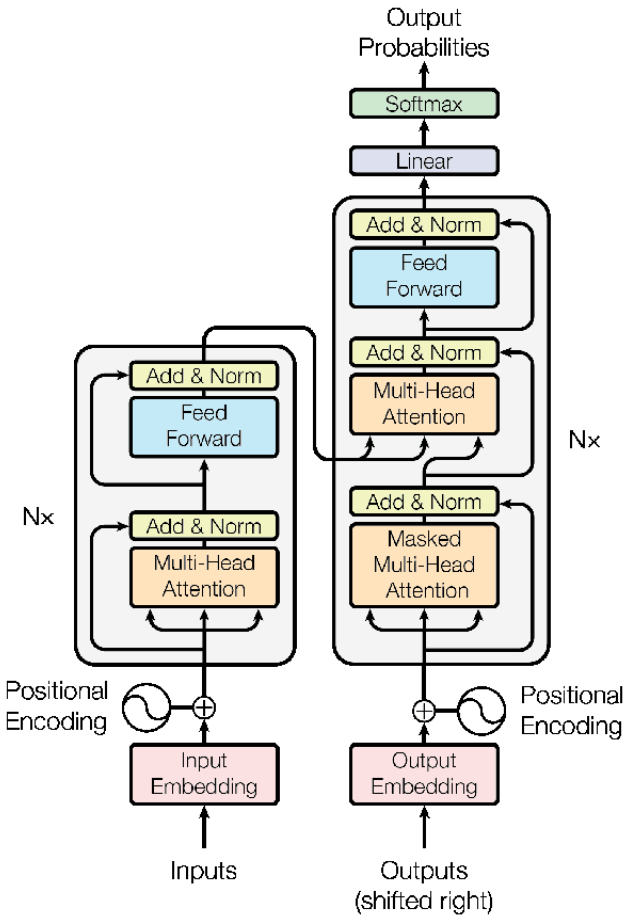
Ձևափոխիչ

Հայկ Կարապետյան

1 Ուշադրությունն այն ամենն է, ինչ ձեզ հարկավոր է

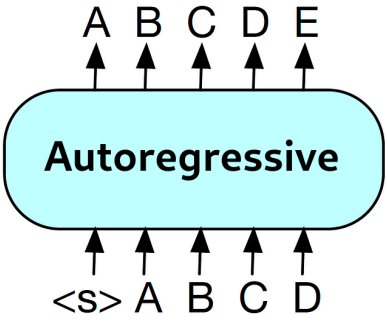
RNN ցանցերում առաջանում էր vanishing gradient խնդիր և առաջին անդամը այնքան ձևափոխությունների էր ենթարկվում history-ի մեջ, որ գրեթե ինֆորմացիա չէր մնում դրա մասին: Այդ պատճառով ներմուծեցինք LSTM և GRU բլոկները: Բայց դրանք, որոշ չափով էին լուծում այդ խնդիրը: Վերջապես attention մեխանիզմի միջոցով կարողացանք լուծել այդ խնդիրը և բոլոր NLP խնդիրները լուծվում էին RNN և attention համդրությունով: RNN-ը ունի մեկ թերություն: Հնարավոր չէ զուգահեռացնել հաշվարկային գործընթացը: Ե՛վ ուսուցման ընթացքում, և՛ թեստավորման ընթացքում, ամեն բլոկ պետք է սպասի նախորդ բլոկի history-ին: Եվ GPU-ի ամբողջ հզորությունը չենք կարող օգտագործել, զուգահեռացման բացակայության պատճառով: 2017 թվականին հայտնվեց նոր եղանակ NLP խնդիրները լուծելու: Հոդվածը, որի մեջ նկարագրված է այդ եղանակը կոչվում է՝ «Ուշադրությունն այն ամենն է, ինչ ձեզ հարկավոր է» (Attention is all you need): Այդ եղանակը կոչվում է ձևափոխիչ (transformer): Հիմնական գաղափարը RNN բլոկեր չօգտագործելն է: RNN բլոկերը օգտագործում էինք history ունենալու համար նախորդ տվյալներից, բայց self attention շերտը լուծում է այդ խնդիրը: Attention շերտի միջոցով մենք կարողանում ենք ինֆորմացիա քաղել մյուս անդամներից առանց history պահելու: Transformer ցանցի կառուցվածքը կարող եք տեսնել նկար 1-ում:



Նկար 1: Transformer ցանցի կառուցվածք

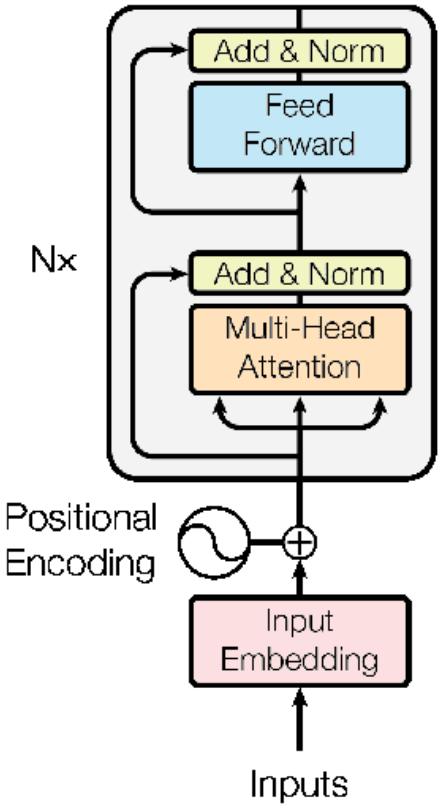
Հիմա ավելի մոտիկից ծանոթանանք այս ցանցի կառուցվածքին:

Առաջին հերթին հասկանանք, թե այն ինչպես է աշխատելու թեստավորման ժամանակ: Մասնավորապես ինչ են լինելու մուտքային և ելքային արժեքները: Transformer ցանցի միջոցով փորձելու ենք լուծել թարգմանության խնդիր: Մեր մուտքային տվյալները կլինեն հայերեն նախադասություններ, իսկ ելքայինները՝ անգլերեն նախադասություններ: Սկզբից հայերեն նախադասությունը բաժանելու ենք token-ների և ձախ մասում փոխանցենք, որպես input: Աջ մասում, որպես մուտքային տվյալ փոխանցելու ենք <start> token-ը: Եվ աջ մասը սկսելու է թարգմանել նախադասությունը: Transformer-ը ավտոռեգրեսիվ (autoregressive) մոդել է: Ի՞նչ է դա նշանակում: Երբ աջ մասում մուտքում փոխանցում ենք <start> token-ը, այն մեզ վերադարձնում է ելքային հավանականություններ, որոնցից ընտրում ենք ամենահավանական token-ը (կամ ընտրությունը կատարում ենք այլ ձև) և միացնելով <start> token-ին՝ նորից փոխանցում ենք աջ մասում, որպես input (Նկար 2): Եվ այդպես կրկնում ենք այնքան ժամանակ, քանի դեռ սահմանված իտերացիաների քանակը չի լրացել կամ գուշակված token-ը եղել է <end>-ը:



Նկար 2: Autoregressive մոդելի աշխատելու գործընթացը

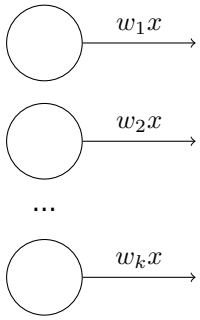
Transformer-ի ձախ մասը կոչվում է Encoder, իսկ աջ մասը՝ Decoder: Ծանոթանանք Encoder-ի կառուցվածքին (Նկար 3):



Նկար 3: Transformer ցանցի, Encoder հատվածի կառուցվածքը

2 Encoder

Encoder-ի առաջին շերտը Embedding շերտն է, որը ինչպես գիտենք, մուտքային one hot վեկտորները վերածում է ավելի շատ ինֆորմացիա պարունակող վեկտորների։ Դրանից հետո գումարում ենք positional encoding-ները, որոնք ինֆորմացիա են ավելացնում անդամի տեղի մասին նախադասության մեջ։ Embedding շերտին և Positional encoding-ին կարող եք ավելի լավ ծանոթանալ առանձին լեկցիայում։ Հաջորդ շերտը Multi-Head Attention շերտն է, որը ամեն անդամի համար հաշվում էր, թե որ անդամներն են ավելի կարևոր և ստանում էր վեկտորներ, որոնցից յուրաքանչյուրը ինֆորմացիա ունի բոլոր անդամների մասին։ Multi-Head Attention-ին նույնպես կարող եք ծանոթանալ առանձին լեկցիայում։ Ստացված վեկտորները փոխանցում ենք "Գումարման և Նորմավորման" (Add & Norm) շերտին։ Այս շերտում, Attention շերտից դուրս եկած վեկտորներին, գումարում ենք մինչ Attention շերտը եղած վեկտորները։ Դրա շնորհիվ սկզբնական անդամների մասին ինֆորմացիան չի կորչում։ Այս տեխնիկան շատ նման է resnet մեթոդի, որի ժամանակ, սկզբնական նկարի մասին ինֆորմացիան խորը ցանցերում չէր կորչում հենց գումարման միջոցով։ Գումարումից հետո ստացված վեկտորը, անցկացնում ենք նորմավորման շերտի միջով։ Այս գործընթացը կոչվում է layer normalization։ Ունենք շերտ, որի ելքային արժեքներն են, $w_1, x, w_2x, \ldots, w_kx$ (Գծագիր 1)։



Գծագիր 1։ k նեյրոնից բաղկացած շերտ

Layer Normalization-ը նորմավորում է ելքային արժեքները։ Այսինքն հաշվում է ելքային արժեքների միջինը, ստանդարտ շեղումը և կատարում անհրաժեշտ գործողությունները։

$$\mu = \frac{1}{k}\sum_{i=1}^{k} w_kx$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{k}(w_kx - \mu)}{k}}$$

$$x'_k = \frac{w_kx - \mu}{\sigma}$$

$$x'_k = \gamma x'_k + \beta$$

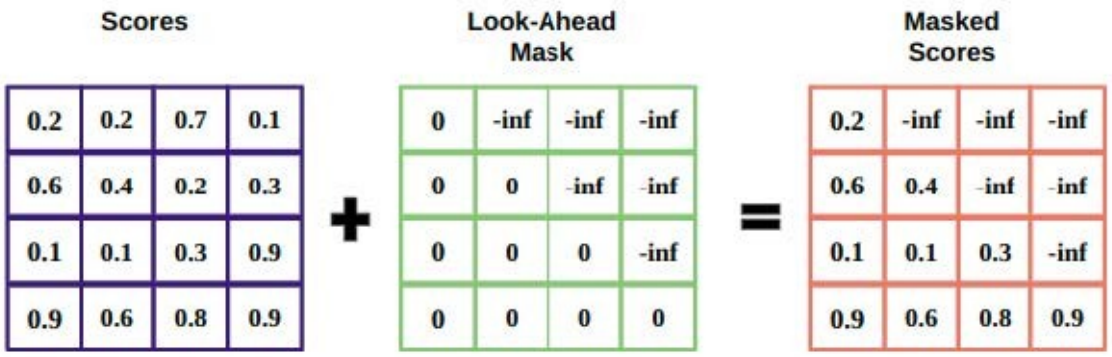
Այս շերտը շատ նման է batch normalization շերտին, բայց ի տարբերություն այդ շերտի այն նորմավորում է շերտի բոլոր նեյրոնների ելքային արժեքները իրար հետ, իսկ batch normalization-ի դեպքում, նորմավորում ենք նույն նեյրոնի ելքային արժեքները ամբողջ batch-ի համար։ γ և β արժեքները այստեղ նույնպես ուսուցանվող պարամետրեր են։ Encoder-ի հատվածից մնաց դիտարկենք Feed Forward Network շերտը։ Այն սովորական fully connected ցանց է և մուտքային տվյալների հետ կատարում է հետևյալ գործողությունը։

$$FFN(x) = W_2ReLU(W_1x + b_1) + b_2$$

Feed Forward շերտից հետո նորից կատարում ենք Add & Norm գործողությունը։ Encoder-ի հատվածը այսքանով ավարտվում է և այն վերադարձնում է այնքան output, ինչքան եղել է մուտքային տվյալների երկարությունը։ Օրինակ՝ նախադասությունը բաղկացած է եղել 10 անդամից։ Encoder-ից նույնպես դուրս կգա 10 հատ վեկտոր։

3 Քողարկված բազմագլուխ ուշադրություն

Սովորական self attention-ի դեպքում ամեն անդամ կարողանում էր հասկանալ, մյուս անդամները ինչքանով են կարևոր իր համար և նշանակություն չուներ այդ անդամները գտնվում են իրենից հետո, թե առաջ: Տեքստի գեներացման խնդիրներում, ամեն պահի մենք չգիտենք, թե հաջորդ բառը ինչ պետք է լինի: Օրինակ՝ ունենք հայերեն բառերը, անգլերեն թարգմանելու խնդիրը: Ամեն պահի մենք փորձում ենք գուշակել հաջորդ անգլերեն բառը: Եվ ուսուցումը նույնպես անհրաժեշտ է այդ կերպ կազմակերպել: Transformer-ի առավելությունը RNN-ից նաև այն էր, որ կարող էինք ուսուցումը զուգահեռացնել: Ուսուցման ընթացքում, եթե Multi-Head Attention-ին մուտքում տանք միանգամից բոլոր անգլերեն բառերը, այն ուղղակի կսովորի մուտքային անգլերեն բառերը առանց փոփոխելու ելքում վերադարձնել: Այսինքն ոչինչ չի սովորի: Այդ պատճառով անհրաժեշտ է այնպես կազմակերպել ուսուցումը, որպեսզի առաջին անգլերեն բառի թարգմանության ժամանակ, այն չիմանա առաջին անգլերեն բառը, իսկ երկրորդի ժամանակ միայն իմանա առաջին բառը և այդպես շարունակ: Դրա համար սովորական Multi-Head Attention-ի փոխարեն օգտագործում ենք քողարկված բազմագլուխ ուշադրություն (Masked Multi-Head Attention): Այն ամեն քայլի միայն ուշադրություն (attention) է դարձնելու նախորդ անդամներին: Հիմա դիտարկենք, թե ինչպես ենք ուշադրություն դարձնելու միայն նախորդ անդամներին մաթեմատիկայի տեսանկյունից (Նկար 4):



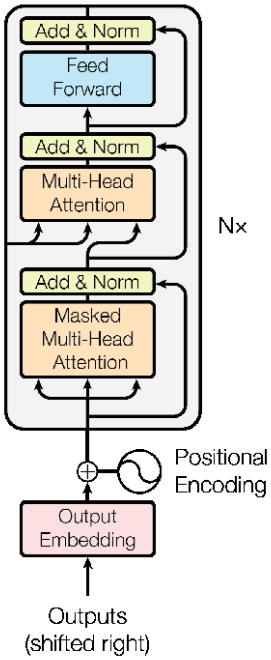
Նկար 4: Masked Multi-Head Attention շերտի քողարկում

Այս դեպքում ունենք 4 մուտքային անդամ: Առաջին տողում գրված են $q_1 \cdot k_1, \ldots, q_1 \cdot k_4$, արժեքները, իսկ 4-րդ տողում՝ $q_4 \cdot k_1, \ldots, q_4 \cdot k_4$ արժեքները: Մեզ անհրաժեշտ է առաջին անդամը ինֆորմացիա ունենա միայն իր մասին, դրա համար $q_2 \cdot k_2, q_3 \cdot k_3, q_4 \cdot k_4$ արժեքները քողարկում ենք: Իսկ 4-րդ անդամը ունի ինֆորմացիա բոլորի մասին: Իսկ ինչպե՞ս է մինուս անվերջությունը օգնում քողարկել անդամները: $q \cdot k$ արժեքները անցնում էին softmax ֆունկցիայի միջով՝ α-ները ստանալու նպատակով: Որտեղ գրված լինի մինուս անվերջություն (-inf), այդ արժեքները կզրոյանան, իսկ մնացած արժեքներնի հավանականությունները կմնան նույնը: Վերջում α-ները value-ներով բազմապատկելիս, նախորդ անդամների արժեքները հաշվի չենք առնի:

$$v_1' = \alpha_{11} v_1 + \alpha_{12} v_2 + \alpha_{13} v_3 + \alpha_{14} v_4, \quad \alpha_{12} = \alpha_{13} = \alpha_{14} = 0$$

4 Decoder

Transformer-ի երկրորդ հատվածը decoer-ն է (Նկար 5)։



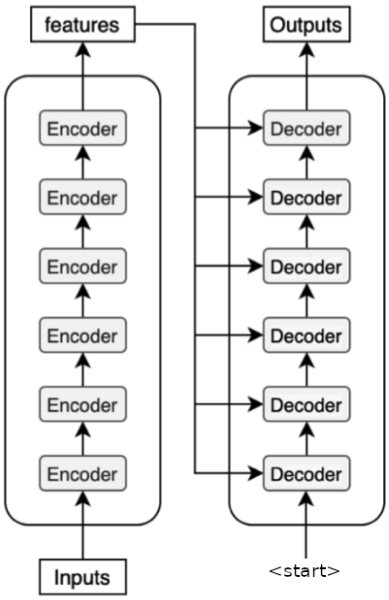
Նկար 5: Transformer ցանցի, Decoder հատվածի կառուցվածքը

Սկզբից նորից ունենք Embedding և Positional Encdoing շերտերը։ Embedding շերտը Encoder-ի շերտում եղածից տարբերվում է կշիռներով։ Իսկ Positional Encoding-ը նորից հաշվում ենք սինուսների և կոսինուսների միջոցով։ Դրանից հետո անդամների վեկտորները անցնում են վերը քննարկված Masked Multi-Head Attention շերտի միջով։ Դրանից հետո նորից փոխանցում ենք Add & Norm շերտին։ Դրանից հետո ունենք Multi-Head Attention շերտ։ Այս շերտին անհրաժեշտ է փոխանցել Q, K, V արժեքներ։ Սովորական դեպքում վերցնում էինք մուտքային վեկտորները և բազմապատկում W_q, W_k, W_v մատրիցներով։Այստեղ մեզ անհրաժեշտ է Encoder-ում ստացված վեկտորները, ինչ որ կերպով փոխանցել Decoder-ին։ Encoder-ը ինֆորմացիա ունի հայերեն բառերի մասին և դրանք անգլերեն թարգմանելիս մեզ հարկավոր է այդ ինֆորմացիան։ Պատկերված նկար 5-ում երևում է, որ Q, K, V արժեքներից երկուսը ստանում ենք Encoder-ից, իսկ մեկը ստանում ենք Decoder-ի Add & Norm շերտից։ Իսկ ո՞ր երկուսն ենք ստանում Encoder-ից։ Վերհիշենք Q, K, V արժեքների անվանումները։ Հարց, հուշում և արժեք։ Անգլերեն բառը թարգմանելիս մեզ պետք է հարցնել հայերեն բառերի մասին ինֆորմացիա և ստանալ համապատասխան հուշումներն ու արժեքները։ Այսինքն Q-ն ստանալու ենք Add & Norm շերտի արժեքները W_q-ով բազմապատկելիս, իսկ K-ն և V-ն Encoder-ից դուրս եկած արժեքները W_k-ով և W_v-ով բազմապատկելիս։ Ստանալով Q, K, V արժեքները կկատարենք նույն գործողությունները, ինչ սովորական Multi-Head Attention-ի դեպքում։

$$Multihead = Concat(head_1, ..., head_h)W^o$$
$$head_i = Attention(xW_i^Q, xW_i^K, xW_i^V)$$
$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Add & Norm-ից ստացված output-ը փոխանցում ենք Feed Forward շերտին, որը դիտարկել ենք Encoder-ի կառուցվածքում։ Feed Forward շերտի output-ը նորից անցկացնում ենք Add & Norm շերտի միջով և ստանում վերջնական output-ը։ Նկար 1-ում կարող ենք տեսնել, որ Encoder և Decoder կառուցվածքների կողքին գրված է $N\times$։ Դա նշանակում է, որ Encoder-ի կառուցվածքը կարող ենք մի քանի անգամ կրկնել։ Առաջին Encoder-ի output-ը կարող ենք

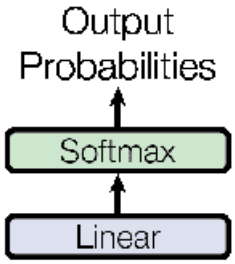
փոխանցել երկրորդ Encoder-ին, որպես մուտք և նույնը ճիշտ է Decoder-ի դեպքում: Միակ բանը, որին պետք է ուշադրություն դարձնել այս դեպքում այն է, որ բոլոր Decoder-ների Multi-Head Attention շերտը՝ K և V արժեքները ստանում է վերջին Encoder-ի output-ից (Նկար 6):



Նկար 6: Encoder և Decoder կառուցվածքները վեց հատ իրար վրա (N=6)

5 Output շերտ

Նկար 1-ում Decoder-ի output-ը փոխանցում ենք Linear շերտին, որը սովորական Dense շերտ է և վերջում կիրառում ենք softmax ակտիվացիոն ֆունկցիա (Նկար 7):



Նկար 7: Transformer մոդելի Output շերտ

Իսկ քա՞նի նեյրոն ենք ունենալու Linear շերտում: Թարգմանության խնդրում մեզ անհրաժեշտ է ամեն պահի գուշակել հաջորդ թարգմանված անդամը (բառը): Tokenizer-ը ունենալով բառարան (vocabulary), կարողանում էր նախադասությունը բաժանել անդամների և ամեն անդամի համապատասխանեցնում էր ինդեքս (one-hot վեկտոր):Այժմ թարգմանություն կատարելիս անհրաժեշտ է ամեն պահի գուշակել ամենահավանական թարգմանված անդամը: Այսինքն մեր Linear շերտում ունենալու ենք vocabulary_size քանակի նեյրոններ: Softmax անելուց հետո ընտրելու ենք ամենահավանական անդամը: Օրինակ՝ ելքում ունենում ենք հետևյալ հավանականային վեկտոր:

$$output = [0.2, 0.6, 0.1, \dots]$$

Այս վեկտորից կարող ենք ասել, որ թարգմանված բառը բառարանի 2-րդ բառն է (0.6):

6 Եզրափակում

Այսպիսով RNN ցանցերին փոխարինեցին մոդելներ, որոնք չունեին ռեկուրենտ բլոկներ և միայն հիմնված էին Attention շերտերի վրա: Transformer մոդելը կազմված է երկու հիմնական բաղադրիչներից՝ Encoder և Decoder: Սկզբում Embedding շերտը և Positional Encoding-ը բառերի one-hot վեկտորները վերածում են ինֆորմացիոն վեկտորների: Encoder և Decoder կառուցվածքներում ամենակարևորը բաղադրիչը Multi-Head Attention շերտն է: Իսկ Decoder-ում ի տարբերություն Encoder-ի ունենք նաև Masked Multi-Head Attention շերտ, որը միայն ուշադրություն է դարձնում նախորդ անդամներին: Վերջում ստանում ենք հավանականային վեկտոր, որը ցույց է տալիս, թե ամենահավանական անդամը որն է: Ստացած անդամ միացնում ենք մեր մուտքին և նորից փոխանցում Decoder-ին: Օրինակ՝ Decoder-ին սկզբից փոխանցում ենք ”<start>” token-ը և output-ում ամենահավանական token-ը ”I”-ն է: Հետո Decoder-ին մուտքում տալիս enq ”<start> I” անդամները և output ստանում ենք ”study” token-ը: Այս գործողությունը կա՛մ կատարում ենք ֆիքսված քանակով (օրինակ 10 անգամ), կա՛մ մինչ <end> token ստանալը (”<start> I study DL.”) անդամները մուտքում փոխանցելիս output-ում կստանանք ”<end>” token-ը: Transformer-ը գրեթե ամբողջությամբ զուգահեռացնում է բազմապատկման գործողությունները, ի տարբերություն RNN ցանցերի: Բայց մեկ է թեստավորման ժամանակ ամեն անգամ output-ը միացնում ենք մուտք և այդ գործողությունը կատարում մի քանի անգամ, իսկ ուսուցման ժամանակ Decoder-ը չի կարող շարունակել գործողությունները, քանի դեռ վերջին Encoder կառուցվածքը չի վերադարձրել output-ը: