Attention Is All You Need

1 Introduction

1. Introduction:

- Establishes the use of recurrent neural networks (RNNs), long short-term memory (LSTM), and gated recurrent neural networks as state-of-the-art approaches in sequence modeling and transduction problems.
- Mentions various efforts to push the boundaries of recurrent language models and encoder-decoder architectures.

2. Contributions and Key People:

- Describes the equal contributions and involvement of different individuals in the development and evaluation of the proposed idea.
- Notes the affiliation and contributions of individuals from Google Brain and Google Research.
- Provides information about the conference where this work was presented.

3. Limitations of Recurrent Models:

- Explains the sequential nature of recurrent models and their inherent difficulty in parallelization, particularly with longer sequence lengths.
- Mentions recent efforts to improve computational efficiency through factorization tricks and conditional computation, but states that the constraint of sequential computation remains.

4. Attention Mechanisms in Sequence Modeling:

- Highlights the use of attention mechanisms in sequence modeling and transduction models to capture dependencies without regard to the distance between elements.
- Points out the usual combination of attention mechanisms with recurrent networks, except for a few cases.

5. Proposal of the Transformer Model:

- Introduces the Transformer as a model architecture that eliminates recurrence and relies solely on an attention mechanism to establish global dependencies between input and output.
- Emphasizes the Transformer's significant parallelization capabilities and its ability to achieve state-of-the-art translation quality with relatively short training time.

2 Background

1. Background of reducing sequential computation:

- The goal of reducing sequential computation forms the foundation of various models like Extended Neural GPU, ByteNet, and ConvS2S.
- These models use convolutional neural networks to compute hidden representations in parallel for all input and output positions.
- The number of operations required to relate signals from distant positions is reduced in the Transformer as compared to ConvS2S and ByteNet.
- However, this reduction in operations leads to a decrease in effective resolution, which is countered by Multi-Head Attention.

2. Self-attention mechanism:

- Self-attention, also known as intra-attention, is an attention mechanism that relates different positions within a single sequence to compute a representation of the sequence.
- Self-attention has been successfully used in various tasks such as reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence representations.

3. End-to-end memory networks and comparison to previous models:

- End-to-end memory networks use a recurrent attention mechanism and have performed well on simple-language question answering and language modeling tasks.
- The Transformer is the first transduction model that relies entirely on self-attention to compute representations of its input and output, without using sequence-aligned RNNs or convolution.
- The advantages of self-attention over models like Extended Neural GPU, ByteNet, and ConvS2S will be discussed in the following sections.

3 Model Architecture

1. Model Architecture:

- Most competitive neural sequence transduction models have an encoder-decoder structure.
- The encoder maps an input sequence of symbol representations to a sequence of continuous representations.
- The decoder generates an output sequence of symbols based on the continuous representations.
- The model is auto-regressive, meaning it uses previously generated symbols as additional input when generating the next symbol.

2. The Transformer Architecture:

- The Transformer follows the general encoder-decoder structure.
- It utilizes stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.
- Figure 1 shows the visual representation of the encoder and decoder components.

3.1 Encoder and Decoder Stacks

1. Encoder Stack:

- The encoder stack consists of N = 6 identical layers.
- Each layer has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.
- Residual connections and layer normalization are employed around each sub-layer.
- The output dimensions of all sub-layers and embedding layers are dmodel = 512.

2. Decoder Stack:

- The decoder stack also consists of N = 6 identical layers.
- In addition to the two sub-layers in each encoder layer, the decoder has a third sub-layer.
- The third sub-layer performs multi-head attention over the output of the encoder stack.
- Similar to the encoder, residual connections and layer normalization are used around each sub-layer.
- The self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions.
- Masking and offsetting of output embeddings ensure that predictions for a position depend only on known outputs at positions less than that position.

3.2 Attention

1. Introduction of Attention:

- The text introduces the concept of attention.

2. Definition of Attention Function:

- An attention function is described as a mapping between a query and a set of key-value pairs.
- The query, keys, values, and output are all vectors.

3. Computation of Output:

- The output is computed as a weighted sum of the values.
- The weight assigned to each value is determined by a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

1. Introduction to Scaled Dot-Product Attention

- We call our particular attention "Scaled Dot-Product Attention".
- The input consists of queries and keys of dimension ${dk}$, and values of dimension ${dv}$.

2. Computation of Scaled Dot-Product Attention

- We compute the dot products of the query with all keys, divide each by ${dk}$, and apply a softmax function to obtain the weights on the values.
- In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix ${Q}$.
- The keys and values are also packed together into matrices ${K}$ and ${V}$.
- The matrix of outputs is computed using the formula: $$\operatorname{Attention}(Q, K, V)=\operatorname{softmax}\left(\frac{Q K^{T}}{\sqrt{d_{k}}}\right) V$$

3. Comparison of Additive and Dot-Product Attention

- The two most commonly used attention functions are additive attention and dot-product (multiplicative) attention.
- Dot-product attention is identical to our algorithm, except for the scaling factor of ${\frac{1}{\sqrt{d_{k}}}}$.
- Additive attention computes the compatibility function using a feed-forward network with a single hidden layer.
- While the two mechanisms perform similarly for small values of ${d_{k}}$, additive attention outperforms dot-product attention without scaling for larger values of ${d_{k}}$.

4. Potential Issues with Dot-Product Attention

- For large values of ${d_{k}}$, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.
- To counteract this effect, we scale the dot products by ${d_{k}}$.

3.2.2 Multi-Head Attention

1. Introduction to Multi-Head Attention

- Instead of performing a single attention function with dmodel-dimensional keys, values, and queries, it is beneficial to linearly project them h times with different learned linear projections.
- The projected versions of queries, keys, and values are then used to perform the attention function in parallel, resulting in dv-dimensional output values.

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

2. Explanation of Dot Product and Variance

- To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance.
- The dot product, q · k = $${ }_{i=1}^{d_{k}} q_{i} k_{i}$$, has a mean of 0 and a variance of dk.

3. Mathematical representation of the MultiHead function

- The MultiHead function is represented as:

$$\operatorname{MultiHead}(Q, K, V)=\operatorname{Concat}(\text{head}_1, ..., \text{head}_h) W^O$$

- Where each head is computed as:

$$\text{head}_i = \operatorname{Attention}(Q W_i^Q, K W_i^K, V W_i^V)$$

- The parameter matrices for the projections are denoted as: $$W_i^Q \in \mathbb{R}^{d_{\text {model }} \times d_{k}}, W_i^K \in \mathbb{R}^{d_{\text {model }} \times d_{k}}, W_i^V \in \mathbb{R}^{d_{\text {model }} \times d_{v}}$$
- The final projection matrix is represented as: $$W^O \in \mathbb{R}^{d_v \times d_{\text{model}}}$$

4. Choice of Parameters in the Multi-Head Attention

- In this work, h = 8 parallel attention layers, or heads, are employed.
- Each head uses dk = dv = dmodel/h = 64.
- Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

1. Applications of Attention in the Model:

- The text discusses the applications of attention in the model.

2. Encoder-decoder Attention Layers:

- These layers involve queries from the previous decoder layer and memory keys and values from the output of the encoder.
- This allows each position in the decoder to attend to all positions in the input sequence.
- This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.

3. Encoder Self-attention Layers:

- The encoder also contains self-attention layers.
- In these layers, all keys, values, and queries come from the output of the previous layer in the encoder.
- Each position in the encoder can attend to all positions in the previous layer of the encoder.

4. Decoder Self-attention Layers:

- Similar to the encoder self-attention layers, the decoder self-attention layers allow each position in the decoder to attend to all positions in the decoder up to and including that position.

5. Preventing Leftward Information Flow:

- To preserve the auto-regressive property in the decoder, leftward information flow needs to be prevented.
- This is implemented inside scaled dot-product attention by masking out values in the input of the softmax that correspond to illegal connections.
- Specific details and a reference figure are mentioned.

3.3 Position-wise Feed-Forward Networks

1. Introduction to Position-wise Feed-Forward Networks

- In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network.
- The feed-forward network is applied to each position separately and identically.
- It consists of two linear transformations with a ReLU activation in between.
- The feed-forward network can be represented by the equation: $$\operatorname{FFN}(x)=\max \left(0, x W_{1}+b_{1}\right) W_{2}+b_{2}$$

2. Explanation of the Equation

- The equation represents the feed-forward network function $\operatorname{FFN}(x)$.
- It consists of two linear transformations.
- The first linear transformation is given by: $$x W_{1}+b_{1}$$
- The ReLU activation function $\max(0, \cdot)$ is applied element-wise to the output of the first linear transformation.
- The second linear transformation is given by: $$\text{output of ReLU activation function} \times W_{2}+b_{2}$$

3. Variation in Parameters

- While the linear transformations are the same across different positions, they use different parameters from layer to layer.
- This means that each layer in the encoder and decoder has its own set of parameters for the feed-forward network.

4. Convolution Interpretation

- Another way of describing the feed-forward network is as two convolutions with kernel size 1.

5. Dimensionality

- The dimensionality of the input and output of the feed-forward network is $d_{\text{model}} = 512$.
- The inner-layer of the feed-forward network has dimensionality $d_{\text{ff}} = 2048$.

3.4 Embeddings and Softmax

1. Embeddings and Softmax in Sequence Transduction Models:

- Learned embeddings are used to convert input tokens and output tokens to vectors of dimension dmodel.
- A learned linear transformation and softmax function are used to convert the decoder output to predicted next-token probabilities.
- In the model discussed, the same weight matrix is shared between the two embedding layers and the pre-softmax linear transformation.
- The weights in the embedding layers are multiplied by dmodel.

2. Reference to a Previous Study:

- The approach of sharing the weight matrix between embedding layers and the pre-softmax linear transformation is similar to a study mentioned as [(<>)24].

3.5 Positional Encoding

1. Introduction to Positional Encoding

- Since the model contains no recurrence and no convolution, the order of the sequence needs to be encoded.
- "Positional encodings" are added to the input embeddings.

2. Table of Maximum Path Lengths and Complexity per Layer

- Table 1: Maximum path lengths, per-layer complexity, and minimum number of sequential operations for different layer types.

- Lists the layer types, their complexity per layer, sequential operations, and maximum path length.
- Provides values for self-attention, recurrent, convolutional, and restricted self-attention layers.

3. Details of Positional Encoding

- Positional encodings are added to the bottoms of the encoder and decoder stacks.
- The positional encodings have the same dimension as the embeddings.
- There are choices of positional encodings, including learned and fixed options.

4. Sine and Cosine Functions for Positional Encoding

- The chosen positional encoding involves using sine and cosine functions of different frequencies.
- Formulas for positional encodings: $$PE_{(pos, 2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}})$$ and $$PE_{(pos, 2i+1)} = \cos(\frac{pos}{10000^{2i/d_{model}}})$$
- Each dimension of the positional encoding corresponds to a sinusoid with wavelengths forming a geometric progression.
- The sinusoidal encoding allows the model to easily learn to attend by relative positions.

5. Comparison of Learned and Sinusoidal Positional Embeddings

- Learned positional embeddings were also tested and found to produce similar results.
- The sinusoidal version was chosen because it may allow the model to handle longer sequence lengths than encountered during training.

4 Why Self-Attention

1. Introduction:

- The section discusses the comparison between self-attention layers, recurrent layers, and convolutional layers.
- These layers are used for mapping variable-length sequences to another sequence with equal length.

2. Desiderata for Self-Attention:

- Three desiderata are considered to motivate the use of self-attention: computational complexity per layer, parallelizability, and path length between long-range dependencies in the network.

3. Comparison of Computational Complexity:

- Self-attention layers have a constant number of sequentially executed operations, while recurrent layers require O(n) sequential operations.
- Self-attention layers are faster than recurrent layers when the sequence length is smaller than the representation dimensionality.
- To address computational performance for very long sequences, self-attention can be restricted to a neighborhood of size r around the output position.

4. Comparison with Convolutional Layers:

- A single convolutional layer with kernel width $k < n$ does not connect all input and output positions, requiring a stack of O(n/k) or O(logk(n)) convolutional layers.
- Convolutional layers are generally more expensive than recurrent layers, but separable convolutions decrease the complexity considerably compared to contiguous kernels.

5. Interpretable Models:

- Self-attention could lead to more interpretable models.
- Attention distributions from models can provide insights into the syntactic and semantic structure of sentences.

5 Training

1. Training Regime:

- This section focuses on describing the training regime for the models.
- It implies that there are specific methods or approaches for training the models, but does not provide further details.

5.1 Training Data and Batching

1. Training Data and Dataset Description:

- The training data used is the standard WMT 2014 English-German dataset, which consists of about 4.5 million sentence pairs.
- Byte-pair encoding is used to encode the sentences, resulting in a shared source-target vocabulary of approximately 37,000 tokens.
- For English-French, a significantly larger WMT 2014 English-French dataset was used, consisting of 36 million sentences.
- The tokens in English-French were split into a 32,000-word piece vocabulary.

2. Batching of Sentence Pairs:

- Sentence pairs were batched together based on approximate sequence length.
- Each training batch contained a set of sentence pairs with around 25,000 source tokens and 25,000 target tokens.

5.2 Hardware and Schedule

1. Hardware and Training Speed:

- The models were trained on one machine with 8 NVIDIA P100 GPUs.
- The training step for the base models took approximately 0.4 seconds each.
- The base models were trained for a total of 100,000 steps or 12 hours.
- The big models, as described in Table 3, had a longer step time of 1.0 seconds.
- The big models were trained for 300,000 steps, equivalent to 3.5 days.

5.3 Optimizer

1. Introduction to the Optimizer

- We used the Adam optimizer [(17)] with $\beta_1=0.9$, $\beta_2=0.98$, and $\epsilon=10^{-9}$.
- The learning rate was varied over the course of training according to the formula:

2. Formula for Learning Rate

- Learning Rate: $\text{lrate} = d_{\text{model}}^{-0.5} \cdot \min(\text{step\_num}^{-0.5}, \text{step\_num} \cdot \text{warmup\_steps}^{-1.5})$

3. Explanation of Learning Rate Formula

- This formula corresponds to increasing the learning rate linearly for the first warmup_steps training steps, and decreasing it thereafter proportionally to the inverse square root of the step number.
- We used $\text{warmup\_steps} = 4000$.

5.4 Regularization

1. Regularization:

- Three types of regularization are employed during training.
- Residual Dropout is applied to the output of each sub-layer before it is added to the sub-layer input and normalized.
- Dropout is also applied to the sums of the embeddings and positional encodings in both the encoder and decoder stacks.
- For the base model, a dropout rate of Pdrop = 0.1 is used.

2. Comparison of BLEU scores and training cost:

- Table 2 presents comparisons of BLEU scores and training costs of various models on English-to-German and English-to-French newstest2014 tests.

- The Transformer model achieves better BLEU scores than previous state-of-the-art models while having a fraction of the training cost.
- The table lists the models, their BLEU scores, and their training costs in FLOPs.

3. Label Smoothing:

- During training, label smoothing is employed with a value of □ls = 0.1.
- Label smoothing may hurt perplexity but improves accuracy and BLEU score.

6 Results

1. Machine Translation:

- The big transformer model outperforms previously reported models on the WMT 2014 English-to-German translation task, achieving a new state-of-the-art BLEU score of 28.4.
- The configuration and training time of this model are described.
- The base model also surpasses all previously published models and ensembles, at a lower training cost.

2. Machine Translation (Continued):

- The big model achieves a BLEU score of 41.0 on the WMT 2014 English-to-French translation task, outperforming previously published single models at a lower training cost.
- An adjustment in the dropout rate is mentioned for this model.

3. Hyperparameters and Inference:

- The process of averaging the last few checkpoints to obtain the final model is described.
- Details on the hyperparameters used in beam search and length penalty are provided.
- The maximum output length during inference is set.
- Termination of inference is mentioned if possible.

4. Results Summary:

- Table 2 summarizes the results and compares the translation quality and training costs to other model architectures from the literature.
- The estimation of the floating point operations used to train a model is explained.

6.2 Model Variations

1. Introduction to Model Variations

- We want to evaluate the importance of different components of the Transformer model.
- We varied our base model in different ways and measured the change in performance on English-to-German translation on the development set.

- We used beam search for evaluation.

2. Variation in Attention Heads and Dimensions

- Table 3 shows the variations on the Transformer architecture.
- In rows (A), we vary the number of attention heads and the attention key and value dimensions, while keeping the amount of computation constant.
- Single-head attention is 0.9 BLEU worse than the best setting, and having too many heads also decreases quality.

3. Performance on English-to-German Translation

- The metrics for performance on English-to-German translation are listed in Table 3.
- Perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

4. Further Variations in Model Parameters

- Rows (B) show the variation in attention key size, where reducing the key size hurts model quality.
- Rows (C) and (D) demonstrate that bigger models perform better, and dropout is useful for avoiding overfitting.
- In row (E), we replace sinusoidal positional encoding with learned positional embeddings and observe similar results to the base model.

7 Conclusion

1. Introduction to the Transformer:

- The Transformer is presented as the first sequence transduction model based entirely on attention.
- It replaces the recurrent layers commonly used in encoder-decoder architectures with multi-headed self-attention.

2. Advantages of the Transformer for translation tasks:

- The Transformer can be trained much faster than architectures based on recurrent or convolutional layers.
- It achieves a new state of the art in translation tasks such as English-to-German and English-to-French.

3. Future plans for attention-based models:

- The authors are excited about the future of attention-based models and plan to apply them to other tasks.

- They intend to extend the Transformer to handle input and output modalities other than text, such as images, audio, and video.
- Investigating local, restricted attention mechanisms to efficiently handle large inputs and outputs is another goal.
- Making generation less sequential is also a research goal.

4. Availability of the code and acknowledgements:

- The code used to train and evaluate the models is available at the provided GitHub link.
- The authors express their gratitude to Nal Kalchbrenner and Stephan Gouws for their comments, corrections, and inspiration.