Image Caption Generator

Philip Paterson

May 2024

1 Introduction - Problem Statement

In our final project, we program an automatic caption generator that accepts images as inputs and outputs captions associated with said image. The image caption generator utilizes a Transformer model that is trained, tested, and validated on the Flickr8K dataset [1].

In this report, the transformer model and its architecture is described, along with the experimental setup and subsequent results. The results are then analyzed and discussed.

2 Model Description

The transformer model architecture has 3 encoder layers and 3 decoder layers. The encoder is the vision transformer, ViT, available on PyTorch [2]. The decoder is a transformer. The output, or memory, of the encoder was fed as input into the decoder (logits were not needed). The numerical descriptions of the architecture are as follows below:

The encoder architecture is as follows:

- Image Size: 256
- Patch Size: 16x16
- Number of Classes: 1000
- Embedding Dimension: 512
- Number of MLP heads: 8
- Feed forwrad (MLP) dimension: 512
- Depth: 3 (corresponding with number of encoder layers)
- Dropout: 0.1
- Embedding dropout: 0.1

The decoder architecture is as follows:

- Embedding Dimension: 512
- Number of attention heads: 8
- Feed Forward Dimension: 512
- Dropout: 0.1
- Number of layers: 3

The entire transformer model visualization can be seen in Figure 1, with the annotation of the specific numbers, such as number of attention heads, above.

The patch embedding, positional encoding, token embedding, encoder layers, and decoder layers layers will be described in this section.

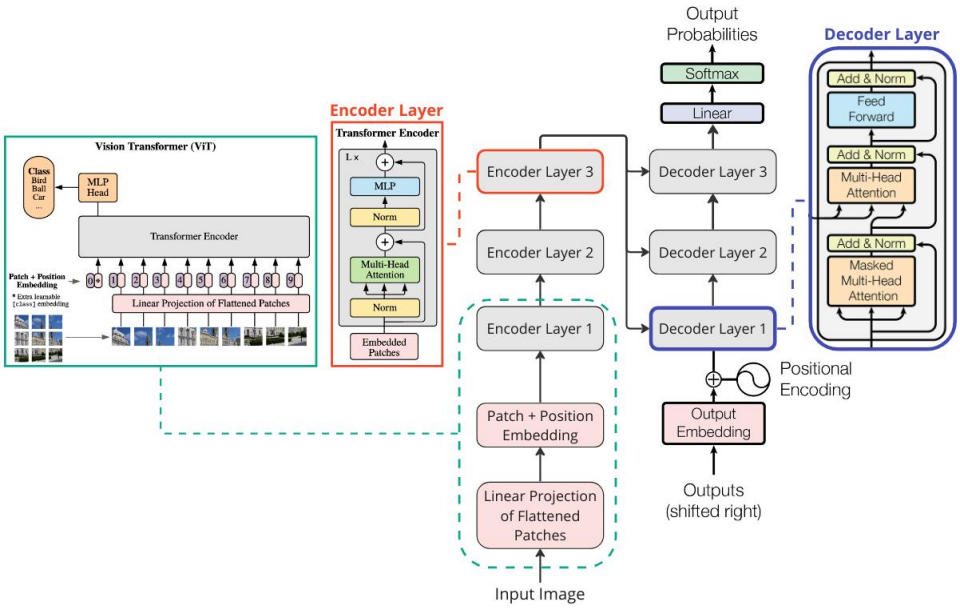

Figure 1: Transformer Architecture Visualization [2, 3]

2.1 Encoder

The transformer encoder is made up of layers of multiheaded self-attention (MSA) and multi-layer perceptron (MLP) blocks that alternate (equations 2 and 3). The MLP has two layers with GELU nonlinearity.

2.1.1 Patch Embedding and Position Encoding

The input images to the encoder are embedded through pach embedding, positional encoding, and token embedding [2].

The input images, of resolution (H, W) and C channels, of the encoder are embedded by reshaping the images into a sequence of 2D patches that is flat, where $X_p \in \mathbb{R}^{N\times(P^2\cdot C)}$ is the sequence and (P, P) is the resolution of the patches. Afterwards, these patches go through standard 1-D positional encoding, and then are token embedded, which is described below for standard input embedding in a transformer.

The equations governing the encoder are shown below, and are discussed at the beginning of this encoder section [2].

$$Z_0 = [x_{\text{class}}; x_p^1 E; x_p^2 E; \ldots; x_p^N E] + E_{\text{pos}}, \quad E \in \mathbb{R}^{(P^2\cdot C)\times D}, \quad E_{\text{pos}} \in \mathbb{R}^{(N+1)\times D} \tag{1}$$

$$z'_\ell = \text{MSA}(\text{LN}(z_{\ell-1})) + z_{\ell-1}, \quad \ell = 1 \ldots L \tag{2}$$

$$z_\ell = \text{MLP}(\text{LN}(z'_\ell)) + z'_\ell, \quad \ell = 1 \ldots L \tag{3}$$

$$y = \text{LN}(z_L^0) \tag{4}$$

Z_0 shows the sequence of embedded patches with the prepended learnable embedding. y is the image representation.

2.2 Input Embedding

The input sequence is embedded by tokenizing the input, performing token embedding, performing position encoding, and summing the results of the token and position encoding together.

2.2.1 Token Embedding

Token embedding produces linear features for each token. Let $\mathbf{T}_x \in \mathbb{R}^{d_t \times d_x}$ be the token embedding, $\mathbf{W}_x^t \in \mathbb{R}^{d_t \times d_i}$ be the token embedding weight matrix, and $\mathbf{X} = [x_1, x_2, ..., x_{d_x}] \in \mathbb{R}^{d_i \times d_x}$ be the input sequence. Thus, $\mathbf{T}_x$ is calculated as follows:

$$\mathbf{T} = \mathbf{W}^t \mathbf{X}$$

2.2.2 Position Encoding

The position of each token is encoded to maintain the ordering. Let $\mathbf{P}_x = [p_1, p_2, ..., p_{d_x}]$ be the position encoding matrix, where

$$p_i = \begin{bmatrix} \sin(\omega_1 \cdot i) \\ \cos(\omega_1 \cdot i) \\ \sin(\omega_2 \cdot i) \\ \cos(\omega_2 \cdot i) \\ \vdots \\ \sin(\omega_{d/2} \cdot i) \\ \cos(\omega_{d/2} \cdot i) \end{bmatrix}_{d \times 1},$$

and $\omega_j = \frac{1}{10000^{2j/d}}$ where $j = 1, 2, ..., \frac{d_p}{2}$.

2.2.3 Encoder/decoder input

The input to the encoder/decoder, $\mathbf{E}_x$, is then calculated as follows:

$$\mathbf{E}_x = \mathbf{T}_x + \mathbf{P}_x \in \mathbb{R}^{d_e \times d_x}$$

2.3 Encoder/Decoder Modules

2.3.1 Attention Layer

The output of the attention layer, $\tilde{Z}_x = [\tilde{Z}_1, x, \tilde{Z}_2, x, ..., \tilde{Z}_{N_H}, x]$, where N_H is the number of attention heads, is calculated where

$$\tilde{\mathbf{Z}}_x = \text{Attention}(\mathbf{E}_x) = \text{attention}(\mathbf{Q}_x, \mathbf{K}_x, \mathbf{V}_x) = \mathbf{V}_x \sigma_M \left(\frac{\mathbf{K}_x^T \mathbf{Q}_x}{\sqrt{d_k}} \right) \in \mathbb{R}^{d_e \times d_x},$$

where

$$\begin{aligned} Q_x^{d_q \times d_x} &= [q_1, q_2, ..., q_{d_x}] = W_x^q E_x, W_x^q \in \mathbb{R}^{d_q \times d_e}, \\ K_x^{d_k \times d_x} &= [k_1, k_2, ..., k_{d_x}] = W_x^k E_x, W_x^k \in \mathbb{R}^{d_k \times d_e}, \\ V_x^{d_v \times d_x} &= [v_1, v_2, ..., v_{d_x}] = W_x^v E_x, W_x^v \in \mathbb{R}^{d_v \times d_e}. \end{aligned}$$

2.3.2 Residual Connection and Normalization Layer

The input to the multi-headed attention layer is then added back to its output, where the resulting sum is subsequently normalized to result in $\mathbf{Z}_x$. The process is described as follows:

$$\mathbf{Z}_x = \text{Norm}(\mathbf{E}_x + \tilde{\mathbf{Z}}_x) \in \mathbb{R}^{d_e \times d_x} \text{ where } \tilde{\mathbf{Z}}_x = \text{attention}(\mathbf{E}_x)$$

$$\text{where } \mathbf{Z}_x[][i] = \gamma \frac{\mathbf{E}_x[][i] + \tilde{\mathbf{Z}}_x[][i] - \mu[i]}{\sigma[i]} + \beta$$

2.3.3 Position-wise Feed-forward Module

The feed-forward network has two fully connected layers, with a ReLU activation function applied between them, along with a residual connection and normalization layer.

The output of the two fully connected layers, $\mathbf{F}'_x$, can be computed as

$$\mathbf{F}'_x = W_{2x}\text{ReLU}(W_{1x}Z_x + b_{1x}) + b_{2x}$$

where $W_{1x} \in \mathbb{R}^{d_f \times d_e}, b_{1x} \in \mathbb{R}^{d_f \times d_x}, W_{2x} \in \mathbb{R}^{d_e \times d_f}, b_{2y} \in \mathbb{R}^{d_e \times d_x}$.

The output for the feed-forward module, $\mathbf{F}_x$, is calculated as follows,

$$\mathbf{F}_x = \text{Norm}(\mathbf{Z}_x + \mathbf{F}'_x) \in \mathbb{R}^{d_e \times d_x}$$

2.4 Decoder

2.4.1 Input Tokenization

The input sequences are tokenized depending on their language. The English language is tokenized using the spaCy tokenizer `en_core_web_sm` [4].

2.4.2 Attention

Referencing Figure 1, the decoder layer shares similar individual component functionality as the encoder layer, with the exception of the masked multi-head attention

$$\mathbf{A}_y = \sigma_M\left(\frac{K_y^T Q_y + M}{\sqrt{d_k}}\right),$$

where $M[j][i] = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{else} \end{cases}$

3 Experimental Setup

Here I will describe the setup of the experiments.

3.1 The Flickr8k Dataset Split

A part of the Flickr8k dataset that has images and five associated english captions was used to train, test, and validate the transformer model [1]. The dataset is split so that the training set contains 6,000 entities, the test set contains 500 entities, and the validation set contains 1,000 sentences.

3.2 Hyper parameters

- Batch size: 32
- Epochs: 20
- Optimizer: Adam Optimizer (`torch.optim.Adam`)
  - Learning Rate = 0.0001
  - betas = (0.9, 0.98)
  - eps = 1e-9

Additionally, when I loaded in the data, I performed

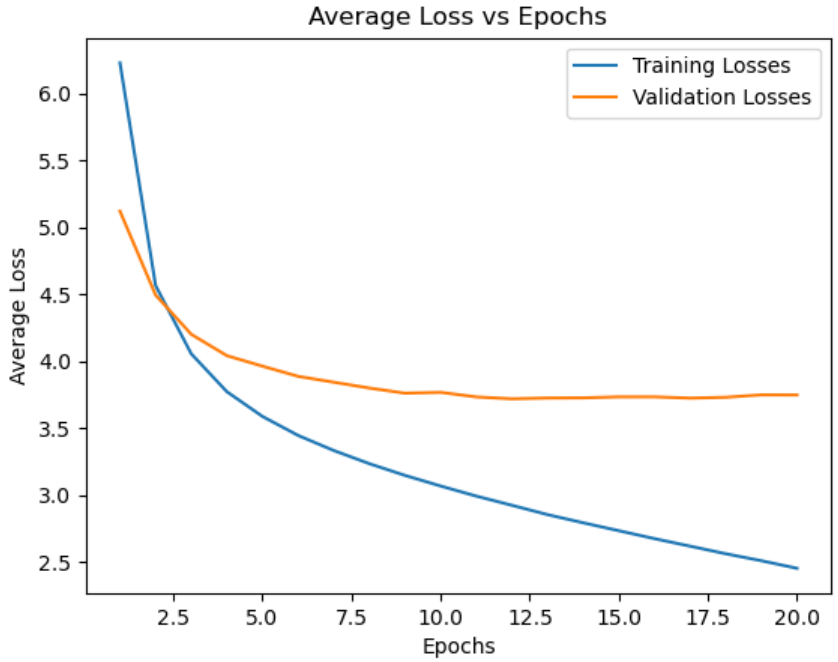

Figure 2: Average Loss vs Epochs

3.3 Weight Initialization

The weight initialization was performed using the PyTorch `nn.init.xavier_uniform_` function [3]. In this PyTorch function, the weights are initialized by sampling from a Xavier uniform distribution, or from $\mathcal{U}(-a, a)$, where

$$a = \text{gain} \times \sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}}.$$

4 Experimental Results

Average BLEU Score on the Testing dataset: 2.24 (out of 100)

The following figures describe my experimental results:

- Average training and validation loss versus epochs: Figure 2
- BLEU Scores vs epochs: Figure 3

5 Analysis

Based on my setup as described in the previous sections, I was able to achieve a BLEU score of about 2.24 on my testing dataset after saving the model with the parameters that had the best BLEU score on the testing dataset.

This BLEU score is unsatisfactory, and there is definitely room for improvement in my model. However, I found that no matter how much I changed the transformer model architecture, I could not achieve a much higher BLEU score than I already did due to the limited resources of my computing machine, and inability to train for long periods of time over long epochs. A good future step would be to train the transformer model on a more powerful machine.

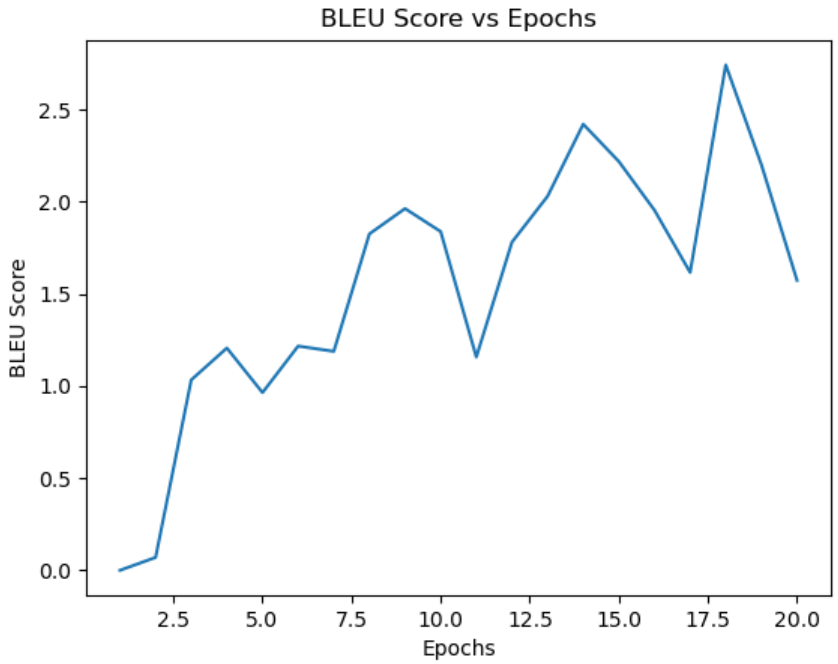

Figure 3: BLEU Score vs Epochs on Testing Dataset

Regarding the predicted captions, a common issue I ran into was that the model consistently predicted the caption to be "a man in a [colored] shirt" whenever a person was involved. However, it could more easily distinguish between a person and a dog, meaning that while the vision transformer could identify if the main subject of the image is a person or animal, it oftentimes could not identify the specifics of said person or animal in their activities and the context of their setting.

5.1 Performance Behavior of Differing Architectures

Here I will describe how the performance behavior differes based on the differing aspects of the model architecture.

- **Image Sizes:** Larger images allowed the model to perform better as they allow for the transformer model to capture more detail from the image. However, these larger images made the model take longer to train and run, and larger images consume more memory.
- **Patch Sizes:** Smaller patch sizes allowed for more accuracy when it came to the predictions because the model is able to pick up on finer details, but at a significant cost of computation, which my machine was limited by. Others findings confirms these results as well [5].
- **Dimension Sizes of Embedding:** Larger embedding dimension sizes allowed for better accuracy of the model, but at a computational cost.
- **Feedforward layers:** Altering the feedforward layers did not achieve much, but may show slight improvements by increasing their size, but not too much as an overly-complex model could lead to over-fitting.
- **Number of encoders:** By increasing the number of encoders, the model's accuracy was able to slightly improve, but at a computational cost. Increasing the number of encoders allowed for my model to capture the more complex details of the images. However, adding too many encoders could cause overfitting.

- **Number of decoders:** By increasing the number of decoders, accuracy was able to slightly improve, but at a computational cost. Increasing the number of decoders allowed for my model to generate more complex sentences. However, adding too many decoders could cause overfitting.
- **Number of attention heads:** Increasing the number of attention heads for my transformer improved on the accuracy, but came at a computational cost, which others findings support [6].

Of course, I would have liked to make my model as sufficiently complex as it needed to be, specifically by focusing on adding more encoders and decoders and increasing the number of attention heads. However, my machine was severely limited in memory and compute time, and could not handle running more complex models, which is why I was not able to achieve a satisfactory BLEU score. Of course, adding complexity must be closely monitored by observing the validation losses and testing accuracy over the epochs, as increasing complexity can lead to more overfitting.

To make more accurate predictions, the transformer architecture would need to be improved, as outlined above, and be run on larger datasets.

6 Conclusion

In conclusion, an automatic image caption generator was created utilizing a transformer model trained on images and their associated English captions from the Flickr8K dataset [1]. The particular transformer architecture utilized has 3 encoder and 3 decoder layers, with the encoder being the Vision Transformer [2] and the decoder being a standard transformer that I defined. The model was able to achieve a BLEU score of 2.24 on the testing dataset after training.

Next steps to build off of this work would be to improve the performance of the transformer model by achieving a higher BLEU score. BLEU scores from 60-70 (out of 100) are regarded as some of the highest that can be achieved, so there is definitely room for improvement [7].

For future steps, one improve the accuracy by experimenting with different transformer model architectures, as discussed in the analysis section, employing more sophisticated methods to avoid overfitting, or training the model off of larger datasets. One could stand to reason that training a larger model with larger datasets would provide significant improvements on the accuracy of the model.

References

[1] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of Artificial Intelligence Research*, vol. 47, p. 853–899, Aug 2013.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[4] [Online]. Available: https://spacy.io/models/en

[5] Synced, "Meet google's flexivit: A flexible vision transformer for all patch sizes," Dec 2022. [Online]. Available: https://syncedreview.com/2022/12/21/meet-googles-flexivit-a-flexible-vision-transformer-for-all-patch-sizes/#:~:text=The%20patch%20size%20informs%20a,size%20they%20were%20trained%20on.

[6] K. Doshi, "Transformers explained visually (part 3): Multi-head attention, deep dive," Jun 2021. [Online]. Available: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853#:~:

text=Multiple%20Attention%20Heads&text=All%20of%20these%20similar%20Attention,and%20nuances%20for%20each%20word.

[7] ——, "Foundations of nlp explained-bleu score and wer metrics," May 2021. [Online]. Available: https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b#:~:text=Bleu%20Scores%20are%20between%200,rarely%20achieve%20a%20perfect%20match.