Անդրադարձ նեյրոնային ցանցեր

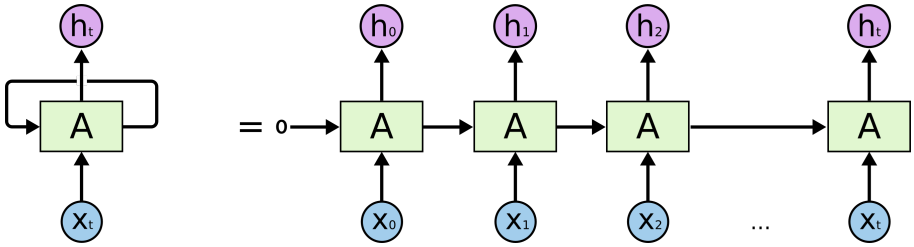
Հայկ Կարապետյան

1 Հաջորդական տվյալներ

Ծանոթանանք տվյալների մի տեսակի հետ, որը կոչվում է հաջորդական տվյալներ (sequential data)։ Դա նշանակում է, որ ամեն նոր տվյալ ինչ որ կերպ կապ ունի նախորդ տվյալների հետ։ Օրինակ՝ տեքստ, խոսք, ձայնագրություն, տեսանյութ։ Տեսանյութը կարող ենք դիտարկել, որպես նկարների հաջորդականություն։ Մարդու աչքը վարկյանում կարողանում է ընկալել միջինում 24 կադր (24 FPS[1])։ Նկարը նույնպես կարող ենք ներկայացնել, որպես հաջորդական տվյալ։ Կարող ենք նկարի ամեն մի տող դիտարկել, որպես մեկ տվյալ և կստացվի պիքսելների խմբերի հաջորդականություն։ Կարո՞ղ ենք կիրառել dense շերտեր, հաջորդական տվյալների հետ։ Ունենք հետևյալ խնդիրը։ Մուտքում ստանում ենք նախադասություն և պետք է թարգմանենք այդ նախադասությունը։ Նախադասության ամեն անդամի (բառ, կետադրման նշաններ) վրա կարող ենք կիրառել word2vec ալգորիթմը, այն վեկտոր դարձնելու համար։ Եթե նախադասության մեջ ունենանք 10 անդամ և մեր word2vec ալգորիթմը ամեն անդամի համար վերադարձնի 100 երկարությամբ վեկտոր, կարող ենք ստացված վեկտորները միացնել իրար, ստանալ 1000 երկարությամբ վեկտոր և դա մուտքում տալ dense շերտին։ Ի՞նչն է խնդիրը։ Եթե ցանցը ունենա 15 անդամից բաղկացած մուտք, այն չի կարողանա ստացված 1500 երկարությամբ վեկտորը անցկացնել dense շերտի միջով, քանի որ dense շերտի կշիռների չափը ֆիքսված է։ Սա առաջին թերությունն է հաջորդական տվյալների հետ dense շերտեր օգտագործելու։ Երկրորդ թերությունն այն է, որ մենք հաշվի չենք առնում անդամների հերթականությունը։ Այսինքն նախադասության թարգմանությունը վերադարձնելիս՝ այն պետք է սկսի թարգմանել առաջին բառից սկսած, ոչ թե ինչ որ պատահական կերպով։

2 Անդրադարձ ցանցեր

Հաջորդական տվյալների դեպքում կիրառում ենք անդրադարձ նեյրոնային ցանցեր (Recurrent Neural Network, կրճատ՝ RNN։ Նկար 1)։

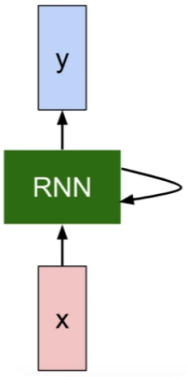


Նկար 1։ Հասարակ RNN

Նկար 1-ում կարող ենք տեսնել հասարակ RNN-ի օրինակ։ x_t մեր մուտքային հաջորդական տվյալներն են։ Օրինակ ունենք տեքստ կազմված (t+1) անդամից։ Ամեն անդամը անցկացնում ենք A բլոկի միջով։ A բլոկը իրենից ներկայացնում է նեյրոնային պարզ ցանց։ Այն կարող է բաղկացած լինել մեկ dense շերտից և ունենալ կամայական ակտիվացիոն ֆունկցիա։ Ամեն քայլի A բլոկին մուտքում տալիս ենք երկու արժեք և այն վերադարձնում է երկու արժեք։ Առաջին քայլում բլոկին փոխանցում ենք x_0-ն և 0-ական վեկտոր, դրանք միացնում ենք իրար (concatenate), անցկացնում մի շերտի միջով ։ Երկրորդ քայլին փոխանցում ենք x_1-ը և h_0-ն։ Ամեն քայլին $h_t = W[x_t, h_{t-1}]$ և հաջորդին բլոկին, որպես մուտք տալիս ենք x_{t+1}-ն և h_t-ը։

1. FPS (Frames Per Second) = Մի վարկյանում կադրերի քանակ

Ամեն բլոկ վերադարձնում է երկու output, որոնք իրար հավասար են և հավասար են h_t-ի: h_t-ն կոչվում է history: Այսինքն այն ինֆորմացիա է պարունակում հաջորդական տվյալների նախորդ անդամների մասին: Առաջին քայլում history-ն 0 է: Այսինքն RNN-ի կառուցվածքը կարողանում է լուծել dense շերտերի ժամանակ առաջացող խնդիրները՝ բոլոր բառերի համար ունենում ենք նույն բլոկը և ինչքան բառերի քանակը շատացնենք այնքան բլոկերի քանակը կշատանա և երկրորդ խնդիրը, երբ անդամների հերթականությունը նշանակություն ունի: Սկզբից RNN բլոկին մուտքում տալիս ենք առաջին անդամ, հետո երկրորդ անդամը և առաջինից ստացված history-ին, մինչև t-րդ անդամը և (t-1)-րդ history-ն:
Ամեն բլոկ նաև կարող է վերադարձնել տարբեր՝ իրար ոչ հավասար output-ներ (Նկար 2):



$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Նկար 2: Երկու output ունեցող RNN, որոնք իրարից տարբերվում են

Ամեն քայլի RNN բլոկին փոխանցում ենք այդ պահին հաջորդական անդամը (x_t) և նախորդ տվյալների history-ն (h_{t-1}): Կարող ենք ասել, որ RNN բլոկը ֆունկցիա է, որը մուտքում ընդունում է history-ն և տվյալ պահի անդամը և վերադարձնում է այդ պահին output-ը (թարգմանված բառը՝ y_t) և նոր հիշողությունը (h_t):

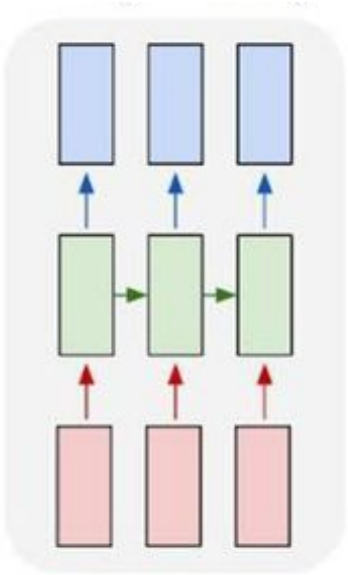
$$y_t, h_t = F(x_t, h_{t-1})$$

Նկար 2-ում պատկերված ցանցը, history-ն վերադարձնելու համար տվյալ անդամը և նախորդ history-ն բազմապատկում է կշիռներով և կիրառում տանգես հիպերբոլական ակտիվացիոն ֆունկցիա: Վերջնական output-ը ստանալու համար նոր հիշողությունը բազմապատկում է կշիռով: Այստեղ կարող ենք նաև տեսնել, որ history-ն և x_t-ն չեն միավորվում և դառնում մի վեկտոր, այլ ամեն մեկը առանձին առանձին բազմապատկում ենք W-ով և գումարում իրար: Այսինքն մուտքային history-ի և x_t-ի հետ կարող ենք կատարել ցանկացած գործողություն, միայն թե վերադարձնենք երկու output: h_t-ն ստանալիս, անպայման անհրաժեշտ է օգտագործել x_t-ն, որպեսզի history-ն ինֆորմացիա պարունակի նաև այս պահի տվյալի մասին:

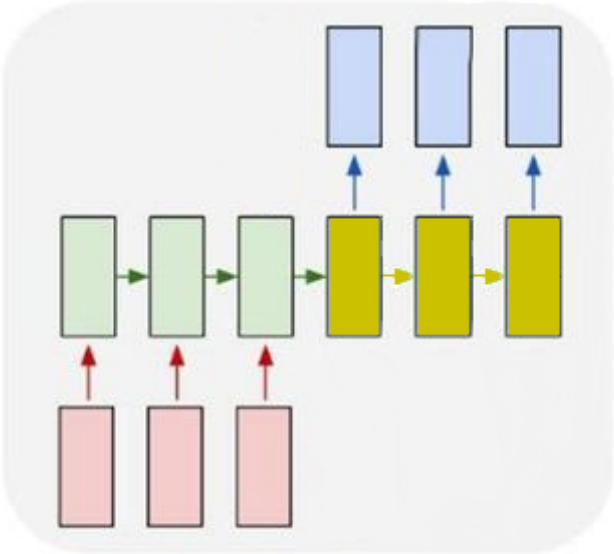
3 RNN-ի տեսակներ

3.1 Many to many

Այս RNN ցանցի դեպքում մենք ունենք հաջորդական տվյալներ և դրանք ցանցի միջով անցկացնելիս ցանկանում ենք նորից ստանալ հաջորդական տվյալներ: Օրինակ ունենք հայերեն տեքստ և այն ցանկանում ենք թարգմանել հայերեն: Ե՛վ մուտքային տվյալները, և՛ ելքային տվյալները հաջորդական են: Many to many RNN կարող ենք իրականացնել երկու եղանակով: Առաջին եղանակը պատկերված է նկար 3-ում: Այս դեպքում ամեն բառի թարգմանության համար օգտվում ենք այդ պահի բառից և նրա նախորդ բառերի history-ից: Այսինքն առաջին բառի թարգմանության ժամանակ ինֆորմացիա չենք ունենում երկրորդ կամ երրորդ բառերի մասին, որը այնքան էլ լավ չէ: Երկրորդ եղանակը պատկերված է նկար 4-ում:



Նկար 3: Ամեն հաջորդական տվյալ մուտք է բլոկի համար և ամեն բլոկ վերադարձնում է ուրիշ հաջորդական տվյալներ



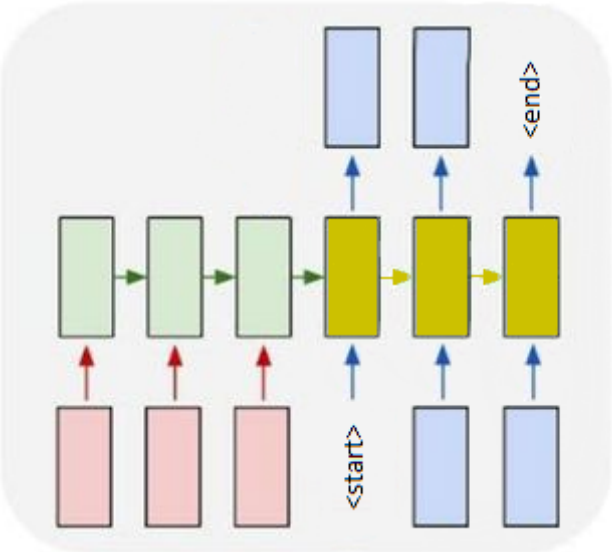
Նկար 4: Ամեն ելքային բլոկ ինֆորմացիա ունի բոլոր մուտքային տվյալների մասին

Այս դեպքում արդեն առաջին բառի թարգմանության համար օգտագործվել է բոլոր բառերի մասին ինֆորմացիան: History-ին իր մեջ ինֆորմացիա է պարունակում բոլոր բառերի մասին: Առաջին երեք բլոկի համար ամեն ինչ պարզ է՝ առաջին բլոկի համար history-ին զրոյական վեկտոր է և ստանում է մուտքում առաջին տվյալը (բառը), իսկ մնացած երկու բլոկը ունեն իրենց համապատասխան մուտքային տվյալները և history-ին: Կանաչ բլոկերը կարող են վերադարձնել output, բայց դրանք մեզ հետաքրքիր չեն, քանի որ չեն պարունակում ինֆորմացիա բոլոր բառերի մասին (բացառությամբ վերջին բլոկի): Դեղին բլոկերի և կանաչ բլոկերի ցանցը կարող է տարբերվել: Ցանցի կառուցվածքը (architecture) կարող ենք թողնել նույնը, բայց կշիռները տարբեր են: Կանաչ բլոկերը մուտքում ընդունելու են անգլերեն բառերը, իսկ դեղին բլոկերը թարգմանելու են հայերեն: Դեղին բլոկերը մուտքում ստանում են միայն history և չունեն մուտքային հաջորդական տվյալներ: Երկրորդ դեղին բլոկը միայն history-ի միջոցով գիտի, թե նախորդ բլոկը ինչ բառ է թարգմանել: Այդ պատճառով բացի նախորդ բլոկի history-ին փոխանցելուց, մուտքում տալիս ենք նաև նախորդ բլոկի output-ը: Այսինքն 1-ին դեղին

բլոկի կապույտ output-ը փոխանցելու ենք 2-րդ բլոկին, որպես մուտք: 3-րդի համար արդեն կփոխանցենք 2-րդ բլոկի output-ը: Առաջանում է երկու հարց.

1. Ի՞նչ է լինելու առաջին դեղին բլոկի մուտքը
2. Ե՞րբ ենք դադարելու նախադասության թարգմանություն

Երկու խնդիրներն էլ լուծելու ենք հատուկ մասնիկների (special tokens) միջոցով: Tokenizer-ը նախադասությունը բաժանում էր մասնիկների և ուներ ֆիքսված բառարան: Ավելացնենք այդ բառարանի մեջ երկու մասնիկ՝ <start>, <end>: Առաջին նախադասությունը ցույց է տալիս, որ պետք է սկսել նախադասությունը թարգմանել: Այս token-ը փոխանցելու ենք առաջին դեղին բլոկին՝ վերջին կանաչ բլոկի history-ի հետ միասին: Իսկ երբ պետք է դադարենք թարգմանությունը ցույց է տալիս <end> token-ը: Մեր դեղին բլոկերը վերադարձնում են (vocab_size) երկարությամբ վեկտոր: Այդ վեկտորը հավանականային վեկտոր է և ամեն ինդեքսի արժեք ցույց է տալիս, թե ինչքան հավանականությամբ է բառի թարգմանությունը բառարանի նույն ինդեքսով բառը: Օրինակ՝ [0.3, 0.5, 0.1, ...] հավանականային վեկտորը ցույց է տալիս, որ ամենահավանական թարգմանությունը բառարանի երկրորդ բառն է: Երբ ամենամեծ հավանականությունը ունենա <end> token-ի ինդեքսը, այդ դեպքում թարգմանությունը կդադարեցնենք (Նկար 5):



Նկար 5: Ամեն ելքային բլոկ ինֆորմացիա ունի բոլոր մուտքային տվյալների մասին

Հիմա դիտարկենք այս ցանցը ուսուցանման և թեստավորման ժամանակ թարգմանության խնդրի վրա:
Մուտքային նախադասությունը հետևյալն է՝ "I study DL", իսկ համապատասխան թարգմանությունը՝ "Ես սովորում եմ DL": Անգլերեն նախադասությունը անցկացնելու ենք tokenizer-ի միջով, ստանանք մասնիկները՝ ["I", "study", "DL"], ամեն մասնիկի համապատասխանեցնենք one-hot վեկտոր և փոխանցենք կանաչ բլոկներին: Վերջին կանաչ բլոկի history-ին և <start> token-ի one-hot վեկտորը փոխանցելու ենք առաջին դեղին բլոկին: Դրանից դուրս եկած output-ը համեմատելու ենք (cross-entropy loss) իրական label-ի հետ: Իրական label-ը "Ես" բառի one-hot վեկտորն է, իսկ գուշակությունը նույն չափի հավանականային վեկտոր: Ուսուցման ժամանակ, քանի որ գիտենք իրական թարգմանությունը, ամեն հաջորդ դեղին բլոկին փոխանցելու ենք իրական նախորդ բառը: Օրինակ ուսուցման ընթացքում առաջին դեղին բլոկը գուշակել է [0, 0, 0, 1, 0] վեկտորը, բայց իրականը [0, 1, 0, 0, 0] վեկտորն է: Երկրորդ դեղին բլոկին փոխանցելու ենք իրական վեկտորը: Նույն կերպ իրական թարգմանության վերջին բառը փոխանցելիս ստացված output-ը համեմատելու ենք <end> token-ի հետ և ավարտենք այդ տվյալով (նախադասությամբ) ուսուցումը, անցնենք մյուս տվյալներին (նախադասություններին): Ուսուցման բոլոր տվյալների սկզբից ավելացնելու ենք <start> token-ը, իսկ վերջից <end> token-ը:

["Ես", "սովորում", "եմ", "DL"] → [<start>, "Ես", "սովորում", "եմ", "DL", <end>]

Թեստավորման ընթացքում առաջին դեղին բլոկին փոխանցելու ենք <start> token-ը և ամեն անգամ դուրս եկած output-ը փոխանցելու ենք հաջորդ դեղին բլոկին։ Այս գործողությունը կատարելու ենք այնքան ժամանակ, մինչև output-ի մեջ <end> token-ի հավանականությունը լինի ամենամեծը։

3.2 Many to one

Այս RNN-ի դեպքում մուտքում ունենալու ենք հաջորդական տվյալներ և արդյունքում վերադարձնելու ենք մի թիվ (վեկտոր)։ Օրինակ ցանկանում ենք հասկանալ նորությունը դրական է, թե բացասական։ Այս դեպքում մեր label-ները 0 և 1 են։ Մուտքային տվյալը տեքստային նորությունն է, իսկ label-ը 0 եթե բացասական նորություն է և 1 եթե դրական է։ Tokenizer-ը կիրառելու ենք ամբողջ տեքստի վրա և ստացված token-ները փոխանցելու ենք ցանցին։ Լավ կամ վատ հասկանալու խնդիրը կոչվում է զգացմունքային վերլուծություն (sentiment analysis)։

3.3 One to many

Այս RNN-ի դեպքում մուտքում ունենք մի թիվ կամ վեկտոր և արդյունքում ցանկանում ենք ստանալ հաջորդական տվյալներ։ Օրինակ ցանցին մուտքում փոխանցելիս մարդու անունը՝ ստանանք աքրոստիկոս։ One to one RNN-ը համարժեք է սովորական dense շերտին։
Տեքստերի հետ փորձարկումներ կարող եք կատարել aitestkitchen.withgoogle.com կայքում։