Improved Transformer Architecture for Sequence to Sequence Translation

Austin Wang
Adviser: Prof. Karthik Narasimhan

Abstract

For many years, recurrent [21, 1, 26] and convolutional [7] neural networks have been firmly established as state of the art approaches to problems in machine translation (MT). The transformer is a recently developed neural network architecture that demonstrates significant improvements over existing architectures for MT; it is easier to train, highly parallelizable, and achieves better translation scores [25]. In contrast to previous architectures, the transformer relies entirely on attention mechanisms. Attention allows specific locations in an input sentence to inform the transformer's translation, but it lacks the ability to encode higher-level features from the input sentence such as positional dependence [17]. In this work, we present an improved transformer architecture: the Convolutional Multi-headed attentive Encoder Transformer or COMET. COMET uses multi-layer convolutions to build high-level representations of the input sentence and multi-headed attention mechanisms to facilitate focused queries within the input. On the Multi30k German to English translation task, COMET demonstrates a 0.33 BLEU increase over the standard transformer architecture.

1. Introduction

Machine translation (MT) is the computational task of generating translations from one language to another. While MT is still inferior to human translation, the application of neural networks to translation tasks (neural machine translation, or NMT) is quickly closing this gap [26, 23].

Recurrent neural networks (RNNs) were one of the earliest proposed NMT architectures and have demonstrated state of the art performance on several translation tasks [1, 21, 26]. An RNN reads the input sequence word by word and encodes into a vector representation. This vector is fed

into another RNN which decodes it into a sequence in the target language. A critical disadvantage of this approach is the compression of the input sequence into a fixed length vector, and empirical results have shown a drop in performance in these architectures for long sequences [4].

To improve the performance of RNNs on long sequences, the attention mechanism was developed [1]. Rather then compressing the entire input into a single vector, the encoder produces a sequence of vectors and the decoder learns to pick out a subset of these vectors adaptively while generating the output sequence.

While the RNN encoder-decoder with attention has demonstrated state of the art results in translation quality [26], the RNN must read the input sequentially. This implies an $O(n)$ runtime and difficulty in parallelizing the model. Furthermore, RNNs suffer from vanishing gradients which make them difficult to train [2]. For these reasons, highly parallelizable convolutional architectures such as ByteNet [12] and ConvS2S [7] were developed, establishing new state of the arts in translation scores. The convolutional architecture allows for easier access to information in the input sequence. However, the number of convolutional layers required to increase the receptive field to the entire length of the input sequence increases linearly for ConvS2S and logarithmically for ByteNet. This implies that the relationships between distant elements in the input are only coarsely available to the model, since these elements only interact in higher convolutional layers [17].

The transformer attempts to mitigate the issues of convolutional architectures, while remaining parallelizable [25]. It relies entirely on attention and self-attention mechanisms. These mechanisms allow the transformer to query any word of the input sequence to inform its translation, regardless of the distance to the word or the input length. The transformer reduces the cost of training by almost an order of magnitude and beats previous convolutional architectures in translation score. While excellent at pinpointing specific locations of interest in the input, attention mechanisms struggles to incorporate surrounding information such as positional dependencies in the input [17].

We developed the Convolutional Multi-headed attentive Encoder Transformer or COMET architecture to combine the best features from convolutional and self-attention architectures. The

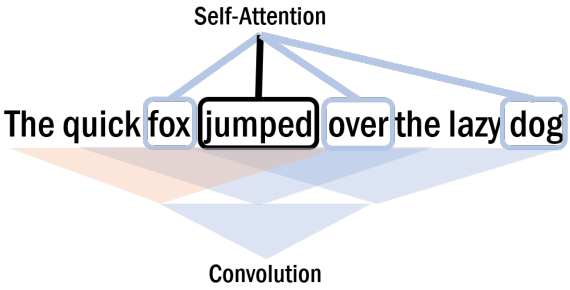

Figure 1: Example in which convolutions may complement self-attention. When encoding *jumped*, the self-attention mechanism might attend to the words that *jumped* act on: *fox, over* and *dog*. However, since positional information is only weakly incorporated in the transformer, it may confound the subject and object. Such examples can be found in the results section. However, the convolution over *The quick fox jumped* (in red) may provide the model with the positional information necessary to conclude that *fox* is the subject.

convolutional architectures are able to capture positional dependencies and other high level features, but this information is blurred by multiple layers of convolution. On the other hand, the attention mechanisms in the transformer allow immediate access to any location within the input, at the cost of sacrificing wider context. In COMET, the encoder uses self-attention to isolate specific points of interest in the input, while convolutional structures provide a high-level picture of the input.

Intuitively, convolutional structures act as a wide-angle lens that allows the model to see how different components in the input relate to each other, while self-attention is a telephoto lens that allows the model to zero in on specific details. By combining the two, COMET is able to build a more detailed picture of the landscape. This intuition is supported empirically. On the Multi30k German to English translation task, COMET outperforms the transformer by over 0.3 BLEU.

2. Background

2.1. Attention

The attention mechanism maps a query and a set of key-value pairs to an output [25]. The output is a weighted sum of the values, and these weights are calculated using a compatibility function between the query and the key. Intuitively, the value and query encode information about specific elements in a sequence (or sequences). The compatibility function uses the query and the key to determine how important the element corresponding to the key is relative to the element corresponding to the

query. The compatibility function then scales the weights accordingly, assigning more weight to values of greater importance.

In the transformer and COMET, we use a dot-product attention [16] with scaling [25]. The query, key, and value are vectors. Since a sentence is composed of multiple words, the vectors are packed into matrices Q, K, V respectively. The compatibility function is simply the softmax of the dot product between Q, K scaled by the square root of the key dimension:

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Self-attention is the application of the attention mechanism to a single source. That is, the queries, keys, and values all correspond to elements within the same sequence. Contrast this with more general attention mechanisms which can occur between different sequences. By attending to the input, self-attention mechanisms are able to pick up on the syntactic and semantic aspects of the input sequence.

Multi-headed attention, is the use of multiple attention functions in parallel. This allows the model to build different representations of the sequence. As with the transformer, in COMET we use multi-headed self-attention in the encoder, and a combination of multi-headed attention and multi-headed self-attention in the decoder. For more information on self and multi-headed attention, refer to the original transformer paper [25].

2.2. Convolutional Structures

In COMET we employ a combination of dilated convolution and gated activation. The convolutional structure is similar to that of ByteNet [12] and SNAIL [17]. Dilated convolutions are a way to increase the receptive field of a kernel without increasing the kernel size by introducing spaces within the kernel [28]. A dilated convolution with dilation factor l can be expressed as:

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t) \tag{2}$$

and note that $l = 1$ is just a regular discrete convolution. Similar to SNAIL, in COMET we use linearly increasing dilation to increase the receptive field without significantly increasing computational cost.

Gated activation is a way to control the relevance of an output by multiplying each element in the output by a scaling factor $\in (0,1)$. We use two convolutions followed by a gated activation [24], which we will refer to as a gated convolution:

$$\text{GatedConv}(x) = f(k_f * x) \otimes \sigma(k_\sigma * x) \tag{3}$$

where f in our case is the tanh function, σ is the sigmoid function, and $\otimes$ denotes element-wise multiplication. The output $\sigma(k_\sigma * x)$ acts as gates by maping each element in the preactivation to a scaling factor.

2.3. Other Work

All translation models described above either employ convolutional or self-attention mechanisms exclusively to encode the input. We are aware of only one example in which self-attention and convolution have been used simultaneously - the SNAIL architecture for meta-learning [17]. However, SNAIL is not designed for MT and is primarily concerned with learning temporal dependencies. To generate the representation for element e_t at time t, SNAIL employs a temporal convolution on a subset of previous items $\{e_{t'}, t' \leq t\}$. In COMET, we use a 1-dimensional spatial convolution over the input sentence, so that any element e_t in the input can depend on elements both ahead and behind e_t in the sequence.

Other work has improved the transformer in either its ability to adapt to long sequences [5], or its runtime [3], but to the best of the author's knowledge, this is the first time self-attention and convolution has been combined in an encoder-decoder architecture for MT.

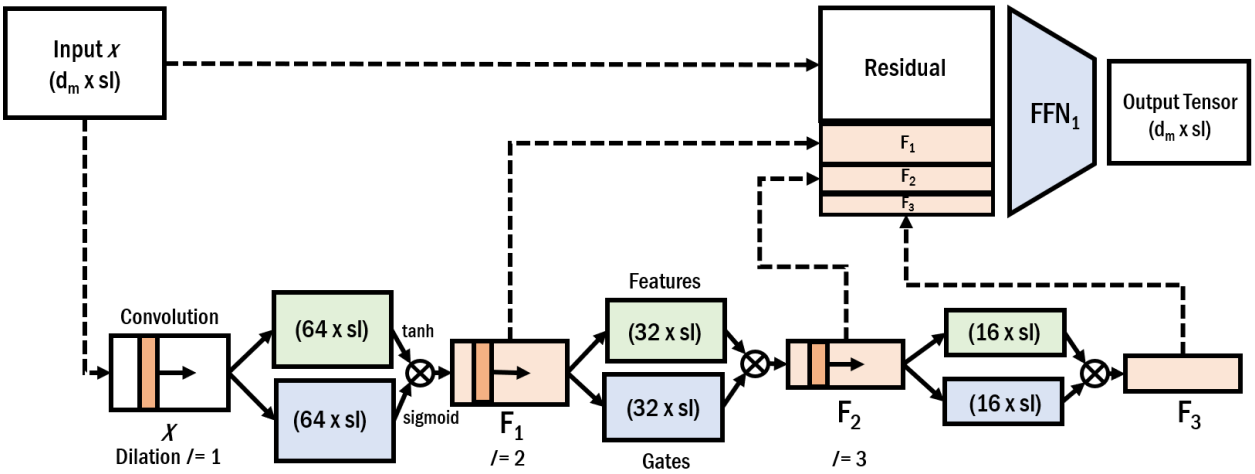

Figure 2: A single convolutional subunit. The input tensor x is passed through three successive gated convolution layers with 64, 32, and 16 features respectively. At each layer, we use a kernel with size 3, and we increase the dilation rate linearly from 1 to 3 to cheaply increase the receptive field. The resulting feature maps F_1, F_2, F_3 are concatenated with the input and passed through a single-layer feed-forward net with ReLU activation. Note that the input and output dimensions are identical.

3. Model Architecture

We describe the best performing COMET architecture here and defer specifics of model variations to the results section. Let sl refer to the length of the input sequence and unless otherwise specified, let the dimension of the model $d_m = 256$. In both the COMET and transformer architectures, we maintain the invariant that all inputs and outputs to subunits are in $\mathbb{R}^{d_m \times sl}$.

3.1. Convolutional Subunit

A single convolutional layer consists of a 1-dimensional gated convolution along the sl dimension of the input tensor. Each layer has kernel size 3, input dimension d_{in}, dilation rate l and and number of output features n_f. We pad both ends of the input tensor with l zeros in the sl dimension to maintain the dimensionality invariant and generate an output tensor with size $n_f \times sl$.

Let $g_{d_{in},n_f,l} : \mathbb{R}^{d_{in} \times sl} \to \mathbb{R}^{n_f \times sl}$ denote the gated convolution as described. To capture interactions between distant elements in the input sequence, we do a multi-layer convolution to generate three sets of feature maps with 64, 32, and 16 features respectively. To aid training speed we employ a batch normalization on the output of all gated convolutions [11].

`MultiLayerConv(`x`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Feature maps $F_1 \in \mathbb{R}^{64 \times sl}, F_2 \in \mathbb{R}^{32 \times sl}, F_3 \in \mathbb{R}^{16 \times sl}$

$F_1 =$`BatchNorm(`$g_{d_m,64,1}(x)$`)`
$F_2 =$`BatchNorm(`$g_{64,32,2}(F_1)$`)`
$F_3 =$`BatchNorm(`$g_{32,16,3}(F_2)$`)`

return F_1, F_2, F_3

The feature maps F_1, F_2, F_3 from a single convolutional subunit has receptive fields 3, 7, and 13 respectively. However, due to the stacking of subunits, the effective receptive field is much larger. The effects of varying the number of convolutional layers is discussed in model variations in the results section. At higher levels, the feature maps are able to see more of the input sequence and thus incorporate distant interactions. However, the information available is coarser. To give the model access to information of different resolutions, we concatenate all three feature maps to the input to generate a new tensor with dimension $(d_m + 112) \times sl$. To maintain the dimensionality invariant, we pass the resulting tensor through a one layer feed-forward network with ReLU activation:

$$\text{FFN}_1(x) = \max(\vec{0}, Wx + b) \tag{4}$$

where $W \in \mathbb{R}^{d_m \times (d_m + 112)}$ to produce an output tensor of dimension $d_m \times sl$. To facilitate weight updating when the pre-activation is negative, we use a leaky ReLU [27]. The full convolutional subunit can then be expressed as:

`ConvUnit(`x`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Tensor $y \in \mathbb{R}^{d_m \times sl}$

$F_1, F_2, F_3 =$ `MultiLayerConv(`x`)`
$x' =$ `concat(`F_1, F_2, F_3, x`)`
$y =$ `FFN`$_1(x')$

return y

3.2. Self-Attention Subunit

The self-attention subunit is very similar to the one used in the transformer architecture. To allow COMET to attend to multiple locations in the input sequence, we use a multi-headed attention, in which h attention layers are run simultaneously. The input queries, keys, and values, $q,k,v \in \mathbb{R}^{d_m \times sl}$ are first passed through affine layers, $Wx+b$, $W \in \mathbb{R}^{d_m \times d_m}, b \in \mathbb{R}^{d_m \times 1}$. Each resulting tensor is then divided into h tensors along the d_m dimension with size $d_h \times sl$ where $d_h = \frac{d_m}{h}$ is the head dimension. Each tuple (q_i, k_i, v_i) is passed to a single head (attention layer) that is independent of all other heads. The results are concatenated into a tensor of size $d_m \times sl$, and then passed into a final affine layer.

The dimensionality of each head is reduced to d_h to ensure that the computational cost does not increase significantly with h. In both the COMET and transformer architectures, we use 8 heads to produce a head dimension of $d_h = 32$. The multi-headed attention function is as follows:

`MultiHeadedAtten(`q,k,v`)`

---

input : Tensor $q,k,v \in \mathbb{R}^{d_m \times sl}$

output : Tensor $y' \in \mathbb{R}^{d_m \times sl}$

$$q',k',v' = W_q x + b_q,\ W_k x + b_k,\ W_v x + b_v$$

$$q' = \begin{bmatrix} q'_1 \\ \vdots \\ q'_h \end{bmatrix},\ k' = \begin{bmatrix} k'_1 \\ \vdots \\ k'_h \end{bmatrix},\ v' = \begin{bmatrix} v'_1 \\ \vdots \\ v'_h \end{bmatrix}$$

$$y = \begin{bmatrix} \text{Attention}(q_1,k_1,v_1) \\ \vdots \\ \text{Attention}(q_h,k_h,v_h) \end{bmatrix}$$

$$y' = W_y y + b_y$$

return y'

Self-attention is simply when all keys, values, and queries are derived from the input sequence.

`SelfAttenUnit(`x`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Tensor $y \in \mathbb{R}^{d_m \times sl}$

$y =$ `MultiHeadedAtten(`x, x, x`)`

return y

3.3. Embeddings and Positional Encoding

As with other sequence to sequence models, COMET uses a learned embedding to transform each input token to a vector of dimension d_m. The output of each embedding is scaled by $\sqrt{d_m}$. In both the transformer and COMET models, we do not use weight sharing between the input and output embedding layers.

To provide the self-attention layers with positional information, after embedding the source and target sequences, we sum the embeddings with a positional encoding before feeding the sum to the encoder or decoder. In both COMET and the transformer, we use sinusoidal positional encoding [25]. Let $PE \in \mathbb{R}^{d_m \times sl}$ be the positional encoding matrix. We define PE as:

$$\begin{aligned} PE[2i, p] &= \sin\left(\frac{p}{10000^{\frac{2i}{d_m}}}\right) \\ PE[2i+1, p] &= \cos\left(\frac{p}{10000^{\frac{2i}{d_m}}}\right) \end{aligned} \tag{5}$$

3.4. Encoder

A single encoder unit consists of a self-attention subunit followed by a gated convolution subunit. At the output of each subunit, we use a residual connection [9] followed by layer normalization [14]. The full encoder consists of stacking N encoder units. In both COMET and the transformer, we use $N = 3$.

`Encoder(x)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Tensor $\in \mathbb{R}^{d_m \times sl}$

Function `EncoderUnit(x)`**:**

  $x' = \texttt{LayerNorm}\big(x + \texttt{SelfAttenUnit}(x)\big)$

  $y = \texttt{LayerNorm}\big(x' + \texttt{ConvUnit}\ (x')\big)$

  return y

for $i \leftarrow 1$ **to** N **do**

  $x = \texttt{EncoderUnit}(x)$

return x

3.5. Decoder

The decoder unit is identical to the one in the transformer. The input is passed through a self-attention unit followed by an attention unit that attends to the outputs of the decoder. The output to the attention unit is passed through a two layer feed forward network with ReLU activation:

$$\text{FFN}_2(x) = W_2\big(\max(\vec{0}, W_1 x + b_1)\big) + b_2 \tag{6}$$

where $W_1 \in \mathbb{R}^{d_{ff} \times d_m}, b_1 \in \mathbb{R}^{d_{ff} \times 1}, W_2 \in \mathbb{R}^{d_m \times d_{ff}}, b_2 \in \mathbb{R}^{d_m \times 1}$, and d_{ff} is the number of neurons in the hidden layer. For COMET and transformer we use $d_{ff} = 2048$. As with the encoder unit, we use residual connections followed by layer normalization after every subunit. The decoder consists of stacking $N = 3$ decoder units. To prevent the decoder from attending to subsequent positions, we apply appropriate masking to the self-attention unit.

`Decoder(`x, y`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$, Decoder output $y \in \mathbb{R}^{d_m \times sl}$
output : Tensor $\in \mathbb{R}^{d_m \times sl}$

Function `DecoderUnit(`x`)`**:**
  $x' = \texttt{LayerNorm}\big(x + \texttt{SelfAttenUnit}(x)\big)$
  $z = \texttt{LayerNorm}\big(x' + \texttt{MultiHeadedAtten}\ (x', y, y)\big)$
  $z' = \texttt{FNN}_2(z)$
  return z'
for $i \leftarrow 1$ **to** N **do**
  $x = \texttt{DecoderUnit}(x)$
return x

3.6. COMET

On the encoder side, the input sequence first passes through an embedder and positional encoder before being passed to the encoder. The decoder is auto-regressive, taking as input the previously generated tokens [8]. These tokens, which we denote intermediate outputs, are passed through an embedder and positional encoder and along with the encoder outputs, is passed to the decoder.

To convert the final decoder output into a a prediction for the next token, we pass the decoder output through an affine layer that projects each decoder output $y \in \mathbb{R}^{d_m}$ to another vector $y' \in \mathbb{R}^{s_t}$, where s_t is the size of the target vocabulary. Each element in y' corresponds to a separate token in the target vocabulary, and we apply a softmax on y' to produce a likelihood for each word in the target vocabulary.

$$\text{TokenLikelihoods} = \text{softmax}(Wy + b) \tag{7}$$

`COMET(`x, y`)`

input : Input tokens $x \in [s_i]^{sl}$, Intermediate output $y \in [s_t]^k$
output : Next token likelihoods $z \in (0,1)^{s_t}$

  $x' = \texttt{Encoder(PositionalEncoder(Embedder(}x\texttt{)))}$
  $y' = \texttt{Decoder(PositionalEncoder(Embedder(}y\texttt{)), }x'\texttt{)}$
  $likelihoods = \texttt{TokenLikelihoods(}y'\texttt{)}$

return *likelihoods*

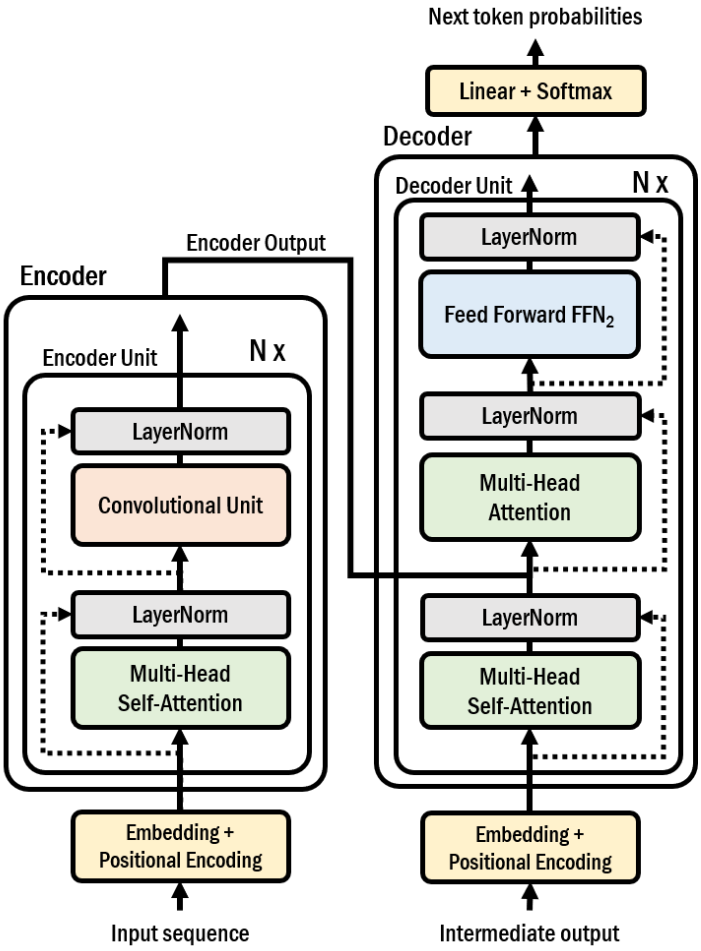

Figure 3: The full COMET architecture. The input sequence passes through an embedder and positional encoder before being passed to the encoder. The encoder consists of N successive encoder units. The intermediate output passes through an embedder and positional encoder before being passed to the decoder along with the encoder output. The decoder consists of N successive decoder units. Finally, the output passes through a linear layer with softmax to predict the next token. Note that the only key difference between COMET and the transformer is the replacement of the feed-forward net in the encoder unit with a convolutional subunit.

where s_i is the size of the input vocabulary, and s_t is the size of the target vocabulary.

4. Training and Evaluation

4.1. Datasets and Preprocessing

The models were trained on the Multi30k dataset on a German to English translation task [6]. The dataset consists of 30,000 parallel sentence pairs in German and English. The dataset was split into a training set of 29,000 sentence pairs and a validation set of 1,000 sentence pairs. For testing, we

used the 2016 edition Multi30k test set. Tokenization was done using SpaCy [10].

4.2. Training

The models were implemented using PyTorch [19] and trained using stochastic gradient descent with cross-entropy loss and batch size of 10. We use the ADAM optimizer [13] with $\beta_1 = 0.9, \beta_2 = 0.98, \varepsilon = 10^{-9}$ and learning rate of 10^{-4}.

We applied dropout [20] to the output of each unit before the residual connection and layer normalization, as well as to all embeddings and positional encodings in both the encoder and decoder. For all models, we used a dropout rate of 0.1.

The models were either trained a NVIDIA GTX1050 or NVIDIA TESLA K80 GPU. Models were trained until validation loss started increasing for at least two consecutive epochs. The model state corresponding to the lowest validation loss was used for further evaluation.

4.3. Evaluation

Translations were generated using a greedy decoding method. The current sentence s is initialized containing only a start of sentence (SOS) token. The next token t is selected to be the word of maximum likelihood in the model output and the current sequence is updated $s = [s\, t]$. The updated sequence is then fed into the decoder again to predict the next token, and this continues until the end of sentence (EOS) token is produced.

$$t = \arg\max_i \left(\texttt{COMET}(x, s)[i]\right) \tag{8}$$

Models were evaluated on the validation set for development and test set for evaluation. For each input sequence, a candidate translation is produced by greedy decoding. The candidate is scored against the target (reference) translation using BLEU score [18]. The final BLEU score is then reported as an average of all BLEU scores in the evaluation dataset. Due to some short sentence lengths resulting in zero BLEU scores, we employ a smoothing function which adds 1 to both the numerator and denominator [15].

Model	BLEU score (development)
COMET	**38.65**
Temporal Convolutions	37.71
FFN-1	38.10
FFN-2	38.46
Source EOS/SOS token	37.78
4-Layer Convolutional Subunit	38.17
Transformer	**38.33**
$d_{ff} = 1024$	38.37*
$N = 4$	37.10
$N = 2$	37.70
$d_m = 512$	37.63
dropout = 0.3	37.89
	BLEU score (test)
COMET	**38.69**
Transformer	**38.36**

Table 1: BLEU scores of different COMET and transformer architectures. The COMET model is the one described in the model architecture section. The baseline transformer architecture uses parameters $N = 3, d_m = 256, d_{ff} = 2048$, and each modification indicates a specific variation on each parameter while holding other parameters constant. Note that $d_{ff} = 1024$ performed inconsistently, so we use $d_{ff} = 2048$ in all models.

5. Results

5.1. COMET vs Transformer

To obtain a baseline transformer score on which we compare the performance of COMET, we experimented with different transformer configurations, some of which are listed in table 1. We found that a transformer with $N = 3, d_m = 256, d_{ff} = 2048$ and dropout = 0.1 achieved the highest BLEU score on the development set. We use the exact same parameters in COMET where relevant to facilitate comparison between the two models.

From table 1, we see that compared to the best transformer model, COMET achieves a 0.33 BLEU score increase on the Multi30k German to English translation task. The quantitative improvement in performance is supported by qualitative evidence as shown in table 2. On many challenging input sentences, COMET preserves the overall meaning of the source sentence, whereas the corresponding translations from the transformer were often unintelligible. We hypothesize that the convolutional

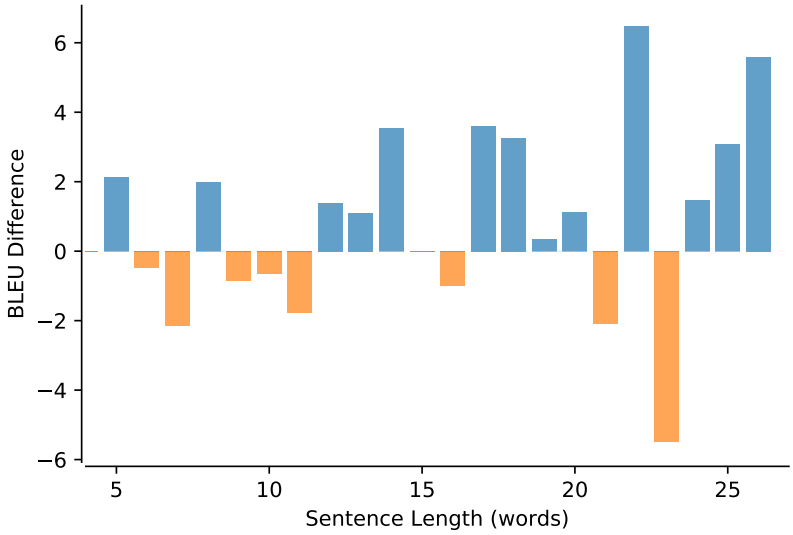

Figure 4: BLEU differential between COMET and the transformer on the test set by input sentence length. A positive differential indicates a higher BLEU score by COMET. Lengths with less than three examples were removed.

layers in the encoder captures high-level features in the input sequence. The decoder then tries to retain these features when generating the output sequence, resulting in translations that are closer to the meaning of the input sentence.

We also hypothesized that positional dependencies and high level features are more informative as sentence length increases. This hypothesis is supported by the difference in BLEU score between COMET and the transformer on translations of input sentence with various lengths. In figure 4 we see that while the transformer generally performed better on very short sentence of less than 12 words, COMET consistently outperforms the transformer on input sentences with 12 or more words.

5.2. Temporal Convolutions

To evaluate the effects of convolutional architectures in the decoder, we replaced the feed-forward net in the decoder with a temporal convolution unit. The convolutional unit convolves over the intermediate output, providing positional dependencies and other high level information. To prevent the model from convolving over subsequent elements in the target, we pad the start of the sequence with $2l$ zeros to implement a temporal convolutional [24]. Aside from the modification to the

Translator	Translation
Reference	A lady with tattoos is taking a picture of a painting with her smartphone.
Transformer	A woman with tattoos is taking a picture of a painting of a painting.
COMET	A woman with tattoos is taking a picture of a painting with her cellphone.
Reference	A brown dog walks in the grass with its tongue hanging out.
Transformer	A brown dog is running through grass hanging out of her tongue
COMET	A brown dog is walking through the grass with his tongue hanging out.

Table 2: Two example translations in which positional dependencies are important. In both cases, COMET preserved the meaning of the original sentence, whereas the transformer mistranslates both instances. In the second example, the tongue is hanging out of the dog. However, without strong positional information, the transformer inverts the subject (tongue) and object (dog) in its translation.

padding, all other features including dilation, number of layers, number of feature maps and kernel size are identical to that of the convolutional unit in the encoder.

Convolutions in the decoder were detrimental to model performance as a whole. This suggests that high-level features of the intermediate output may be less beneficial to accurate sequence generation, and may in fact confound the decoder. However, we did not experiment extensively with temporal convolutions in the decoder, and further modifications might improve model performance.

5.3. Deep FFN Layers

We experimented with increasing the representation power of the encoder by introducing additional feed-forward components into the convolutional units in the encoder. In the first modification to the convolutional unit denoted FFN-1 in table 1, we pass the concatenation of the features maps F_1, F_2, and F_3 through a two layer feed-forward network with hidden layer size $d_{ff} = 512$, ReLU activation, and output dimension equal to input dimension. We then employ a residual connection followed by a layer normalization before passing the result through a single-layer feed-forward net. During training we employ a dropout of 0.1 on the output of both feed-forward nets.

`ConvUnitFFN-1(`x`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Tensor $y \in \mathbb{R}^{d_m \times sl}$

$$F_1, F_2, F_3 = \texttt{MultiLayerConv}(x)$$
$$F' = \texttt{concat}(F_1, F_2, F_3)$$
$$x' = \texttt{LayerNorm}\big(F' + \texttt{FFN}_2(F')\big)$$
$$y = \texttt{FFN}_1\left(\begin{bmatrix} x' \\ x \end{bmatrix}\right)$$

return y

In the second modificaiton which we denote FFN-2, we pass the concatenation of feature maps F_1, F_2, F_3 and the input x through a 2-layer feed-forward net with hidden layer dimension $d_{ff} = 1024$.

`ConvUnitFFN-2(`x`)`

input : Tensor $x \in \mathbb{R}^{d_m \times sl}$

output : Tensor $y \in \mathbb{R}^{d_m \times sl}$

$$F_1, F_2, F_3 = \texttt{MultiLayerConv}(x)$$
$$x' = \texttt{concat}(F_1, F_2, F_3, x)$$
$$y = \texttt{FFN}_2(x')$$

return y

Neither modification increased BLEU score from the standard COMET architecture, suggesting the the simple dimensionality reducing 1-layer feed-forward network is sufficient in extracting the high-level features from the input sequence. Introducing extra parameters into the convolutional unit may have caused the model to overfit on the training data, limiting its ability to generalize.

5.4. Other Modifications

While the 1-dimensional convolution in the encoder provides information on positional dependencies, information such as the absolute position of the elements are only weakly incorporated by the positional encodings in the transformer. To allow the model to access information about an element's relative position to the start and end of the input sequence, we appended an SOS and EOS token to the beginning and end of the input sequence respectively. When the convolution occurs

near the beginning or end of the input sequence, such information can be incorporated into the resulting feature maps.

While such a modification drastically reduced the validation loss to 1.30 per 100 tokens (compared to a loss of 1.33 for COMET and 1.34 for the transformer), we observed a significant drop in BLEU score. This may be due to the model becoming overly confident, which decreases perplexity, but may hurt BLEU [25]. For further work, penalizing over-confidence with label smoothing may improve performance [22].

A similar result was observed when we increased the depth of the convolutional subunit to 4. We generated an extra feature map F_4 by convolving over F_3 with a kernel of size 3 and dilation $l = 4$. We then concatenate all feature maps F_1, F_2, F_3, F_4 with the input x and pass the concatenated tensor into a single layer feed forward network. Again, we observed low validation loss of 1.30 per 100 tokens, but a significant drop in BLEU score, which may also be caused by model over-confidence.

6. Conclusions

In this work we present COMET, a novel neural architecture for sequence to sequence translation that builds on the previous transformer architecture. COMET combines convolutional units to extract high-level features from the input, and self-attention mechanisms to facilitate focused querying of any location within the input. By combining these complementary mechanisms, COMET achieves better translation performance, beating the transformer by 0.33 BLEU on the Multi30k German to English translation task.

6.1. Limitations and Future Work

There are numerous limitations in this work that present exciting opportunities for further work. First, to facilitate rapid prototyping and model development, we selected a small dataset. While this drastically reduced training time, the results of this study is not directly comparable with other works in the literature which mainly use the WMT 2014 dataset [21, 7, 25].

Although COMET beats the transformer by a significant margin, we have only scratched the surface of convolutional self-attention architectures. The model variations that we tried, some

of which we presented in the results section, are only cursory initial explorations of potential architectures. This work has demonstrated the potential of combining convolutions and self-attention for MT, and future work should be focused on zeroing in on the best ways to incorporate these two powerful mechanisms.

6.2. Acknowledgements

I would like to thank my advisor, Prof. Karthik Narasimhan for his invaluable guidance throughout the course of this project, my fellow classmates and friends of the *IW06: Natural Language Generation* seminar for their support, and the staff of the Princeton IW program for making this partnership a possibility.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[2] Y. Bengio *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[3] R. Child *et al.*, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[4] K. Cho *et al.*, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[5] Z. Dai *et al.*, "Transformer-xl: Language modeling with longer-term dependency," 2018.

[6] D. Elliott *et al.*, "Multi30k: Multilingual english-german image descriptions," in *Proceedings of the 5th Workshop on Vision and Language*. Association for Computational Linguistics, 2016, pp. 70–74. Available: http://www.aclweb.org/anthology/W16-3210

[7] J. Gehring *et al.*, "Convolutional sequence to sequence learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1243–1252.

[8] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[9] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] M. Honnibal and I. Montani, "spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing," *To appear*, 2017.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[12] N. Kalchbrenner *et al.*, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[14] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[15] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 605.

[16] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[17] N. Mishra *et al.*, "A simple neural attentive meta-learner," *arXiv preprint arXiv:1707.03141*, 2017.

[18] K. Papineni *et al.*, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 311–318.

[19] A. Paszke *et al.*, "Automatic differentiation in pytorch," 2017.

[20] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[22] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[23] A. Toral and V. M. Sánchez-Cartagena, "A multifaceted evaluation of neural versus phrase-based machine translation for 9 language directions," *arXiv preprint arXiv:1701.02901*, 2017.

[24] A. Van den Oord *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, 2016, pp. 4790–4798.

[25] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[26] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[27] B. Xu *et al.*, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[28] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.