CS 182/282A: Designing/Visualizing and Understanding Deep Neural Networks

Lecture 18: Transformer Review, Fine-tuning, Embeddings

3 April 2023

Lecturer: Professor Anant Sahai *Scribe: Zeqing Jin, Buyi Zhang*

In this lecture, RNN, attention, and transformer topics covered in the previous lectures were reviewed. New topics including fine-tuning and embeddings were discussed.

1 Transformer from the RNN-based encoder/decoder

A typical RNN-based encoder and decoder architecture is shown in Fig.1. The RNN blocks are stacks of RNNs, which are weight shared across time. However, in the current structure, the state has to carry a lot of information which may not able to focus on important pieces.

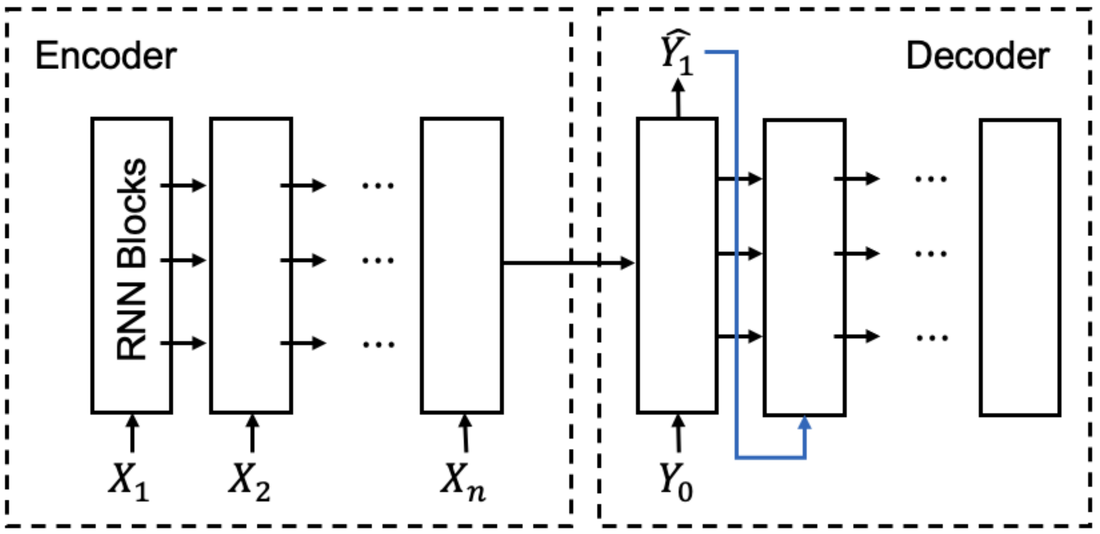

Figure 1: RNN-based encoder and decoder architecture

Therefore, the attention mechanism is introduced in the following steps:

Step 1: Use Attention (Cross-Attention) to give the decoder access to the relevant local context in input.

Step 2: Use Attention (Self-Attention) to give the decoder causal access to output so far.

There is a natural intellectual question that comes up: What now remains is the role of the state in the decoder as we have a way to get access content in the encoder and decoder so far. Why do we need the state at all? The answer to this question leads to the paper *Attention Is All You Need* (`https://tinyurl.com/ykc86uwf`).

Step 3: Remove RNNs from Decoder.

Step 4: Introduce Attention (Self-Attention) in Encoder (Conceptual Step: Keep it causal like Step 2 above). This allows RNNs to be removed.

Therefore, RNN blocks become Transformer blocks with different types of structures shown in Fig. 2. With all the parts, attention, residual connection and MLP, different kinds of interconnections are possible and lead to different structures.

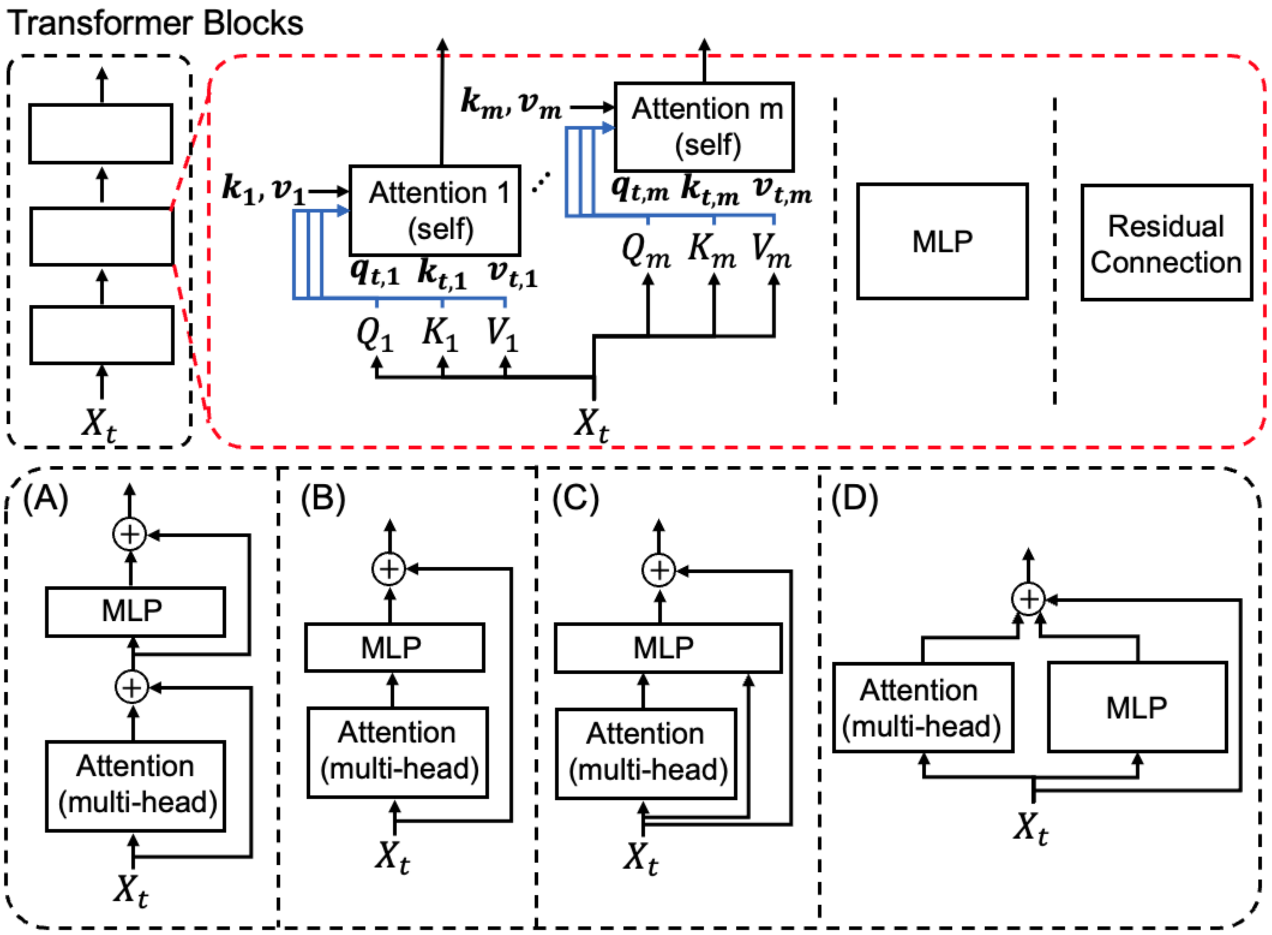

Figure 2: Transformer Blocks and different structures. Bounding box with red dashed line indicates a zoom-in view of the transformer block, which includes multi-head attention, multilayer perceptron (MLP), and residual connection. The sub-figures (A, B, C, D) show examples of the structure constructed by parts in the red bounding box.

In Fig. 2, structures A, C, and D are feasible structures, where structure A is a serial interconnection of multi-head attention block and MLP block, and structure D works like a parallel connection. Structure B is problematic and undesired owing to two reasons: (1)

structure A has two residual connections, where two blocks can learn separately, while structure B only has one residual connection, possibly jumping both blocks and learning nothing. (2) The raw information of X_t is missing after the attention block (the information could be a lot of information in the past without X_t) when fed into the MLP block in structure B. Structure D will have similar effect with structure A since the attention in this layer will have serial connections with the MLP next layer. But structure D is easier to be implemented and convenient in large transformer models.

Step 5: Remove causality from encoder, which is an intrinsic implementation constraint of RNN. It means instead of thinking of $\boldsymbol{k_i}$ and $\boldsymbol{v_i}$ pairs as only coming from the past, allow self-attention to see them even from later on in the sequence (use non-causal Self-Attention).

Aside note: Normalizations can be added after the ResNet connection. Besides, normalization can also be added when generating queries and keys in one recent paper (`https://tinyurl.com/yeyu7d59`).

Model training: The next question is how to train this model. For example, the causal access in Step 2. A naive way is sequentially going through all the decoder RNN blocks (Epoch 1, all the information in the encoder and the first block in the decoder; Epoch 2, all the information in the encoder and the first two blocks in the decoder; etc). However, under this method, all gradients and information are held in the memory of the GPU, which is not ideal and efficient for the utilization of the processors. Therefore, we need parallelism processing during the training process using causal masking. One way to realize causal masking is adding minus infinity to the undesired terms as these terms will be zeroed out after the softmax operation $\sigma(x)_i = e^{x_i} / \sum_{i=1}^{n} e^{x_i}$.

2 Fine-tuning and Embedding

2.1 Fine-tuning

Before we talk about the applications of transformer architectures like BERT (Bidirectional Encoder Representations from Transformers, encoder by itself) and GPT (Generative pre-trained transformers, decoder by itself), one issue of the Transformer and its training is that the transformer architectures are very hungry for training data, as they have a much weaker inductive bias. One way to solve the data shortage is to take advantage of unlabeled data like self-supervision. Another way is amortization across tasks. Many tasks are using the same kind of data, particularly in the field of language processing. So in practice, the transformer is pre-trained on some amalgam of tasks and possibly self-supervision. Then do the adjustment (fine-tuning) for the task we specifically care about.

The basic kind of fine-tuning or adapting a neural network to another task is introduced here. It is called the "Adding a new head" approach, illustrated in Fig. 3A.

1) Decapitate or remove the old head.

2) Add a new head with random initialization.

3) Train new head on task-specific data, while freezing the rest of the net.

The outputs of the neural network which are before the last linear layer to do classification or regression tasks are interpreted as the learned features or embeddings of the input. For example, historically, people decapitated the neural network trained from classification tasks with the ImageNet dataset and trained on medical image classification with limited labeled medical images. As a result, the performance is superior compared to training a new network from the scratch only on the task-specific data. One comment is that you can choose anywhere along the neural network as the end of the embedding and remove the following structure as the old head.

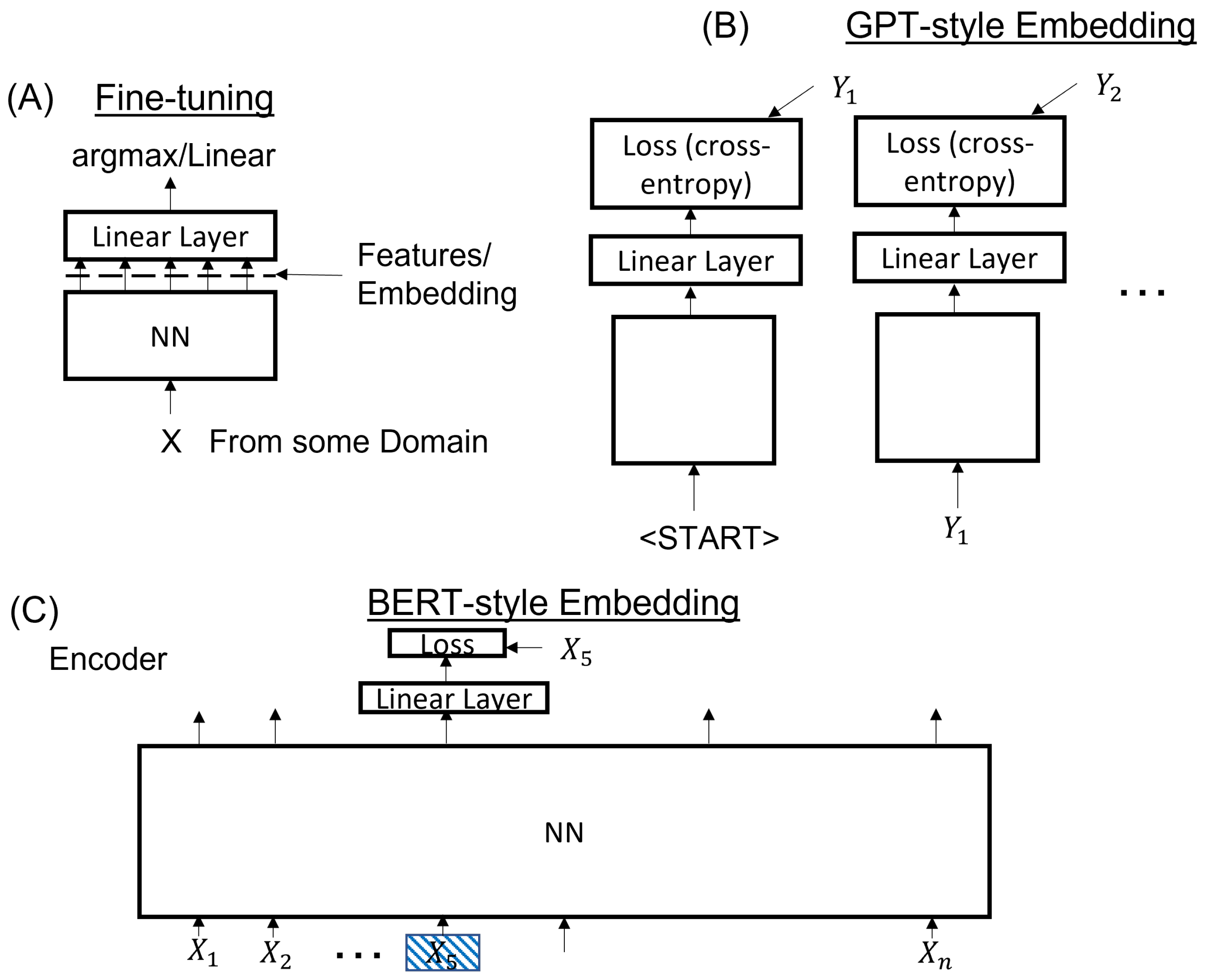

Figure 3: Fine tunning, GPT style Embedding and BERT style Embedding

2.2 Embeddings

Then the next section is about how we can learn an embedding for language models. There are two kind of approaches: GPT style and BERT style.

GPT style is the self-supervise or auto-complete by next word/token prediction like the system-id style. Before talking about the GPT style, we introduce the concept of tokens. We would like to have the inputs as a sequence like Y_1, Y_2, Y_3, Y_4, ... ($Y_i \in \mathbf{R}^d$). For language problems, we need to get vectors from the text i.e. collections of the characters and this is called the problem of tokenization and token embedding. The characters with variable length are turned into a vector with fixed length. This is done by parsing the input string into segments of tokens and followed by a look-up table that could be tuned/learned. The details of tokenization is not included here but they are related to Huffman coding. The tokenization is done at both input and output sides.

The GPT style to find embeddings is basically self-supervised by the next token prediction as shown in Fig. 3B. The deep neural network starts with "start" input and outputs with a loss (cross-entropy) between the first prediction and Y_1. Next, the network has Y_1 as the input and tries to predict the next token Y_2. This follows the same method in system-id problem. One benefit is that all the text could be used as the training data to do the next token prediction. The underlying lower dimensional structure or the embedding of language is presumably learned after training. The attributes of language are distilled from the auto-complete (next token prediction). One comment is that way more features than the training data may be obtained after this but that should be okay in our context. We may assume the regularization works.

Next, the BERT style is briefly introduced here as shown in Fig. 3C. Based on the encoder architecture, BERT style does the data augmentation, such as the noisy and masked auto-encoding we have learned. Some fractions of the input (5-15%) are masked and the task is to reconstruct the embedding. For example, The input like X_5 could be replaced with the mask which is a real vector. The BERT style embedding was a huge language model development when it first came out.