Encoders for Generateing Latent Space from EEG Channels

Yingqi Ding & Ruyue Hong

May 24, 2019

Contents

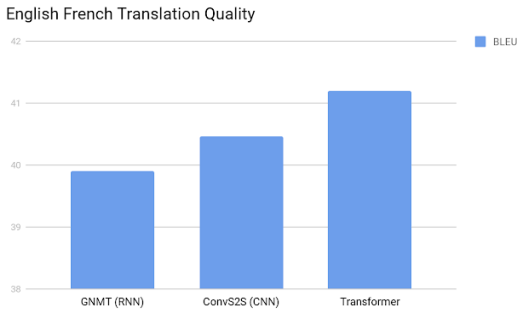
1 Introduction

1.1 Background & Motivation

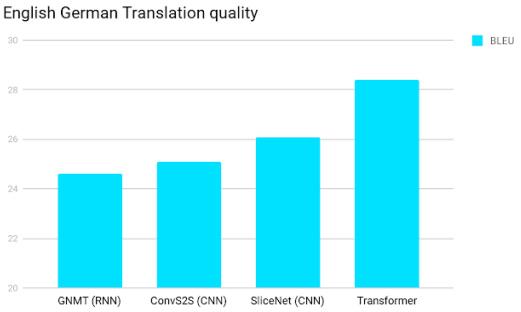
The project aims to study Neurofeedback, i.e. EEG electroencephalogram biofeedback, which is a type of biofeedback that measures brain waves from a computer-based program. We will use sound or visual signals to reorganize or retrain the brain signals, where positive feedback represents desirable brain activities and negative feedback represents undesirable brain activities. By responding to this process, clients learn to regulate and improve their brain function and to alleviate symptoms of various neurological and mental health disorders. During the past few decades, there has been a rapid rise of neurofeedback-based treatments for many neurological, psychological and neurodevelopmental conditions and the research in this direction is rapidly growing. The future applications of the neurofeedback is very wide and promising, as it will be helpful for people with various forms of therapy, which include but not limited to seizure conditions, behavior disorders, attention deficits, autism, ongoing developmental delays, acquired brain injuries, birth trauma, anxiety, depression, post-traumatic stress disorder, stress-related problems, and insomnia or interrupted sleep patterns, as well as those with age-related cognitive loss.

1.2 Abstract

In this project, we focus on a brand new type of model - Transformer Network - which is originally designed and implemented by Google for translating languages. Just like translating English into German, we are translating EEG signals into latent space that can be used to generate images and even videos, using networks such as BigGAN. So what is the Transformer Network and why do we need it? Neural networks, in particular recurrent neural networks (RNNs), are now at the core of the leading approaches to language understanding tasks such as language modeling, machine translation and question answering. In "Attention Is All You Need", Google introduces the Transformer, a novel neural network architecture based on a self-attention mechanism that we believe to be particularly well suited for language understanding. Regarding the language translation task, Transformer outperforms both recurrent and convolutional models on academic English to German and English to French translation benchmarks. On top of higher translation quality, the Transformer requires less computation to train and is a much better fit for modern machine learning hardware, speeding up training by up to an order of magnitude.



(a) BLEU scores (higher is better) of single models on the standard WMT newstest2014 English to French translation benchmark.



(b) BLEU scores (higher is better) of single models on the standard WMT newstest2014 English to German translation benchmark.

With this powerful new Transformer model, we aims to modify its implementation and use the attention mechanism to encode EEG signals. Most of our code are adopted from the Annotated Transformer originally written by people from Harvard NLP research group. We mainly change the input layer and the output layer, as well as omit the embedding and prune some unnecessary operations specific for languages but not for time series sequences. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. End-to-end memory networks are based on a recurrent attention mechanism instead of sequencealigned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks. To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequencealigned RNNs or convolution. In this paper, we are going to introduce every detail about how our modified Transformer for time series sequences works, including its architecture, the magic of attention, mathematical intuition, implementation, etc., as well as represent some preliminary results and suggest some further applications.

2 Dataset

2.1 Linear Dynamic System Dataset

$$x_0 = \mathcal{N}(\mu_{init}, \sigma_{init}) \tag{1}$$

$$x_{t+1} = \mathcal{N}(Ax_t + B\mu_t, \sigma_{state}) \tag{2}$$

$$y_t = \mathcal{N}(Cx_t + D\mu_t, \sigma_{obs}) \tag{3}$$

We use a toy dataset generated by the above formula. And obtain the input data and output data as [1000,300,10] and [1000,300,2] respectively. The three parameters stand for batch size, time stamps, dimension. Below is the visualization of data.

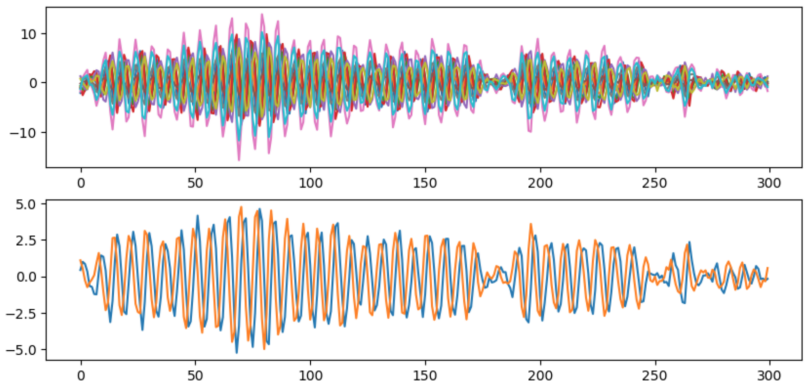


Figure 2: Dynamic Linear System Example.

2.2 EEG Dataset

Our EEG dataset is [4710,1000,32] and output imgae latent space is [4710,60,256].

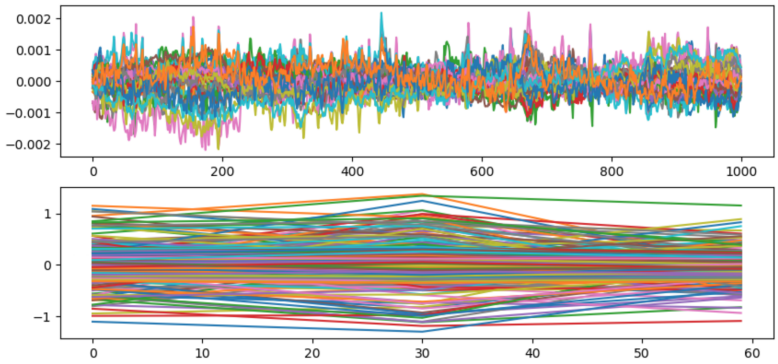


Figure 3: Electroencephalography(EEG).

3 The Transformer [1]

The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder then generates an

output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves, respectively.

3.1 Architecture

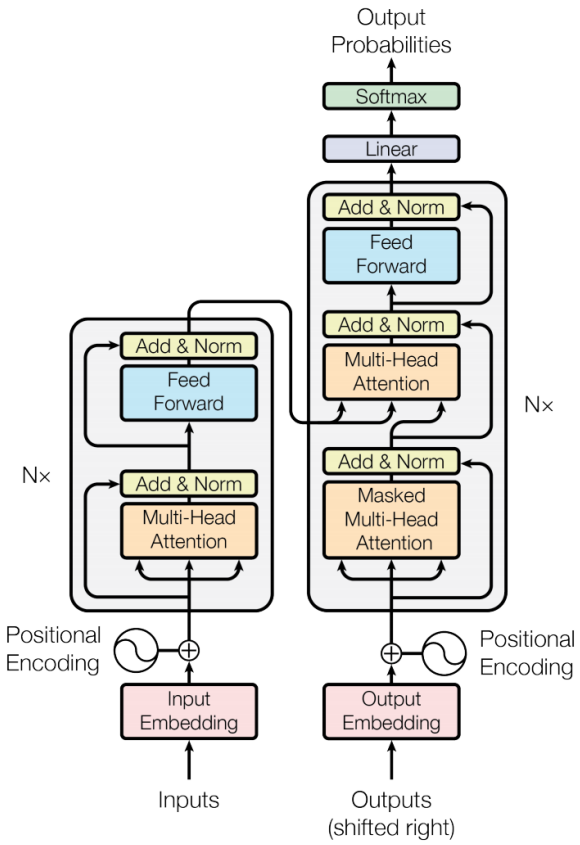


Figure 4: The Transformer - model architecture.

3.2 Encoder Stack

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function

implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{model} = 512$.

3.3 Decoder Stack

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.4 Self-Attention

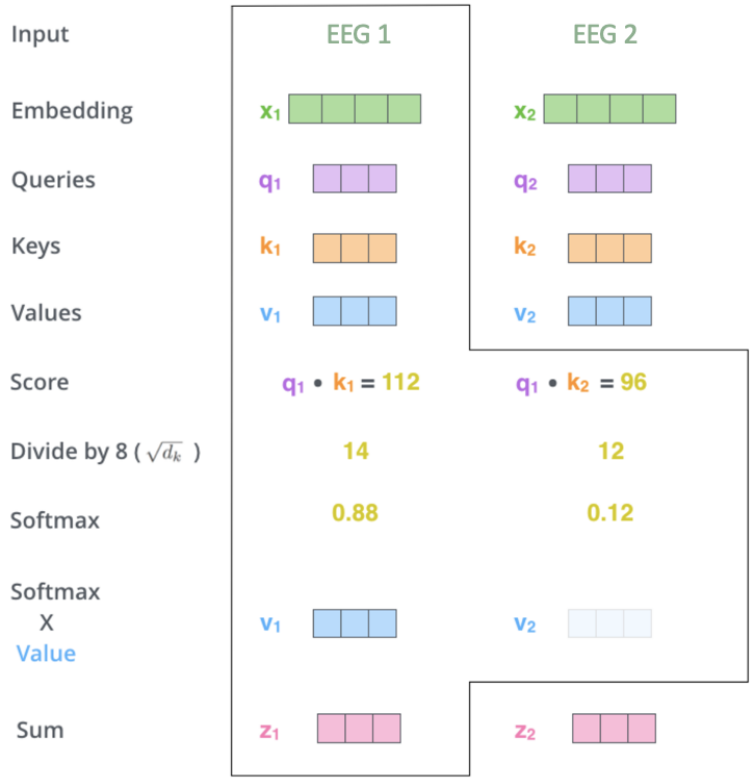


Figure 5: The procedure to calculate output in self-attention layer.[2]

In calculating self-attention, firstly, we need to create three vectors from each of the encoder's input vectors (in this case, the embedding of each EEG channels). So for each channel, we create a Query vector, a Key vector, and a Value vector. The three vectors are created by multiplying the embedding by three trained matrices.

Secondly, we need calculate a score. Say we're calculating the self-attention for the first EEG in this example. We need to score each EEG of the input sequence against this EEG. The score indicates how much we should focus on other parts of the input sequence as we encode a EEG at a certain position. The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.

Thirdly, we need to divide the scores by the square root of the dimension of the key vectors used due to having more stable gradients. Then, we pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1. This softmax score indicates the probability that each EEG will be expressed at this position. It is no doubt that the EEG which is at this position will get the highest probability, but sometimes another relevant EEG can also achive a high score here.

Fourthly, we will multiply the softmax score to each value vector. The reason of this action is to keep the EEG value we want to focus on and ignore irrelevant EEGs by multiply an extremely small the softmax score number like 0.001.

The last step is to sum up all the weighted value vectors, which is the output of the self-attention layer at this position (for the first EEG).

3.5 Multi-head Attention

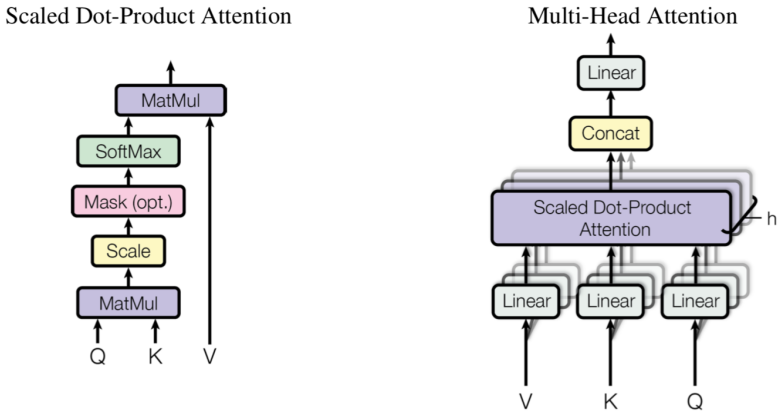


Figure 6: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Then we add a mechanism called "multi-headed" attention to the previous self-attention layer. There are two reasons for this amazing mechanism:

- It helps the model to focus on EEGs on different positions. It is true that in the self-embedding layer, it contains a little bit of the other EEGs. However, it is still dominated by the actual EEG itself.
- With the help of multi-headed attention, we can access multiple sets of Q/K/V weight matrices, which are all initialized randomly. After traning, each matrix can project the output of lower layer into different representation subspace.

3.6 Positional Encoding

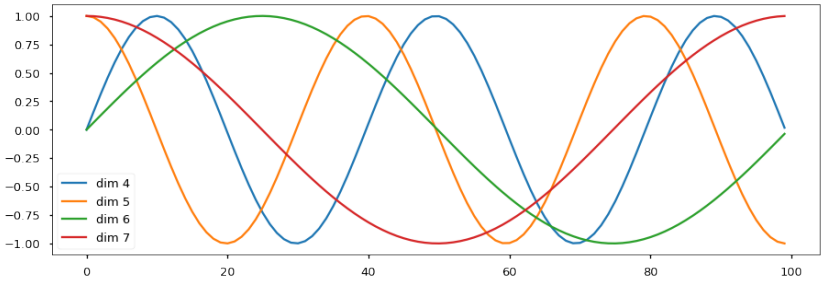


Figure 7: The positional encoding will add in a sine wave based on position. The frequency and offset of the wave is different for each dimension.[3]

Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed. In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a *sinusoid*. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the

model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

3.7 Optimizer & Loss Function

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

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

This 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 $warmup\_steps = 4000$. We used the mean squared error (MSE) as the loss function.

4 Result

4.1 Linear Dynamic System Results

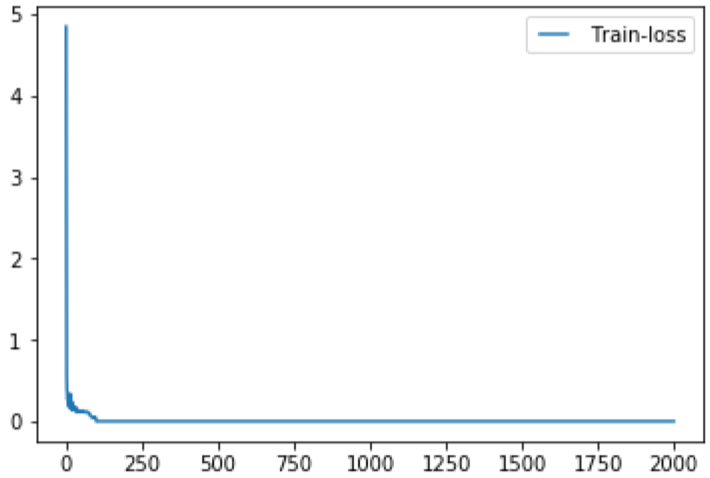


Figure 8: Training loss for 2000 epochs.

The training loss decreases dramatically in the first 100 epochs and converges at early stage. The final loss is very close to 0.

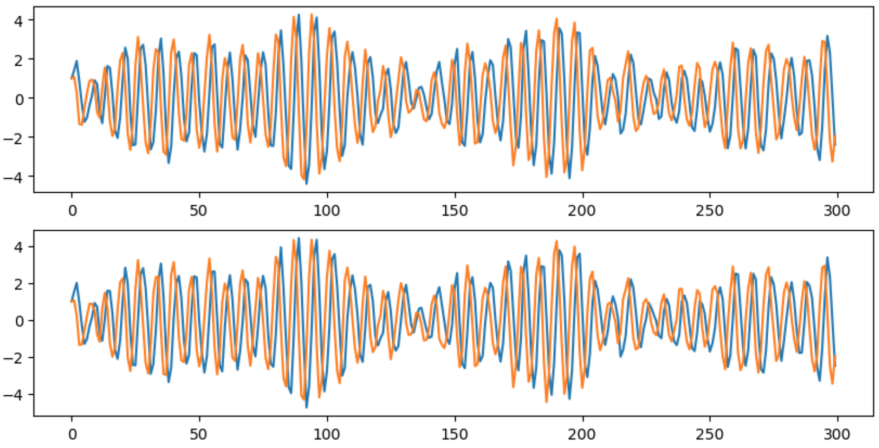


Figure 9: LDS prediction visualization. The upper one is the predicted graph. Another one is the true output .

From the visualization above, we can tell the model really learns the inner pattern within linear dynamic system data.

4.2 EEG Results

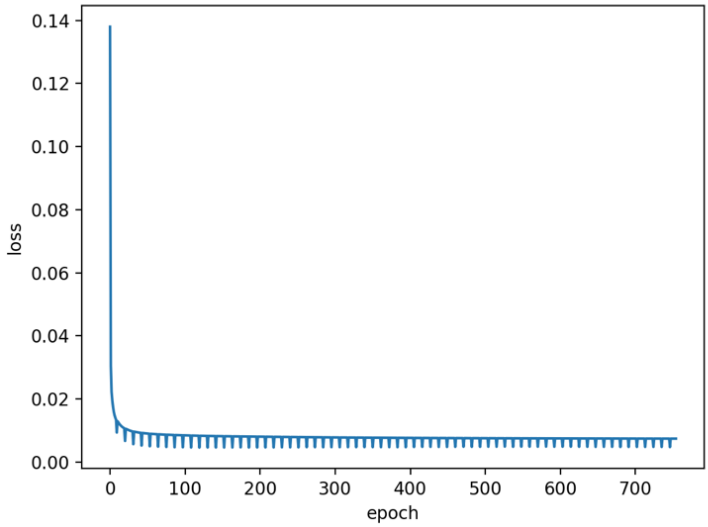


Figure 10: EEG loss for 684 epochs.

The loss graph for EEG is somewhat similar to loss graph for Linear Dynamic System data. The loss decreases quickly in the first 50 epochs. But later, the loss vibrated

around 0.00741 during training process.

The architecture for both datasets are the same. Only 3 encoder layers and 3 decoder layers due to memory limitations and computational time restrictions. The linear dynamic system data is much simpler than EEG signals, thus less layer structure can still make the model learn well in LDS data. But EEG is much more complicated, maybe a deeper layer transformer can help the model learn a better inner pattern.

5 Future Discussion

We are very excited about the future potential of the Transformer and have already started applying it to other problems involving very different inputs and outputs, such as images and video. We hope you'll look forward to seeing what can be done with the Transformer. In this project, we intend to create a new form of Neurofeedback using a trainable end-to-end neural network that is capable of generating meaningful video from EEG signals. As a result, Deep Neurofeedback (DNF) is designed as a new a methodology to overcome limitations of existing classic neurofeedback systems. DNF consists of an EEG headset, an encoder network to encode the EEG signals into the feedback latent-space and a decoder/generator network that maps the latent space signals into a sequence of images and audio. The decoder network is a pre-trained generative model that can generate high-dimensional display signals from a pretrained latent space. Once the encoder is trained, the user will be able to generate images from their EEG and change them dynamically in a closed feedback loop. Although we do not expect high-fidelity reconstruction of images from EEG signals, we hypothesise that this method will allow the users to flexibly control the pixels or audio samples on the screen and achieve much higher level of control than traditional methods. Generating images or audio in real-time requires high levels of parallelization which can be achieved using multiple GPUs.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. pages 5998–6008, 2017.

[2] Jay Alammar. The illustrated transformer. 2018.

[3] Alexander Rush. The annotated transformer. pages 52–60, July 2018.