Different Attention Mechanisms

Ritesh Kumar Maurya

2024-05-24

- In this blog, we will try to understand how different attention mechanisms work alongside vanilla attention from 2017 paper
- For the code part, you can checkout this link

0.1 Issue with RNNs

- Informaton Bottleneck:- If sentence is too much long, they needed to compress all the previous token information in just a single vector h
- Sequential Nature:- All the tokens being processed one by one

0.2 Quotient rule

- Will be used while finding out the derivative of softmax
- $\left(\frac{u}{v}\right)' = \frac{u'v-uv'}{v^2}$.

0.3 Transformers

- Given a sequence of length N, where each token is represented by Dmodel numbers, we compute three matrices
  - Query = XWq (what I'm looking for)
  - Key = XWk (what I have)
  - Value = XWv (the information I'll give if I'm relevant)
  - Wq and Wk are of same shape and to maintain the symmetry Wv is also kept to be of same shape, Wq,Wk [Dmodel, Dk] and Wv [Dmodel, Dv]
- Now to find out which key is more relevant to given queries we do matrix multiplication and the finally apply softmax to get probability
  - score = QueryKey^T

- attn = softmax(score/sqrt(Dk))
- Why scaling factor
  * this score dot product has variance of Dk, which causes instability in backpropagation
    - lets say we have two vectors q and k of dimension dk
    - E[qi]=E[ki] = 0
    - var[qi]=var[ki] = 1
    - then E[q.k] = 0 but var[q.k] = dk link
  * the derivative of softmax contains some terms which if not normalizaied lead to gradient nearly 0 becuase of this larger variance
  *
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$
  *
$$\frac{\partial \text{Softmax}(z)_i}{\partial z_j} = \text{Softmax}(z)_i(\delta_{ij} - \text{Softmax}(z)_j)$$
  * If z_i is much larger than other elements, $\text{Softmax}(z)_i \approx 1$.
  * If z_i is much smaller, $\text{Softmax}(z)_i \approx 0$.
  * So incase of jacobian matrix, considering a simple case where query 0 attends all the keys
    - Diagonal: $s_1(1 - s_1) \approx 1(0) = 0$.
    - Off-Diagonal: $-s_1 s_2 \approx -1(0) = 0$.
  * to enforce unit variance, we apply normalization by dividng the score by standard deviation i.e. sqrt(Dk)

0.4 Single Head Vanilla Attention PyTorch Code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import time

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class SingleHeadVanillaAttention(nn.Module):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.Wq = nn.Linear(in_features = self.in_features, out_features = self.out_features)
```

```
        self.Wk = nn.Linear(in_features = self.in_features, out_features = self.out_features)
        self.Wv = nn.Linear(in_features = self.in_features, out_features = self.out_features)

    def forward(self, x):
        query = self.Wq(x) # [B, N, Dmodel] * [Dmodel, Dk] -> [B, N, Dk] TC=O(B*N*Dmodel*Dk)
        key = self.Wk(x)   #[B, N, Dmodel] * [Dmodel, Dk] -> [B, N, Dk] TC=O(B*N*Dmodel*Dk)
        value = self.Wv(x) # [B, N, Dmodel] * [Dmodel, Dk] -> [B, N, Dk] TC=O(B*N*Dmodel*Dk)

        scores = torch.matmul(query, key.transpose(-1, -2))/(self.out_features**0.5) # [B, N

        attn = F.softmax(scores, dim=-1) # [B, N, N]

        final = torch.matmul(attn, value) # [B, N, N] * [B, N, Dk] -> [B, N, Dk]  TC=O(B*N^2
        return final # [B, N, Dk]

Dmodel = 4096
Dk = 4096

model = SingleHeadVanillaAttention(in_features = Dmodel, out_features = Dk).requires_grad_(Fa

n_tokens = 2048
inp = torch.randn(1, n_tokens, Dmodel, device=device)

torch.cuda.synchronize()
start = time.time()
k=50
for _ in range(k):
    with torch.no_grad():
        out = model(inp)
torch.cuda.synchronize()
print(f"Time taken: {(time.time()-start)/k:.3f}")
```

```
Time taken: 0.285
```

0.5 Linear Attention

- Instead of using softmax, it tries to make it more general where this softmax function can be replaced by any other function
- Kernel is some function which takes in two input vectors (query and key) and gives a single scalar output i.e. similarity score.

- This softmax can also be replaced by some kernel which is some kind of similarity function (i.e. to have property of a similarity function).
- Kernel can be decomposed into linear functions (feature functions)
    - K(a, b) = phi(a)Tphi(b)
- Feature function takes a vector as input and projects it into new feature space
- The basic idea is instead of doing complicated non-linear function like softmax, cant we just project a and b into highr dimensional space and just do the linear inner product
- The attention mechanism is defined as $A_l(x) = V' = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V$.
- We can make the above equation more generalised by using a simlarity function
- $V_i' = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)}$
- where we can think of $\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$
- only constraint on sim is to give non-negative outputs
- given a kernel with feature representation $\varphi(x)$, we can rewrite the above equation
- $V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}$
- By making the use of associative property of matrix multiplication, we can take out Q from summation, because it is independet of j
- $V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}$ (4)
- And finally we can convert it into vectorized format
- $\left(\phi(Q)\phi(K)^T\right)V = \phi(Q)\left(\phi(K)^T V\right)$
- It is linear because we can compute numerator and denominator and use it for all the queries [The eqn before vectorized one]
- For softmax attention, complexity is O(N^2 * max(D, M)), where D is dimensionality for query and key, M is for value
- For linear attention, complexity is O(NCM), where C is feature map projection dim and M is for value
- Linearization of exact spftmax attention is not feasible because of the exponential kernel, which is of infinte dimension and cant be stored on a computer (expansion of e^x is infinitely many terms)
- Thats why it is better to use some kind of polynomial feature representation

- The author chose elu instead of relu to avoid setting gradients to 0 when input was negative