i$ is mapped to an index in the dictionary using a tokenization function $\tau : \mathcal{V} \to \{1, \ldots, K\}$, such that

$$\tau(\mathbf{v}_i) = k_i \quad \text{with} \quad \mathbf{v}_i = w_{k_i}.$$

This index is then used to construct a **one-hot encoded vector** $\mathbf{o}_i \in \{0, 1\}^K$, where the k_i-th component is set to 1 and all others are 0.

This one-hot vector is then projected into a dense, learnable embedding space using an embedding matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, and positional information (Vaswani et al. 2017) is added

$$\mathbf{x}_i = \mathbf{E}\mathbf{o}_i + \mathbf{r}_i.$$

GPT Architecture

Embedding and positional encoding

We begin by defining a dictionary

$$\mathcal{V} = \{w_1, w_2, \ldots, w_K\},$$

which is a fixed, finite set of words indexed from 1 to K. A word $\mathbf{v}_i$ is mapped to an index in the dictionary using a tokenization function $\tau : \mathcal{V} \to \{1, \ldots, K\}$, such that

$$\tau(\mathbf{v}_i) = k_i \quad \text{with} \quad \mathbf{v}_i = w_{k_i}.$$

This index is then used to construct a **one-hot encoded vector** $\mathbf{o}_i \in \{0, 1\}^K$, where the k_i-th component is set to 1 and all others are 0.

This one-hot vector is then projected into a dense, learnable embedding space using an embedding matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, and positional information (Vaswani et al. 2017) is added

$$\mathbf{x}_i = \mathbf{E}\mathbf{o}_i + \mathbf{r}_i.$$

GPT Architecture

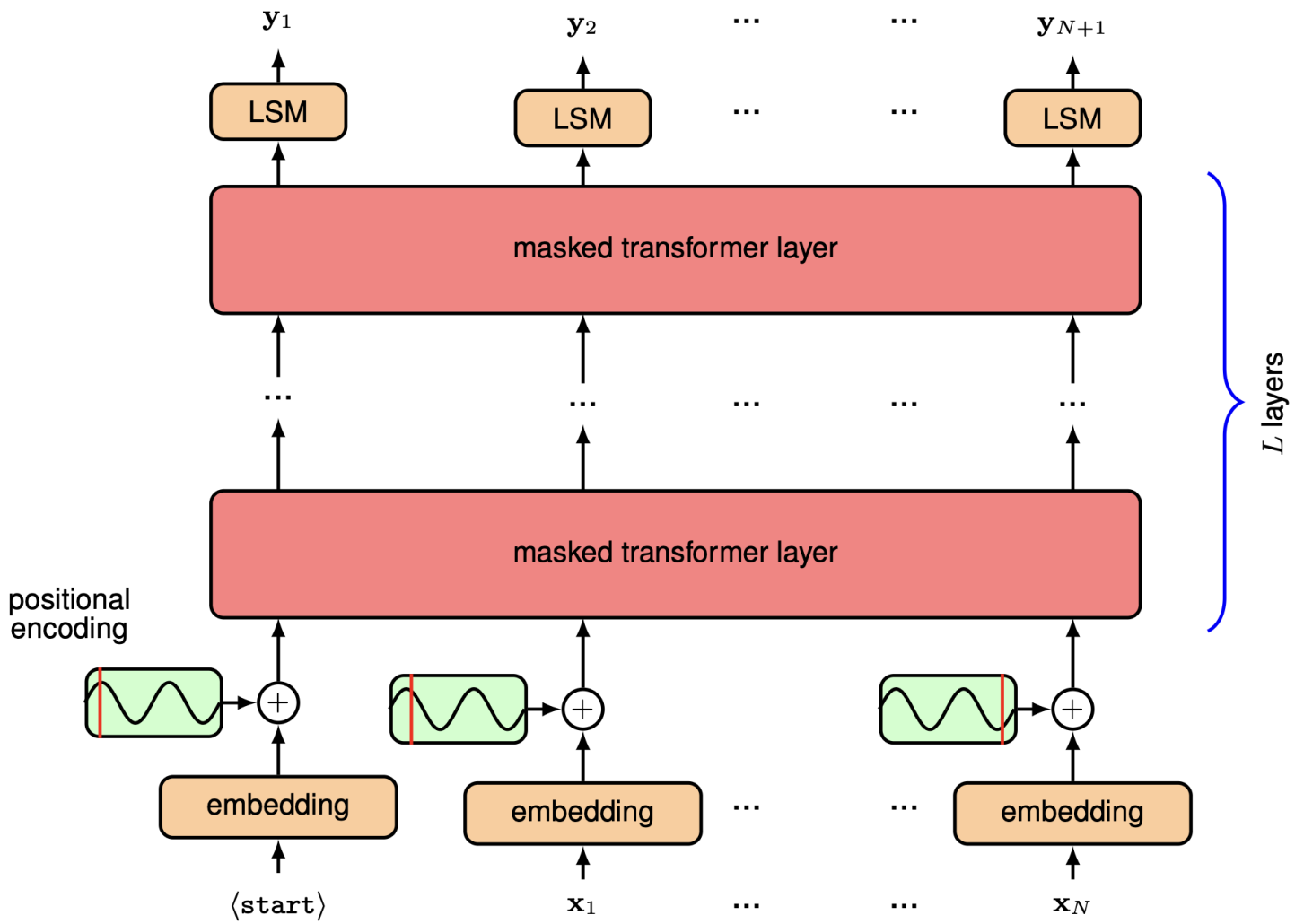


Figure: Architecture of a GPT decoder transformer network. Here 'LSM' stands for linear-softmax and denotes a linear transformation whose learnable parameters are shared across the token positions, followed by a softmax activation function.

Gen AI for Chained Extremes

How do we apply GPT to generate sequences of extreme events?[2]

- Let $(X_1, \ldots, X_d) \in \mathcal{X}^d$ be a chain of extreme events, e.g. $\mathcal{X} = \mathbb{R}_+$.
- Define a finite partition $\{\mathcal{C}_k\}_{k=1}^K$ of $\mathcal{X}$, and a one-hot tokenization function $\tau : \mathcal{X} \to \{0,1\}^K$ defined as

$$\tau(X_i) = (I(X_i \in \mathcal{C}_1), \ldots, I(X_i \in \mathcal{C}_K)), \quad i = 1, \ldots, d.$$

- Each token is then projected into an embedding space using a learnable matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, positional encoding $\mathbf{r}_i$ and extremal encoding[3] are added

$$\mathbf{s}_i = \mathbf{E}\tau(X_i) + \mathbf{r}_i + \mathbf{e}_i.$$

- Instead of modeling the joint density of $(X_1, \ldots, X_d)$, we model the probability that each X_i falls in a set $\mathcal{C}_k$, using:

$$P(X_1 \in \mathcal{C}_{k_1}, \ldots, X_d \in \mathcal{C}_{k_d}) = \prod_{i=1}^{d} P(X_i \in \mathcal{C}_{k_i} \mid X_{1:i-1}).$$

[2] Ongoing work with **Miguel de Carvalho** and **Ronny Vallejos**.

[3] e.g. based on extreme quantile regression models.

Gen AI for Chained Extremes

How do we apply GPT to generate sequences of extreme events?[2]

- Let $(X_1, \ldots, X_d) \in \mathcal{X}^d$ be a chain of extreme events, e.g. $\mathcal{X} = \mathbb{R}_+$.
- Define a finite partition $\{\mathcal{C}_k\}_{k=1}^K$ of $\mathcal{X}$, and a one-hot tokenization function $\tau : \mathcal{X} \to \{0,1\}^K$ defined as

$$\tau(X_i) = (I(X_i \in \mathcal{C}_1), \ldots, I(X_i \in \mathcal{C}_K)), \quad i = 1, \ldots, d.$$

- Each token is then projected into an embedding space using a learnable matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, positional encoding $\mathbf{r}_i$ and extremal encoding[3] are added

$$\mathbf{s}_i = \mathbf{E}\tau(X_i) + \mathbf{r}_i + \mathbf{e}_i.$$

- Instead of modeling the joint density of $(X_1, \ldots, X_d)$, we model the probability that each X_i falls in a set $\mathcal{C}_k$, using:

$$P(X_1 \in \mathcal{C}_{k_1}, \ldots, X_d \in \mathcal{C}_{k_d}) = \prod_{i=1}^{d} P(X_i \in \mathcal{C}_{k_i} \mid X_{1:i-1}).$$

[2] Ongoing work with **Miguel de Carvalho** and **Ronny Vallejos**.

[3] e.g. based on extreme quantile regression models.

Gen AI for Chained Extremes

How do we apply GPT to generate sequences of extreme events?[2]

- Let $(X_1, \ldots, X_d) \in \mathcal{X}^d$ be a chain of extreme events, e.g. $\mathcal{X} = \mathbb{R}_+$.
- Define a finite partition $\{\mathcal{C}_k\}_{k=1}^K$ of $\mathcal{X}$, and a one-hot tokenization function $\tau : \mathcal{X} \to \{0,1\}^K$ defined as

$$\tau(X_i) = (I(X_i \in \mathcal{C}_1), \ldots, I(X_i \in \mathcal{C}_K)), \quad i = 1, \ldots, d.$$

- Each token is then projected into an embedding space using a learnable matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, positional encoding $\mathbf{r}_i$ and extremal encoding[3] are added

$$\mathbf{s}_i = \mathbf{E}\tau(X_i) + \mathbf{r}_i + \mathbf{e}_i.$$

- Instead of modeling the joint density of $(X_1, \ldots, X_d)$, we model the probability that each X_i falls in a set $\mathcal{C}_k$, using:

$$P(X_1 \in \mathcal{C}_{k_1}, \ldots, X_d \in \mathcal{C}_{k_d}) = \prod_{i=1}^{d} P(X_i \in \mathcal{C}_{k_i} \mid X_{1:i-1}).$$

[2] Ongoing work with **Miguel de Carvalho** and **Ronny Vallejos**.

[3] e.g. based on extreme quantile regression models.

Gen AI for Chained Extremes

How do we apply GPT to generate sequences of extreme events?[2]

- Let $(X_1, \ldots, X_d) \in \mathcal{X}^d$ be a chain of extreme events, e.g. $\mathcal{X} = \mathbb{R}_+$.
- Define a finite partition $\{\mathcal{C}_k\}_{k=1}^K$ of $\mathcal{X}$, and a one-hot tokenization function $\tau : \mathcal{X} \to \{0,1\}^K$ defined as

$$\tau(X_i) = (I(X_i \in \mathcal{C}_1), \ldots, I(X_i \in \mathcal{C}_K)), \quad i = 1, \ldots, d.$$

- Each token is then projected into an embedding space using a learnable matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, positional encoding $\mathbf{r}_i$ and extremal encoding[3] are added

$$\mathbf{s}_i = \mathbf{E}\tau(X_i) + \mathbf{r}_i + \mathbf{e}_i.$$

- Instead of modeling the joint density of $(X_1, \ldots, X_d)$, we model the probability that each X_i falls in a set $\mathcal{C}_k$, using:

$$P(X_1 \in \mathcal{C}_{k_1}, \ldots, X_d \in \mathcal{C}_{k_d}) = \prod_{i=1}^{d} P(X_i \in \mathcal{C}_{k_i} \mid X_{1:i-1}).$$

[2] Ongoing work with **Miguel de Carvalho** and **Ronny Vallejos**.

[3] e.g. based on extreme quantile regression models.

Gen AI for Chained Extremes

How do we apply GPT to generate sequences of extreme events?[2]

- Let $(X_1, \ldots, X_d) \in \mathcal{X}^d$ be a chain of extreme events, e.g. $\mathcal{X} = \mathbb{R}_+$.
- Define a finite partition $\{\mathcal{C}_k\}_{k=1}^K$ of $\mathcal{X}$, and a one-hot tokenization function $\tau : \mathcal{X} \to \{0,1\}^K$ defined as

$$\tau(X_i) = (I(X_i \in \mathcal{C}_1), \ldots, I(X_i \in \mathcal{C}_K)), \quad i = 1, \ldots, d.$$

- Each token is then projected into an embedding space using a learnable matrix $\mathbf{E} \in \mathbb{R}^{D \times K}$, positional encoding $\mathbf{r}_i$ and extremal encoding[3] are added

$$\mathbf{s}_i = \mathbf{E}\tau(X_i) + \mathbf{r}_i + \mathbf{e}_i.$$

- Instead of modeling the joint density of $(X_1, \ldots, X_d)$, we model the probability that each X_i falls in a set $\mathcal{C}_k$, using:

$$P(X_1 \in \mathcal{C}_{k_1}, \ldots, X_d \in \mathcal{C}_{k_d}) = \prod_{i=1}^{d} P(X_i \in \mathcal{C}_{k_i} \mid X_{1:i-1}).$$

[2] Ongoing work with **Miguel de Carvalho** and **Ronny Vallejos**.

[3] e.g. based on extreme quantile regression models.

How do we apply GPT to generate sequences of extreme events?

- Each conditional $P(X_i \in \mathcal{C}_k \mid X_{i-L:i-1})$ is approximated by a **decoder Transformer**[4] $\mathrm{TF}_{\boldsymbol{\theta}}$, which maps the input embeddings $(\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})$:

$$\widetilde{\mathbf{S}}_i = \mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{S}_i), \quad \text{with } \mathbf{S}_i = (\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})^{\mathrm{T}},$$

- This representation is then passed through a learnable linear projection $\mathbf{W} \in \mathbb{R}^{K \times D}$ to obtain the logits $\mathbf{z}_i = (z_{i1}, \ldots, z_{iK}) \in \mathbb{R}^K$:

$$\mathbf{z}_i = \mathbf{W}\widetilde{\mathbf{S}}_i.$$

- Finally, the softmax layer converts the logits into probabilities:

$$P_{\boldsymbol{\theta}}(X_i \in \mathcal{C}_k \mid X_{i-L:i-1}) = \frac{\exp(z_{ik})}{\sum_{j=1}^{K} \exp(z_{ij})}, \quad k = 1, \ldots, K.$$

- The model is trained by minimizing the empirical negative log-likelihood over a **large** dataset $\mathcal{D} = \{(X_1^{(t)}, \ldots, X_d^{(t)})\}_{t=1}^{N}$:

$$\hat{\ell}(\boldsymbol{\theta}) = -\frac{1}{Nd} \sum_{t=1}^{N} \sum_{i=1}^{d} \log P_{\boldsymbol{\theta}}(X_i^{(t)} \in \mathcal{C}_{k_i^{(t)}} \mid X_{i-L:i-1}^{(t)}).$$

[4] Just like ChatGPT!

How do we apply GPT to generate sequences of extreme events?

- Each conditional $P(X_i \in \mathcal{C}_k \mid X_{i-L:i-1})$ is approximated by a **decoder Transformer**[4] $\mathrm{TF}_{\boldsymbol{\theta}}$, which maps the input embeddings $(\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})$:

$$\widetilde{\mathbf{S}}_i = \mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{S}_i), \quad \text{with } \mathbf{S}_i = (\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})^{\mathrm{T}},$$

- This representation is then passed through a learnable linear projection $\mathbf{W} \in \mathbb{R}^{K \times D}$ to obtain the logits $\mathbf{z}_i = (z_{i1}, \ldots, z_{iK}) \in \mathbb{R}^K$:

$$\mathbf{z}_i = \mathbf{W}\widetilde{\mathbf{S}}_i.$$

- Finally, the softmax layer converts the logits into probabilities:

$$P_{\boldsymbol{\theta}}(X_i \in \mathcal{C}_k \mid X_{i-L:i-1}) = \frac{\exp(z_{ik})}{\sum_{j=1}^{K} \exp(z_{ij})}, \quad k = 1, \ldots, K.$$

- The model is trained by minimizing the empirical negative log-likelihood over a **large** dataset $\mathcal{D} = \{(X_1^{(t)}, \ldots, X_d^{(t)})\}_{t=1}^{N}$:

$$\hat{\ell}(\boldsymbol{\theta}) = -\frac{1}{Nd} \sum_{t=1}^{N} \sum_{i=1}^{d} \log P_{\boldsymbol{\theta}}(X_i^{(t)} \in \mathcal{C}_{k_i^{(t)}} \mid X_{i-L:i-1}^{(t)}).$$

[4] Just like ChatGPT!

How do we apply GPT to generate sequences of extreme events?

- Each conditional $P(X_i \in \mathcal{C}_k \mid X_{i-L:i-1})$ is approximated by a **decoder Transformer**[4] $\mathrm{TF}_{\boldsymbol{\theta}}$, which maps the input embeddings $(\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})$:

$$\widetilde{\mathbf{S}}_i = \mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{S}_i), \quad \text{with } \mathbf{S}_i = (\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})^{\mathrm{T}},$$

- This representation is then passed through a learnable linear projection $\mathbf{W} \in \mathbb{R}^{K \times D}$ to obtain the logits $\mathbf{z}_i = (z_{i1}, \ldots, z_{iK}) \in \mathbb{R}^K$:

$$\mathbf{z}_i = \mathbf{W}\widetilde{\mathbf{S}}_i.$$

- Finally, the softmax layer converts the logits into probabilities:

$$P_{\boldsymbol{\theta}}(X_i \in \mathcal{C}_k \mid X_{i-L:i-1}) = \frac{\exp(z_{ik})}{\sum_{j=1}^{K} \exp(z_{ij})}, \quad k = 1, \ldots, K.$$

- The model is trained by minimizing the empirical negative log-likelihood over a **large** dataset $\mathcal{D} = \{(X_1^{(t)}, \ldots, X_d^{(t)})\}_{t=1}^{N}$:

$$\hat{\ell}(\boldsymbol{\theta}) = -\frac{1}{Nd} \sum_{t=1}^{N} \sum_{i=1}^{d} \log P_{\boldsymbol{\theta}}(X_i^{(t)} \in \mathcal{C}_{k_i^{(t)}} \mid X_{i-L:i-1}^{(t)}).$$

[4] Just like ChatGPT!

How do we apply GPT to generate sequences of extreme events?

- Each conditional $P(X_i \in \mathcal{C}_k \mid X_{i-L:i-1})$ is approximated by a **decoder Transformer**[4] $\mathrm{TF}_{\boldsymbol{\theta}}$, which maps the input embeddings $(\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})$:

$$\widetilde{\mathbf{S}}_i = \mathrm{TF}_{\boldsymbol{\theta}}(\mathbf{S}_i), \quad \text{with } \mathbf{S}_i = (\mathbf{s}_{i-L}, \ldots, \mathbf{s}_{i-1})^{\mathrm{T}},$$

- This representation is then passed through a learnable linear projection $\mathbf{W} \in \mathbb{R}^{K \times D}$ to obtain the logits $\mathbf{z}_i = (z_{i1}, \ldots, z_{iK}) \in \mathbb{R}^K$:

$$\mathbf{z}_i = \mathbf{W}\widetilde{\mathbf{S}}_i.$$

- Finally, the softmax layer converts the logits into probabilities:

$$P_{\boldsymbol{\theta}}(X_i \in \mathcal{C}_k \mid X_{i-L:i-1}) = \frac{\exp(z_{ik})}{\sum_{j=1}^{K} \exp(z_{ij})}, \quad k = 1, \ldots, K.$$

- The model is trained by minimizing the empirical negative log-likelihood over a **large** dataset $\mathcal{D} = \{(X_1^{(t)}, \ldots, X_d^{(t)})\}_{t=1}^{N}$:

$$\hat{\ell}(\boldsymbol{\theta}) = -\frac{1}{Nd} \sum_{t=1}^{N} \sum_{i=1}^{d} \log P_{\boldsymbol{\theta}}(X_i^{(t)} \in \mathcal{C}_{k_i^{(t)}} \mid X_{i-L:i-1}^{(t)}).$$

[4] Just like ChatGPT!

References

1) Vaswani et al. (2017). *Attention is All You Need*. arXiv:1706.03762.
2) Ba et al. (2016). *Layer Normalization*. arXiv:1607.06450.
3) Garg et al. (2022). *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes*. arXiv:2208.01066v3.
4) Akyürek et al. (2023). *What Learning Algorithm Is In-Context Learning? Investigations with Linear Models*. arXiv:2301.05207.
5) Bai et al. (2023). *Transformers as Statisticians: Provable In-Context Learning with In-Context Algorithm Selection*. arXiv:2306.11644
6) Bishop & Bishop (2024). *Deep Learning: Foundations and Concepts*. Springer.

Exploring Transformer-Based Models for Cascading Extremes

Clemente Ferrer

clemente.ferrer@usm.cl

Department of Mathematics
Universidad Técnica Federico Santa María

Joint work with **Miguel de Carvalho** & **Ronny Vallejos**

Workshop on Generative AI for Extreme Events

Edinburgh, UK
June 13, 2025