Ինքնաուշադրություն

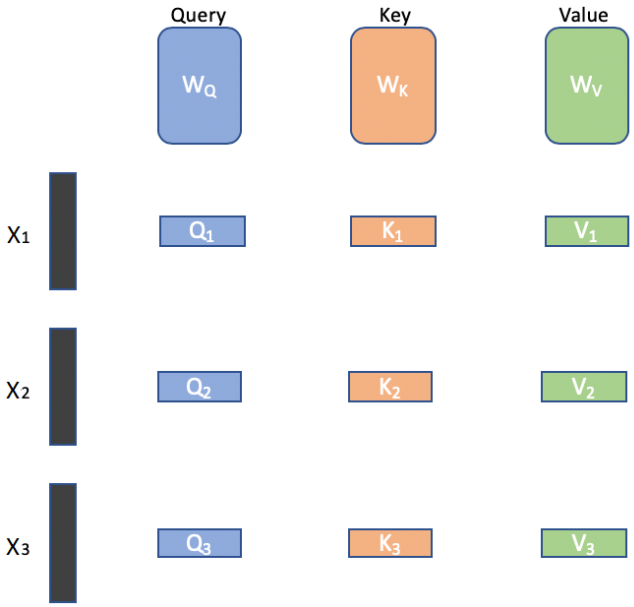
Հայկ Կարապետյան

RNN-ում առաջանում էր vanishing gradient խնդիր, որի պատճառով որոշում կայացնելիս սկզբնական տվյալներին հաշվի չէինք առնում (long term problem)։ Ներմուծելով attention գաղափարը, լուծեցինք այդ խնդիրը։ Այսօր կծանոթանանք attention-ի մեկ այլ տեսակի հետ, որը կոչվում է ինքնաուշադրություն (self attention)։ Attention մեխանիզմը նաև անվանում են շերտ, քանի որ առկա են ուսուցանվողդ պարամետրեր։ Այն սովորական attention-ից տարբերվում է α-ների ստացման եղանակով։ Սովորական attention-ի դեպքում α-ն ստանալու համար վերցնում էինք առաջին BiRNN-ի history-ն (h') և երկրորդ RNN-ի history-ն (s_{t-1}) և ստանում էինք α_t (Բանաձև 1)։

$$\alpha_t = W_1^T(W_2 s_{t-1} + W_3 h'_t) \tag{1}$$

Այստեղ ամեն h' տեղյակ է նախորդներից, բայց մեկ է ինչ որ ձևափոխությունների ենթարկված։ Այդ պատճառով self attention-ը առաջարկում է α-ները ստանալ, օգտագործելով միայն h'-երը։ Դիտարկենք self attention շերտը, երբ մեր մուտքային տվյալները նախադասություններ են և մենք լուծում ենք թարգմանության խնդիր։ Նախադասության ամեն անդամի (token) համար ունենք իրեն համապատասխան embedding-ը և այդ embedding-ի չափը հիպերպարամետր է, որը մենք ենք ընտրում։ Վերցնենք embedding-ի չափը d_emb։ Այժմ ամեն անդամի embedding-ը ներկայացնենք երեք ավելի փոքր embedding-ներով։ Այդ embedding-ները ունեն ”հարց” (query), ”հուշում” (key) և ”արժեք” (value) անվանումները։ Նախադասության t-րդ անդամի embedding-ը նշանակենք x_t-ով։ x_t-ն d_emb չափողականության վեկտոր է։ n չափողականության վեկտոր ասելով հասկանում ենք n չափողականությամբ տարածության վեկտոր, որը նույնն է, որ վեկտորը բաղկացած է n հատ էլեմենտից։ x_t վեկտորի միջոցով ստանանք երեք՝ ավելի փոքր չափողականության վեկտորներ (Նկար 1)։

$$Q_t = W_q x_t \ (query),\ K_t = W_k x_t \ (key),\ V_t = W_v x_t \ (value)$$



Նկար 1: x_t embedding-ը վերածում ենք երեք ավելի փոքր embedding-ների

Հիմա հասկանանք , թե ինչպես ենք օգտագործելու այս արժեքները α-ները ստանալու համար։ Ամեն նախադասության անդամը ունի իրեն համապատասխան q, k, t վեկտորները։ Առաջին անդամը ուզում է հասկանալ, թե իր համար ինչքան կարևոր են մնացած անդամները և բոլորին հարցնում է այդ մասին (query)։ Նախադասության մյուս անդամները, որպես պատասխան

վերադարձնում են հուշում, թե ինչքան կարևոր են առաջին բառի համար (key)։ Առաջին անդամի համար կստանանք հետևյալ զույգերը։

$$(q_1, k_1),\ (q_1, k_2),\ \ldots,\ (q_1, k_t)$$

Երկու վեկտորների նմանությունը կարող ենք հաշվել իրենց կազմած անկյունով կամ իրենց կազմած անկյունի կոսինուսով (cosine similarity)։

$$cos(a, b) = \frac{a \cdot b}{||a|| \cdot ||b||}$$

a-ն և b-ն վեկտորներ են։ Համարիչում գրված են դրանց կետային բազմապատկումը (dot product), այսինքն a-ի անդամները հերթով բազմապատկում ենք b-ի անդամների հետ։ Բազմապատկումը կարող ենք կատարել միայն այն դեպքում, երբ a և b վեկտորների չափողականությունը նույնը լինի։ Հայտարարը նորմավորման համար է, որպեսզի այդ վեկտորները դարձնենք միավոր վեկտորներ, հետո նոր հաշվենք կազմած անկյան կոսինուսը։ Համարիչից հետևում է, որ երկու վեկտորների կետային բազմապատկումը իրենից ներկայացնում է, ինչ որ նմանություն։ Այդ պատճառով հասկանալու համար, թե ամեն բառի համար մյուսը ինչքան է կարևոր, query և key վեկտրոները կետային բազմապատկելու ենք իրար հետ։ Արդյունքում կստանանք։

$$q_1 \cdot k_1,\ q_1 \cdot k_2,\ \ldots,\ q_1 \cdot k_t$$

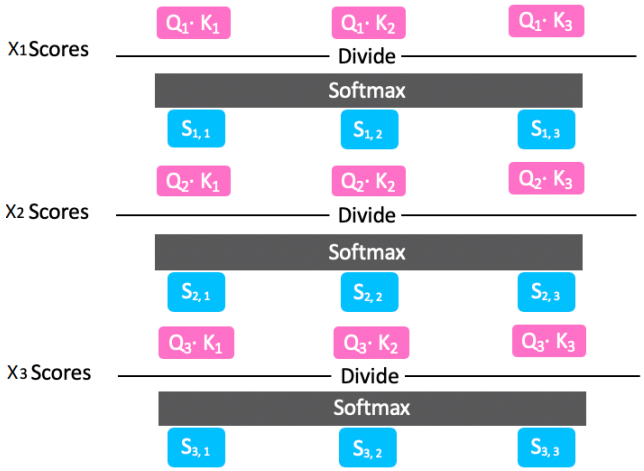
W_k և W_q մատրիցների չափերը պետք է նույնը լինեն՝ (d_{emb}, d_k), որպեսզի կարողանանք k և q վեկտորների միջև կատարել կետային բազմապատկում։ Որպեսզի այս արժեքները ցույց տան ինչ որ տոկոսային կարևորություն, կիրառենք softmax ակտիվացիոն ֆունկցիա։

$$s_{1t} = \frac{e^{q_1 \cdot k_t}}{\sum_{i=1}^{t} e^{q_1 \cdot k_i}}$$

$q \cdot k$ բազմապատկման արժեքը մեծանալու է d_k-ի մեծանալուն զուգահեռ, քանի որ վեկտորների չափողականության մեծացմանը զուգընթացի կաճի նաև գումարելիների քանակը կետային բազմապատկում կատարելիս։ Մեծ արժեքների դեպքում կարող են առաջանալ խնդիրներ։ Softmax ակտիվացիոն ֆունկցիան մեծ արժեքների դեպքում կարող է շատ դանդաղ և ոչ ճշգրիտ աշխատել էքսպոնենտի պատճառով։ Գրադիենտային վայրեջքի դեպքում հնարավոր է մեծ արժեքներ և անկայուն ուսուցում։ Այդ պատճառով արժեքները փոքր պահելու նպատակով բաժանում ենք $\sqrt{d_k}$-ի։

$$s_{1t} = \frac{e^{q_1 \cdot k_t / \sqrt{d_k}}}{\sum_{i=1}^{t} e^{q_1 \cdot k_i / \sqrt{d_k}}}$$

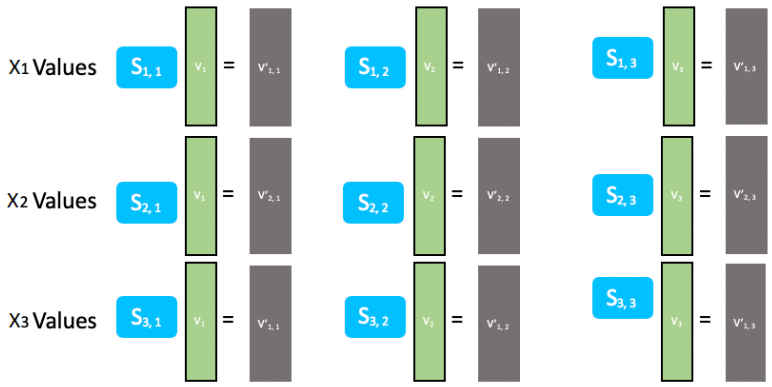
Այս գործողությունները պատկերավոր կարող եք տեսնել նկար 2-ում: Ստացված s-երը 0-ից մեկ միջակայքից են և դրանց գումարը հավասար է մեկի: s_{11}-ը ինֆորմացիա է պարունակում, թե առաջին բառի թարգմանության համար, առաջին բառը ինչքան կարևոր է և համապատասխանաբար s_{1t}-ն ցույց է տալիս, թե առաջին բառի թարգմանության համար t-րդ բառը ինչքան կարևոր է: Ամեն անդամի համար ստացել էինք երեք embedding՝ query, key, value: Հասկացանք,



Նկար 2: query և key վեկտորները կետային բազմապատկում ենք, ապա բաժանում d_k-ի և կիրառում softmax

որ query և key embedding-ները օգտագործվում են հասկանալու համար մյուս անդամների կարևորութ-յունը: Value-ն ցույց է տալիս այդ անդամի արժեքը և նոր embedding ստանալու համար անհրա-ժեշտ է s-ը բազմապատկել value-ով (Նկար 3):

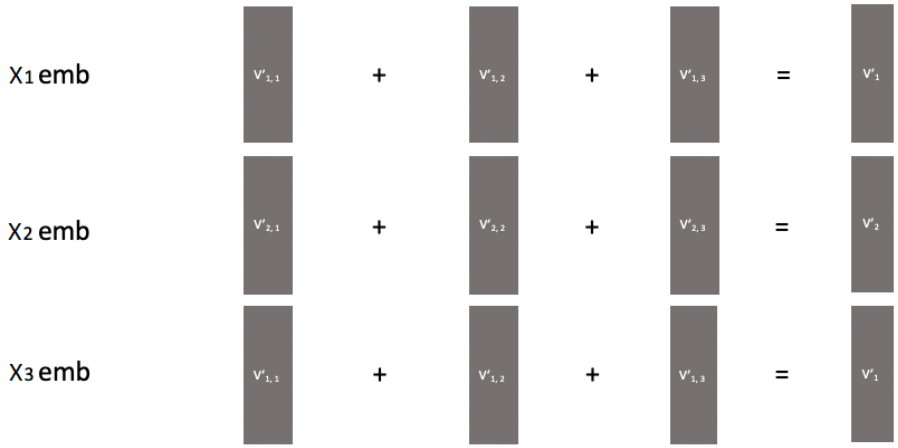
$$v'_{1t} = s_{11} v_t$$



Նկար 3: Softmax-ի արժեքները բազմապատկում ենք value embedding վեկտորներով

Ամեն անդամի համար նոր embedding ստանալու համար, իրեն համապատասխան ինդեքսով v' արժեքները գումարելու ենք իրար (Նկար 4)։

$$v'_{11} = s_{11}v_1 + s_{12}v_2 + \cdots + s_{t1}v_t$$



Նկար 4: Նոր embedding վեկտորների ստացում

W_v մատրիցի չափը կարող է տարբերվել W_k և W_q մատրիցների չափերից։ Attention-ի շերտը կարող ենք վեկտորներով գրել հետևյալ կերպ։

$$Attention(q, k, v) = \sum_i \frac{e^{q \cdot k_i / \sqrt{d_k}}}{\sum_j e^{q \cdot k_j / \sqrt{d_k}}} v_i$$

Կամ կարող ենք գրել մատրիցների տեսքով

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Այսպիսով self attention-ի միջոցով ամեն անդամ հասկանում է, թե իր կողքին եղած անդամներից, որոնք են կարևոր և ինչքան չափով, իսկ որոնք կարևոր չեն։