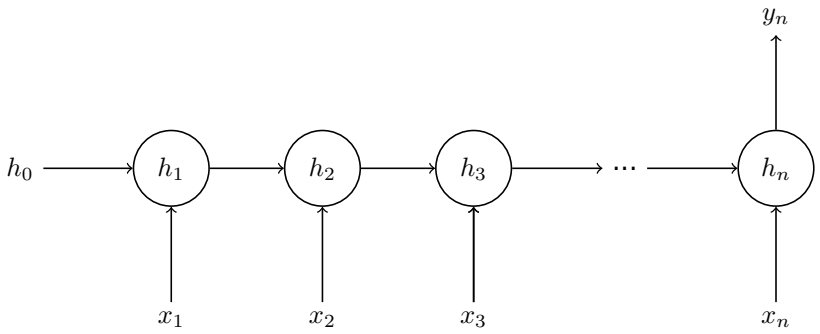
Երկարատև և կարճատև հիշողություն

Հայկ Կարապետյան

RNN ցանցերը նախատեսված են հաջորդական (sequential) տվյալների հետ գործողությունները կատարելու համար: Օրինակ՝ մեկնաբանության տեքստը օգտագործելով հասկանանք, թե մեկնաբանությունը լավն է, թե ոչ (sentiment analysis): Հիմա դիտարկենք, թե ինչ խնդիր կարող է առաջանալ երկար տեքստերի դեպքում:

1 Երկարաժամկետ հիշողության խնդիրը

Դիտարկենք գծագիր 1-ում պատկերված RNN ցանցը:



Գծագիր 1: RNN ցանց մեկ ելքով

Ունենք n բառից բաղացած տեքստ և ցանկանում ենք վերջում ստանալ մեկ ելք (1՝ դրական տեքստ, 0՝ բացասական տեքստ): History-ին և output-ը ստացվում են հետևյալ բանձևով.

$$h_i = \sigma(w_1 x_i + w_2 h_{i-1})$$
$$y_n = h_n = \sigma(w_1 x_n + w_2 h_{n-1})$$

Որտեղ w_1, w_2-ը ուսուցանվող կշիռներ են: Այս ցանցում կորստի ֆունկցիան վերցնելու ենք binary cross-entropy:

$$L = -yln(h_n) - (1-y)ln((1-h_n)),\ y \in \{0,1\}$$

$y-$ը իրական պիտակն է, իսկ h_n-ը ցանցի գուշակած պիտակը

Դիտարկենք կորստի ֆունկցիայի ածանցյալը, երբ պիտակը=1

$$L(y=1) = -ln(h_n)$$

$$\frac{\partial L}{\partial w_2} = -\frac{1}{h_n}\frac{\partial h_n}{w_2} = -\frac{1}{h_n}\sigma'(w_1 x_i + w_2 h_{n-1})\frac{\partial (w_1 x_i + w_2 h_{n-1})}{\partial w_2} =$$
$$= -\frac{1}{h_n}\sigma(...)(1-\sigma(...))\left(h_{n-1} + w_2\frac{\partial h_{n-1}}{\partial w_2}\right)$$
$$\frac{\partial h_{n-1}}{\partial w_2} = \sigma(...)(1-\sigma(...))\left(h_{n-2} + w_2\frac{\partial h_{n-2}}{\partial w_2}\right)$$

Ըստ w_2-ի ածանցյալները հաշվելիս կարող ենք տեսնել, որ առաջանում է w_2 գործակից: Առաջացած գործակիցները բազմապատկվելով ստանալու ենք w_2^n: w_2-ի մեկից մեծ լինելու դեպքում w_2^n-ը շատ է մեծանալու (exploding gradient), իսկ 1-ից փոքր լինելու դեպքում ձգտելու է զրոյի (vanishing gradient): Այստեղ կարող ենք տեսնել, որ սկզբի արժեքները w_2-ով չենք բազմապատկում և ունենում ենք գումարելի, որը զրո չի դառնա: Դա ավելի պարզ է երևում է, երբ կորուստի հաշվումը գրում ենք հետևյալ կերպ.

Սկզբից հաշվենք կորստի ածանցյալը ըստ առաջին history-ի

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_n}\frac{\partial h_n}{\partial h_{n-1}}\frac{\partial h_{n-1}}{\partial h_{n-2}}...\frac{\partial h_2}{\partial h_1}$$
$$\frac{\partial h_n}{\partial h_{n-1}} = w_2\sigma(...)(1-\sigma(...))$$
$$\frac{\partial L}{\partial h_1} = w_2^n\prod_{i=1}^{n}\sigma(...)(1-\sigma(...)) \quad (1)$$

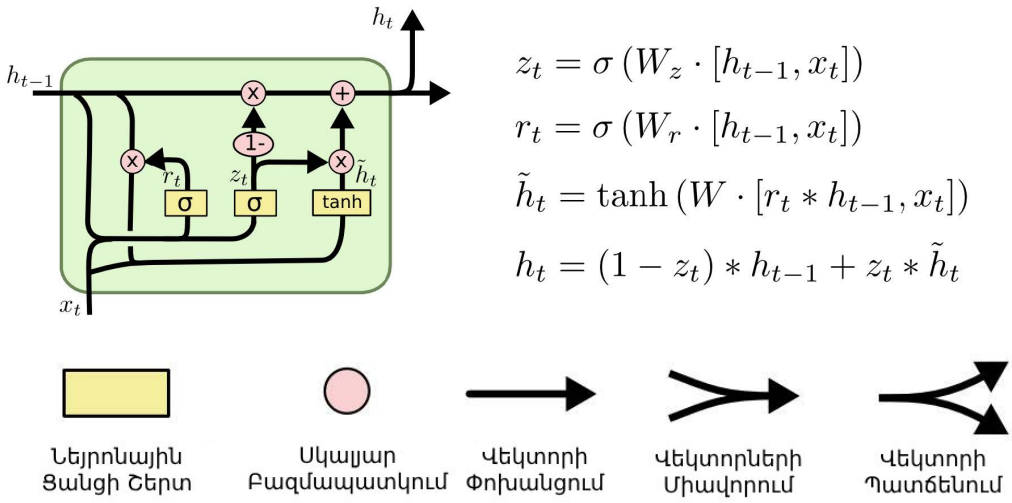
Հիմա հաշվենք կորստի ածանցյալը ըստ w_2-ի

$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial h_1}\frac{\partial h_1}{\partial w_2} + \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial w_2}} + \cdots + \boxed{\frac{\partial L}{\partial h_n}\frac{\partial h_n}{\partial w_2}}$$

Կարմիրով նշված հատվածը հստակ երևում է, որ ձգտում է 0-ի (ըստ 1 բանաձևի)։ Դա կոչվում է երկարատև հիշողություն։ Դրանից հետևում է, որ վերջնական ելքի վրա (պիտակ) առաջին բառը ազդեցություն չի ունենում և ըստ առաջին բառի կշիռները չենք թարմացնում։ Իսկ կանաչով նշված հատվածը 0 չէ և դրանք վերջին բառերի ածանցյալներն են։ Այսինքն վերջին բառերը հաշվի առնելով մենք թարմացնում ենք կշիռներն այնպես, որպեսզի գուշակած պիտակի և իրական պիտակի միջև կորուստը լինի նվազագույնը։ Եվ երկար մեկնաբանությունների դասակարգման խնդրում, ցանցը որոշում կայացնելիս ուշադրություն չի դարձնի առաջին բառերին։ Այսպիսով սովորական RNN ցանցերում մենք միայն ունենք կարճատև հիշողություն։

2 GRU

Սովորական RNN-ում հասկացանք, որ սկզբի մուտքային տվյալները հաշվի չենք առնում որոշում կայացնելիս և դրա պատճառն այն էր, որ սկզբնական history-ին շատ փոփոխությունների էր ենթարկվում (բազմապատկվում w_2-ով) և ինֆորմացիան կորչում էր։ Այդ խնդիրը լուծելու համար առաջացել է GRU բլոկը (Նկար 1)։ Քայլ առ քայլ հասկանանք այս բլոկի կառուցվածքը։



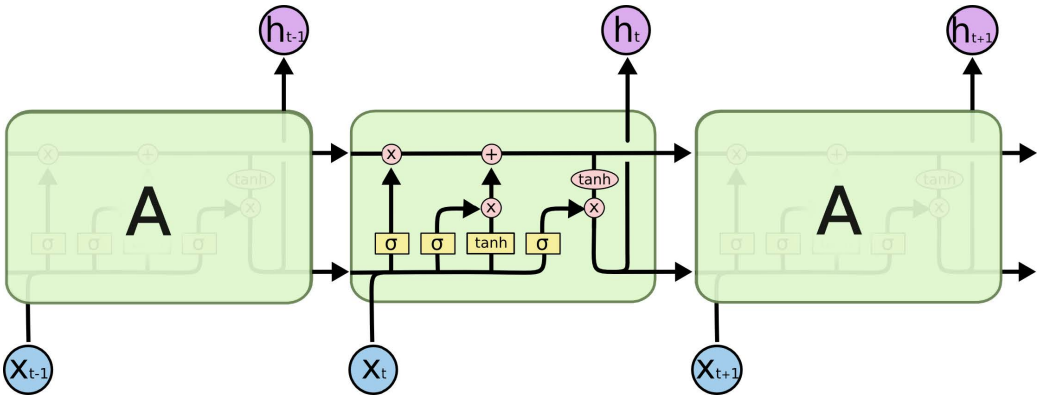
Նկար 1: GRU բլոկ

Սկզբից history-ին և մուտքային տվյալը միացնում ենք իրար (concat), բազմապատկում W_z-ով և կիրառում sigmoid ակտիվացիոն ֆունկցիա։ $W_z[h_{t-1}, x_t]$-ն համարժեք է $W_{zh}h_{t-1} + W_{zx}x_t$ գործողությանը։ Ստացված արդյունքը 0-ից 1 միջակայքի թիվ է և արդյունքը վերագրում ենք z_t-ին։ Նույն գործողությունը կատարում ենք ևս մեկ անգամ, բայց արդեն ուրիշ կշռով՝ W_r։ Հիմա նոր հիշողությունը ստանալու համար կատարենք հետևյալ գործողությունները։ Բազմապատկենք նախորդ հիշողությունը r_t-ով։ Միացնենք x_t-ին, բազմապատկենք W-ով և կիրառենք tanh ակտիվացիոն ֆունկցիա։ Սովորական RNN-ի դեպքում h_{t-1}-ը չէր բազմապատկվում r_t-ով։ Այս բազմապատկումը ցույց է տալիս, թե նախորդ հիշողության ինչքան մասն ենք ցանկանում

հաշվի առնել: r_t-ն ստացվում է sigmoid ակտիվացիոն ֆունկցիայի միջոցով և ինչքան մոտ է 1-ի, այնքան ավելի շատ ենք ցանկանում հաշվի առնել հիշողությունը: GRU բլոկի հայտնվելուց առաջ եղել է Mini GRU բլոկը, որը հենց output-ում վերադարձնում էր $\tilde{h_t}$-ն: Իսկ ինչո՞ւ ենք վերջում բազմապատկում z_t-ով: RNN-ի խնդիրն այն էր, որ մենք հիշողությունը բազմապատկում էինք w-ով և այդ պատճառով սկզբի բառերը հաշվի չէինք առնում որոշում կայացնելիս: Այստեղ հիշողության հետ ոչ մի փոփոխություն չենք կատարում, միայն բազմապատկում ենք $(1 - z_t)$-ով, որը ցույց կտա, թե ինչքան է պետք հաշվի առնել նախորդ տվյալների (բառերի) history-ին: z_t-ն ուսուցանվում է և ցանցը ինքն է որոշելու ամեն տվյալի մուտքում ինչքան ուշադրություն դարձնի հիշողությանը, ինչքան նոր եկած տվյալին: W_z, W_r, W կշիռների չափերը պետք է նույնը լինեն, որպեսզի կարողանանք գործողությունները կատարել: r_t-ն բազմապատկվում է h_{t-1}-ով, $(1 - z_t)$-ն բազմապատկվում է h_{t-1}-ով այդ պատճառով W_z-ի և W_r-ի չափերը պետք է նույնը լինեն: W_z-ը և W_r-ը բազմապատկվում են $[h_{t-1}, x_t]$-ով: $r_t * h_{t-1}$-ը սկալյար բազմապատկում է և չափը չի փոփոխվում (առաջին էլեմենտները հերթով բազմապատկվում են երկրորդ վեկտորի էլեմենտներով): W-ն բազմապատկվում է $[r_t * h_{t-1}, x_t]$, դրանից էլ կհետևի որ W_z, W_r, W կշիռների չափերը նույնն են:
Նորից ֆիքսենք, որ հիշողությունը չենք բազմապատկում w-ով, միայն անհրաժեշտության դեպքում բազմապատկում ենք 0-ից 1 թվով: Ինչքան շատ հաշվի ենք առնում հիշողությունը, այնքան քիչ հաշվի ենք առնում տվյալ պահին եկած տվյալը և հակառակը: Դա տեղի է ունենում այն պատճառով, որ գործակիցների գումարը պետք է լինի 1:

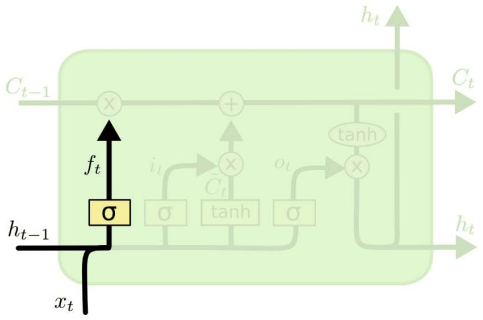
3 LSTM

GRU-ի նման գոյություն ունի ևս մի բլոկ, որը կոչվում է LSTM (Long Short Term Memory): Դիտարկենք LSTM-ի կառուցվածքը (Նկար 2):



Նկար 2: LSTM բլոկ

Սա առաջին recurrent ցանցն է, որը ունի երկու history: Հերթով դիտարկենք LSTM բլոկի ամեն հատված:

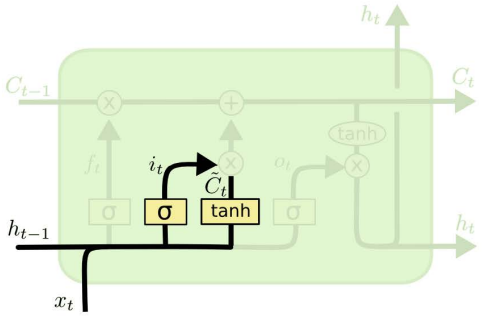


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Քայլ 1

Այստեղ նույնպես history-ին և տվյալ պահի տվյալը բազմապատկում ենք կշռով, գումարում

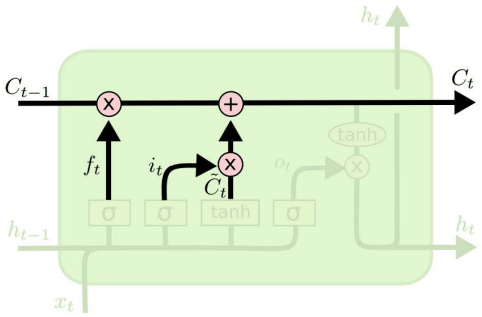
բայես և կիրառում sigmoid ակտիվացիոն ֆունկցիա: Արդյունքը վերագրում f_t փոփոխականին:



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Քայլ 2

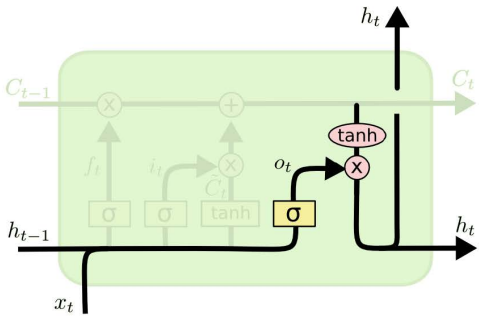
Հաջորդ քայլով նորից բազմապատկում ենք կշռով, գումարում բայես և կիրառում sigmoid ակտիվացիոն ֆունկցիա: Արդյունքը վերագրում i_t փոփոխականին: Հետո նույն գործողությունը կիրառում ենք, բայց արդեն tanh ակտիվացիոն ֆունկցիայով և վերագրում $\tilde{C}_t$ փոփոխականին:



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Քայլ 3

Հաջորդ քայլով հաշվի ենք առնում նախորդ history-ին f_t չափով, իսկ ներկայիս տվյալը i_t չափով: f_t-ն անվանում են forget gate (մոռացման դարպաս), իսկ i_t-ն input gate (մուտքային դարպաս): C_t-ն պարունակում է ինֆորմացիա և՛ հիշողության մասին, և՛ նոր եկած տվյալի և ինքն է որոշում, թե որին ինչքան ուշադրություն դարձնի: Ի տարբերություն GRU-ի, այստեղ կարող ենք երկուսին էլ շատ ուշադրություն դարձնել, քանի որ նրանց գումարը պարտադիր չէ, որ լինի 1: Այստեղ նույնպես հիշողությունը չենք բազմապատկում w կշռով, այլ բազմապատկում ենք 0-ից 1 թվով: Իսկ ինչո՞ւ ենք $\tilde{C}_t$-ն հաշվելիս օգտագործում tanh ակտիվացիոն ֆունկցիա: Դա այն պատճառով է, որպեսզի կարողանանք C_{t-1} փոքրացնել: Եթե C_{t-1}-ը միշտ բազմապատկեինք զրոյից մեկ թվով և գումարեինք դրական թիվ, այն գնալով կմեծանար: Այդ պատճառով C_{t-1}-ին գումարում ենք -1-ից 1 միջակայքի թիվ:



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Քայլ 4

Հաջորդ քայլով հաշվում ենք երկրորդ հիշողությունը։ Դրա համար հաշվում ենք output gate-ը (ելքային դարպաս), որը նորից կշիռով բազմապատկում է history-ին և մուտքային տվյալը, գումարում բայես և կիրառում sigmoid ակտիվացիոն ֆունկցիա։ Ստացված արժեքը բազմապատկում ենք $tanh(C_t)$-ի հետ և վերադարձնում, որպես output և history։
Այսպիսով և՛ GRU բլոկը, և՛ LSTM բլոկը փորձում են լուծել vanishing gradient-ի խնդիրը, որը առաջանում է սովորական RNN ցանցերում և սկզբնական տվյալների մասին ինֆորմացիան այնքան է ձևափոխվում, որը output-ի վրա ազդեցություն չի ունենում։ Բայց vanishing gradient-ի խնդիրը վերջնականապես չի լուծվում և դրա լուծումը կտեսնենք ուշադրության մոդելներում (attention models)։