Attention is all you need

1- Introduction

- LSTM, and RNN, factor computation along the symbol positions of the input and the output sequences.
- generates $h_t = f(h_{t-1})$
  - ↳ precludes parallelization (becomes critical with long sequences) due to the sequential computation.

2- Background

Models such as ENGPU, ByteNet and ConvS2S use CNN as basic building block and aim at reducing sequential computation.

- difficult to learn long range dependencies (linearly for ConvS2S and log for Bytenet)
  - ↳ Transformers reduces it to constant number of operations.

3- Model Architecture

Most competitive neural sequence models have an encoder-decoder structure. The encoder maps an input sequence of symbol representations $(x_1, \dots, x_n)$ to continuous representation $z = (z_1, \dots, z_n)$. Given z, the decoder generates $(y_1, \dots, y_n)$. At each step the model is auto-regressive.

3.1- Encoder Decoder Stacks

Encoder: $N = 6$ identical layers.

- each layer has 2 sub-layer:
  - Multi-Head Self-Attention
  - Position-wise FFN.

} Residual connection and Layer Norm

$\text{LayerNorm}(x + \text{SubLayer}(x))$

↳ outputs $\in \mathbb{R}^{d=512}$

NN

Decoder: $N = 6$ layers

- 3 sublayer:
  - Multi-Head Attention
  - position wise FNN
  - MH Self Attention (with mask)

3.2- Attention

3.2.1- Scaled Dot-Product Attention

$Q, K \in \mathbb{R}^{d_k}$, $V \in \mathbb{R}^{d_v}$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$S = QK^T$: each row S_i contains attention scores for token i

each row A_i is a probability distribution across all input tokens defining how much attention token i pays to others.

$A_{ij} \geq 0$, $\sum_j A_{ij} = 1$

with

- Q: query vector (what to look for) $= XW^Q$, W^Q
- K: key vector (what each token offers) $= XW^K$, W^K
- V: value vector (content to share) $= XW^V$, W^V

$\in \mathbb{R}^{d_{\text{model}}=512} > d_k$

all are computed using learnable linear transformations.

What it does?

- weigh the importance of different parts of a sequence

Why scaling?

- if d_k is too large, $q_i^T k_j$ tends to have a large variance → softmax saturates.

If $q_i, k_j \sim \mathcal{N}(0,1)$, then $\mathbb{E}[q_i^T k_j] = 0$

but $\text{Var}(q_i^T k_j) = d_k$

Deriving gradients of Attention

Attention is fully differentiable → fully trainable via backprop

We need: $\frac{\partial \mathcal{L}}{\partial Q}, \frac{\partial \mathcal{L}}{\partial K}, \frac{\partial \mathcal{L}}{\partial V}$

is $\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial Z} \cdot \frac{\partial Z}{\partial A} \cdot \frac{\partial A}{\partial Q,K} \cdot \frac{\partial Q,K}{\partial X}$

- $\frac{\partial \mathcal{L}}{\partial V} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$ with $\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \hat{y}_i$
- $\frac{\partial \mathcal{L}}{\partial A} = \left(\frac{\partial \mathcal{L}}{\partial Z} @ V^T\right) \odot \text{softmax}'(A)$ → $\frac{\partial \mathcal{L}}{\partial W^V} = X^T @ \frac{\partial \mathcal{L}}{\partial V}$
- $\frac{\partial \mathcal{L}}{\partial Q} = \frac{1}{\sqrt{d_k}}\left(\frac{\partial \mathcal{L}}{\partial A}\right)K$ → $\frac{\partial \mathcal{L}}{\partial W^Q} = X^T @ \frac{\partial \mathcal{L}}{\partial Q}$
- $\frac{\partial \mathcal{L}}{\partial K} = \frac{1}{\sqrt{d_k}}\left(\frac{\partial \mathcal{L}}{\partial A}\right)^T Q$ → $\frac{\partial \mathcal{L}}{\partial W^K} = X^T @ \frac{\partial \mathcal{L}}{\partial K}$

then update params

3.1.1- Layer normalization (1911.07013)

- normalize the distribution of intermediate layers → reducing vanishing/exploding gradients

PROOF:

$$y_i = \gamma_i \left[\frac{x_i - \frac{1}{d}\sum_j x_j}{\sqrt{\frac{1}{d}\sum_j (x_i - \frac{1}{d}\sum_k x_k)^2 + \varepsilon}}\right] + \beta_i, \text{ with } \gamma_i, \beta_i \in \mathbb{R}^d \text{ learnable.}$$

$\Rightarrow$ y_i has a bounded mean and variance $\tilde{0}$

$\text{Var}\left[\frac{\partial \mathcal{L}}{\partial x}\right]$ is more consistant

- enables a faster training and convergence

PROOF

for convex fcts, GD satisfies:

$f(x^{(t)}) - f(x^*) < \frac{C}{t}$ with training speed depending on C.

with L: gradient Lipschitz cst

- how rapidly the gradient changes

$< \frac{L \cdot \|x^{(0)} - x^*\|^2}{2t}$

- $\|x^{(0)} - x^*\|$ the initial distance from the optimum

LayerNorm

given $X \in \mathbb{R}^d$, LayerNorm re-centers and re-scale X as:

$$\hat{X} = g @ \mathcal{N}(X) + b, \quad \mathcal{N}(X) = \frac{X - \mu}{\sigma}, \quad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

$\text{LayerNorm}(x) = \gamma \hat{X} + \beta$, where $\gamma, \beta \in \mathbb{R}^d$ are learnable params which were proven to cause overfitting, bc they are learned from the training set and they ignore the testing data distribution. 1911.07013 shows that LayerNorm offers lower training loss but higher validation loss than w/o γ, β.

The solution: AdaNorm

Let $y = \mathcal{N}(x) = \frac{x - \mu}{\sigma}$ be the normalized vector.

Let $\phi(y)$ be a function replacing the γ and β:

$z = \phi(y) @ y = \phi(\mathcal{N}(x)) @ \mathcal{N}(x)$ where

z is the output of AdaNorm.

3.3 - Position Wise Feed-Foward Networks

Let's remember that Attention is a linear process : Q, K, V are linear projections of X.

→ FNN introduces non-linearity letting the model transform each token representation independently, enables deep hierarchical representations and makes the model capable of learning complex mappings

$$FFN(x) = \sigma(xW_1 + b_1)W_2 + b_2$$

is (GeLU, ReLU, max, sigmoid, etc...)

Let $z_1 = xW_1 + b_1$, $z_2 = \sigma(z_1)W_2 + b_2$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial z_2} \cdot h^T \in \mathbb{R}^{d_{model} \times d_{ff}}$$

with $\mathcal{L} = -\sum_{i=1}^{d} y_i \log(\hat{y}_i)$

$$\frac{\partial \mathcal{L}}{\partial z_2} = \delta_2 \text{ (coming from upper layer)}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial z_2} \in \mathbb{R}^{d_{model}}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial z_1} \cdot x^T \in \mathbb{R}^{d_{ff} \times d_{model}}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial z_1} \in \mathbb{R}^{d_{ff}}$$

$$\frac{\partial \mathcal{L}}{\partial z_1} = \delta_1 \odot \text{ReLU}'(z_1)$$

Masked Multi-Head Attention

- ensures that the model can attend to tokens$_{0:i-1}$ when predicting token$_i$, to not leak any info.

→ -∞ becomes 0 after softmax, so the model won't give any importance to those tokens.

Cross-Attention

- integrates and aligns information from different sequences.
- takes as an input the src and target and captures their dependencies.
- Q from target, K and V from s
- example with translation: aligns source and target words. Generates the translation word by word.

$Q \in \mathbb{R}^{n \times d_k} = X_{dec} W^Q$

$K, V \in \mathbb{R}^{m \times d_k}$

$K = X_{enc} W^K$

$V = X_{enc} W^V$

$$\text{Cross Attention}(X_{enc}, X_{dec}) = \text{softmax}\left(\frac{X_{dec} W^Q \; @ \; X_{enc} W^{K^T}}{\sqrt{d_k}}\right) X_{enc} W^V$$