Դևիսի և Փաթնեմի մեթոդը (ej 69)

Ենթադրենք՝ -ը դիզյունկտների բազմություն է։ Մեթոդը, ըստ էության, բաղկացած է հետևյալ չորս կանոններից`

1. *Տավտոլոգիայի կանոն՝* -ից ջնջում ենք բոլոր տավտոլոգիա հիմնական դիզյունկտները։ Մնացած բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե -ը անհամատեղելի է:
2. *Մեկ լիտերալ դիզյունկտների կանոն՝* եթե -ում գոյություն ունի մեկ լիտերալ պարունակող հիմնական դիզյունկտ , ապա -ը ստացվում է -ից՝ ջնջելով այն հիմնական դիզյունկտները, որոնք պարունակում են : Եթե -ը դատարկ է, ապա -ը համատեղելի է: Հակառակ դեպքում, կառուցում ենք -ը՝ -ից ջնջելով -ի մուտքերը: -ը անհամատեղելի է, այն և միայն այն դեպքում, եթե -ը նույնպես անհամատեղելի է: Նշենք, որ եթե -ը մեկ լիտերալ հիմնական դիզյունկտ է, ապա այն ջնջելիս կվերածվի -ի։
3. *Մաքուր լիտերալների կանոն`* -ի հիմնական դիզյունկտում գտնվող լիտերալը կոչվում է *մաքուր*-ում, այն և միայն այն դեպում, եթե -ը չի հանդիպում -ի որևէ հիմնական դիզյունկտում: Եթե -ը մաքուր լիտերալ է, ապա ջնջում ենք բոլոր հիմնական դիզյունկտները, որոնք պարունակում են ։ Մնացած բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե -ը անհամատեղելի է:
4. *Բաժանման կանոն`* եթե  բազմությունը կարելի է ներկայացնել հետևյալ տեսքով՝ , որտեղ -ի​ն և -ը ազատ են -ից և -ից, ապա ստանում ենք երկու բազմություն՝  *և* , -ը անհամատեղելի է, այն և միայն այն դեպքում, երբ *-*ը անհամատեղելի է, այսինքն՝ և -ը, և ​-ը անհամատեղելի են:

Վերոհիշյալ կանոնները շատ կարևոր են: Հետագայում կտեսնենք, որ այս կանոններն ունեն ավելի լայն կիրառություն: Բերենք օրինակներ՝ այս կանոնների օգտագործումը ցույց տալու համար:

Օրինակ՝ ցույց տանք, որ -ը անհամատեղելի է:

Քանի, որ վերջնական բանաձևը պարունակում է դատարկ դիզյունկտ ,ապա -ը անհամատեղելի է:

Օրինակ՝ ցույց տանք, որ -ը համատեղելի է:

Քանի որ բաժանման երկու բազմություններն էլ համատեղելի են, ապա -ը նույնպես համատեղելի է:

Օրինակ՝ ցույց տանք, որ -ը համատեղելի է:

Այսպիսով -ը համատեղելի է։

Ռեզոլյուցիայի մեթոդը տրամաբանակ արտահայտություններում 5.2(ej 77)

Ռեզոլյուցիայի մեթոդը, ըստ էության, Դևիսի և Փաթնեմի մեկ լիտերալ դիզյունկտների կանոնի ընդհանրացումն է, որը տրված է 4-րդ գլխի § 4.6-ում:

Օրինակ դիտարկենք հետևալ դիզունկտները՝

Օգտագործելով մեկ լիտերալ դիզունկտների կանոնը, -ից և -ից մենք կարող ենք ստանալ նոր դիսյունկտ

Մեկ լիտերալ դիզյունկտների կանոնը մեզ անհրաժեշտ է, որպեսզի նախ որոշենք, արդյոք կա լիտերալների հակադիր զույգ (օրինակ՝ ) -ում և (օրինակ՝  ) -ում, ապա ջնջենք այդ զույգը -ից և -ից, որպեսզի ստանանք նոր դիզյունկտ , որը -ն է:

Վերոհիշյալ կանոնը ընդհանրացնելով և այն կիրառելով դիզյունկտների ցանկացած զույգի նկատմամբ (ոչ պարտադիր միայն մեկ լիտերալ պարունակող), մենք ստանում ենք հետևյալ կանոնը, որը կանվանենք ***ռեզոլյուցիայի կանոն***:

Ցանկացած երկու դիզյունկտների համար՝ և , եթե գոյություն ունի ​լիտերալ -ում, որը հակադիր է ​ լիտերալին -ում, ապա ջնջելով ​-ը -ից և ​-ը -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան: Ստացված դիզյունկտը կոչվում է -ի և -ի *ռեզոլվենտ*:

Օրինակ դիտարկենք հետևյալ դիզյունկտները՝

-ը պարունակում է լիտերալ, որը հակադիր է -ում գտնվող լիտերալին: Ուստի, ջնջելով -ն -ից և -ն -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան`, ստացված ռեզոլվենտը կլինի ։

Ռեզոլվենտի կարևոր հատկությունն այն է, որ ցանկացած ռեզոլվենտ, որը ստացվում է երկու դիզյունկտներից՝  և , -ի և -ի տրամաբանական հետևանքն է: Այս հատկությունը հաստատվում է հետևյալ թեորեմով`

Թեորեմ 5.1: Եթե տրված են երկու դիզյունկտներ՝ և , ապա -ի և -ի ռեզոլվենտը -ն -ի և -ի տրամաբանական հետևանքն է:

Ապացույց՝ ենթադրենք , և , որտեղ և -ը լիտերալների դիզյունկցիաներ են: Ենթադրենք, որ -ը և -ը ճշմարիտ են ինտերպրետացիայում: Մենք ցանկանում ենք ապացուցել, որ -ի և -ի ռեզոլվենտը՝ -ն, նույնպես ճշմարիտ է -ում: Ապացույցի համար նշենք, որ -ը կամ -ը կեղծ են -ում։ Եթե -ը կեղծ է -ում, ապա -ը կարող է ճշմարիտ լինել միայն այն դեպքում, եթե ​-ը ճշմարիտ է -ում: Նույն կերպ, եթե -ը կեղծ է -ում, ապա -ը կարող է ճշմարիտ լինել միայն այն դեպքում, եթե -ը ճշմարիտ է -ում։ Ոեզոլվենտը՝ ​, կլինի ճշմարիտ -ում, եթե ​-ը կամ -ը ճշմարիտ է -ում: Քանի որ ​-ը կամ -ը պետք է ճշմարիտ լինեն -ում, ապա -ն նույնպես ճշմարիտ է-ում։ Դա այն է, ինչ պետք էր ապացուցել:

**Սահմանում`** Ենթադրենք՝ -ը դիզյունկտների բազմություն է: -ից -ի ռեզոլյուցիոն արտածումը դիզյունկտների վերջավոր հաջորդականություն է՝ որտեղ յուրաքանչյուր ​-ն կամ պատկանում է -ին, կամ նախորդ դիզյունկտների ռեզոլվենտն է, և : -ից (դատարկ դիզյունկտ) արտածումը կոչվում է -ի հերքում (կամ -ի անհամատեղելիության ապացույց):

Մենք ասում ենք, որ դիզյունկտը կարող է արտածվել կամ ստացվել -ից, եթե գոյություն ունի -ի արտածում -ից:

Օրինակ դիտարկենք բազմություն՝

(1)-ից և (2)-ից կարող ենք ստանալ ռեզոլվենտ՝ : (4)-ից և (3)-ից կարող ենք ստանալ ռեզոլվենտ՝ ։ Քանի, որ -ն ստացվում է -ից ռեզոլյուցիայի կանոնի կիրառմամբ, ապա համաձայն Թեորեմ 5.1-ի, -ը -ի տրամաբանական հետևանքն է: Ուստի, -ը անհամատեղելի է։

§ 5.3. (ej 79) Փոխարինում և ունիֆիկացիա

Մենք դիտարկեցինք *ռեզոլյուցիայի մեթոդը* տրամաբանական արտահայտությունների համար: Հաջորդիվ մենք այն կտարածենք *առաջին կարգի տրամաբանության* վրա: Նշել ենք, որ ռեզոլյուցիայի կանոնի կիրառման հիմնական պահը հակադիր լիտերալների գտնելն է երկու դիզյունկտներում: Երբ դիզյունկտները չեն պարունակում փոփոխականներ, ապա դա շատ պարզ է: Սակայն, երբ դիզյունկտները պարունակում են փոփոխականներ, ապա խնդիրը բարդանում է: Օրինակի համար դիտարկենք հետևյալ դիզյունկտները՝

Չկա որևէ լիտերալ ​-ում, որը հակադիր լինի -ի որևէ լիտերալի: Սակայն, եթե մենք ​-ում x-ը փոխարինենք -ով, իսկ ​-ում x-ը փոխարինենք -ն, ապա կստանանք՝

Գիտենք, որ ​-ը և ​-ը համապատասխանաբար ​-ի և ​-ի հիմնական օրինակներն են, իսկ -ն և -ն հակադիր են միմյանց: Ուստի, -ից և ​-ից մենք կարող ենք ստանալ ռեզոլվենտ՝

Ընդհանուր դեպքում, եթե ​-ում x-ը փոխարինենք -ով, ապա կստանանք՝

Կրկին ​-ը ​-ի օրինակ է: Միևնույն ժամանակ, ​-ում -ը հակադիր է ​-ում -ին: Ուստի, մենք կարող ենք ստանալ ռեզոլվենտ ​-ից և ​-ից:

-ը ​-ի օրինակ է: Փոփոխականները ​-ում և -ում համապատասխան թերմերով փոխարինելով, ինչպես նշված է վերևում, մենք կարող ենք ստեղծել նոր դիզյունկտներ ​-ից և ​-ից: Բացի այդ, ​-ը *ամենաընդհանուր դիզյունկտն* է այն իմաստով, որ վերը նշված գործընթացով ստացված բոլոր այլ դիզյունկտները -ի օրինակներ են: ​-ը նույնպես կանվանենք -ի և ​-ի ռեզոլվենտ:

**Սահմանում`** *փոխարինումը* (подстановка) վերջավոր բազմություն է՝ , որտեղ՝ յուրաքանչյուր -ն փոփոխական է, յուրաքանչյուր -ն թերմ է, որը տարբերվում է -ից, բոլոր ​-երը տարբեր են: Եթե -ը հիմնական թերմեր են (այսինքն՝ չեն պարունակում փոփոխականներ), ապա փոխարինումը կոչվում է հիմնական փոխարինում: Փոխարինումը, որը չի պարունակում որևէ տարր, կոչվում է դատարկ փոխարինում և նշանակվում է -ով: Փոխարինումը գրելու համար մենք կօգտագործենք հունարեն տառեր (օրինակ՝ ):

Օրինակ հետևալ երկու բազմությունները հանդիսանում են փոխարինում՝

**Սահմանում`** ենթադրենք -ը փոխարինում է, և -ն արտահայտություն է: Այդ դեպքում -ն արտահայտություն է, որը ստացվում է -ից՝ -ում *​-*ի բոլոր հանդիպումները միաժամանակ փոխարինելով -ով: -ն կոչվում է -ի *օրինակ*: (Նշենք, որ օրինակի այս սահմանումը համատեղելի է գլուխ 4-ում տրված դիզյունկտի հիմնական օրինակի սահմանման հետ:)

Օրինակ՝ ենթադրենք և: Այդ դեպքում։

**Սահմանում` ե**նթադրենք և երկու փոխարինումներ են: Այդ դեպքում -ի և -ի **կոմպոզիցիան** (նշանակում ենք ) այն փոխարինումն է, որը ստացվում է հետևալ բազմությունից՝

Ջնջելով բոլոր այն -երը որոնց համար , և բոլոր -երը որոնց համար (այսինքն՝  ​-ն արդեն առկա է -ում)։

Օրինակ ՝ ենթադրենք

Այդ դեպքում՝ ։ Սակայն, քանի որ (այսինքն ), ապա պետք է հեռացնել բազմությունից։ Բացի այդ, քանի որ -ը և ​-ը առկա են -ում, ապա ​-ը և ​-ը (այսինքն՝  -ը և -ը) նույնպես պետք է հեռացվեն: Այսպիսով, մենք ստանում ենք՝

**Սահմանում`** փոխարինումը -ն կոչվում է ունիֆիկատոր (unifier) բազմության համար, այն և միայն այն դեպքում, երբ ։ Ասում են, որ բազմությունը *ունիֆիկացվող* է, եթե բազմության համար գոյություն ունի ունիֆիկատոր:

**Սահմանում`** ունիֆիկատոր -ն բազմության համար կոչվում է ամենաընդհանուր ունիֆիկատոր (most general unifier, MGU), այն և միայն այն դեպքում, երբ ցանկացած այլ ունիֆիկատոր -ի համար գոյություն ունի փոխարինում , այնպես որ՝ ։

Օրինակ՝ բազմությունը ունիֆիկացվող է քանի, որ հանդիսանում է ունիֆիկատոր նրա համար։

§ 5.4. Ունիֆիկացման ալգորիթմ (ej 82)

Այս պարբերությունում կներկայացնենք ունիֆիկացման ալգորիթմ, որը թույլ է տալիս գտնել ամենաընդհանուր ունիֆիկատորը վերջավոր ունիֆիկացվող բազմության համար: Եթե բազմությունը չի ունիֆիկացվում, ալգորիթմը կհայտնաբերի նաև այդ փաստը:

**Սահմանում`** ոչ դատարկ արտահայտությունների բազմության -ի անհամապատասխանությունների բազմությունը ստացվում է գտնելով առաջին (ձախից) դիրքը, որտեղ -ի բոլոր արտահայտությունները չունեն նույն սիմվոլը, այնուհետև յուրաքանչյուր արտահայտությունից դուրս գրելով այն ենթաարտահայտությունը, որը սկսվում է այդ դիրքում գտնվող սիմվոլից։ Այս ենթաարտահայտությունների բազմությունը կոչվում է -ի *անհամապատասխանությունների բազմություն*:

Օրինակ՝ եթե -ն հետևյալ բազմությունն է՝ , ապա առաջին դիրքը, որտեղ -ի բոլոր արտահայտությունները չունեն նույն սիմվոլը, հինգերորդ դիրքն է, քանի որ բոլոր արտահայտությունները ունեն նույն առաջին չորս սիմվոլները՝ : Այսպիսով, անհամապատասխանությունների բազմությունը բաղկացած է համապատասխան ենթաարտահայտություններից, որոնք սկսվում են հինգերորդ դիրքից, և դա հետևյալ բազմությունն է՝ ։

*Ունիֆիկացման ալգորիթմ`*

*Քայլ* 1. Բազմություններ ։

*Քայլ* 2. Եթե -ն միալիտերալ դիզյունկտ է, ապա -ն -ի *ամենաընդհանուր ունիֆիկատորն* է: Հակառակ դեպքում, գտնել -ի անհամապատասխանությունների բազմությունը​:

*Քայլ* 3. Եթե -ում գոյություն ունեն և տարրեր, այնպիսին որ -ն փոփոխական է, որն չի հայտնվում -ում, ապա անցնել քայլ 4-ին: Հակառակ դեպքում, ավարտել՝ -ն չի ունիֆիկացվում:

*Քայլ* 4. Սահմանել և : (Նշենք, որ )։

*Քայլ* 5. -ին վերագրել  արժեքը և անցնել քայլ 2-ին:

Օրինակ՝ գտնել ամենաընդհանուր ունիֆիկատորը

1. և ։ Քանի, որ ​-ն միալիտերալ դիզյունկտ չէ, ուստի ​-ն -ի ամենաընդհանուր ունիֆիկատոր չէ:
2. Անհամապատասխանությունների բազմությունը՝ : -ում գոյություն ունի փոփոխական : որը չի հանդիպում *-*ում։
3. Սահմանենք՝

,

1. ​-ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն -ի համար։
2. -ից կգտնենք և *։*
3. Սահմանենք՝

,

1. -ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն -ի համար։ : -ից կգտնենք և :
2. Սահմանենք՝

,

1. Քանի, որ​-ը միալիտերալ դիզյունկտ է, ապա -ն -ի ամենաընդհանուր ունիֆիկատորն է:

Թեորեմ 5.2 (Ունիֆիկացման թեորեմ)՝ եթե-ն վերջավոր ոչ դատարկ ունիֆիկացվող արտահայտությունների բազմություն է, ապա ունիֆիկացման ալգորիթմը միշտ կավարտվի քայլ 2-ում, և վերջին -ն կլինի -ի ամենաընդհանուր ունիֆիկատորը:

§ 5.5. Ռեզոլյուցիայի մեթոդ առաջին կարգի տրամաբանական արտահայտությունների համար (ej 85)

Նախորդ պարբերությունում ներկայացված ունիֆիկացման ալգորիթմի շնորհիվ մենք կարող ենք այժմ դիտարկել առաջին կարգի տրամաբանության համար ռեզոլյուցիայի մեթոդը:

**Սահմանում`** եթե դիզյունկտ -ի երկու կամ ավելի լիտերալներ (նույն նշանով) ունեն ամենաընդհանուր ունիֆիկատոր , ապա -ն կոչվում է -ի *սոսնձում*: Եթե -ն միալիտերալ դիզյունկտ է, ապա սոսնձումը կոչվում է *միալիտերալ սոսնձում*:

Օրինակ՝ ենթադրենք : Այդ դեպքում առաջին և երկրորդ լիտերալները (ընդգծված) ունեն ամենաընդհանուր ունիֆիկատոր : Ուստի, -ը -ի սոսնձումն է:

**Սահմանում`** եթե ​-ը և ​-ը երկու դիզյունկտներ են (կոչվում են դիզյունկտ-նախադրյալներ), որոնք չունեն ընդհանուր փոփոխականներ: Թող ​-ը և ​-ը լինեն երկու լիտերալներ համապատասխանաբար ​-ում և ​-ում: Եթե -ը և ​-ն ունեն ամենաընդհանուր ունիֆիկատոր , ապա դիզյունկտը՝

կոչվում է -ի և -ի (երկուական) ռեզոլվենտ: ​-ը և ​-ը կոչվում են *կրճատվող լիտերալներ*:

Օրինակ՝ ենթադրենք , : Քանի որ -ը ներառված է և՛ ​-ում, և՛ ​-ում, մենք փոխարինում ենք ​-ում -ը -ով՝ ։ Ընտրում ենք՝ , : Քանի որ :, ապա ​-ը և ​-ն ունեն ամենաընդհանուր ունիֆիկատոր: Այսպիսով՝

Ուստի, *-*ն -ի և ​-ի երկուական ռեզոլվենտն է։ -ը և -ն կրճատվող լիտերալներն են:

https://www.phantastike.com/math/matlogika\_chang\_li/djvu/view/

**1. Ներածություն**

Հաջորդականությունից հաջորդականություն (Sequence-to-Sequence, Seq2Seq) մոդելները խորը ուսուցման կարևոր ճյուղներից են, որոնք նախատեսված են տեքստի, ձայնի կամ այլ հաջորդական տվյալների մշակման համար: Այս մոդելները լայնորեն կիրառվում են մեքենայական թարգմանության, տեքստի ամփոփման, զրույցի համակարգերի և այլ խնդիրների լուծման համար:

Seq2Seq մոդելը հիմնված է կոդավորիչ-ապակոդավորիչ (Encoder-Decoder) ճարտարապետության վրա: Կոդավորիչը մուտքագրվում է հաջորդական տվյալները և վերափոխում է դրանք ֆիքսված երկարության վեկտորի, որը պարունակում է բոլոր անհրաժեշտ ինֆորմացիան, իսկ ապակոդավորիչը այդ վեկտորի հիման վրա ստեղծում է համապատասխան ելքային հաջորդականություն: Այս մոտեցումը հնարավորություն է տալիս արդյունավետ կերպով լուծել այնպիսի խնդիրներ, որտեղ առաջնային պահանջը է հերթական ներկայացումը:

**2. Հիմնական Սկզբունքներ**

Seq2Seq մոդելի հիմնական գաղափարը հաջորդական տվյալների մեկ ձևից մյուսին փոխակերպումն է՝ պահպանելով դրանց բովանդակային և իմաստային կապը։ Այս մոդելը սովորաբար բաղկացած է երկու հիմնական բաղադրիչից՝ կոդավորիչից և ապակոդավորիչից։

Առաջին փուլում կոդավորիչը վերլուծում է մուտքային հաջորդականությունը (օրինակ՝ նախադասություն) և վերափոխում այն համառոտ, ֆիքսված չափսի վեկտորային ներկայացման։ Այդ վեկտորը պետք է ներառի մուտքային տեղեկության հիմնական իմաստը և կառուցվածքը։

Երկրորդ փուլում ապակոդավորիչը ստանում է այդ ներկայացումը և գեներացնում է համապատասխան ելքային հաջորդականություն՝ օրինակ թարգմանված նախադասություն կամ ամփոփում։

Մոդելի նպատակն է ապահովել մուտքի և ելքի միջև իմաստային համապատասխանություն՝ առանց մուտքային տվյալների կորստի։ Սա իրականացվում է ոչ միայն տվյալների վերափոխման միջոցով, այլ նաև՝ խորքային վերլուծությամբ, որը թույլ է տալիս մոդելին հաշվի առնել ամբողջ կոնտեքստը և շարունակաբար բարելավել արդյունքի ճշգրտությունը։

Այս մոտեցումը թույլ է տալիս մոդելին արդյունավետ սովորել և գեներացնել ճիշտ շարքեր՝ նվազեցնելով սխալների հավանականությունը և բարձրացնելով գեներացված տեքստի որակը։

**3. Մոդելի ճարտարապետությունը**

**3.1. Կոդավորիչ (Encoder)**

Կոդավորիչը հանդիսանում է մոդելի առաջին փուլը, որը պատասխանատու է մուտքային հաջորդականության վերլուծության և դրա իմաստային ներկայացման համար։ Այն ընդունում է մուտքային շարքի տարրերը (օրինակ՝ բառեր կամ սիմվոլներ) և յուրաքանչյուրը վերափոխում է վեկտորային ներկայացման՝ օգտագործելով նեյրոնային ցանցեր։

Այս ներկայացումները փոխանցվում են հաջորդ շերտերին՝ աստիճանաբար հավաքագրվելով և կենտրոնացվելով վերջին թաքնված վիճակում (hidden state), որը հանդիսանում է ամբողջ մուտքի համառոտ և կոնտեքստով հարստացված ամփոփումը։

Կոդավորիչը սովորաբար կառուցվում է հետևյալ նեյրոնային ցանցերի վրա՝

* **RNN (Recurrent Neural Network)** – Հիմնական մեխանիզմ հաջորդականությունների մշակման համար։
* **LSTM (Long Short-Term Memory)** – Առավել բարելավված տարբերակ, որը կարողանում է պահպանել երկարատև կախվածություններ։
* **GRU (Gated Recurrent Unit)** – Մեկ այլ արդյունավետ տարբերակ, որն ունի LSTM-ի նման հատկություններ՝ ավելի պարզ կառուցվածքով։

Կոդավորիչի հիմնական խնդիրն է ոչ միայն վերափոխել մուտքային տվյալները, այլև պահպանել դրանց **ժամանակային համատեքստը**, **բառերի հերթականությունը**, **կախվածությունները**, ինչպես նաև պատրաստել այնպիսի ներկայացում, որը դեկոդավորիչը կարող է արդյունավետ կերպով օգտագործել ելքային հաջորդականության գեներացման համար։

Հաճախ կոդավորիչը բաղկացած է մեկից ավելի շերտերից՝ ինչը թույլ է տալիս մոդելին վերամշակել տվյալները տարբեր մակարդակներում՝ սկսած մակերեսային հատկություններից մինչև խորքային իմաստաբանական կառուցվածքներ։

Այսպիսով, կոդավորիչը հանդիսանում է հիմք ամբողջ մոդելի արդյունավետության համար, քանի որ հենց այն է ապահովում ելքային հատվածի հիմքը՝ մուտքի կոնտեքստի ներկայացման տեսքով։

**3.2. Ապակոդավորիչ (Decoder)**

Ապակոդավորիչը մոդելի երկրորդ բաղադրիչն է, որը պատասխանատու է կոդավորիչից ստացված կոնտեքստային ներկայացման հիման վրա ելքային հաջորդականության գեներացման համար։ Այն սկսում է իր աշխատանքը կոդավորիչի վերջնական թաքնված վիճակից (hidden state), որն ամփոփում է մուտքային շարքի ամբողջ տեղեկատվությունը։

Ապակոդավորիչը գեներացնում է ելքային շարքը *բառ առ բառ* կամ *սիմվոլ առ սիմվոլ* եղանակով՝ յուրաքանչյուր քայլում հաշվի առնելով ինչպես նախորդ գեներացված բառը, այնպես էլ ընդհանուր կոնտեքստը։ Այս գործընթացը ռեկուրսիվ է, քանի որ յուրաքանչյուր նոր արտադրվող տարրը կախված է նախորդներից։

Ելքային շղթայի ճշգրիտ կառուցման համար ապակոդավորիչում հաճախ օգտագործվում են հետևյալ մեխանիզմները՝

* *Ելքային պարամետրերի վերահսկում (output control)* – ապակոդավորիչը պետք է հաշվի առնի սեփական նախորդ ելքերը՝ կանխելու համար սխալների կուտակումը։
* *Teacher forcing* – ուսուցման փուլում ապակոդավորիչին հաղորդվում են իրական (ground truth) նախորդ բառերը, փոխարինելով իր կողմից գեներացվածներին, ինչը օգնում է արագ ու կայուն ուսուցման։
* *Ապակոդավորիչի վիճակի թարմացում* – յուրաքանչյուր նոր գեներացված տարր թարմացնում է ներքին վիճակը, որի հիման վրա որոշվում է հաջորդ տարրը։

Բարդ լեզվաբանական խնդիրների ժամանակ՝ ապակոդավորիչը պետք է ոչ միայն շարունակաբար վերահսկի արդյունքը, այլև ունենա հնարավորություններ՝ վերանայել և հարմարեցնել իր ներքին ներկայացումները։ Դրան աջակցելու համար սովորաբար ապակոդավորիչը աշխատում է *միջշերտային կապերով* (skip connections), *ավտոնոմ կառավարմամբ* և երբեմն *ուշադրության մեխանիզմով*(attention mechanism), որը ելքը ստեղծելիս մոդելին թույլ է տալիս կենտրոնանալ մուտքի տարբեր հատվածների վրա։

**3.3. Ուշադրության (Attention) մեխանիզմ**

Seq2Seq մոդելի զարգացման առանցքային փուլերից մեկն եղել է ուշադրության մեխանիզմի (attention mechanism) ներմուծումը, որը հնարավորություն է տալիս հաղթահարել սովորական կոդավորիչ-ապակոդավորիչ ճարտարապետության սահմանափակումները։

Երբ մուտքային հաջորդականությունը երկար է կամ բարդ, միակ վեկտորով ամբողջական իմաստի ամփոփումը դառնում է ոչ արդյունավետ։ Ուշադրության մեխանիզմը լուծում է այս խնդիրը՝ թույլ տալով, որ ապակոդավորիչը յուրաքանչյուր պահին կենտրոնանա մուտքի այն տարրերի վրա, որոնք առավել կարևոր են տվյալ բառը գեներացնելու համար։

Այս մեխանիզմի հիմնական գաղափարն է՝ յուրաքանչյուր ելքային քայլի ժամանակ հաշվարկել *ուշադրության* *կշիռներ (attention weights)* մուտքային բոլոր տարրերի նկատմամբ՝ ըստ դրանց համապատասխանության տվյալ պահին ապակոդավորիչի վիճակին։ Այս կշիռները ցույց են տալիս, թե որքան "ուշադրություն" պետք է դարձնել մուտքի յուրաքանչյուր տարրին։

Ուշադրության մեխանիզմը բաղկացած է հետևյալ քայլերից`

1. *Մուտքագրման կոդավորում (Input Encoding)*՝ մուտքային տվյալները վերածվում են ներկայացման հատուկ ձևի, որը հարմար է ուշադրության մեխանիզմի մշակման համար:
2. *Հարցման վեկտորի ստեղծում (Query Generation)*՝ մոդելը ստեղծում է *հարցման վեկտոր* (query vector)՝ հիմնվելով իր ընթացիկ վիճակի կամ կոնտեքստի վրա: Այս վեկտորը ցույց է տալիս, թե մոդելը ինչ տեղեկության վրա է ցանկանում կենտրոնանալ կամ փորձում ստանալ մուտքային տվյալներից:
3. *Բանալի-արժեք զույգերի ստեղծում (Key-Value Pair Creation)*՝ մուտքային ներկայացումները բաժանվում են բանալի-արժեք զույգերի: Բանալիները օգտագործվում են համապատասխանության կամ կարևորության որոշման համար: Արժեքները պահում են իրական տվյալները կամ տեղեկատվությունը:
4. *Նմանության հաշվարկ (Similarity Computation)*` hաշվարկվում է հարցման վեկտորի և յուրաքանչյուր բանալու նմանությունը՝ օգտագործելով տարբեր մեթոդներ, ինչպիսիք են՝ *կետային արտադրյալ* (Dot Product), *ընդհանուր*՝ օգտագործելով կշիռների մատրիցա (General), *կոնկատենացիա* (Concat)`

որտեղ՝

* -ը կոդավորիչի թաքնված վիճակն է -րդ դիրքում։
* -ն ապակոդավորիչի թաքնված վիճակն է -րդ դիրքում։
* -ն կշիռների մատրիցն է։
* -ն կշիռների վեկտորն է:

1. *Ուշադրության կշիռների հաշվարկ (Attention Weights Calculation)*՝ նմանության գնահատականներն անցնում են softmax ֆունկցիայի միջով, ստանալու համար ուշադրության կշիռները (attention weights), որոնք ցույց են տալիս յուրաքանչյուր մուտքային տարրի հարաբերական կարևորությունը։ ։
2. *Կշռված գումար (Weighted Sum)*՝ ուշադրության կշիռները կիրառվում են համապատասխան արժեքների (values) վրա, ստեղծելով կշռված գումար։ Այս քայլը համախմբում է մուտքի այն տեղեկատվությունը, որը ուշադրության մեխանիզմը համարում է կարևոր՝

Որտեղ՝ -ը կոդավորիչում բանալի-արժեք զույգերի (կոդավորիչի թաքնված վիճակների) ընդհանուր քանակն է։

1. *Կոնտեքստային վեկտոր* *(Context Vector)*՝ կշռված գումարը ծառայում է որպես կոնտեքստային վեկտոր, որը ներկայացնում է մուտքից ստացված կարևոր տեղեկատվությունը։ Այն ներառում է տվյալ քայլի կամ առաջադրանքի համար անհրաժեշտ կոնտեքստը։
2. *Ինտեգրումը մոդելի հետ*՝ կոնտեքստային վեկտորը միավորվում է մոդելի ընթացիկ վիճակի կամ թաքնված ներկայացման հետ՝ մոդելի հաջորդ քայլերի կամ շերտերի համար տրամադրելով լրացուցիչ տեղեկություն կամ կոնտեքստ։
3. *Կրկնություն*՝ Քայլեր 2-ից 8-ը կրկնվում են մոդելի յուրաքանչյուր քայլի կամ իտերացիայի ընթացքում, ինչը հնարավորություն է տալիս ուշադրության մեխանիզմին դինամիկ կերպով կենտրոնանալ մուտքային հաջորդականության տարբեր հատվածների վրա։