Deep Learning HW3

Yuanhe Guo

February 2024

1 Theory

1.1 Attention

(a) $\boldsymbol{H} = \boldsymbol{V} softargmax_\beta(\boldsymbol{K^T Q}) \in \mathbb{R}^{t \times n}$

(b) The higher β, the more focused the attention will be, vice versa.

(c) When β goes to $+\infty$ (temperature gets cold), the output of $softargmax_\beta$ goes to onehot encoding, thus preserving one value vector v. Here we are referring to hard attention. In fully connected layer, we can use a linear transformation with no bias and weight being identity matrix.

(d) When β goes to 0 (temperature goes hot), the output of $softargmax_\beta$ goes to the average value. Here we are referring to soft attention. In fully connected layer, we can use a linear transformation with no bias and weight being a scalar matrix.

(e) First let's write out the $\hat{\boldsymbol{H}}$ after perturbation.
$\hat{\boldsymbol{H}} = \boldsymbol{V} softargmax_\beta([\boldsymbol{k_1}^T\boldsymbol{Q}, ...(\boldsymbol{k_i} + \boldsymbol{\epsilon})^T\boldsymbol{Q}, ...\boldsymbol{k_m}^T\boldsymbol{Q}])$
Given $\mathbb{E}(\boldsymbol{\epsilon}) = 0$, we know $\mathbb{E}[(\boldsymbol{k_i} + \boldsymbol{\epsilon})^T\boldsymbol{Q}] = \boldsymbol{k_i}^T\boldsymbol{Q}$. Therefore, a small perturbation to one of the $\boldsymbol{k}$ won't change $\boldsymbol{H}$.

(f) Suppose we change the i-th $\boldsymbol{k}$. Now the new $\hat{\boldsymbol{H}}$ is
$\hat{\boldsymbol{H}} = \boldsymbol{V} softargmax_\beta([\boldsymbol{k_1}^T\boldsymbol{Q}, ...a\boldsymbol{k_i}^T\boldsymbol{Q}, ...\boldsymbol{k_m}^T\boldsymbol{Q}])$
Let's consider the $softargmax_\beta$ function:

$$softargmax_\beta(\boldsymbol{K}^T\boldsymbol{Q}) \begin{cases} \frac{exp(\beta \boldsymbol{k_t}^T\boldsymbol{Q})}{\Sigma_{j=1}^{i-1} exp(\beta \boldsymbol{k_j}^T\boldsymbol{Q}) + \Sigma_{j=i+1}^{m} exp(\beta \boldsymbol{k_j}^T\boldsymbol{Q}) + exp(\beta a \boldsymbol{k_i}^T\boldsymbol{Q})} & \text{if } t \neq i \\ \frac{exp(\beta a \boldsymbol{k_t}^T\boldsymbol{Q})}{\Sigma_{j=1}^{i-1} exp(\beta \boldsymbol{k_j}^T\boldsymbol{Q}) + \Sigma_{j=i+1}^{m} exp(\beta \boldsymbol{k_j}^T\boldsymbol{Q}) + exp(\beta a \boldsymbol{k_i}^T\boldsymbol{Q})} & \text{if } t = i \end{cases}$$

As a result, the $\boldsymbol{h}$ value for the i-th k becomes larger, while all other values become smaller.

1.2 Multi-headed Attention

(a) Let the concatted matrix of attentions to be $\boldsymbol{S}$.

$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{V}}^{\boldsymbol{1}} softargmax_{\beta}((\boldsymbol{K}\boldsymbol{W}_{\boldsymbol{K}}^{\boldsymbol{1}})^T \boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{Q}}^{\boldsymbol{1}}) \\ ... \\ \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{V}}^{\boldsymbol{h}} softargmax_{\beta}((\boldsymbol{K}\boldsymbol{W}_{\boldsymbol{K}}^{\boldsymbol{h}})^T \boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{Q}}^{\boldsymbol{h}}) \end{pmatrix}$$

Then we apply an linear transformation to get $\boldsymbol{H}$

$$\boldsymbol{H} = \boldsymbol{S}\boldsymbol{W}^{\boldsymbol{0}}$$

(b) Both conv1d and multi-headed attention applies weight to input in order to project it into a new matrix, then connects the result to a linear layer.

1.3 Self Attention

(a) Consider the i-th head. $\boldsymbol{Q}^{\boldsymbol{i}} = \boldsymbol{C}\boldsymbol{W}_{\boldsymbol{Q}}^{\boldsymbol{i}}, \boldsymbol{K}^{\boldsymbol{i}} = \boldsymbol{C}\boldsymbol{W}_{\boldsymbol{K}}^{\boldsymbol{i}}, \boldsymbol{V}^{\boldsymbol{i}} = \boldsymbol{C}\boldsymbol{V}_{\boldsymbol{K}}^{\boldsymbol{i}}$.
$\boldsymbol{S}^{\boldsymbol{i}} = Attention(\boldsymbol{Q}^{\boldsymbol{i}}, \boldsymbol{K}^{\boldsymbol{i}}, \boldsymbol{V}^{\boldsymbol{i}}) = \boldsymbol{V}^{\boldsymbol{i}} softargmax_{\beta}((\boldsymbol{K}^{\boldsymbol{i}})^T \boldsymbol{Q}^{\boldsymbol{i}})$
$\boldsymbol{H} = \boldsymbol{S}\boldsymbol{W}^{\boldsymbol{0}}$

(b) Since self-attention handles data simultaneously, we need positional encoding when the input is sequential. For example, the position of each word in a text input matters, so we want to add positional encoding.

(c) If $\boldsymbol{Q} = \boldsymbol{K}$, and $\boldsymbol{Q}$ is an orthogonal matrix, then $\boldsymbol{K}^T\boldsymbol{Q} = \boldsymbol{Q}^{-1}\boldsymbol{Q} = \boldsymbol{I}$. Now the attention behaves like an identity layer.

(d) When the result of $softargmax_{\beta}(\boldsymbol{K}^T\boldsymbol{Q})$ is a diagonal matrix, then the self attention just does linear projection to each location. The proper name for "running" linear layer should be "1Dconvolution", which applies linear projection one element at a time.

(e) Let's consider a special situation. The attention score calculated by $softargmax_{\beta}(\boldsymbol{K}^T\boldsymbol{Q}$ for each location (as well as its relative neighbors) are the same. In this way, each self attention head acts like a kernel in a convolution layer, which applies a shared weight to one location and its neighbors.

1.4 Transformer

(a) The way of handling input sequence is different between Transformer and previous models. Previous models knows the neighbors of current input by using the latent of previous inputs, resulting in sequential processing. Transformer takes in sequence in parallel, and gets to know the neighbors of one element by using attention mechanism, while the position information is preserved by positional encoding. Compared with previous models, Transformer is able to handle a large input sequence efficiently.

(b) Self-attention is an attention mechanism that relates different positions of an input sequence to output a representation. It has three significant usages. First, self-attention reduces computational complexity per layer when compared with recurrent layers. Second, self-attention can parallelized a large amount of computation. Third, self-attention enables the network to better learn long-range dependencies.

(c) Multi-head attention learns to linearly projects queries, keys values h times, and apply attention to each version of queries, keys and values to get h heads. After that, all heads are concat together. The benefit is that multi-head attention allows the model to learn from different representations in parallel.

(d) Feed-forward networks used in transformer are position-wise. One feed-forward network consists of two linear transformations and a ReLU in the middle. This enables the model to utilize different parameters across layers at the same position.

(e) Layer normalization normalize the layer with means and variance of all hidden units in that layer. This technique reduces the undesired correlated changes caused by the output of one layer. It also overcomes the constraints on mini-batch size of batch normalization. Transformer architecture applies layer normalization after residual connection in both "encoder" and "decoder".

1.5 Vision Transformer

(a) Vision Transformer turns image into sequence by embedding image patches, while CNN turns image into latent by weight sharing. In ViT, the process of patching images is a convolution layer whose kernel size and stride is the same.

(b) There are three main differences. First, ViT takes images as input, and Transformer takes words as input. Second, ViT has "encoder" but doesn't have "decoder", so there's only one part of attention in ViT model. Third, the output of ViT is the predicted class, which is in different modality of the input.

(c) In ViT, positional embeddings are used for preserving locality information in the image. Positional embeddings in ViT are learned 1D embeddings, while in Transformer, the embedding is a combination of sine and cosine functions.

(d) ViT connects a one hidden layer MLP to the output of multi-head attention as classification head during pre-training. During fine-tuning, the classification head is just a one layer MLP.