Vanilla rnn:

* No parallelism
* Cannot handel bottleneck
* Vanishing gradient

Attention:

* Partly vanishing gradient
* Satisfy bottleneck
* Cannot satisfy seq access and parametrical access

Attention is a genericic concept: can also be used to a single rnn

Encoder- Decoder:

Encoder states : keys/values

Decoder state : query

Keys: important word in doc that represent it

We can retrive the required portion form the

**Is attention all we need: ( cannot address seq access and parmetrical access)**

* **Attention is not only to solve sequence to sequence problem – generic solution to address other problem concept too – can also be applied to single rnn**

In encoder decoder model, keys and queries are the hidden state layer representation itself.

Calculate the dot product of query with all keys to get the attention score.

Pass attention score to attention distribution, the similarity score of distribution act as weight. Return the corresponding value relating to the output.

Why not remove the seq links if all are connected to one other, but we lose the sequence, i.e temporal dependence will not be preserved, i.e position info owill be lost– shuffeling the hidden states also produce the same output.

How to address them?.

**Self attention mechanism:**

Generate 3 vec from each hidden state – key, value, query

Every query multiplied with all keys - results in attention score – passed to softmax get attention distribution, these act as weights and each is multiplied with corresponding v and summation them to produce a1 – attention vector.

All these occure in a layes and there are multiple layer, - for every query, key, value pair is unique. i.e matrices shared across tokens but not pairs.

**Self Attention to Transformer:**

* **Positional encode:** to encode position of tokens

We fed position ecoding to the selfattention along with word embedding

* **Multi-headed attention: - to capture different aspect**

Attention head1, attention head2 to generate attention vector for different hidden they generate two attention vector a1, a2.( each attention is used to catpture different aspects(like wether noun, pronoun, etc).

This results in two vector of size d , but we can pass only one vector of size d to the next layer, for this we concatenate them resulting in a 2d vector which is passed to a linear function – matrix of 2d x d to generate a d sized matrix.

Time complexity: O(n^2)

* Adding nonlinearities:

Self attention is linear. Though we are using a non linear function. After simplifying the terms mathematically we can see that the non linearity just causes a coefficient over the seq of terms resulting in a net linear solution.

Why we need non linear layer?

How to tackle this?

Add non linearity through a feed forward layer.

Position wise feed forward network – pass to matrix w1 first do some nonlinear action and pass to matrix w2

Feed forward layer Considered as internal memory of transformer

* Mask attention: decoder of the hidden state willnot have any information of future hidden state, Encoder is given so each word is having access to every other word

In traing time future decoder states are known but in test time don’t know. How to make masking of future states in this in traing time also.

Practically will be having q matrix containg all queries and k matrix containg all key.

To not allow hidden state to know the futre state we almost dissect the upper right corner during q x h matric mul for this we multiply with another matrix

Cross attention layer:

Lec: 16:

Recap:

Self attention operation is a linear operation – so include non linearity – add feed forward network – position wise feed forward network,

In each encoder block - self attention layer and feed forward layer

In each Decoder block – self attention, mask self attention( only access to left side i.e future states not allowed).

To gave an interaction between encoder and decoder layer – query will come from the decoder and key,value form encoder. Through normal self attention we allow decoder interact with encoder.

Transformer access token in parallel – pos info not maintained , to maintain pos info we maintain positional enmbedding

**Positional encoding:**

Position aware embedding( embedding with signal) = pos encoding + embedding

Summing lead to parameter reduction

Pos aware fed to first layer of encoder/decoder – input sent to first block of transformer is pos aware format( not added to other layers in transformer).

Problem 1: how to address the issue of multi variable sentence length

Problem 1: how to address the issue of multi variable sentence length, how to calculate the max length as the no of tokens should be predefined and we have no access to that and changes if increase the training set.

Problem 2:

In sseq 1 last pos embedding is, say of vec 3. In seq 2 last pos embedding is of vec 30, but this seq also contanin pos vector 3 and model may misintruptly think seq 1 vec 3 same as that of seq 2 vec 3. how to let model know the end of seq.

Sol: Divide pos embedding by max token in a seq.

Now same pos in variable length sentence are different. i.e same pos in seq of diff length is different. As pos is caluculated as pos/max len

Representation of pos using binary representation . but each bit repeats itself after certain periodicity.

How to encode them: This is captured using sin function.

I = element pos in pos vector

Pos = pos in seq

d model = model size

upper part of vec – high freq sin wave

lower part – low freq sin wave

why i : to capture different values for different element as sin func is repetative

low freq – differ slightly

high freq – differ significantly

1st by sin, 2nd by cosine, 3rd by sin …

If I is multiple of 2 – sin

I not multiple by 2 – coine

I start with 0 so the above seq

Pos rel of nearby words high , and vice versa

Kind of dic, where every ele denotes a pos vector, when seq is fed corresponding pos encoding from dic and add and will be passed. When back propagation is applied position embedding are updated.

Properties of positional embedding:

* Monotonicity: proximity higher if words are near
* Translation embedding: proximity of word with same difference is equal ( like 1, 3 = 4,6)
* Symmentry:

Absolute pos embedding: sin representation embedding, pos aware embedding

Relativve pos encoding:

preserve relative pos b/w words.

R – relative dist b/w m and n i.e m-n , if m, n fall out the range of r min and r max r min and r max considered accordingly

Pr added with value and key: how to generate pr?

Transformer-XL:

1. replace n with p m-n in query key dot product
2. pm replaced by u and v due to diff context

like this many version of changes were introduced in T5 paper and DeBERTa paper

combining both relative and absolute positon: (here complex space is used to represent embedding).

diffence in angel takes care of relative position

rotation takes care of absolute position

Rotatory positon embedding( RoPE) : see ppt

Given funxtion fq and fk we need to generate such that they depend on word embedding m and pos m, word embedding n and pos n rspv and dot product of that should equal relative position between the word m and word n (i.e m-n).

Here multiplying rotatory matrix with vector leads to difference in ang but mag remain the same.

Rotation mtrix are orthogonal

M – query pos, n- key pos

Pos info captured by imaginary comp e pow im/ntheta

Rotation matrix takes care of abs position of m and n and self attention takes care of relative pos b/w m and n

The pos info is multiplicative while the earlier version we saw additive version approach

Properties of RoPE:

Long term decay.

Computationally efficient realization of rotatory matrix multiplication

Transformr Architecture:

Residual connection : helps in convergence and vanishing gradient

Add Norm:

When we try to make the model faster we use to smooth the vector input( i.e we make the vector follow a certain distribution(like make val fall b/w 0 and 1)).

Norm:

* Stabilized traingn
* Accelerated convergence
* Prevent overfitting
* Improvr gradient flow

Types of normalization: in transformer we use layer norm

Normalization is between 0 to 1

Standardization: mean =0 , variance = 1

Often standardization called as normalization

Batch Normalization: j

Learn Add Norm layer in transformer architecture:

Batch and layer normalization:

Batch should be reprenstative enough i.e batch size should be large enough to get correct output.

Why Normalization: