ML TA hours

HW10

Colab experiment

2024.12.03

This week's colab homework

1. Self Attention

2. Multihead Self-Attention

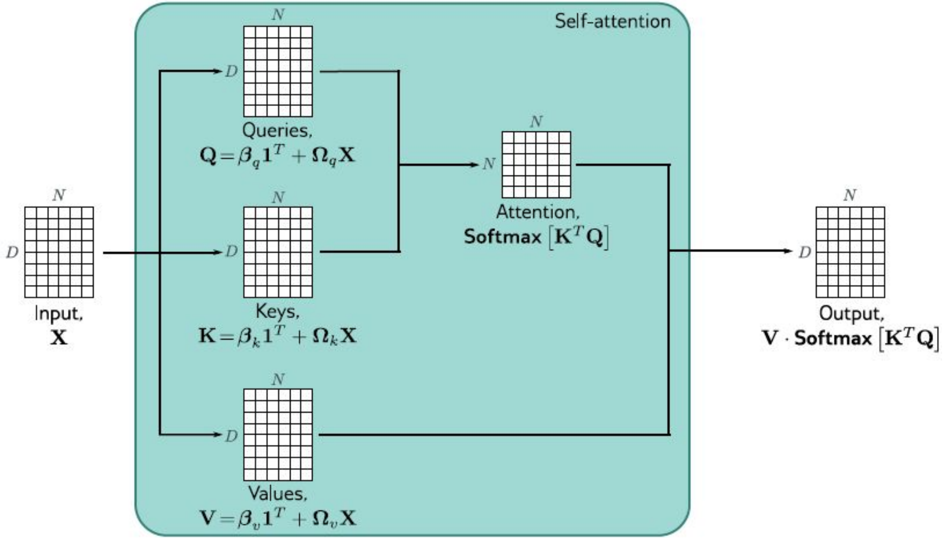
3. Transformer

Part 1

1. Self Attention

2. Multihead Self-Attention

3. Transformer



TODO - compute the queries, keys, and values for each input

```python
# Make three lists to store queries, keys, and values
all_queries = []
all_keys = []
all_values = []
# For every input
for x in all_x:
    # TODO -- compute the keys, queries and values.
    # Replace these three lines
    query = np.ones_like(x)
    key = np.ones_like(x)
    value = np.ones_like(x)


    all_queries.append(query)
    all_keys.append(key)
    all_values.append(value)
```

Hint:

$$\mathbf{v}_m = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_m,$$
$$\mathbf{q}_n = \boldsymbol{\beta}_q + \boldsymbol{\Omega}_q \mathbf{x}_n$$
$$\mathbf{k}_m = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{x}_m,$$

TODO - complete the softmax function

```
def softmax(items_in):

    # TODO Compute the elements of items_out
    # Replace this line
    items_out = items_in.copy()


    return items_out ;
```

Hint:

$$
\begin{aligned}
a[\mathbf{x}_m, \mathbf{x}_n] &= \mathrm{softmax}_m\left[\mathbf{k}_\bullet^T \mathbf{q}_n\right] \\
&= \frac{\exp\left[\mathbf{k}_m^T \mathbf{q}_n\right]}{\sum_{m'=1}^{N} \exp\left[\mathbf{k}_{m'}^T \mathbf{q}_n\right]},
\end{aligned}
$$

TODO - compute the self attention values

```
# Create emptymlist for output
all_x_prime = []

# For each output
for n in range(N):
    # Create list for dot products of query N with all keys
    all_km_qn = []
    # Compute the dot products
    for key in all_keys:
        # TODO -- compute the appropriate dot product
        # Replace this line
        dot_product = 1


        # Store dot product
        all_km_qn.append(dot_product)

    # Compute dot product
    attention = softmax(all_km_qn)
    # Print result (should be positive sum to one)
    print("Attentions for output ", n)
    print(attention)

    # TODO: Compute a weighted sum of all of the values according to the attention
    # (equation 12.3)
    # Replace this line
    x_prime = np.zeros((D,1))


    all_x_prime.append(x_prime)
```

Hint:

$$\mathbf{sa}_n[\mathbf{x}_1,\ldots,\mathbf{x}_N] = \sum_{m=1}^{N} a[\mathbf{x}_m,\mathbf{x}_n]\mathbf{v}_m.$$

The result will be like this (You can check if your answer is correct)

```
# Print out true values to check you have it correct
print("x_prime_0_calculated:", all_x_prime[0].transpose())
print("x_prime_0_true: [[ 0.94744244 -0.24348429 -0.91310441 -0.44522983]]")
print("x_prime_1_calculated:", all_x_prime[1].transpose())
print("x_prime_1_true: [[ 1.64201168 -0.08470004  4.02764044  2.18690791]]")
print("x_prime_2_calculated:", all_x_prime[2].transpose())
print("x_prime_2_true: [[ 1.61949281 -0.06641533  3.96863308  2.15858316]]")
```

TODO - compute self attention in matrix form

```
# Now let's compute self attention in matrix form
def self_attention(X,omega_v, omega_q, omega_k, beta_v, beta_q, beta_k):

    # TODO -- Write this function
    # 1. Compute queries, keys, and values
    # 2. Compute dot products
    # 3. Apply softmax to calculate attentions
    # 4. Weight values by attentions
    # Replace this line
    X_prime = np.zeros_like(X);



    return X_prime
```

Hint:

$$
\begin{aligned}
\mathbf{V}[\mathbf{X}] &= \boldsymbol{\beta}_v \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_v \mathbf{X} \\
\mathbf{Q}[\mathbf{X}] &= \boldsymbol{\beta}_q \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_q \mathbf{X} \\
\mathbf{K}[\mathbf{X}] &= \boldsymbol{\beta}_k \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_k \mathbf{X},
\end{aligned}
$$

$$\mathbf{Sa}[\mathbf{X}] = \mathbf{V}[\mathbf{X}] \cdot \mathbf{Softmax}\Big[\mathbf{K}[\mathbf{X}]^T \mathbf{Q}[\mathbf{X}]\Big]$$

The result will be like this (You can check if your answer is correct)

The answer should be the same as before.

```
[[ 0.94744244  1.64201168  1.61949281]
 [-0.24348429 -0.08470004 -0.06641533]
 [-0.91310441  4.02764044  3.96863308]
 [-0.44522983  2.18690791  2.15858316]]
```

Summarize what you need to do in the part 1

1. Complete all TODO parts as directed

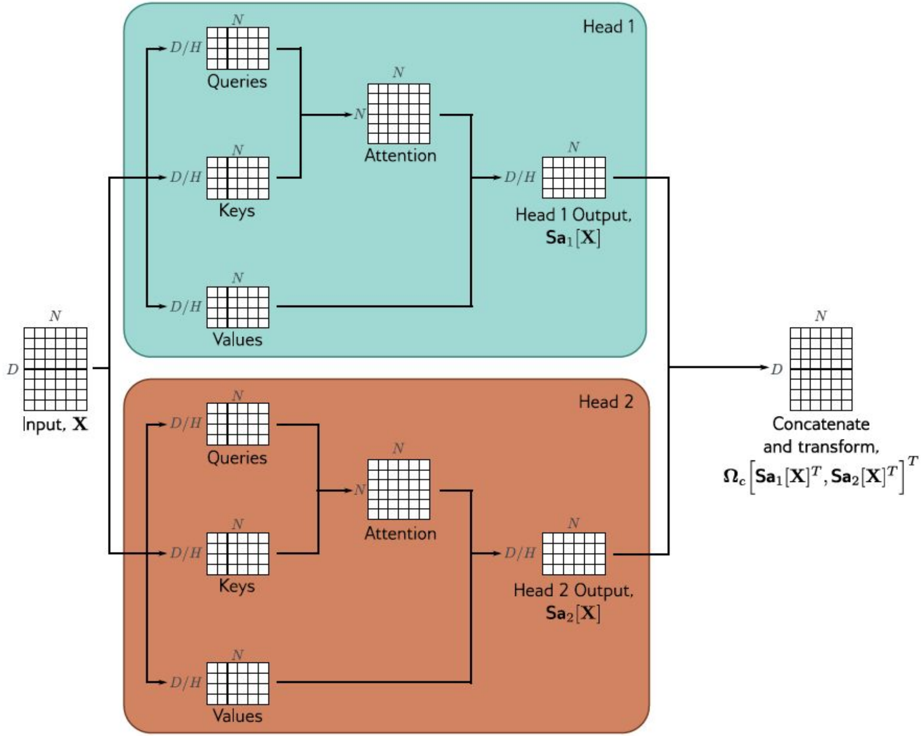
Part 2

1. Self Attention

2. Multihead Self-Attention

3. Transformer

(two heads)



TODO - compute multihead self attention in matrix form

```
# Now let's compute self attention in matrix form
def multihead_scaled_self_attention(X,omega_v1, omega_q1, omega_k1, beta_v1, beta_q1, beta_k1, omega_v2, omega_q2, omega_k2, beta_v2, beta_q2, beta_k2, omega_c):

  # TODO Write the multihead scaled self-attention mechanism.
  # 1. Compute queries, keys, and values
  # 2. Apply softmax to calculate attentions and weight values by attentions
  # 3. Concatenate the self-attentions and apply linear transformation to combine them
  # Replace this line
  X_prime = np.zeros_like(X) ;


  return X_prime
```

Hint:

$$\begin{aligned}\mathbf{V}_h &= \boldsymbol{\beta}_{vh}\mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{vh}\mathbf{X} \\ \mathbf{Q}_h &= \boldsymbol{\beta}_{qh}\mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{qh}\mathbf{X} \\ \mathbf{K}_h &= \boldsymbol{\beta}_{kh}\mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{kh}\mathbf{X}.\end{aligned}$$

$$\mathbf{Sa}_h[\mathbf{X}] = \mathbf{V}_h \cdot \mathbf{Softmax}\left[\frac{\mathbf{K}_h^T\mathbf{Q}_h}{\sqrt{D_q}}\right]$$

$$\mathbf{MhSa}[\mathbf{X}] = \boldsymbol{\Omega}_c\left[\mathbf{Sa}_1[\mathbf{X}]^T, \mathbf{Sa}_2[\mathbf{X}]^T, \ldots, \mathbf{Sa}_H[\mathbf{X}]^T\right]^T$$

The result will be like this (You can check if your answer is correct)

```
True values:
[[-21.207  -5.373 -20.933  -9.179 -11.319 -17.812]
 [ -1.995   7.906 -10.516   3.452   9.863  -7.24 ]
 [  5.479   1.115   9.244   0.453   5.656   7.089]
 [ -7.413  -7.416   0.363  -5.573  -6.736  -0.848]
 [-11.261  -9.937  -4.848  -8.915 -13.378  -5.761]
 [  3.548  10.036  -2.244   1.604  12.113  -2.557]
 [  4.888  -5.814   2.407   3.228  -4.232   3.71 ]
 [  1.248  18.894  -6.409   3.224  19.717  -5.629]]
```

Summarize what you need to do in the part 2

1. Complete all TODO parts as directed

Part 3

1. Self Attention

2. Multihead Self-Attention

3. Transformer

Dataset : Wikitext-2

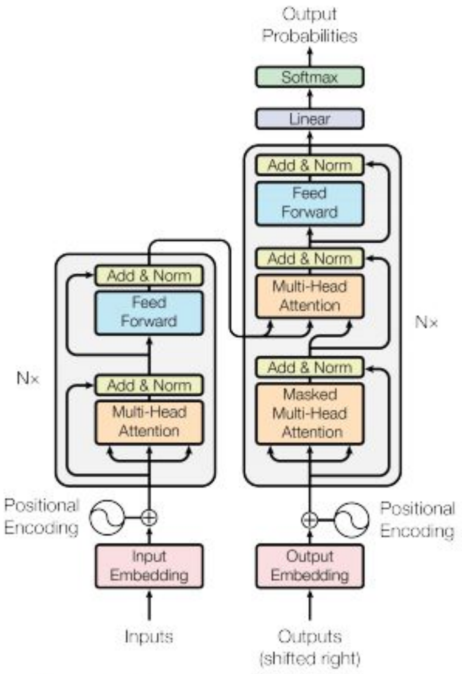


Figure 1: The Transformer - model architecture.

Notice

```
# Uncomment one of the following that works for you.

# device = torch.device("cuda")
device = torch.device("mps")
# device = torch.device("cpu")
```

- please set your device

```
from google.colab import drive
drive.mount('/content/drive')
import sys
sys.path.append('/content/drive/MyDrive/Week14/Week14/') # Change to your own path
import data
```

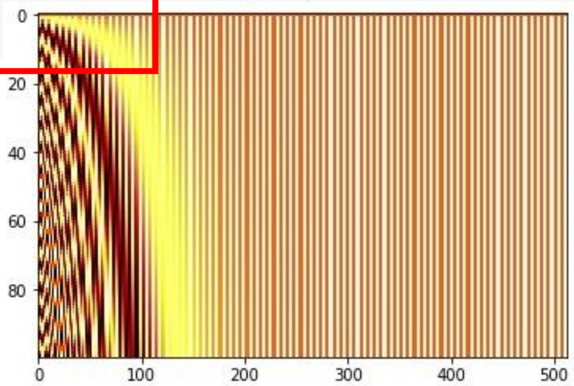
```
ve already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

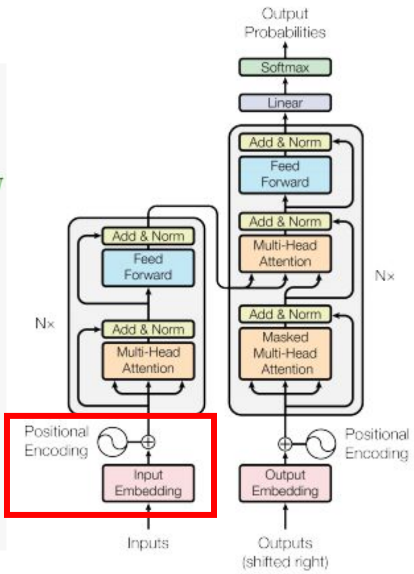
```
corpus = data.Corpus('/content/drive/My Drive/Week14/Week14/data/wikitext-2') # Change to your own path

def batchify(data, bsz):
    nbatch = data.size(0) // bsz
    data = data.narrow(0, 0, nbatch * bsz)
    data = data.view(bsz, -1).t().contiguous()
    return data.to(device)

train_data = batchify(corpus.train, batch_size)
val_data = batchify(corpus.valid, eval_batch_size)
test_data = batchify(corpus.test, eval_batch_size)
ntokens = len(corpus.dictionary)
```

- Pay attention to the paths of the module and dataset here.
- The methods and paths will differ when using Colab and a local environment

TODO - Complete positional encoding function

```python
# Define positional encoding used in the transformer model

##########################################################################################
# [TODO]: Build a positional encoding function that can be used in the TransformerModel below
##########################################################################################
class PositionalEncoding(nn.Module):

    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)


    def forward(self, x):


        return self.dropout(x)
```





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

Build the Transformer model && Training

- This part does not require implementation, but please read and understand it.
- Write a description of the method for this part.

TODO - use the transfer model to generate new text

```
g.set_state(initial_state)
input = torch.randint(ntokens, (1, 1), dtype=torch.long, generator=g).to(device)


generated_text = ""

###############################################################################
# [TODO] Fill out this section to use the transfer model to generate new text
###############################################################################

for i in range(num_words):



    generated_text = generated_text + word + " "
print(generated_text)
```

Hints :
1. Predict next word probabilities.
2. Scale probabilities with temperature.
3. Sample the next word index.
4. Add sampled word to the input.
5. Find the word for the index.
6. Add word to the output text.

gradually generate text with a length of 100

Like text solitaire

E.g.,

" "

↓

" The "

↓

" The cat "

↓

" The cat is "

↓

" The cat is cute "

↓

......

The result might look like this

```
walks nucleus a state holding from
to by , " However 1 introduction
low Yue tour that @-@ distracting
studied latter , around . . to than
the highlight Saving . Raj µm to ,
hold elite I and the Chevalier
percent until Tech most of ) " . .
for to special are the this meet
Beat a is sign crime lead for Ramon
and November art horn old Link the
touches the those author nine and
greater Beaumont cover son flats .
been C . debut Prussian , by the .
once the persuade , week most are
```

Summarize what you need to do in the part 3

1. Complete all TODO parts as directed
2. Description of the methodology (Transformer Model, Training)
3. Conclusions and discussions

Submission

- After executing your code, download the .ipynb file and submit it to NTU COOL
- Submitted file name: student ID_week14_colab1_homework.ipynb

  student ID_week14_colab2_homework.ipynb

  student ID_week14_colab3_homework.ipynb

- HW10 Deadline : 2024/12/10 23:59 (Monday night)
- No late submission

- If there are any questions

  Email: r12945048@ntu.edu.tw (add “[ML HW10]” to the beginning of the title.)