The Transformer Architecture

Jindřich Libovický

March 7, 2024

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Today's Learning Outcomes

After today's class, you should be able to

- Explain **the building blocks** of the Transformer architecture to a non-technical person;
- Describe the Transformer architecture using **equations**, especially the self-attention block;
- **Implement** the Transformer architecture (in PyTorch or another framework that does automated differentiation).

Today's Programe

1. Quick recap quiz [5 min]
2. Lecture on the Transformer architecture [25 min]
3. Live coding session in PyTorch [45 min]
4. Lecture on architecture tweaks [15 min]

Architecture Tweaks

Activation Functions in the FF layer

- Origial Transformer paper ReLU $\max(x, 0)$

Activation Functions in the FF layer

- Origial Transformer paper ReLU $\max(x, 0)$
- GPT-2, BLOOM uses GELU (Hendrycks and Gimpel, 2016)

$$\text{GELU}(x) = x\Phi(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

Activation Functions in the FF layer

- Origial Transformer paper ReLU $\max(x, 0)$
- GPT-2, BLOOM uses GELU (Hendrycks and Gimpel, 2016)

$$\text{GELU}(x) = x\Phi(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

- Gemini (and its open source variant Gemma) uses GEGLU (Shazeer, 2020):

$$\text{GEGLU}(x, W, V, b, c) = \text{GELU}(xW + b) \otimes (xV + c)$$

Activation Functions in the FF layer

- Origial Transformer paper ReLU $\max(x, 0)$
- GPT-2, BLOOM uses GELU (Hendrycks and Gimpel, 2016)

$$\text{GELU}(x) = x\Phi(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

- Gemini (and its open source variant Gemma) uses GEGLU (Shazeer, 2020):

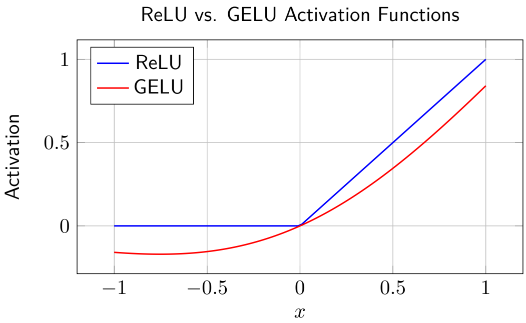
$$\text{GEGLU}(x, W, V, b, c) = \text{GELU}(xW + b) \otimes (xV + c)$$

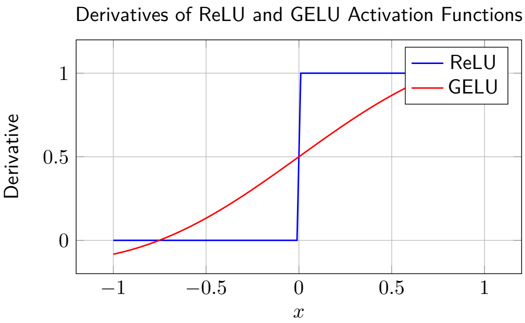
- LLaMA2 uses SwiGLU

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_\beta(xW + b) \otimes (xV + c)$$

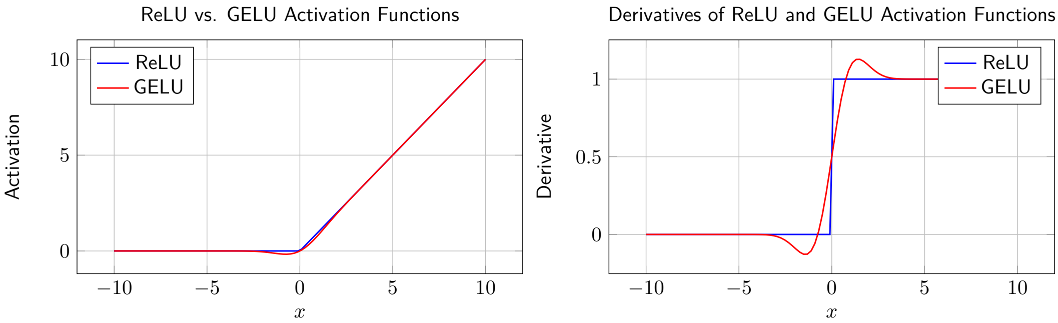
$$\text{Swish}_\beta(x) = x\,\text{sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}}$$

ReLU vs. GELU [-1; 1]

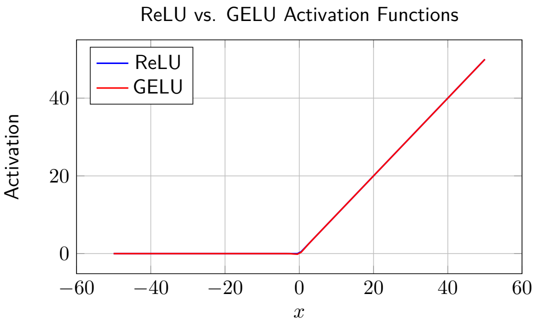


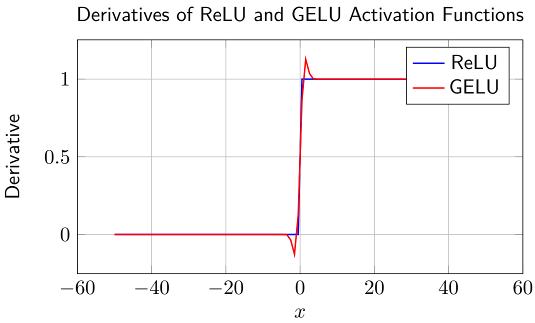


ReLU vs. GELU [-10; 10]

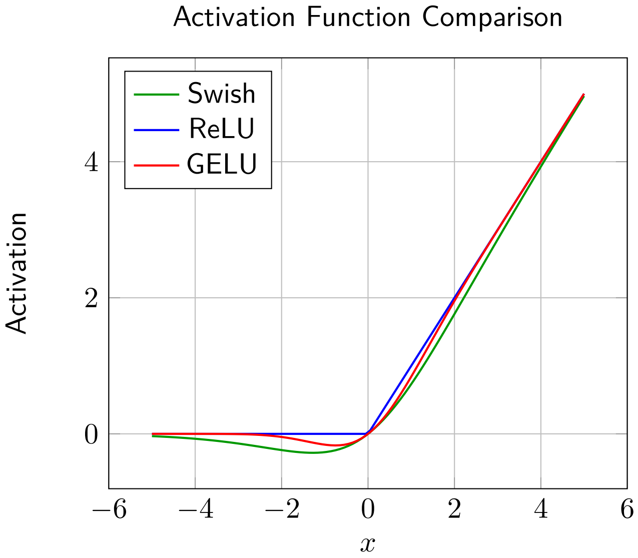


ReLU vs. GELU [-50; 50]





Swish activation function

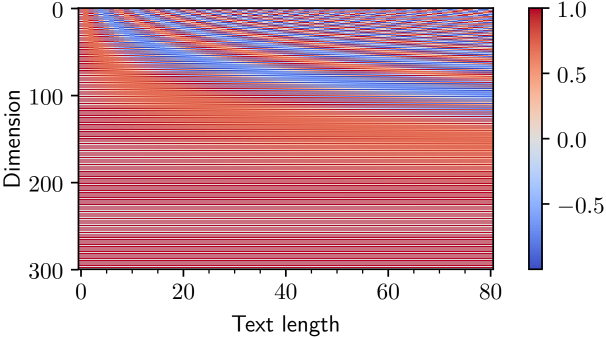


Position Embeddings

- Original 2017 paper: Sinusoidal absolute embeddings
- Often trained *absolute embeddings* embeddings (e.g., BERT or GPT-2 has that)
- Relative position encoding (e.g., T5)
- RoPE rotary positional encoding (e.g., BLOOM, LLaMA 2, PaLM, Gemma)

Sinusoidal Position Embeddings

$$\text{pos}(i) = \begin{cases} \sin\left(\frac{t}{10^4}^{\frac{i}{d}}\right), & \text{if } i \mod 2 = 0 \\ \cos\left(\frac{t}{10^4}^{\frac{i-1}{d}}\right), & \text{otherwise} \end{cases}$$



Rotary Position Embedding: Formulas

- Relative position embeddings: applied during QK multiplication

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021.
URL `https://arxiv.org/abs/2104.09864`

Rotary Position Embedding: Formulas

- Relative position embeddings: applied during QK multiplication
- RoPE: Analytical version of relative position embeddings

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding.
CoRR, abs/2104.09864, 2021.
URL `https://arxiv.org/abs/2104.09864`

Rotary Position Embedding: Formulas

- Relative position embeddings: applied during QK multiplication
- RoPE: Analytical version of relative position embeddings
- Designed s.t.: when we multiply projection W_Q and W_K by the matrix position matrix, than $q_m^T k_n$ multiplication will have information about $m - n$

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding.
CoRR, abs/2104.09864, 2021.
URL `https://arxiv.org/abs/2104.09864`

Rotary Position Embedding: Formulas

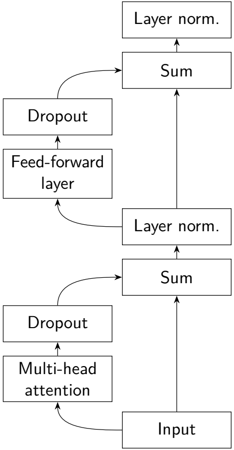
$$R^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$$

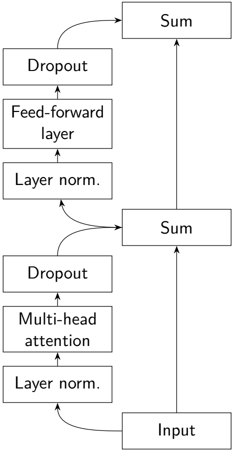
$$q_m^\intercal k_n = (R^d_{\Theta,m} W_q x_m)^\intercal (R^d_{\Theta,n} W_k x_n) = x^\intercal W_q R^d_{\Theta,n-m} W_k x_n$$

Pre- vs. Post-Layer Normalization

Original: Post-normalization



Now more common: Pre-normalization (Xiong et al., 2020)



Root Mean Square Layer Normalization

Instead of normalizing $\gamma \odot \left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\right) + \beta$

$$\gamma \odot \left(\frac{x}{\mathsf{RMS}(x)}\right), +\beta \qquad \text{where } \mathsf{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} a_i^2}$$

It is faster and does approximately the same thing as the original layer norm.

Biao Zhang and Rico Sennrich. Root mean square layer normalization.
In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019.
URL `https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html`

References I

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. URL http://arxiv.org/abs/1606.08415.

Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. *CoRR*, abs/2002.04745, 2020. URL https://arxiv.org/abs/2002.04745.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.