Deep learning HW 4

wbg231

October 2023

1 theory

1.1 Attention

This question tests your intuitive understanding of attention and its property.

(a) (1pt) Given queries $Q \in \mathbb{R}^{d\times n}, K \in \mathbb{R}^{d\times m}$ and $V \in \mathbb{R}^{t\times M}$ what is the output H of the standard dot-product attention? (You can use the softargmaxβ function directly. It is applied to the column of each matrix).

- we can calculate our attention matrix $A \in \mathbb{R}^{m\times n}$ as $A = softargmax_\beta(K^T Q)$
- then find out the output $H \in \mathbb{R}^{t\times n}$ as $H = VA$

(b) (2pts) Explain how the scale β influence the output of the attention? And what β is conveniently to use?

- β controls how uniformly spread the attention is. If β is small, the attention is more spread. If β is large, the attention is more concentrated.
- this can be seen by the softargmax function

$$sofrargnax_\beta(z)_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{d} e^{-\beta z_j}}$$

- in this context we like to set $\beta = \frac{1}{\sqrt{d}}$ where $Q \in \mathbb{R}^{d\times t}$ and $K \in \mathbb{R}^{d\times n}$
- we do this since we are taking the dot product $K^T Q$ whose scale will grow at a rate of $\sqrt{d}$ and we want to keep the attention matrix A from growing too large
- at $\beta = 1$ we get the argmin function
- at $\beta = 0$ we get a uniform distribution

(c) One advantage of the attention operation is that it is really easy to preserve a value vector $\mathbf{v}$ to the output $\mathbf{h}$. Explain in what situation, the outputs preserves the value vectors. Also, what should the scale β be if we just

want the attention operation to preserve value vectors. Which of the four types of attention we are referring to? How can this be done when using fully connected architectures?

- we know that our output is $H = VA$ where $A = sofrargnax_\beta(K^T Q)$
- $H = VA = V$ when $\beta \to 1$ this can be seen as $A_i = softargmax(K^T Q)_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{d} e^{-\beta z_j}}$
- that means that our matrix A will have columns and rows that all sum to one and thus our output $h \in H \to h \in V \quad \forall h \in H$
- this means our network is only paying attention to a few samples
- we can do this in a fully concentrated network by having a space weight matrix W
- this is hard attention

(d) On the other hand, the attention operation can also dilute different value vectors $\mathbf{v}$ to generate new output $\mathbf{h}$. Explain in what situation the outputs is spread version of the value vectors. Also, what should the scale β be if we just want the attention operation to diffuse as much as possible. Which of the four types of attention we are referring to? How can this be done when using fully connected architectures?

- on the other hand as $\beta \to 0$ we can see that $H_{i,j} = softargmax_0(K^t Q)_{i,j} = \frac{1}{d} \quad \forall i, j$
- we can do this with a fully concentrated network by having a dense weight matrix W with the same value for all weights
- this is soft attention

(e) If we have a small perturbation to one of the $\mathbf{k}$ (you could assume the perturbation is a zero-mean Gaussian with small variance, so the new $\hat{\mathbf{k}} = \mathbf{k} + \epsilon$), how will the output of the $\mathbf{H}$ change?

- call $K^{'} = [k_1^t, \cdots, k_a^t + \epsilon, \cdots, k_m^t]$ then we can write

  $Z^{'} = softargmax_\beta(K'^T Q) = softargmax_\beta([k_1^T Q, \cdots k_a^T Q + \epsilon Q \cdots k_m^T Q])$

- meaning that we can write $Z_i^{'} = \begin{cases} \frac{e^{-\beta z_i}}{\sum_{j \neq a} e^{-\beta z_j} + e^{-\beta z_a - \beta \epsilon Q}} & \text{if } i \neq a \\ \frac{e^{-\beta z_a - \beta \epsilon}}{\sum_{j \neq a} e^{-\beta z_j} + e^{-\beta z_a - \beta \epsilon Q}} & \text{if } i = a \end{cases} \in \mathbb{R}^{m \times n}$
- then we have $H^{'} = VZ'$ meaning that each value V was scaled proportionally to the dot product of the perturbation and the query matrix
- in this case regardless of the scale of z_a it's attention is scaled down proportionally to the perturbation

(f) If we have a large perturbation that it elongates one key so the $\hat{k} = \alpha k$ for $\alpha > 1$, how will the output of the H change?

- call $K^{'} = [k_1^t, \cdots, \alpha k_a^t, \cdots, k_m^t]$ then we can write

  $Z^{'} = softargmax_\beta(K^{'T}Q) = softargmax_\beta([k_1^TQ, \cdots \alpha k_a^TQ \cdots k_m^TQ])$
- meaning that we can write $Z_i^{'} = \begin{cases} \frac{e^{-\beta z_i}}{\sum_{j \neq a} e^{-\beta z_j} + e^{-\alpha\beta z_a}} & \text{if } i \neq a \\ \frac{e^{-\alpha\beta z_a}}{\sum_{j \neq a} e^{-\beta z_j} + e^{-\beta\alpha z_a}} & \text{if } i = a \end{cases} \in \mathbb{R}^{m \times n}$
- so in other words the attention paid to the perturbed key is scaled down by α
- and the attention to all other keys are scaled up by $\frac{\alpha}{m-1}$
- in this case the magnitude of perturbation is effected by the scale of z_a

1.2 Multi-headed Attention

This question tests your intuitive understanding of Multi-headed Attention and its property

(a) Given queries $Q \in \mathbb{R}^{d \times n}, K \in \mathbb{R}^{d \times m}$ and $V \in \mathbb{R}^{t \times m}$, what is the output H of the standard multi-headed scaled dot-product attention? Assume we have h heads.

- yeah so our intermediate outputs can be written as $O = \begin{pmatrix} O_1 \\ \cdots \\ O_h \end{pmatrix}$
- where $O_i = (VW_i^V)softargmax_\beta((KW_i^K)^T(QW_i^Q))$
- then our final output is just a linear combination of thoe intermediate outputs that is $H = OW_i^O$

(b) Is there anything similar to multi-headed attention for convolutional networks? Explain why do you think they are similar. (Hint: read the conv1d document from PyTorch: link)

- yeah it is similar to using multiple kernels in CNNs. They are similar as both are using multiple linear projections to extract different features from the input then ultimately concatenating them into one output

1.3 Self Attention

(a) Given an input $C \in \mathbb{R}^{e \times n}$, what is the queries Q, the keys K and the values V and the output H of the standard multi-headed scaled dot-product self-attention? Assume we have h heads. (You can name and define the weight matrices by yourself)

- if this is the case we can write $Q = \begin{pmatrix} Q_1 \\ \cdots \\ Q_h \end{pmatrix} = \begin{pmatrix} W_1^Q \\ \cdots \\ W_h^Q \end{pmatrix} C \in \mathbb{R}^{d'h \times n}$ and each $Q_i \in \mathbb{R}^{d'h \times e}$
- we can write $K = \begin{pmatrix} K_1 \\ \cdots \\ K_h \end{pmatrix} = \begin{pmatrix} W_1^K \\ \cdots \\ W_h^K \end{pmatrix} C \in \mathbb{R}^{d'h \times n}$ and each $K_i \in \mathbb{R}^{d'h \times e}$
- and values as $V = \begin{pmatrix} V_1 \\ \cdots \\ V_h \end{pmatrix} = \begin{pmatrix} W_1^V \\ \cdots \\ W_h^V \end{pmatrix} C \in \mathbb{R}^{d''h \times n}$ and each $V_i \in \mathbb{R}^{d''h \times e}$
- so finally this gives us outputs of the form $O_i = (VW_i^V) softargmax_\beta((KW_i^K)^T(QW_i^Q))$ meaning that $O = VA$ where $A_i = softargmax_\beta(K_i^T Q_i)$
- then finally we have $H = W^O O$ where $W^0 \in \mathbb{R}^{e \times hd}$ and $O \in \mathbb{R}^{hd \times n}$ meaning that $H \in \mathbb{R}^{e \times n}$

(b) Explain when we need the positional encoding for self-attention and when we don't. (You can read about it at link)

- we want positional encoding when there is some ordered structure in our data that we want to keep track of this could be the order of words in a sentence or the order of pixels in an image
- if the order does not matter as might be the case with something like medical records where we only have one observation of each person we do not need positional encoding

(c) Show us one situation that the self attention layer behaves like an identity layer or permutation layer.

- an example of this could be if we are setting our beta quite high, and thus we are getting our A matrix to be an argmax, in this case we are only permuting our data

(d) Show us one situation that the self attention layer behaves like an "running" linear layer (it applies the linear projection to each location). What is the proper name for a "running" linear layer.

- if $\beta = 0$ then we are uniformly paying attention to all elements and thus have this type of behavior

(e) Show us one situation that the self attention layer behaves like an convolution layer with a kernel larger than 1. You can assume we use positional encoding

- if our β is neither super large nor super small then we are taking a weighted average of our data and thus have this type of behavior

1.4 Transformer

Read the original paper on the Transformer model: "Attention is All You Need" by Vaswani et al. (2017).

(a) Explain the primary differences between the Transformer architecture and previous sequence-to-sequence models (such as RNNs and LSTMs).

- Transformers do not use any recurrence or convolutional layers

(b) Explain the concept of self-attention and its importance in the Transformer model

- self attention is as a mechanism for relating information within a single sequence of input x
- in the transformer model self attention is used in both the encoding and decoding steps
- in the encoding step all keys values and queries are passed through the encoder, with the last layer of the encoder outputing a sequence of encodings repersenting all keys, values and queries. then passing that through a self attention layer allows the model to learn dependencies between the different elements of the input sequence (key, values and queries)
- similarly we use a self attention layer on the output of the decoder step allowing us to learn global dependencies of the outptus

(c) Explain the feed-forward neural networks used in the model and their purpose.

- there is a feedward neural network in the encoder and decoder step, which computes

$$ffn(x) = max(0, xW_1 + b_1)W_2 + b_2$$

- the parameters ie weights and biases are the same at each position of the sequence but vary at each layer.
- this allows the model to have non-linearities varying at each layer

(d) Describe the multi-head attention mechanism and its benefits.

- multi headed attention means that we preform the attention function in parallel on h triplets of (Q, K, V) producing h output values $H_1 \cdots H_n$ which we again linearly combine with some learned weight matrix W^0
- multi-headed attention allows our model to learn from multiple from the representations of dirent subspaces at the same time, that is the model is able to learn from seperate pieces of information at the same time potentially gaining more robust representations

(e) Describe the layer normalization technique and its use in the Transformer architecture.

- at each sublayer of the encoder and decoder they have a skip connections with a layer normalization function. This normalization functions normalizes (that is scales the mean and variance) of our previous layer. this is implemented as

$$h^l = layernorm(h^{l-1} + f_0^{l-1}(h^{l-1}))$$

  where h^i is the hidden representation of the ith sublayer, and f_θ^l is our models lth layer
- this mitigates shifts in scale throughout our model and improves training speed

1.5 Vision Transformer

Read the paper on the Transformer model: "An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale".

(a) What is the key difference between the Vision Transformer (ViT) and traditional convolutional neural networks (CNNs) in terms of handling input images? Can you spot a convolution layer in the ViT architecture?

- unlike standard CNNs the Vit architecture does not use repeated convolutional layers to extract features from the image
- instead it breaks the immage into fixed patches and takes a linear projection on them (which is a convolution) then passes the resulting representations through a transformer encoder

(b) Explain the differences between the Vision Transformer and the Transformer introduced in the original paper.

- the vision transformer does not have a decoder step and instead just passes the learned encodings through an mlp to produce the output
- the vision transformer also has the fist step being a convolution which is not in the original transformer
-

(c) What is the role of positional embeddings in the Vision Transformer model, and how do they differ from positional encodings used in the original Transformer architecture?

- in the original transformer architecture uses positional embeddings that are sums of sinusoids based on position in the sequence
- the vision transfer just uses spacial the position of the patch in the image as the positional embedding

(d) How does the Vision Transformer model generate the final classification output? Describe the process and components involved in this step.

- when passing data to the encoder architecture, they add to there fixed size patches $X_1 \cdots X_n$ a classification parameter X_0 then pass that all through the encoder.
- then finally at the last later of the encoder they have $Z_0^L \cdots Z_n^L$ and make there final prediction as $y = MLP(layernorm(Z_0^L))$ (that is the layer norm of the final layers classification parameters passed through an mlp)