**Մեքենայական ուսուցման վրա հիմնված կոդի օպտիմալացում. ազդեցությունների և հնարավորությունների ուսումնասիրություն ծրագրային հավելվածի մշակման օրինակով**

1. **Մեքենայական ուսուցման մեթոդոլոգիաները կոդի օպտիմիզացիայի մեջ։**

1․1 Մեքենայական ուսուցման հիմնական գաղափարները և կիրառումը կոդի օպտիմիզացիայում

1․2 Համեմատական վերլուծություն առկա ալգորիթմների միջև։

**2․ Ամրապնդման ուսուցման (RL) մեթոդներ։**

2․1 RL մեթոդների էությունը և կիրառելիության ոլորտները։

2․2 Կոդի օպտիմիզացիայի համար RL մեթոդի մշակում։

**3․ Ծրագրային հավելվածի նախագծում և մշակում։**

3․1 Հավելվածի բնույթը, աշխատանքի սկզունքը։

3․2 Հավելվածի նախագծման և մշակման ընթացքը։

3․3 Ստացված արդյունքների համեմատություն և գնահատում։

# Ներածություն

# Մեքենայական ուսուցման (ML) արագ զարգացող ոլորտը հեղափոխում է բազմաթիվ ոլորտներ՝ փոփոխելով ավանդական մոտեցումները և հնարավորություն տալով բեկումնային առաջընթացներ գրանցելու: ML-ի շնորհիվ խորը փոփոխությունների ենթարկվող տիրույթներից մեկը կոդի օպտիմիզացումն է, որը ծրագրային ապահովման մշակման կարևորագույն ասպեկտն է՝ ուղղված կոդի արդյունավետ աշխատանքի և որակի բարձրացմանը: Կոդի օպտիմիզացումը միշտ էլ էական է եղել ծրագրային ապահովման կատարողականը բարելավելու, ռեսուրսների սպառումը նվազեցնելու և բարձր արձագանքման աստիճանի ապահովման համար: Ավանդաբար, կոդերի օպտիմալացումը մեծապես հիմնված է ստատիկ վերլուծության, էվրիստիկ մեթոդների և պրոֆիլով առաջնորդվող օպտիմալացումների (PGO) վրա: Ստատիկ վերլուծությունը ներառում է կոդի ուսումնասիրություն՝ առանց այն գործարկելու, անարդյունավետության բացահայտում և կատարողականությունը բարելավելու համար նախապես սահմանված կանոնների կիրառում: Էվրիստիկ մեթոդներն օգտագործում են փորձի վրա հիմնված տեխնիկա՝ կոդերի հատվածների օպտիմալացման վերաբերյալ խելամիտ կռահումներ անելու համար, մինչդեռ PGO-ն օգտագործում է գործարկման ժամանակի տվյալները՝ օպտիմալացման գործընթացը ուղղորդելու համար՝ կենտրոնանալով առավել հաճախ կատարվող ուղիների վրա: Այնուամենայնիվ, այս ավանդական մեթոդները, չնայած իրենց արդյունավետությանը, ունեն իրենց բնորոշ սահմանափակումներ: Ստատիկ վերլուծությունը կարող է բաց թողնել դինամիկ վարքագիծը և համատեքստին հատուկ օպտիմալացումները: Էվրիստիկ մեթոդները հաճախ հիմնվում են մշակողների փորձի և ինտուիցիայի վրա, ինչը հանգեցնում է արդյունքների փոփոխականության: PGO-ն, չնայած հզոր մեթոդ լինելուն, կարող է ծանրաբեռնված լինել և միշտ չէ, որ լավ հարմարվում է փոփոխվող աշխատանքային ծանրաբեռնվածությանը կամ կատարման նոր միջավայրերին: Մինչդեռ, ML-ի, հատկապես ուժեղացման ուսուցման (RL) գալուստով, նոր մեթոդներ և հնարավորություններ են ի հայտ գալիս կոդի օպտիմալացման համար, որոնք առաջարկում են ավանդական տեխնիկայի հնարավորությունները գերազանցելու ներուժ: Ամրապնդման ուսուցումը, որը ML-ի ենթաբազմություն է, հատկապես խոստումնալից է այս առումով: RL-ը հիմնական էությունը կայանում է գործակալի ուսուցման մեջ, որոշումների հաջորդական կայացման համար՝ պարգևատրելով ցանկալի գործողությունները և պատժելով անցանկալիներին: Այս մոտեցումն իր էությամբ հարմարեցված է կոդի օպտիմալացմանը, որտեղ գործակալը կարող է վերապատրաստվել՝ ուսումնասիրելու տարբեր օպտիմալացման ռազմավարություններ և սովորելու, թե որոնք են տալիս կատարողականի լավագույն բարելավումները: RL-ի ներուժը կոդի օպտիմալացման մեջ միայն տեսական չէ: Վերջին բարելավումները և փորձարարական կիրառությունները ցույց են տվել RL-ի կարողությունը՝ հասնելու զգալի արդյունավետության: Օրինակ, Google-ի AutoML-ը, որն օգտագործում է RL-ը՝ նեյրոնային ցանցերի ճարտարապետությունը նախագծելու համար, մի քանի չափանիշներով գերազանցել է մարդու կողմից նախագծված մոդելներին: Նմանապես, RL-ի վրա հիմնված կոդերի օպտիմալացման մոտեցումները ցույց են տվել խոստումնալից արդյունքներ՝ կրճատելով կատարման ժամանակը և հիշողության օգտագործումը ավելի արդյունավետ կերպով, քան ավանդական տեխնիկան:

Կուրսային աշխատանքում **քննարկման նյութ է դարձել** մեքենայական ուսուցման վրա հիմնված կոդի օպտիմալացումը. ծրագրային հավելվածի նախագծման օրինակով ազդեցությունների և հնարավորությունների ուսումնասիրությունը։

**Ուսումնասիրվող թեմայի արդիականությունը։** Կոդերի օպտիմալացման համար մեքենայական ուսուցման, մասնավորապես՝ ամրապնդման ուսուցման օգտագործման թեման ժամանակակից է և շատ կարևոր՝ մի քանի հիմնական պատճառներով: Նախ, քանի որ ծրագրային ապահովումը դառնում է ավելի բարդ, և կատարողականի պահանջները մեծանում են, ավանդական օպտիմալացման մեթոդները ջանում են համընթաց մնալ: Ժամանակակից հավելվածները հաճախ աշխատում են տարբեր միջավայրերում՝ ամպային սերվերներից մինչև ամենատարբեր սարքեր, որոնցից յուրաքանչյուրն ունի եզակի օպտիմալացման կարիքներ: Երկրորդ, մեքենայական ուսուցումը սովորելու և հարմարվելու իր ունակությամբ առաջարկում է օպտիմալացման դինամիկ և խելացի մոտեցում, որը կարող է ավելի արդյունավետ կերպով կարգավորել առաջացող բարդությունները: Բացի այդ, հաշվողական հզորության շարունակական առաջընթացը և տվյալների մեծ հավաքածուների առկայությունը հնարավոր է դարձնում մշակել բարդ ML մոդելներ կոդի օպտիմալացման համար: Առաջադեմ ML տեխնիկայի հետ ծրագրային ապահովման կարևորագույն մարտահրավերների միավորումը, ներկայացնում է հետազոտության առաջադեմ դաշտ, որն ունի ժամանակակից ծրագրային ապահովման համակարգերի արդյունավետությունն ու կատարողականությունը զգալիորեն բարձրացնելու ներուժ:

«Մեքենայական ուսուցման վրա հիմնված կոդի օպտիմալացում. ազդեցությունների և հնարավորությունների ուսումնասիրություն ծրագրային հավելվածի նախագծման օրինակով» թեմայի ուսումնասիրության **նպատակն** է` բացահայտել մեքենայական ուսուցման մեջ առկա հիմնական մեթոդոլոգիաները, ալգորիթմները, դրանց կիրառումը կոդի օպտիմիզացման մեջ, ինչպես նաև կատարել համեմատական վերլուծություններ նրանց միջև ըստ որոշակի չափանիշների, ուսումնասիրել ամրապնդման ուսուցման մեթոդները, նրանց դերն ու նշանակությունը և ի վերջո մշակել RL մեթոդների վրա հիմնված նոր մեթոդ(ալգորիթմ), որը հնարավորություն կտա իրականացնել կոմպիլյատորի հետ համատեղ կոդի օպտիմիզացիա, բարելավելով ինչպես կոդի կատարման ժամանակը, այնպես էլ նրա կողմից զբաղեցրած հիշողության չափը:

Ելնելով մեր նպատակից սահմանել ենք հետևյալ **խնդիրները**՝

* Կատարել մեքենայակ ուսուցման վրա հիմնված կոդի օպտիմիզացման համար առկա ալգորիթմների ուսումնասիրություն։
* Կատարել համեմատական վերլուծություն ալգորիթմների միջև՝ առանձնացնելով հիմնական առավելությունները և թերությունները։
* Մշակել նոր ալգորիթմ հիմնված RL մեթոդների վրա, ըստ առաջադրված պարամետրերի կոդի օպտիմիզացիա կատարելու համար։

**Ուսումնասիրության գիտական նորույթը։** Աշխատանքում փորձ է կատարվել փաստական նյութի վերլուծության հիման վրա կատարել առաջադրված խնդիրների ամբողջական ուսումնասիրություն։

**Աշխատանքի կառուցվածքը և բովանդակությունը։** «Մեքենայական ուսուցման վրա հիմնված կոդի օպտիմալացում. ազդեցությունների և հնարավորությունների ուսումնասիրություն ծրագրային հավելվածի նախագծման օրինակով» աշխատանքը կազմված է երկու գլխից՝ Մեքենայական ուսուցման մեթոդոլոգիաները կոդի օպտիմիզացիայի մեջ, Ամրապնդման ուսուցման (RL) մեթոդներ։

Առաջին գլխում՝ Մեքենայական ուսուցման մեթոդոլոգիաները կոդի օպտիմիզացիայի մեջ, ներկայացված է մեքենայական ուսուցման հիմնական գաղափարը, առկա ալգորիթմների կիրառումը կոդի օպտիմիզացիայում և համեմատական վերլուծության դրանց միջև:

Երկրորդ գլխում՝ Ամրապնդման ուսուցման (RL) մեթոդներ մանրամասն շարադրված է RL մեթոդների էությունը և նշանակությունը(կիրառելիության ոլորտները), RL մեթոդների վրա հիմնված նոր մեթոդի(ալգորիթմի) մշակումը, որը միտված է իրականացնելու կոդի օպտիմիզացիա ըստ առաջադրված պարամետրերի բարելավման։

**Գլուխ 1**

**Մեքենայական ուսուցման մեթոդոլոգիաները կոդի օպտիմիզացիայի մեջ։**

* 1. ***Մեքենայական ուսուցման հիմնական գաղափարները և կիրառումը կոդի օպտիմիզացիայում:***

**Մեքենայական ուսուցումը (Machine Learning, ML)** արհեստական ինտելեկտի (Artificial Intelligence, AI) և համակարգչային գիտության լայնորեն կիրառվող ուղղություններից է, որն առանձնանում է տվյալների և ալգորիթմների կիրառմամբ՝ համակարգերին հնարավորություն տալով ընդօրինակել մարդկային ուսուցման գործընթացը և աստիճանաբար բարելավել իրենց կատարողականությունը։ Մեքենայական ուսուցման մեթոդները լայնորեն օգտագործվում են իրական կյանքում, օրինակ՝ անհատականացված առաջարկությունների համակարգերում, որոնք վերլուծում են օգտագործողի նախընտրությունները և տրամադրում համապատասխան բովանդակություն։

Այս տեխնոլոգիան հիմնված է համակարգերի կարողության վրա՝ սովորել և կատարելագործվել փորձից՝ առանց մարդու ուղղակի միջամտության։ Այն օգտագործվում է մեծածավալ տվյալների (big data) մշակման և վերլուծության համար՝ հնարավորություն տալով ստեղծել ծրագրեր, որոնք կանխատեսումներ են կատարում, օրինակ՝ պահանջարկի կանխատեսումը կամ հիվանդությունների վաղ հայտնաբերումը։ Այս գործընթացը ներառում է ալգորիթմների կիրառում, որոնք մշակվում և բարելավվում են տվյալների հավաքածուների հիման վրա։ Օրինակ, մեքենայական ուսուցման ալգորիթմը կարող է «ուսուցանվել» տվյալների հավաքածուի վրա, որը բաղկացած է ծաղիկների հազարավոր պատկերներից, որոնք պիտակավորված են տարբեր տեսակի ծաղիկներով, որպեսզի այն կարողանա ճիշտ նույնականացնել ծաղիկը նոր լուսանկարում՝ տարբերակիչ այն բնութագրերի հիման վրա, որոնք նա սովորել է այլ նկարներից:

Նման ալգորիթմների արդյունավետ աշխատանքն ապահովելու համար, դրանք սովորաբար պահանջում են բազմակի կատարելագործում, մինչև կկարողանան կազմել գործողությունների ամբողջական և ճշգրիտ ուղեցույց, որը թույլ կտա իրականացնել համապատասխան առաջադրանքները։ Վերապատրաստման բավարար փուլերից հետո ալգորիթմները ձևափոխվում են «մեքենայական ուսուցման մոդելների», որոնք նախագծված են հատուկ խնդիրներ լուծելու համար, ինչպիսիք են՝ պատկերների դասակարգումը, գների կանխատեսումը կամ ռազմավարական որոշումներ կայացնելը՝ օրինակ՝ շախմատում։

Որոշ դեպքերում ալգորիթմները շերտավորվում են միմյանց վրա՝ ստեղծելու բարդ ցանցեր, որոնք թույլ են տալիս կատարել ավելի բարդ առաջադրանքներ, ինչպիսիք են տեքստերի ստեղծումը և չաթ-բոտերի հզորացումը, «խորը ուսուցում» (Deep Learning) կոչվող մեթոդի միջոցով: Խորը ուսուցումը մեթոդ է, որը համակարգիչներին սովորեցնում է տեղեկատվությունը մշակել այնպես, ինչպես մարդու նյարդային գործընթացները: Նեյրոնային ցանցերը բաղկացած են երեք հիմնական շերտերից.

* *Մուտքային շերտ։* Տվյալները մուտքագրվում են մուտքային շերտով:
* *Թաքնված շերտեր։* Թաքնված շերտերը մշակում և տեղափոխում են տվյալները այլ շերտեր:
* *Ելքային շերտ։* Վերջնական արդյունքը կամ կանխատեսումը կատարվում է ելքային շերտում:

Այս ցանցերը գործում են բազմաթիվ կրկնվող ցիկլերով՝ օգտագործելով մեծածավալ տվյալների հավաքածուներ, որոնք նաև հայտնի են որպես վերապատրաստման տվյալներ։ Յուրաքանչյուր կրկնությունից հետո նեյրոնային ցանցն աստիճանաբար բարելավում է իր ճշգրտությունը, ինչը նման է մարդու ուսուցման գործընթացին, որտեղ փորձի միջոցով զարգանում են հմտությունները։

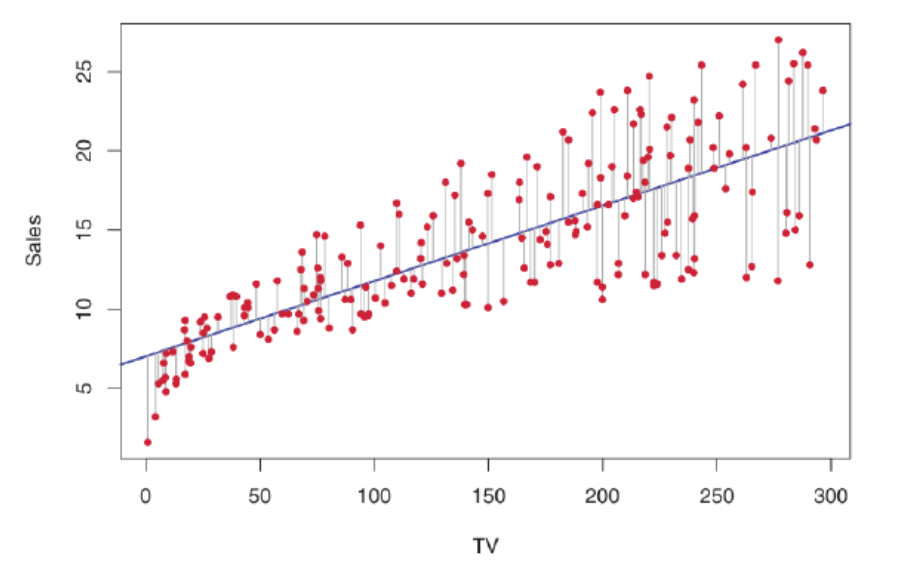
Խորը ուսուցման մոդելները իրենցից ներկայացնում են ֆայլեր, որոնք տվյալների մշակման մասնագետները վարժեցնում են՝ մարդկային նվազագույն միջամտությամբ առաջադրանքներ կատարելու համար: Խորը ուսուցման մոդելները ներառում են նախապես սահմանված քայլերի հավաքածուներ (ալգորիթմներ), որոնք հրահանգում են ֆայլին, թե ինչպես վարվել որոշակի տվյալների հետ: Ուսուցման այս մեթոդը խորը ուսուցման մոդելներին հնարավորություն է տալիս ճանաչել տեքստի, պատկերների կամ հնչյունների ավելի բարդ օրինաչափությունները:

Մեքենայական ուսուցումն ընդհանրապես ներառում է ալգորիթմների և տվյալների համակցություն, որը նպաստում է ինքնավար կամ կիսաինքնավար համակարգերի զարգացմանը։ Խորը ուսուցումը, իր հերթին, մեքենայական ուսուցման ենթաբաժին է, որը հիմնված է ալգորիթմների շերտավորման վրա՝ նեյրոնային ցանցերի միջոցով։ Այդ ցանցերը, որոնք որոշ չափով հիշեցնում են մարդու ուղեղի կառուցվածքը, հնարավորություն են տալիս մեքենաներին իրականացնել բարդ առաջադրանքներ՝ վերամշակելով մեծ քանակությամբ տվյալներ։ Արդյունքում, չնայած մեքենայական ուսուցման հիմքում ընկած ընդհանուր սկզբունքները համեմատաբար պարզ են, մոդելները, որոնք արտադրվում են գործընթացի վերջում, կարող են լինել համապարփակ մշակված և բավականաչափ բարդ:

Համակարգիչների հայտնագործումից ի վեր շրջանառվել է այն միտքը, թե արդյո՞ք դրանք կարող են ինքնուրույն սովորել: Եթե դա հնարավոր լինի, ապա համակարգիչների համար կբացվեն նոր կիրառություններ, հարմարեցման նոր մակարդակներ և հնարավորություններ: Մեքենայական ուսուցման ալգորիթմների միջոցով տեղեկատվության մշակման ավելի խորը ըմբռնումը կարող է նաև նպաստել մարդու սովորելու կարողությունների բարելավմանը: Ներկայիս դրությամբ հիմնովին չի մշակվել այնպիսի ալգորիթմ որը թույլ կտա, որ համակարգիչները սովորեն այնպես, ինչպես մարդիկ են ունակ յուրացնելու նոր ինֆորմացիա կամ հմտություն: Սակայն արդեն մշակվել են բազմաթիվ ալգորիթմներ, որոնք արդյունավետ են որոշակի ուսումնական առաջադրանքների համար: Այս մոտեցումների զարգացումը սկիզբ է դրել ուսուցման ինտեգրման գործընթացի տեսական հիմքի ստեղծմանը: Այնպիսի գործնական խնդրի համար, ինչպիսին է խոսքի ճանաչումը, մեքենայական ուսուցման վրա հիմնված ալգորիթմները գերազանցում են մինչ օրս փորձված բոլոր մոտեցումներին: Նմանապես, **տվյալների մայնինգի** (Data Mining) ոլորտում մեքենայական ուսուցման մեթոդները լայնորեն օգտագործվում են մեծ ծավալի առևտրային տվյալների բազաներից արժեքավոր գիտելիքներ հայտնաբերելու համար: Այս տվյալները կարող են ներառել սարքավորումների պահպանման գրառումներ, վարկային դիմումներ, ֆինանսական գործարքներ, բժշկական գրառումներ և այլն:

Մեքենայական ուսուցումը արհեստական ինտելեկտի ենթաբազմություն է, որը մշակում է ալգորիթմներ և մեթոդներ, որոնք օգտագործվում են համակարգչային համակարգերի կողմից առաջադրանքներ կատարելու համար՝ առանց հստակ հրահանգներ ստանալու: Մեքենայական ուսուցման ալգորիթմը որպես մուտքային տվյալ վերցնում է մի շարք օրինակներ, որոնք կոչվում են փորձարկման հավաքածու (training set), և արդյունքում կառուցում է կանխատեսման մոդել, որը նաև հայտնի է որպես մեքենայական ուսուցման մոդել: Մեքենայական ուսուցումն օգտագործվում է բազմաթիվ ոլորտներում՝ համակարգչային համակարգեր, բնական լեզվի մշակում, կենսաինֆորմատիկա, տնտեսագիտություն և այլն: Կախված առաջադրանքից, ուսուցումը կարող է դասակարգվել մի քանի մեթոդների բազմազանության.

* ***Վերահսկվող ուսուցում։*** Վերահսկվող ուսուցման մեջ յուրաքանչյուր տվյալ բաղկացած է հատկանիշ - պիտակ զույգերից: Հատկանիշները պարունակում են տեղեկատվություն օրինակի մասին, և պիտակը կանխատեսման թիրախ է: Եթե պիտակը դիսկրետ է, ապա առաջադրանքը կոչվում է դասակարգում (օրինակ՝ կանխատեսել՝ արդյոք email-ը սպամ է, թե ոչ): Եթե պիտակը շարունակական է, ապա առաջադրանքը ռեգրեսիա է (օրինակ՝ որևէ տան գնի կանխատեսումը՝ ելնելով դրա հատկանիշներից): Վերահսկվող ուսուցման գործընթացում ալգորիթմը օգտագործում է մուտքագրված տվյալների և դրանց համապատասխան պիտակների հավաքածուն (training set), ուսուցանում է ֆունկցիա, որը կարող է նոր մուտքային տվյալների համար ճիշտ կանխատեսումներ ապահովել: Սա նշանակում է, որ մուտքագրվող տվյալների օրինակները և դրանց արդյունքները նախապես հայտնի են, և համակարգը պետք է կարողանա ընդհանրացնել սովորածը՝ նոր տվյալների հետ կիրառման համար: Յուրաքանչյուր ուսուցման օրինակը բաղկացած է մուտքային օբյեկտից (սովորաբար վեկտորից) և ցանկալի ելքային արժեքներից: Ուսուցման ալգորիթմը վերլուծում է այս տվյալները և ստեղծում ֆունկցիա, որը թույլ է տալիս նոր տվյալների քարտեզագրում՝ ապահովելով համարժեք արդյունքներ: Օպտիմալ դեպքում ալգորիթմը պետք է կարողանա ճիշտ կանխատեսել դասի պիտակները նոր դեպքերի համար, ինչը պահանջում է տվյալների «ողջամիտ» ընդհանրացում: Գծային ռեգրեսիան լավ օրինակ է վերահսկվող ուսուցման համար, քանի որ այն պարզ է և, այնուամենայնիվ, լայնորեն կիրառվող գործնականում: Մյուս կողմից, նեյրոնային ցանցերը հզոր վերահսկվող ուսուցման մոդելներ են, որոնք կարող են մոտավորել բարդ գործառույթները: Գծային ռեգրեսիան մեքենայական ուսուցման ամենապարզ ալգորիթմն է: Ինչպես ենթադրում է նրա անունը, այն օգտագործվում է ռեգրեսիայի առաջադրանքների համար: Դիտարկենք այն դեպքը, երբ հատկանիշի վեկտորները և պիտակները երկչափ են: Գծային ռեգրեսիան գտնում է այն գիծը, որը լավագույնս համապատասխանում է տվյալներին և վերադարձնում է այս գիծը որպես կանխատեսման մոդել: Նկար 1.1-ում ներկայացված է գծային ռեգրեսիայի օրինակ: Կարմիր կետերը ներկայացնում են փորձարկման հավաքածուն, իսկ գիծը վերադարձված մոդելն է: Այն դեպքում, երբ հատկանիշի վեկտորները բազմաչափ են, ալգորիթմը գծի փոխարեն դառնում է հիպերպլան:

******

Նկար 1․1։ Գծային ռեգրեսիայով ապրանքի վաճառքի կանխատեսում՝ օգտագործելով հեռուստատեսային գովազդի բյուջեն։

* ***Չվերահսկվող ուսուցում։*** Այս դեպքում յուրաքանչյուր օրինակը պարունակում է միայն որոշակի առանձնահատկություններ, և ուսուցման ալգորիթմը պետք է հայտնաբերի տվյալների որոշ օգտակար հատկություններ: Առաջադրանքը չի վերահսկվում, քանի որ տվյալները չեն պարունակում տեղեկատվություն այն մասին, թե ինչ կարելի է կանխատեսել: Չվերահսկվող առաջադրանքի օրինակ է կլաստերավորումը, որը դասակարգման առաջադրանք է տվյալների հետ, որոնք չեն պարունակում դասեր: Սա նշանակում է, որ ալգորիթմը պետք է ինքնուրույն հայտնաբերի, թե ինչպես դասակարգել տվյալները: Կիբեռնետիկայի տեսանկյունից սա կիբեռնետիկ փորձերի տեսակներից մեկն է։ Որպես կանոն, սա հարմար է միայն այն առաջադրանքների համար, որոնցում հայտնի են մի շարք օբյեկտների նկարագրություններ, և պահանջվում է հայտնաբերել ներքին հարաբերությունները, կախվածությունները և օրինաչափությունները, որոնք գոյություն ունեն օբյեկտների միջև: Չվերահսկվող ուսուցումը հաճախ հակադրվում է վերահսկվող ուսուցմանը, երբ յուրաքանչյուր ուսումնական օբյեկտի համար պարտադրվում է «ճիշտ պատասխան», և պահանջվում է գտնել մուտքային տվյալների և համակարգի ելքային տվյալների միջև կապը:
* ***Ամրապնդման ուսուցում։*** Ուսուցման ալգորիթմը կատարում է գործողություններ և յուրաքանչյուր գործողության համար ստանում է հաջող իրականացման որոշակի ազդանշան: Նպատակն է առավելագույնի հասցնել հաջող իրականացված գործողությունների քանակը: Այս համատեքստում ալգորիթմը չի մշակում օրինակների ֆիքսված շարք, այլ փոխազդում է միջավայրի հետ՝ առավելագույնի հասցնելով կատարաողականի աստիճանը: Տվյալները ստատիկ չեն, այլ տրամադրվում են դինամիկ կերպով միջավայրի կողմից: Ուսուցման ուժեղացման առաջադրանքի օրինակ է սովորել շախմատ խաղալ առանց հստակ ծրագրավորված լինելու, միայն շախմատային միջավայրի հետ շփվելով: Ամրապնդման ուսուցումը տարբերվում է վերահսկվող ուսուցումից նրանով, որ կարիք չկա պիտակավորված մուտքային/ելքային զույգերի ներկայացման և ոչ օպտիմալ գործողությունների հստակ ուղղման կարիք։ Փոխարենը շեշտը դրվում է հետախուզման (չարտացվող տարածքի) և շահագործման (ներկայիս գիտելիքների) միջև հավասարակշռություն գտնելու վրա։ Մասամբ վերահսկվող ամրապնդման ալգորիթմները կարող են համատեղել վերահսկվող և ամրապնդման ալգորիթմների առավելությունները։ Միջավայրը սովորաբար նշվում է Մարկովի որոշման գործընթացի (MDP) տեսքով, քանի որ այս համատեքստում ուժեղացման ուսուցման շատ ալգորիթմներ օգտագործում են դինամիկ ծրագրավորման տեխնիկա։ Դասական դինամիկ ծրագրավորման մեթոդների և ամրապնդման ուսուցման ալգորիթմների միջև հիմնական տարբերությունն այն է, որ վերջիններս չեն ենթադրում MDP-ի ճշգրիտ մաթեմատիկական մոդելի իմացություն և թիրախավորում են խոշոր MDP-ներին, որտեղ ճշգրիտ մեթոդները դառնում են անիրագործելի։ Մաթեմատիկայի մեջ Մարկովի որոշման գործընթացը (MDP) դիսկրետ ժամանակի ստոխաստիկ կառավարման գործընթաց է: Այն ապահովում է մաթեմատիկական շրջանակ որոշումների կայացման մոդելավորման համար այն իրավիճակներում, երբ արդյունքները մասամբ պատահական են և մասամբ որոշում կայացնողի հսկողության տակ: MDP-ները օգտակար են դինամիկ ծրագրավորման միջոցով լուծված օպտիմալացման խնդիրները ուսումնասիրելու համար։

Այս մեթոդներից յուրաքանչյուրը ներառում է ուսուցման փուլի ընթացքում գործառույթի իրականացումը առավելագույնի կամ նվազագույնի հասցնելու գործընթաց: Այս ֆունկցիան կոչվում է կորստի ֆունկցիա, և այն գնահատում է, թե ինչպես է մեքենայական ուսուցման մոդելը համապատասխանում տվյալներին: Այս համատեքստում ուսուցումը կարող է դիտվել որպես օպտիմալացման խնդիր, սակայն ուսուցման և օպտիմալացման միջև կան կարևոր տարբերություններ:

***Առկա ալգորիթմների և մեթոդների կիրառումը կոդի օպտիմիզացիայում։***

Կոդի օպտիմալացումը այն գործընթացն է, որի ընթացքում ծրագիրը կամ մոդելը վերափոխվում է՝ նպատակ ունենալով բարելավել դրա կատարողականը: Այս գործընթացն անընդհատ վերապատրաստման միջոցով հանգեցնում է գործառույթների առավելագույն կամ նվազագույն արդյունավետության գնահատման, ինչը հանդիսանում է մեքենայական ուսուցման հիմնական սկզբունքներից մեկը՝ ավելի լավ արդյունքների հասնելու համար:

Օպտիմալացումը ծրագրի փոխակերպման տեխնիկա է, որը փորձում է բարելավել կոդը՝ ստիպելով նրան սպառել ավելի քիչ ռեսուրսներ (օրինակ՝ CPU, հիշողություն) և ապահովել բարձր արագություն: Սա իրականանում է՝ փոխարինելով բարձր մակարդակի ընդհանուր ծրագրավորման կառուցվածքները շատ ավելի արդյունավետ ցածր մակարդակի կոդերով: Կոդի օպտիմալացման գործընթացը պետք է հետևի ստորև ներկայացված երեք կանոններին.

* Ճշգրտության պահպանում։ Օպտիմալացված կոդը պետք է լիովին պահպանի սկզբնական կոդի ֆունկցիոնալությունը՝ առանց արդյունքների ճշգրտության վրա բացասական ազդեցություն թողնելու:
* Կախվածությունների խախտման բացառումը։ Օպտիմիզացիան չպետք է խախտի տվյալների հոսքի և ֆունկցիոնալ կախվածությունների ճիշտ հաջորդականությունը:
* Կատարման արդյունավետության բարելավում։ Օպտիմիզացված կոդը պետք է ապահովի էական կատարողականության բարելավում՝ կրճատելով հաշվարկային ծախսերը և առավել արդյունավետ օգտագործելով ռեսուրսները:

Այս մոտեցումների կիրառումը թույլ է տալիս ծրագրավորողներին ոչ միայն բարելավել ծրագրերի կատարողականը, այլև արդյունավետորեն օգտագործել առկա ռեսուրսները՝ լուծելով բարդ խնդիրներ:

Կոդի օպտիմալացումը կարելի է բաժանել երկու հիմնական խմբի՝ մեքենայից անկախ և մեքենայից կախված օպտիմալացումներ:

Մեքենայից անկախ կոդի օպտիմալացում։  
Այս տեսակի օպտիմալացումը կենտրոնանում է կոդի բարելավման վրա՝ անկախ դրա հետագայում կիրառվող սարքավորումներից: Օպտիմալացման այս մոտեցումը նպատակ ունի բարելավել միջանկյալ կոդը՝ ստանալով առավել արդյունավետ վերջնական կոդ:

Մեքենայից անկախ օպտիմալացման ժամանակ կոդի փոփոխությունները չեն ներառում հիշողության կոնկրետ տարածքներ կամ պրոցեսորի ռեգիստրներ: Միջանկյալ կոդի ստեղծման ընթացքում հաճախ հանդիպում են անարդյունավետ կոդի հատվածներ, ինչպիսիք են՝

* Հաստատունների փոխարեն փոփոխականների օգտագործումը
* Փոփոխականների ավելորդ պատճենները
* Արտահայտությունների կրկնությունները

Կոդի օպտիմալացումը հնարավորություն է տալիս հեռացնել նման անարդյունավետությունը՝ կրճատելով հավելյալ բեռնվածությունը և բարելավելով ծրագրի ընդհանուր կառուցվածքը: Այս գործընթացը կարող է հանգեցնել ծրագրի կառուցվածքի զգալի փոփոխության, երբեմն՝ նույնիսկ անճանաչելի մակարդակի: Օրինակ, իրականացվում են գործողություններ, ինչպիսիք են՝

* Ցիկլերի բացումը
* Ֆունկցիաների ներդրումը (Inlining)
* Փոփոխականների վերացումը

Մեքենայից անկախ օպտիմալացման կարևոր առավելությունն այն է, որ այդ գործընթացի արդյունքում ստացված կոդը հարմար է ցանկացած տեսակի սարքավորման համար՝ առանց էական վերամշակման անհրաժեշտության:

Մեքենայից կախված կոդի օպտիմիզացումն ուղղված է ծրագրաշարի արդյունավետության և կատարողականի բարելավմանը՝ օգտագործելով այն ապարատային հարթակի հատուկ առանձնահատկությունները, որի համար նախատեսված է տվյալ կոդը: Այս տեսակի օպտիմալացումը սերտորեն կապված է թիրախային պրոցեսորի ճարտարապետության և հրահանգների հավաքածուի (Instruction Set Architecture, ISA) հետ և սովորաբար իրականացվում է թիրախային մեքենայի համար կոդի գեներացիայից հետո:

Այս գործընթացի ընթացքում կոդը փոխակերպվում է այնպես, որ առավելագույնս համապատասխանի թիրախային սարքավորման առանձնահատկություններին: Մեքենայից կախված օպտիմիզատորները մեծապես ուշադրություն են դարձնում, օրինակ՝

* **Հիշողության հիերարխիայի առավելություններից օգտվելուն**,
* **Պրոցեսորի ռեգիստրների արդյունավետ կառավարմանը**,
* **Կոնկրետ հրահանգների գործարկման ժամանակի նվազեցմանը**,
* **Պարալելիզմի (Parallelism) և պիպելայնինգի (Pipelining) հնարավորություններին:**

Չնայած այս երկու մոտեցումների տարբերություններին, դրանց ընդհանուր նպատակը կոդի կատարողականի բարելավումն է:

* **Մեքենայից կախված օպտիմիզացիան** առավելագույն արդյունավետություն է ապահովում կոնկրետ ապարատային հարթակների վրա՝ ճշգրտելով կոդը հատուկ ճարտարապետության կարիքներին համապատասխան:
* **Մեքենայից անկախ օպտիմիզացիան**, իր հերթին, կենտրոնանում է կոդի շարժունակության (portability) և պահպանելիության (maintainability) վրա՝ առանց թիրախային ապարատային հարթակից կախվածության:

Խոհեմորեն համադրելով երկու մոտեցումները՝ մշակողները կարող են ստեղծել ծրագրաշար, որն ապահովում է՝

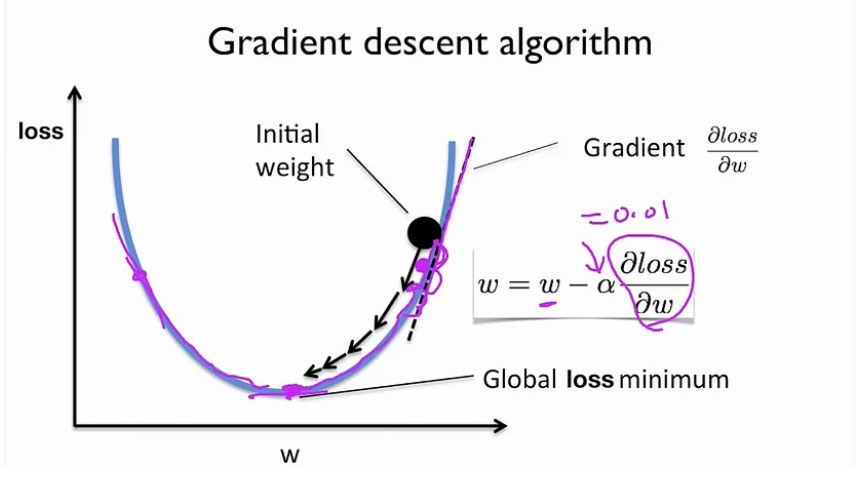
* **Օպտիմալ արդյունավետություն՝ բազմազան սարքավորումների վրա,**
* **Հարմարվողականություն՝ տարբեր հաշվողական միջավայրերում,**
* **Բարձր շարժունակություն՝ ապարատային հարթակների փոփոխության դեպքում:**

Այս բազմաշերտ մոտեցումը նպաստում է ծրագրերի կենսունակության և կատարողականի բարձրացմանը՝ հնարավորինս օգտագործելով ապարատային և ծրագրային ռեսուրսները:

Մեքենայական ուսուցումը օգտվողին թույլ է տալիս համակարգչի ալգորիթմին տրամադրել հսկայական քանակությամբ տվյալներ և թույլ տալ, որ համակարգիչը վերլուծի և կայացնի տվյալների վրա հիմնված առաջարկություններ և որոշումներ: Այնպիսի իրականությունում, որտեղ գրեթե բոլոր ձեռքով իրականացվող (manual) առաջադրանքները ավտոմատացված են, այդ գործընթացների իրականացման սահմանումները փոփոխվում են: Այժմ կան մեքենայական ուսուցման ալգորիթմների տարբեր տեսակներ, որոնցից մի քանիսը կարող են օգնել համակարգիչներին շախմատ խաղալ, վիրահատություններ կատարել և դառնալ ավելի խելացի և անհատական: Մեքենայական ուսուցման բազմաթիվ ալգորիթմներ նախագծվել են ամենատարբեր դինամիկ ժամանակներում՝ օգնելու լուծել իրական աշխարհի բարդ խնդիրները: Մեքենայական ուսուցման ալգորիթմները ավտոմատացված են և ինքնուրույն ենթարկվում են փոփոխությունների՝ ժամանակի ընթացքում շարունակական բարելավմամբ: Մեքենայական ուսուցման ալգորիթմների ինտեգրումը կոդի օպտիմալացման գործընթացներում փոխակերպող մոտեցում է ծրագրային ապահովման մեջ: Օգտագործելով մեքենայական ուսուցման հնարավորությունները՝ մշակողները կարող են ավտոմատացնել և բարելավել կոդի օպտիմալացումը՝ հանգեցնելով կատարողականության և արդյունավետության զգալի բարելավումների: Մեքենայական ուսուցման տեխնիկան առաջարկում է օպտիմիզացման գործընթացն ավտոմատացնելու ներուժ՝ սովորելով օրինաչափություններ կոդի նմուշների և կատարման արդյունքների (execution traces) հսկայական տվյալների հավաքածուներից: Այս ավտոմատացումը հեշտացնում է օպտիմալացման աշխատանքային հոսքը՝ նվազեցնելով ծրագրավորողների աշխատանքը և հնարավորություն տալով նրանց կենտրոնանալ ավելի բարդ խնդիրների իրագործման վրա:

Օպտիմալացումը մեքենայական ուսուցման ալգորիթմների ամենակարևոր մասն է: Այն սկսվում է կորստի ֆունկցիայի/ծախսերի ֆունկցիայի սահմանմամբ և ավարտվում է նվազագույնի հասցնելով կորուստները և ծախսերը՝ օգտագործելով օպտիմալացման ալգորիթմներ: Դրանք օգնում են առավելագույնի կամ նվազագույնի հասցնել սխալի գործառույթը: Մոդելի ներքին պարամետրերը շատ կարևոր դեր են խաղում մոդելի արդյունավետ վերապատրաստման և ճշգրիտ արդյունքներ ստանալու համար: Ահա թե ինչու մոդելի պարամետրերին համապատասխան օգտագործվում են տարբեր օպտիմալացման ալգորիթմներ՝ օպտիմալ արժեքները թարմացնելու և հաշվարկելու համար: Սա, իր հերթին, բարելավում է մոդելի ուսուցման գործընթացը, ինչպես նաև դրա արդյունքը: Մեքենայի ուսուցման համատեքստում օպտիմիզացումը վերաբերում է մոդելի պարամետրերի ճշգրտման գործընթացին, որպեսզի նվազագույնի(կամ առավելագույնի) հասցվի խնդրին համապատասխան օբյեկտիվ ֆունկցիան: Օբյեկտիվ ֆունկցիան չափում է, թե որքան լավ է մոդելը կատարում առաջադրանքը, ինչպես օրինակ՝ նվազագույնի հասցնելով ուսուցման տվյալների հավաքածուի սխալը: Գործընթացը ներառում է պարամետրերի օպտիմալ հավաքածուի հայտնաբերում, որոնք հանգեցնում են մոդելի լավագույն կերպով իրականացմանը: Գոյություն ունեն օպտիմիզացման տարբեր ալգորիթմներ, որոնք օգտագործվում են մեքենայական ուսուցման մեջ՝ գտնելու պարամետրերի օպտիմալ փաթեթը: Այս ալգորիթմները պատասխանատու են վերապատրաստման գործընթացի ընթացքում մոդելի պարամետրերի կրկնվող թարմացման համար: Որոշ ընդհանուր օպտիմալացման ալգորիթմներ ներառում են.

1. ***Gradient Descent։*** Gradient Descent-ը առաջին կարգի կրկնվող օպտիմալացման ալգորիթմ է, որը լայնորեն օգտագործվում է մեքենայական ուսուցման և օպտիմալացման խնդիրներում: Դրա հիմնական նպատակն է նվազագույնի հասցնել դիֆերենցիալ արժեքի կամ կորստի ֆունկցիան՝ կրկնվող մոդելի պարամետրերը կարգավորելու միջոցով (Նկար 1․2):

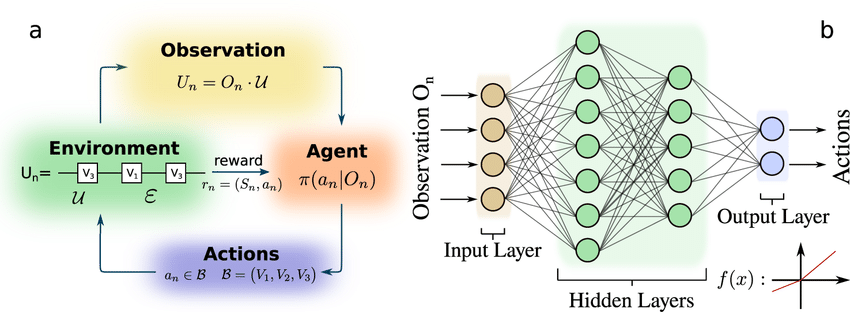


Նկար 1․2։ Gradient Descent ալգորիթմի ներկայացում։

w = w − α \* δ կորուստ​/δw

Ալգորիթմի իրականացման համար նախ և առաջ անհրաժեշտ է կազմել մոդելի պարամետրերի արժեքների (կշիռների) սկզբնական հավաքածուն: Յուրաքանչյուր կրկնության դեպքում պետք է հաշվարկել կորստի ֆունկցիայի գրադիենտը պարամետրերի նկատմամբ, որտեղ w-ը ներկայացնում է պարամետրերը, α-ն ուսուցման արագությունն է, իսկ δloss​/δw-ը նպատակային ֆունկցիայի գրադիենտն է: Անհրաժեշտ է պարզեցնել այն ֆունկցիան, որը պետք է նվազագույնի հասցնել՝ օգտագործելով առաջին կարգի բազմանդամը: Այս δloss/δw արտահայտությունը ներկայացնում է կորստի փոփոխության արագությունը՝ կապված քաշի պարամետրի հետ և կարևոր բաղադրիչ է օպտիմալացման ալգորիթմներում, ինչպիսին է Gradient Descent-ը: Նպատակը կշիռները կարգավորելն է այն ուղղությամբ, որը նվազագույնի է հասցնում կորուստը, իսկ ածանցյալը տեղեկատվություն է տրամադրում պարամետրային տարածության որոշակի կետում կորստի ֆունկցիայի թեքության մասին: Gradient Descent ալգորիթմը նպատակ ունի գտնել ծախսերի կամ կորստի ֆունկցիայի նվազագույն արժեքը: Այս ֆունկցիան ներկայացնում է մոդելի կանխատեսված արժեքների և ուսումնական տվյալների բազայի իրական արժեքների (պիտակների) միջև տարբերությունը:

1. ***Decision Trees:*** Որոշման ծառերը մեքենայական ուսուցման հիմնարար ալգորիթմներից է, որն օգտագործվում է դասակարգման և ռեգրեսիայի առաջադրանքների համար: Նրանք աշխատում են ռեկուրսիվորեն բաժանելով մուտքային տվյալները հատվածների՝ հիմնված առանձնահատուկ արժեքների վրա, ընդ որում յուրաքանչյուր հատված համապատասխանում է ծառի որոշման հանգույցին: Հանգույցների բաժանումը հիմնված է այնպիսի չափանիշների վրա, որոնք նպատակ ունեն առավելագույնի հասցնել տեղեկատվության ստացումը (դասակարգման առաջադրանքներում) կամ նվազագույնի հասցնել անճշտությունը (ռեգեսիոն առաջադրանքներում): Ծառի կառուցման գործընթացը սկսվում է տվյալների բաժանման լավագույն հատկանիշի ընտրությամբ: Դա արվում է բաժանման տարբեր չափանիշների գնահատմամբ, ինչպիսիք են Gini impurity- ն կամ տեղեկատվության ստացումը: Ընտրվում են այն հատկանիշը և բաժանման կետը, որոնք հանգեցնում են առավելագույն տեղեկատվության ստացման կամ նվազագույն անճշտության: Տվյալներն այնուհետև բաժանվում են այս ընտրությունների հիման վրա, և գործընթացը կրկնվում է ռեկուրսիվորեն յուրաքանչյուր բաժանման համար, մինչև չկատարվեն կանգառի որոշակի չափանիշներ, օրինակ՝ հասնել ծառի առավելագույն խորության կամ նմուշների նվազագույն քանակի մեկ հանգույցի համար: Ծառի կառուցումից հետո կիրառվում են այսպես կոչված «էտման» տեխնիկա՝ չափից ավելի հարմարեցումը նվազեցնելու համար: Էտումը ներառում է ճյուղերի կամ հանգույցների հեռացում, որոնք էապես չեն բարելավում ծառի աշխատանքը վավերացման տվյալների բազայում: Այս գործընթացը օգնում է ծառին խուսափել ուսուցման տվյալների անհամապատասխան տեղեկատվությունից՝ ավելի լավ կանխատեսումներ անելով նոր, չտեսնված տվյալների վրա: Երբ ծառը կառուցվի, այն կարող է օգտագործվել նոր տվյալների վերաբերյալ կանխատեսումներ անելու համար: Սկսած արմատային հանգույցից՝ յուրաքանչյուր տվյալի կետ շրջում է ծառի վրայով՝ հիմնվելով առանձնահատուկ արժեքների վրա, մինչև այն հասնի տերևային հանգույցին: Այնուհետև տերևային հանգույցի վրա կանխատեսումը վերագրվում է տվյալի կետին: Կոդի օպտիմալացման համատեքստում առանձնահատկությունները կարող են ներկայացնել կոդի տարբեր բնութագրեր, ինչպիսիք են ցիկլային կառուցվածքները, ֆունկցիաների կանչերը կամ հիշողության հասանելիության օրինաչափությունները: Որոշումների ծառերը կարող են օգտագործվել այս հատկանիշները վերլուծելու և բացահայտելու համար, թե որոնք են ամենակարևոր ազդեցությունը կատարողականության ցուցանիշների վրա, ինչպիսիք են կատարման ժամանակը կամ ռեսուրսների օգտագործումը: Այս տեղեկատվությունը կարող է օգնել օպտիմալացման ջանքերին առաջնահերթություն տալ ամենաազդեցիկ կոդի բաղադրիչների վրա:
2. ***Long Short-Term Memory (LSTM):*** Երկար կարճաժամկետ հիշողության (LSTM) ցանցերը կրկնվող նեյրոնային ցանցի (RNN) տեսակ են, որոնք նախատեսված են հաջորդական տվյալների մոդելավորման և հեռահար կախվածությունները ֆիքսելու համար: Ի տարբերություն ավանդական RNN-ների, LSTM ցանցերը լուծում են անհետացող գրադիենտի խնդիրը՝ ներմուծելով ավելի բարդ ճարտարապետություն, որը կարող է պահպանել տեղեկատվություն երկար ժամանակաշրջաններում: LSTM ցանցերի հիմքում ընկած են հիշողության բջիջները, որոնք ունեն բջջային վիճակ, որը կարող է ժամանակի ընթացքում պահպանել որոշակի տեղեկատվությունը: Այս հիշողության բջիջներն ունեն դարպասներ, որոնք կարգավորում են տեղեկատվության հոսքը, ներառյալ մուտքային դարպասը, մոռացության դարպասը և ելքային դարպասը: Մուտքային դարպասը վերահսկում է նոր տեղեկատվության հոսքը դեպի բջջային վիճակ, մոռացության դարպասը որոշում է, թե որ տեղեկությունը պետք է հեռացվի բջջային վիճակից, իսկ ելքային դարպասը կարգավորում է բջջային վիճակից դուրս եկող տեղեկատվությունը: Յուրաքանչյուր ժամանակային քայլում LSTM ցանցը ստանում է մուտքային տվյալ և հիշողության բջիջների ընթացիկ վիճակ: Մուտքային դարպասը որոշում է, թե որքան նոր տեղեկատվություն պետք է ավելացվի բջջային վիճակին՝ հաշվի առնելով դրա համապատասխանությունը: Մոռացման դարպասը որոշում է, թե նախորդ վիճակից որ տեղեկատվությունը պետք է պահպանվի կամ մոռացվի: Վերջապես, ելքային դարպասը զտում է տեղեկատվությունը ելքում LSTM միավոր արտադրելու համար: Մարզման ընթացքում LSTM ցանցի պարամետրերը, ներառյալ դարպասների կշիռներն ու կողմնակալությունները, օպտիմիզացված են՝ օգտագործելով ***Gradient Descent*** - ը և ժամանակի ընթացքում ետ տարածումը (BPTT): Այս գործընթացը կարգավորում է ցանցի պարամետրերը՝ նվազագույնի հասցնելու կանխատեսված արդյունքների և իրական թիրախների միջև եղած տարբերությունը: LSTM ցանցերի հիմնական առավելություններից մեկը հաջորդական տվյալների մեջ երկարաժամկետ կախվածություններ ֆիքսելու նրանց կարողությունն է: Հիշողության բջիջները թույլ են տալիս համապատասխան տեղեկատվությանը պահպանվել մի քանի ժամանակային քայլերի ընթացքում՝ հնարավորություն տալով ցանցին սովորել տվյալների բարդ օրինաչափություններ և հարաբերություններ։
3. ***Deep Reinforcement Learning-ը (DRL)*** համատեղում է ամրապնդման ուսուցման (RL) սկզբունքները խորը ուսուցման հետ՝ հնարավորություն տալով սովորել և որոշումներ կայացնել բարդ միջավայրերում: Այս մոտեցումը հատկապես հզոր է այնպիսի առաջադրանքների համար, որտեղ լուծումը ներառում է հաջորդական որոշումների կայացում, և որտեղ միջավայրը կարող է մոդելավորվել որպես Մարկովյան որոշումների գործընթաց (MDP): Մարկովի որոշման գործընթացը (MDP) մաթեմատիկական շրջանակ է (framework), որն օգտագործվում է ամրապնդման ուսուցման միջավայրը նկարագրելու համար, որտեղ արդյունքները մասամբ պատահական են և մասամբ որոշում կայացնողի (գործակալի) հսկողության տակ : Deep Reinforcement Learning-ում (DRL) գործակալը փոխազդում է շրջակա միջավայրի հետ՝ յուրաքանչյուր քայլում որոշումներ կայացնելով՝ հիմնված դիտարկումների վրա: Շրջակա միջավայրը ներկայացված է վիճակներով, որոնք ամփոփում են ներկա իրավիճակը կամ կազմաձևումը, որում գտնվում է գործակալը: Գործակալը կատարում է գործողություններ, որոնք առաջացնում են անցումներ մի վիճակից մյուսը, և յուրաքանչյուր գործողություն հանգեցնում է պարգևի, որը ծառայում է որպես գործողության արդյունավետության հետադարձ կապ: Քաղաքականությունը, որին հետևում է գործակալը, թելադրում է, թե ինչ գործողություններ ձեռնարկել՝ ելնելով ներկա վիճակից: Արժեքի ֆունկցիան գնահատում է որոշակի վիճակում գտնվելու կամ որոշակի գործողություն կատարելու ակնկալվող պարգևը՝ օգնելով գործակալին գնահատել իր գործողությունների հնարավոր երկարաժամկետ ձեռքբերումները: Գործակալի առաջնահերթ նպատակն է առավելագույնի հասցնել հավաքական պարգևները ժամանակի ընթացքում, որը հայտնի է որպես վերադարձ, որն առաջնորդում է գործակալին դեպի ռազմավարություններ, որոնք երկարաժամկետ հեռանկարում տալիս են առավել բարենպաստ արդյունքներ(Նկար 1․3):



Նկար 1․3։ Deep Reinforcement Learning ալգորիթմի ներկայացում։

***1.2 Համեմատական վերլուծություն առկա ալգորիթմների միջև։***

Մեքենայական ուսուցման ալգորիթմներն ու մոդելները էական դեր են խաղում կոդի կատարողականի օպտիմալացման գործում՝ ավտոմատացնելով արդյունավետ կոդերի ստացման ուղիներն ու կոնֆիգուրացիաները գտնելու գործընթացը: Ծրագրային ապահովման մշակման ոլորտում կոդի օպտիմիզացումը շատ կարևոր է արդյունավետությունը բարձրացնելու, ծրագրի կատարման ժամանակը նվազեցնելու և ռեսուրսների օգտագործումը նվազագույնի հասցնելու համար: Ավանդաբար, այս գործընթացը իրականացվել է ձեռքով և եղել է ժամանակատար՝ մեծապես հենվելով մշակողների փորձի և ինտուիցիայի վրա: Այնուամենայնիվ, մեքենայական ուսուցման գալուստը ներմուծել է ավտոմատացված և արդյունավետ մեթոդներ կոդի օպտիմալացման համար: Կատարենք համեմատական ուսումնասիրություն հետևյալ չորս նշանավոր մեքենայական ուսուցման ալգորիթմների միջև՝ գրադիենտ ծագում(Gradient Descent), որոշումների ծառեր(Decision Trees), երկար կարճաժամկետ հիշողություն (LSTM) և խորը ամրապնդման ուսուցում (DRL) և դրանց կիրառությունները կոդի օպտիմալացման մեջ: Ուսումնասիրելով դրանց մեխանիզմները և գործնական կիրառման դեպքերը՝ նպատակ ունենք ընդգծել, թե ինչպես կարելի է օգտագործել այդ ալգորիթմները՝ հասնելու կոդի օպտիմալ կատարողանակության աստիճանի և բացահայտելու առավել հարմար ալգորիթմը հատուկ օպտիմալացման առաջադրանքների համար:

Gradient Descent-ը մեքենայական ուսուցման անկյունաքարային ալգորիթմ է, որը լայնորեն օգտագործվում է տարբեր մոդելների օպտիմալացման համար՝ նվազագույնի հասցնելով տրված ծախսերի ֆունկցիան: Այն գործում է կրկնվող պարամետրերը կարգավորելով այն ուղղությամբ, որը նվազեցնում է ծախսերը, դրանով իսկ գտնելով օպտիմալ լուծումը: Կոդի օպտիմիզացման ժամանակ Gradient Descent-ը կարող է օգտագործվել կոդերի բազայի մեջ պարամետրերը ճշգրտելու համար՝ բարելավելու կատարողականի չափումները, ինչպիսիք են կատարման ժամանակը և հիշողության օգտագործումը: Օրինակ, դիտարկեք մի սցենար, որտեղ բարդ մաթեմատիկական ֆունկցիան օպտիմալացման կարիք ունի: Gradient Descent-ը կարող է պարբերաբար փոփոխել ֆունկցիայի պարամետրերը՝ հանգեցնելով կատարողականի զգալի բարելավումների: Դրա կիրառումը հատկապես արդյունավետ է այն իրավիճակներում, երբ օպտիմալացման խնդիրը շարունակական է և տարբերակելի՝ ապահովելով հստակ ուղի դեպի առավել արդյունավետ լուծում:

Որոշումների ծառերը, մյուս կողմից, առաջարկում են այլ մոտեցում: Որպես ոչ պարամետրային վերահսկվող ուսուցման մեթոդ, դրանք գերազանց են դասակարգման և ռեգրեսիայի առաջադրանքների համար: Որոշման ծառերը աշխատում են՝ տվյալները բաժանելով ենթաբազմությունների՝ հիմնված առանձնահատկությունների արժեքների վրա՝ դրանք դարձնելով չափազանց մեկնաբանելի և հեշտ պատկերացնելու համար: Կոդի օպտիմալացման համատեքստում որոշման ծառերը կարող են օգտագործվել՝ առաջնահերթություն տալու համար, թե մեծ կոդի բազայի ո՞ր հատվածներն է պետք օպտիմալացնել: Կոդերի կատարման տեղեկամատյանները (log) վերլուծելով՝ որոշումների ծառը կարող է բացահայտել խցանումները և առաջարկել բարելավման հատուկ հատվածներ: Օրինակ, եթե ծրագրի շրջանակներում որոշակի գործառույթը ճանաչվում է որպես կատարողականի խոչընդոտ, որոշման ծառը կարող է օգնել մշակողներին հասկանալ այն պայմանները, որոնց դեպքում այս ֆունկցիան դառնում է անարդյունավետ՝ դրանով իսկ ուղղորդելով թիրախային օպտիմալացումները, ինչպիսիք են ցիկլի բացումը կամ ալգորիթմական ճշգրտումները: Հիմնական առանձնահատկությունները, որոնք որոշումների ծառերը հարմար են դարձնում այս կոդի օպտիմիզացիայի խնդրի լուծման համար, ներառում են դրանց մեկնաբանելիությունը, ինչպես թվային, այնպես էլ դասակարգային տվյալների մշակման ունակությունը և արդյունավետությունը օպտիմալացման հիմնական ոլորտները բացահայտելու և առաջնահերթության աստիճան տալու համար:

Երկար կարճաժամկետ հիշողության (LSTM) ցանցերը,որոնք հանդիսանում են կրկնվող նեյրոնային ցանցի (RNN) տեսակներ, նախատեսված են հաջորդական տվյալների մշակման և երկարաժամկետ կախվածությունները արձանագրելու համար: Սա LSTM-ները հատկապես օգտակար է դարձնում այն ​​առաջադրանքների համար, որտեղ առկա տվյալները վճռորոշ դեր են խաղում ապագա արդյունքների կանխատեսման գործում: Կոդի օպտիմալացման ժամանակ LSTM-ները կարող են կանխատեսել ապագա կատարողականությունը՝ հիմնվելով կատարման օրինաչափությունների վրա: Օրինակ, LSTM մոդելը կարող է վերլուծել կոդերի անցյալում կատարման ժամանակները և կանխատեսել ռեսուրսների ապագա օգտագործումը՝ հնարավորություն տալով ակտիվ օպտիմալացումներ կատարել: Այս կանխատեսող հնարավորությունը արժեքավոր է կոդի օպտիմալացման համար, որը մշակում է ժամանակային տվյալները կամ այլ հաջորդական մուտքերը՝ թույլ տալով ծրագրավորողներին լուծել աշխատանքի հնարավոր խնդիրները՝ նախքան դրանք գործարկելը: Հիմնական հատկանիշները, որոնք LSTM-ները հարմար են դարձնում առաջադրված խնդրի լուծման համար, ներառում են նրանց երկարաժամկետ կախվածությունները հայտնաբերելու ունակությունը, հաջորդական տվյալների մշակման արդյունավետությունը և առկա միտումների հիման վրա ճշգրիտ կանխատեսումներ կատարելու կայունությունը:

Խորը ամրապնդման ուսուցումը (DRL) միավորում է ամրապնդման ուսուցման սկզբունքները խորը նեյրոնային ցանցերի հետ՝ հնարավորություն տալով մշակել գործակալների, որոնք սովորում են օպտիմալ վարքագիծ իրենց միջավայրի հետ փոխգործակցության միջոցով: Կոդի օպտիմալացման ոլորտում DRL-ը կարող է ավտոմատացնել գործընթացը՝ սովորելով փորձից և հարմարվելով փոփոխվող պայմաններին: DRL գործակալը կարող է վերապատրաստվել՝ օպտիմալացնելու կոմպիլյատորների կարգավորումները կամ կոդի ստեղծման փուլերը՝ շարունակաբար բարելավելով իր մոտեցումը՝ հիմնված կատարողականի հետադարձ կապի վրա: Օրինակ, DRL գործակալը կարող է ուսումնասիրել տարբեր օպտիմալացման ռազմավարություններ, ինչպիսիք են հանգույցի փոխակերպումները կամ հիշողության բաշխման տեխնիկան, և հաջորդական կերպով կատարելագործել իր ընտրությունը՝ առավելագույնի հասցնելու կատարողականի ձեռքբերումները: Այս հարմարվողական և հետախուզական բնույթը DRL-ը դարձնում է բավականին արդյունավետ բարդ և դինամիկ կոդերի բազաների համար, որտեղ ավանդական օպտիմալացման մեթոդները կարող են ավելի քիչ արդյունավետ լինել: Հիմնական հատկանիշները, որոնք DRL-ին նպատակահարմար են դարձնում կոդի օպտիմալացման առաջադրանքի լուծման գործում, ներառում են ժամանակի ընթացքում սովորելու և հարմարվելու նրա կարողությունը, օպտիմալացման ռազմավարությունների լայն շրջանակ ուսումնասիրելու առավելությունը և բարդ, բազմաչափ օպտիմալացման խնդիրներ լուծելու կարողությունը:

Կատարված ուսումնասիրություններից և համեմատական վերլուծություններից վստահ կարելի է արձանագրել, որ թեև մեքենայական ուսուցման թվարկված այլ ալգորիթմներն ունեն իրենց ուժեղ կողմերը, DRL-ն առաջարկում է եզակի առավելություններ կոդի օպտիմալացման խնդրի լուծման ուղղությամբ։

Deep Reinforcement Learning-ը (DRL) հատկապես հարմար է կոդի օպտիմիզացման խնդրի լուծման համար՝ գործարկման ժամանակը և հիշողության օգտագործումը բարելավելու համար: Ի տարբերություն մեքենայական ուսուցման այլ ալգորիթմների, DRL-ը նախագծված է բարդ, դինամիկ միջավայրեր վարելու համար և կարող է ժամանակի ընթացքում հարմարեցնել իր ռազմավարությունները՝ հիմնվելով շարունակական ուսուցման վրա: Սա այն դարձնում է իդեալական ընտրություն կոդի օպտիմալացման ավտոմատացման համար, որը հաճախ ներառում է բազմաթիվ հնարավոր բարելավումների և կոնֆիգուրացիաների փոփոխություններ:

*DRL-ի առանձնահատկություններն ու առավելությունները։*

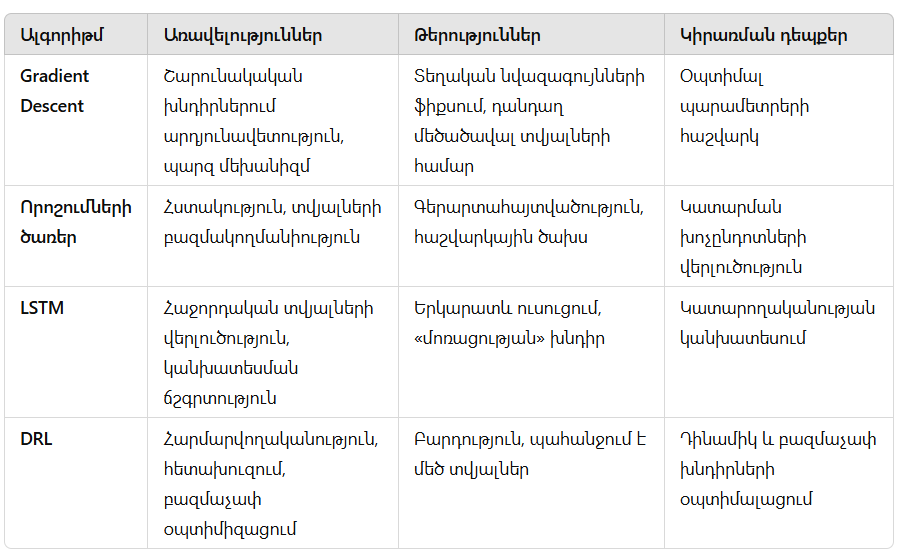
* *Հարմարվողական ուսուցում.* DRL գործակալները կարող են սովորել փորձից և հարմարեցնել իրենց ռազմավարությունները՝ ավելի լավ կատարողականության հասնելու համար: Սա շատ կարևոր է կոդերի օպտիմալացման համար, որտեղ լավագույն ռազմավարությունները կարող են անմիջապես երևալ և կարող են փոխվել կոդի բազայի զարգացմանը զուգընթաց:
* *Հետախուզում և համեմատություն.* DRL արդյունավետորեն հավասարակշռում է նոր օպտիմալացման ռազմավարությունների ուսումնասիրությունը հայտնի, արդյունավետ ռազմավարությունների շահագործման հետ: Այս հաշվեկշիռը DRL-ին թույլ է տալիս բացահայտել կոդի կատարողականը բարելավելու նորարարական ուղիներ, որոնք կարող են չդիտարկվել ավանդական օպտիմալացման մեթոդներով:
* *Կառավարման համալիր, դինամիկ միջավայրեր.* DRL-ը կարող է կառավարել իրական խնդրի կոդերի բազայի բարդությունն ու դինամիզմը, որոնք հաճախ ներառում են բարդ կախվածություններ և տարբեր կատարողական բնութագրեր: Տարբեր տեսակի կոդերի հետ հարմարվելու և իր մոտեցումը շարունակաբար կատարելագործելու կարողությունը նշանակալի առավելություն է:
* *Բազմաչափ օպտիմիզացում.* DRL-ը կարող է միաժամանակ օպտիմիզացնել կատարողականի մի քանի ցուցանիշներ, ինչպիսիք են գործարկման ժամանակը և հիշողության օգտագործումը: Այս ամբողջական մոտեցումը երաշխավորում է, որ մի ուղղությամբ իրականացվող բարելավումները բացասաբար չեն ազդի մյուսի վրա՝ հանգեցնելով ավելի հավասարակշռված և արդյունավետ օպտիմալացման:

Deep Reinforcement Learning-ը (DRL) առանձնանում է որպես կոդի օպտիմալացման ամենահարմար ալգորիթմ՝ աշխատելու ժամանակը և հիշողության օգտագործումը բարելավելու համար: Ժամանակի ընթացքում սովորելու և հարմարվելու, հետախուզումն ու համեմատությունը հավասարակշռելու, բարդ միջավայրերը կարգավորելու և միաժամանակ մի քանի կատարողական ցուցանիշներ օպտիմիզացնելու կարողությունը նրան դարձնում է եզակի առաջադրված խնդրի լուծման համար: Օգտագործելով DRL ալգորիթմների տրամաբանությունը՝ մշակողները կարող են ավտոմատացնել կոդի կատարողականի բարելավման գործընթացը՝ հասնելով արդյունավետության զգալի ձեռքբերումների և ստեղծելով ավելի ամուր, բարձր կատարողական հավելվածներ:

Ուստի, DRL-ի հարմարվողական ուսուցման հնարավորությունները և կայուն օպտիմալացման շրջանակը հզոր լուծում են առաջարկում կոդի օպտիմալացման մարտահրավերների համար: Օգտվելով DRL-ից՝ մշակողները կարող են բարելավել իրենց հավելվածների արդյունավետությունը՝ ապահովելով, որ դրանք արդյունավետ գործարկվեն և արդյունավետ օգտագործեն ռեսուրսները՝ ի վերջո հանգեցնելով ավելի լայնածավալ և հուսալի ծրագրային համակարգերի ստեղծմանը։

Այս ալգորիթմները համեմատելիս նրանցից յուրաքանչյուրը իր հետ բերում է յուրահատուկ ուժեղ կողմեր՝ դարձնելով դրանք առավել նպատակահարմար կոդի օպտիմալացման տարբեր ասպեկտների համար: Gradient Descent-ը առաջատար է շարունակական օպտիմալացման առաջադրանքներում՝ առաջարկելով ճշգրիտ պարամետրերի հաշվարկման հնարավորություններ: Որոշումների ծառերը ապահովում են հստակ և մեկնաբանելի մեթոդ՝ օպտիմիզացում պահանջող կրիտիկական կոդի բաժինները հայտնաբերելու և ըստ առաջնահերթության դասակարգելու համար: LSTM-ները հզոր են կանխատեսող վերլուծության համար՝ հնարավորություն տալով հաջորդական կոդերի օրինաչափությունների ակտիվ օպտիմալացում: Միևնույն ժամանակ, DRL-ն առաջարկում է կայուն շրջանակ՝ օպտիմալացման բարդ առաջադրանքների ավտոմատացման համար՝ հարմարվողական ուսուցման և շարունակական կատարելագործման միջոցով:

Ուստի, կոդի օպտիմալացման համար մեքենայական ուսուցման ալգորիթմի ընտրությունը կախված է կոդերի բազայի հատուկ պահանջներից և բնութագրերից: Gradient Descent-ը իդեալական է շարունակական օպտիմալացման առաջադրանքների համար, որոշումների ծառերը՝ առաջնահերթությունների և խոչընդոտների հայտնաբերման համար, LSTM-ները հաջորդական համատեքստերում կանխատեսելի օպտիմալացման համար, և DRL-ը՝ ամբողջական, հարմարվող օպտիմալացման ռազմավարությունների համար: Օգտագործելով յուրաքանչյուր ալգորիթմի ուժեղ կողմերը՝ մշակողները կարող են հասնել զգալի արդյունավետության և բարձրացնել իրենց կոդի արդյունավետությունը՝ ի վերջո հանգեցնելով ավելի ամուր և բարձր կատարողականությամբ օժտված ծրագրային համակարգերի (Նկար 1․4):



*Նկար 1․4։ Համեմատական վերլուծություն ալգորիթմների միջև։*

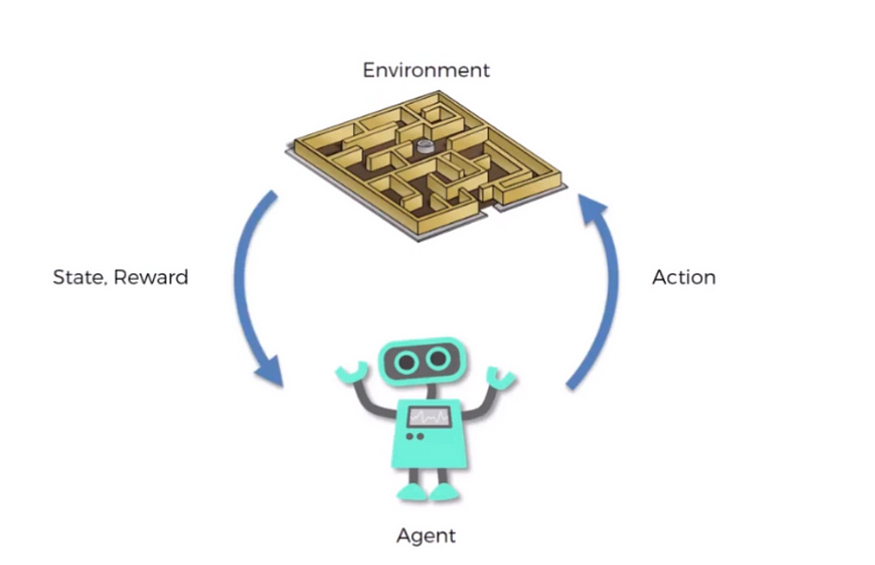
**Գլուխ 2**

**Ամրապնդման ուսուցման (RL) մեթոդներ։**

***2․1 RL մեթոդների էությունը և կիրառելիության ոլորտները։***

Հաշվողական տեխնոլոգիաների արագ զարգացումը նոր հնարավորություններ է բացում արհեստական ինտելեկտի (AI) կիրառման համար՝ ստեղծելով խելացի ալգորիթմներ, որոնք կարող են իմիտացնել մարդու մտածողությունը: AI-ը հիմնված է մեքենաների միջոցով մարդու ինտելեկտի նմանակման գաղափարի վրա, և դրա ենթաոլորտներից մեկը **մեքենայական ուսուցումն** է (ML), որը կենտրոնանում է այնպիսի համակարգերի մշակման վրա, որոնք կարող են ինքնուրույն կատարելագործվել փորձի միջոցով [1]: Առանձնացնում են ML-ի երեք հիմնական կատեգորիա՝ վերահսկվող ուսուցում, չվերահսկվող ուսուցում և ամրապնդման ուսուցում։ Վերահսկվող ուսուցումը կապված է պիտակավորված օրինակներով վերապատրաստված տվյալների շարքից սովորելու հետ: Նպատակն է սովորող գործակալին հնարավորություն տալ ընդհանրացնել իր պատասխանները այն դեպքերին, որոնք ներառված չեն թրեյնինգային հավաքածուում: Փոխարենը, չվերահսկվող ուսուցումը կապված է տվյալների բազայում թաքնված օրինաչափությունների և գիտելիքների հայտնաբերման հետ՝ առանց որևէ վերահսկողության: Վերջապես, ամրապնդման ուսուցումը (RL) ուսուցման տեսակ է, որն առաջնորդվում է որոշակի նպատակով: Գործակալը սովորում է՝ շփվելով անհայտ միջավայրի հետ, շարունակաբար՝ փորձելով և սխալվելով: Գործակալը արձագանք է ստանում շրջակա միջավայրից պարգևատրման (կամ պատժի) միջոցով. այնուհետև, օգտագործում է դա՝ իրեն մարզելու և շրջակա միջավայրի մասին փորձ և գիտելիքներ հավաքելու համար: RL-ի խնդրի մեկ այլ կարևոր հատկանիշն այն է, որ ցանկացած իրավիճակում գործակալը պետք է ընտրություն կատարի շրջակա միջավայրի մասին իր ներկայիս գիտելիքներն օգտագործելու (այդ իրավիճակում նախկինում փորձված գործողություն իրականացնելու) կամ այդ իրավիճակում նախկինում չփորձված գործողություններ ուսումնասիրելու միջև:

Նախքան RL ալգորիթմների և հավելվածների վերաբերյալ ուսումնասիրություն կատարելը, կարևոր է հասկանալ RL-ի հիմնական հասկացությունները (Նկար 2․1)։



Նկար 2․1։ Ամրապնդման ուսուցման հիմնական բաղադրիչները։

* ***Գործակալ։*** Գործակալը խելացի համակարգ կամ ծրագիր է, որը փոխազդում է միջավայրի հետ՝ սովորելու, թե ինչպես հասնել որոշակի նպատակի: Գործակալը սովորում է՝ ստանալով արձագանք շրջակա միջավայրից՝ իր գործողությունների համար պարգևների կամ պատիժների արժանանալու տեսքով: Գործակալի վերջնական նպատակն է սովորել մի քաղաքականություն, որը իրականացնում է շրջակա միջավայրի վիճակներից դեպի գործողություններ քարտեզագրում, որն առավելագույնի է հասցնում նրա երկարաժամկետ ակնկալվող պարգևը:

Գործակալը սովորաբար ունի երեք հիմնական բաղադրիչ՝ քաղաքականություն, արժեքային ֆունկցիա և ուսուցման ալգորիթմ: Քաղաքականությունը գործակալի ռազմավարությունն է՝ շրջակա միջավայրի ներկա վիճակի հիման վրա կատարվելիք գործողությունները ընտրելու համար: Արժեքի ֆունկցիան գնահատում է ակնկալվող երկարաժամկետ պարգևը, որը գործակալը կստանա որոշակի վիճակից և գործողությունից: Ուսուցման ալգորիթմը թարմացնում է գործակալի քաղաքականության և արժեքի ֆունկցիան՝ հիմնվելով շրջակա միջավայրից ստացած հետադարձ կապի վրա:

Գործակալի վարքագիծը բնութագրվում է հետախուզում- շահագործում փոխզիջումով: Գործակալը պետք է ուսումնասիրի շրջակա միջավայրը լավագույն քաղաքականությունը բացահայտելու համար, բայց նա նաև պետք է օգտագործի իր ներկայիս գիտելիքները՝ առավելագույնի հասցնելու իր ակնկալվող պարգևը: Այս փոխզիջումը հաճախ լուծվում է՝ օգտագործելով հետախուզական ռազմավարություն, ինչպիսին է epsilon-greedy կամ Thompson նմուշառումը, որը հավասարակշռում է հետախուզումն ու շահագործումը:

Ընդհանուր առմամբ, գործակալը ապրապնդման ուսուցման համակարգի կենտրոնական բաղադրիչն է և պատասխանատու է սովորելու և շրջական միջավայրի հետ փոխազդելու համար ՝ առաջադրված նպատակներին հասնելու համար:

* ***Շրջակա միջավայր։*** Ամրապնդման ուսուցման մեջ «միջավայր» տերմինը վերաբերում է արտաքին աշխարհին կամ համակարգին, որտեղ գործում է գործակալը: Դա այն միջավայրն է, որի հետ գործակալը շփվում և հետադարձ կապ է ստանում, և գործակալի նպատակն է դասեր քաղել այդ փոխազդեցություններից և օպտիմալացնել իր վարքագիծը՝ որոշակի նպատակի հասնելու համար:

Ամրապնդման ուսուցման միջավայրը կարող է լինել ամեն ինչ՝ վիրտուալ սիմուլյացիայից մինչև ֆիզիկական համակարգ, օրինակ՝ ռոբոտ կամ խաղ: Շրջակա միջավայրը գործակալին տրամադրում է մի շարք դիտարկումներ կամ վիճակներ, որոնք նկարագրում են համակարգի ներկա վիճակը: Այս դիտարկումներն օգտագործվում են գործակալի կողմից որոշումներ կայացնելու այն գործողությունների մասին, որոնք նա պետք է կատարի, և այն գործողությունները, որոնք նա ձեռնարկում է, որպեսզի հանգեցնի շրջակա միջավայրի վիճակի փոփոխությանը:

Շրջակա միջավայրը նաև հետադարձ կապ է տրամադրում գործակալին պարգևների կամ տուգանքների տեսքով՝ հիմնվելով նրա կատարած գործողությունների վրա: Այս պարգևները գործակալին ազդանշան են ծառայում ցուցադրած վարքագիծը ամրապնդելու կամ դրանից հրաժարվելու համար, իսկ գործակալի նպատակն է առավելագույնի հասցնել իր կուտակային պարգևը ժամանակի ընթացքում:

Շրջակա միջավայրը կարևոր դեր է խաղում ուսուցման հաջողության մեջ, քանի որ գործակալի կարողությունն է սովորել և օպտիմալացնել իր վարքագիծը, այն մեծապես կախված է շրջակա միջավայրի որակից և նրա տրամադրած հետադարձ կապից: Հետևաբար, ճշգրիտ և արդյունավետ միջավայրի ստեղծումը կարևոր է ամրապնդման ուսուցման հաջող համակարգերի զարգացման համար:

* ***Վիճակ։*** Վիճակը վերաբերում է ներկա իրավիճակին կամ միջավայրի կոնֆիգուրացիային, որում գտնվում է գործակալը: Այն ներառում է բոլոր համապատասխան տեղեկությունները, որոնք անհրաժեշտ են գործակալին՝ որոշումներ կայացնելու և գործողություններ կատարելու համար՝ առավելագույնի հասցնելու իր պարգևները: Վիճակը կարող է ներկայացվել տարբեր ձևերով, օրինակ՝ փոփոխականների կամ առանձնահատկությունների մի շարք, որոնք նկարագրում են ներկա իրավիճակը, շրջակա միջավայրի ամբողջական պատկերը կամ զգայական մուտքը: Ներկայացման ընտրությունը կախված է շրջակա միջավայրի բարդությունից և առաջադրված խնդրից:

Վիճակը կարևոր է, քանի որ այն որոշում է այն գործողությունները, որոնք կարող է ձեռնարկել գործակալը և այն պարգևները, որոնք նա կստանա: Գործակալի նպատակն է սովորել մի քաղաքականություն, որը քարտեզագրում է վիճակներից գործողություններ, որոնք առավելագույնի են հասցնում դրա կուտակային պարգևը ժամանակի ընթացքում: Հետևաբար, վիճակը ուսուցման գործընթացի կարևոր մասն է, քանի որ այն կազմում է գործակալի որոշումների կայացման գործընթացի հիմքը:

* ***Գործողություն։*** Գործողությունը վերաբերում է գործակալի կողմից ընդունված որոշմանը ի պատասխան շրջակա միջավայրի վիճակի: Դա կոնկրետ քայլ կամ վարքագիծ է, որը գործակալն ընդունում է որոշակի վիճակում՝ հաջորդ վիճակին անցնելու համար: Գործողությունը կարող է լինել ընտրություն հասանելի տարբերակների շարքից կամ տիրույթում շարունակական արժեք:

Օրինակ, շախմատային խաղում գործողությունը կարող է լինել խաղատախտակի որոշակի քառակուսի տեղափոխելը: Ինքնավար մեքենայում գործողությունը կարող է լինել արագացնելը, արգելակելը կամ ղեկը պտտելը: Ռոբոտի գործողությունը կարող է ենթադրել շարժվել դեպի որոշակի դիրք կամ պտտվել որոշակի ուղղությամբ: Գործակալի կողմից գործողության ընտրությունը շատ կարևոր է, քանի որ այն որոշում է շրջակա միջավայրից ստացվող պարգևները: Գործակալի նպատակն է սովորել օպտիմալ քաղաքականություն, որն առավելագույնի հասցնում է կուտակային պարգևները՝ ընտրելով հնարավոր լավագույն գործողությունները յուրաքանչյուր քայլում:

* ***Պարգևատրում։*** Պարգևը հետադարձ ազդանշան է, որը գործակալը ստանում է շրջակա միջավայրից գործողություն կատարելուց հետո: Պարգևատրման նպատակն է ցույց տալ, թե որքանով է գործակալն իր նպատակին հասնելու համար կատարել ճիշտ գործողություն, որը սովորաբար ժամանակի ընթացքում պարգևատրման կուտակային չափման առավելագույնի հասցնելն է:

Պարգևները կարող են լինել դրական, բացասական կամ զրոյական՝ կախված նրանից, թե գործակալի գործողությունը հանգեցրել է ցանկալի, անցանկալի կամ չեզոք արդյունքի: Գործակալի նպատակն է սովորել այնպիսի քաղաքականություն, որն առավելագույնի հասցնի ապագա պարգևների ակնկալվող գումարը կամ ակնկալվող եկամուտը: Պարգևատրման գործառույթի ձևավորումը ուժեղացման ուսուցման կարևոր կողմն է, քանի որ այն ուղղակիորեն ազդում է գործակալի վարքագծի վրա: Լավ մշակված պարգևատրման գործառույթը պետք է խթանի գործակալին հասնելու ցանկալի նպատակին՝ միաժամանակ խուսափելով չնախատեսված վարքագծից: Այնուամենայնիվ, պարգևատրման գործառույթների նախագծումը, որը ճշգրիտ կերպով կներկայացնի ցանկալի վարքագիծը, կարող է դժվար լինել, իսկ ոչ պատշաճ պարգևատրման գործառույթները կարող են հանգեցնել ոչ օպտիմալ կամ նույնիսկ անցանկալի վարքագծի:

* ***Քաղաքականություն։*** Քաղաքականությունը գործառույթ է, որը քարտեզագրում է գործակալի ներկայիս վիճակը տվյալ վիճակում կատարվող գործողության հետ: Քաղաքականությունը սահմանում է գործակալի վարքագիծը կամ ռազմավարությունը շրջակա միջավայրում գործողություններ ընտրելու համար: Քաղաքականությունը կարող է լինել դետերմինիստական, ինչը նշանակում է, որ այն միշտ ընտրում է նույն գործողությունը տվյալ վիճակի համար, կամ ստոխաստիկ, ինչը նշանակում է, որ նա ընտրում է գործողությունները հավանականորեն: Ստոխաստիկ քաղաքականության մեջ յուրաքանչյուր գործողություն կատարելու հավանականությունը տվյալ վիճակում որոշվում է քաղաքականության կողմից: Ամրապնդման ուսուցման նպատակը հաճախ օպտիմալ քաղաքականություն սովորելն է, որը ժամանակի ընթացքում առավելագույնի կհասցնի ակնկալվող կուտակային պարգևը: Սա սովորաբար ձեռք է բերվում փորձի և սխալի միջոցով, որտեղ գործակալը փոխազդում է շրջակա միջավայրի հետ, հետևում է ստացված պարգևներին և անցումներին դեպի նոր վիճակներ և թարմացնում է իր քաղաքականությունը՝ հիմնվելով դիտարկված արդյունքների վրա:
* ***Արժեքի Ֆունկցիա։*** Արժեքի ֆունկցիան գնահատում է վիճակի կամ վիճակ-գործողության զույգի արժեքը: Այն ցույց է տալիս, թե որքան լավ է որոշակի վիճակը կամ գործողությունը գործակալի նպատակին հասնելու առումով: Արժեքի ֆունկցիան ամրապնդման ուսուցման շատ ալգորիթմների կարևոր բաղադրիչն է, քանի որ այն առաջնորդում է գործակալի որոշումների կայացման գործընթացը:

Ամրապնդման ուսուցման մեջ կան երկու տեսակի արժեքային ֆունկցիաներ՝ վիճակ-արժեք ֆունկցիա և գործողություն-արժեք ֆունկցիա: Վիճակի վրա հիմնված ​​արժեքի ֆունկցիա. այն կանխատեսում է, թե որքան պարգև կարող է ակնկալել գործակալը ստանալ տվյալ վիճակից: Վիճակի ​​արժեք ֆունկցիան նշվում է V(s)-ով և սահմանվում է որպես ակնկալվող կուտակային պարգև, որը գործակալը կարող է ստանալ՝ սկսած տվյալ վիճակից և հետևելով ընթացիկ քաղաքականությանը:

Գործողություն-արժեք ֆունկցիա. այն կանխատեսում է, թե որքան պարգև կարող է ակնկալել գործակալը տվյալ վիճակում որոշակի գործողություն կատարելով: Գործողություն-արժեք ֆունկցիան նշվում է Q(s,a)-ով և սահմանվում է որպես ակնկալվող կուտակային պարգև, որը գործակալը կարող է ստանալ՝ սկսած s վիճակից, կատարելով a գործողություն և հետևելով ընթացիկ քաղաքականությանը:

Արժեքի ֆունկցիան գնահատվում է՝ օգտագործելով անցյալի փորձը և թարմացվում է կրկնվող ուսուցման մեթոդների միջոցով, ինչպիսիք են ժամանակային տարբերությունների ուսուցումը կամ Մոնտե Կառլոյի մեթոդները: Արժեքի ֆունկցիայի ճշգրիտ գնահատումը կարևոր է գործակալի համար օպտիմալ որոշումներ կայացնելու և իր երկարաժամկետ պարգևները առավելագույնի հասցնելու համար:

* ***Q-Ֆունկցիա։*** Q ֆունկցիան (նաև հայտնի է որպես գործողություն-արժեք ֆունկցիա) մաթեմատիկական ֆունկցիա է, որն ընդունում է վիճակ-գործակալ զույգը որպես մուտքագրում և ելնում է ակնկալվող երկարաժամկետ պարգևը այդ վիճակում այդ գործողությունը կատարելու և դրանից հետո որոշակի քաղաքականությանը հետևելու համար: Q ֆունկցիան կարևոր բաղադրիչ է ամրապնդման ուսուցման բազմաթիվ ալգորիթմներում, ինչպիսիք են Q-learning-ը և SARSA-ն, քանի որ այն թույլ է տալիս գործակալին գնահատել տարբեր գործողությունների որակը տարբեր վիճակներում: Սովորելով Q ֆունկցիան, գործակալն այնուհետև կարող է ընտրել այնպիսի գործողություններ, որոնք առավելագույնի են հասցնում ակնկալվող երկարաժամկետ պարգևը, ինչը ուժեղացման ուսուցման բազմաթիվ առաջադրանքների նպատակն է: Q ֆունկցիան հաճախ ներկայացված է որպես աղյուսակ կամ ֆունկցիայի մոտավորիչ, օրինակ՝ նեյրոնային ցանց, որը սովորում է փորձի և շրջակա միջավայրի հետ փոխազդեցության միջոցով: Q ֆունկցիայի սովորելու գործընթացը ներառում է Q արժեքների կրկնվող թարմացում՝ հիմնվելով դիտարկված պարգևների և անցումների վրա, մինչև գնահատականները համընկնեն իրական արժեքներին:
* ***Հետախուզում ընդդեմ շահագործման։*** Հետախուզումը և շահագործումը ուժեղացման ուսուցման երկու կարևոր հասկացություններ են, որոնք վերաբերում են այն բանին, թե ինչպես պետք է գործակալը ընտրի գործողություններ, որոնք պետք է կատարի միջավայրում:

Հետախուզումը վերաբերում է գործակալի վարքագծին՝ փորձելու նոր գործողություններ՝ շրջակա միջավայրի մասին ավելին իմանալու և պոտենցիալ ավելի լավ գործողություններ գտնելու համար, որոնք հանգեցնում են ավելի բարձր պարգևների: Մյուս կողմից, շահագործումը վերաբերում է գործակալի վարքագծին՝ ընտրելու գործողությունները, որոնք արդեն փորձված և ապացուցված են, որ հանգեցնում են բարձր պարգևների: Ամրապնդման ուսուցման մարտահրավերն է հավասարակշռել հետախուզումը և շահագործումը, որպեսզի առավելագույնի հասցվի գործակալի երկարաժամկետ վարձատրությունը: Եթե ​​գործակալն օգտագործում է միայն հայտնի լավ գործողությունները, նա կարող է բաց թողնել ավելի լավ գործողությունները, որոնք դեռ չի փորձել: Մյուս կողմից, եթե գործակալը միայն նոր գործողություններ է ուսումնասիրում, այն կարող է բավարար պարգև չհավաքել երկարաժամկետ հեռանկարում լավ գործելու համար: Առաջարկվել են հետախուզման տարբեր ռազմավարություններ, ինչպիսիք են ε-greedy, softmax և Upper Confidence Bound (UCB), ի թիվս այլոց: Այս ռազմավարությունները օգտագործում են տարբեր ուղիներ՝ հետախուզումն ու շահագործումը հավասարակշռելու համար, իսկ ռազմավարության ընտրությունը կախված է կոնկրետ խնդրից:

Կան մի քանի ալգորիթմներ, որոնք օգտագործվում են ամրապնդման ուսուցման մեջ: Ահա ամենատարածված օգտագործվողներից մի քանիսը.

* **Q-ուսուցում:** Q-Learning-ը արժեքի վրա հիմնված RL ալգորիթմ է, որը կենտրոնանում է արժեքի ուսուցման վրա` որոշակի իրավիճակներում որոշակի գործողություններ կատարելու միջոցով, որոնք ներկայացված են Q արժեքներով: Հիմնական գաղափարն է թարմացնել այս Q արժեքները՝ հիմնվելով շրջակա միջավայրից ստացված պարգևների վրա: Ալգորիթմն օգտագործում է կրկնվող գործընթաց՝ յուրաքանչյուր վիճակ-գործողություն զույգի Q արժեքները թարմացնելու համար՝ հիմնվելով շրջակա միջավայրից ստացված դիտարկվող պարգևների վրա: Թարմացումները կատարվում են օգտագործելով Բելմանի հավասարումը, որն արտահայտում է ընթացիկ վիճակի ակնկալվող արժեքը որպես անմիջական պարգևի և հաջորդ վիճակի ակնկալվող արժեքի գումար: Կրկնվող Q-արժեքները թարմացնելով, Q-ուսուցումը համընկնում է օպտիմալ Q-ֆունկցիայի և, հետևաբար, օպտիմալ քաղաքականության հետ: (Նկար 2․2)



*Նկար 2․2։ Բելմանի հավասարում, որտեղ 𝛼-ը սովորելու գործակիցն է, 𝛾-ը զեղչի գործոնն է, r-ը պարգևն է, s՛-ը հաջորդ վիճակն է իսկ 𝑎՛-ը հաջորդ գործողությունն է:*

Q-learning-ի հիմնական առավելություններից մեկն այն է, որ այն կարող է սովորել օպտիմալ քաղաքականություն մեծ վիճակային տարածքներով և ստոխաստիկ պարգևներով միջավայրերում: Այնուամենայնիվ, դա պահանջում է վիճակներ - գործողություններ արժեքների բավարար ուսումնասիրություն՝ խուսափելու համար ոչ օպտիմալ քաղաքականությունից, որը կարող է դժվար լինել որոշ միջավայրերում: Բացի այդ, Q-learning-ը ենթադրում է, որ վիճակի անցման և պարգևատրման գործառույթներն անհայտ են, ինչը միշտ չէ, որ կարող է լինել գործնականում:

* **Deep Q-Network (DQN)։** Deep Q-Network-ը (DQN) խորը ամրապնդման ուսուցման հանրաճանաչ ալգորիթմ է, որը ներդրվել է DeepMind-ի կողմից 2013 թվականին: Այն ընդլայնում է Q-ուսուցման ալգորիթմը՝ աշխատելու մեծ չափերի մուտքային տվյալների հետ՝ օգտագործելով խորը նեյրոնային ցանց՝ Q - ֆունկցիան մոտավորելու համար: DQN-ում գործակալն օգտագործում է նեյրոնային ցանց՝ գնահատելու յուրաքանչյուր հնարավոր գործողության Q արժեքները տվյալ վիճակում: Ցանցը ընդունում է վիճակը որպես մուտքագրում և թողարկում է գնահատված Q արժեքները յուրաքանչյուր հնարավոր գործողության համար: Գործակալն այնուհետև ընտրում է ամենաբարձր գնահատված Q արժեք ունեցող գործողությունը, որը պետք է կատարվի:

DQN-ին վարժեցնելու համար գործակալն օգտագործում է փորձի վերարտադրումը և թիրախային ցանցը: Փորձի վերարտադրումը տեխնիկա է, որտեղ գործակալը պահպանում է անցումները շրջակա միջավայրից բուֆերում և նմուշառում է այդ անցումների պատահական խմբաքանակները՝ նեյրոնային ցանցը մարզելու համար: Սա օգնում է կոտրել հարաբերակցությունը հաջորդական նմուշների միջև և բարելավել ուսումնական գործընթացի կայունությունը: Թիրախային ցանցը առանձին ցանց է, որն օգտագործվում է ուսուցման համար Q արժեքի թիրախներ ստեղծելու համար: Թիրախային ցանցի կշիռները սառեցված են և միայն պարբերաբար թարմացվում են հիմնական ցանցի կշիռներով, ինչը օգնում է կայունացնել ուսումնական գործընթացը:

* **Քաղաքականության գրադիենտ։** Քաղաքականության գրադիենտ ալգորիթմները ամրապնդող ուսուցման մեթոդների դաս են, որոնք սովորում են քաղաքականության գործառույթը՝ ուղղակիորեն օպտիմիզացնելով նպատակային ֆունկցիան, որը չափում է քաղաքականությանը հետևելով ստացված ակնկալվող կուտակային պարգևը: Քաղաքականության գործառույթը քարտեզագրում է վիճակները գործողություններին, իսկ ալգորիթմի նպատակն է գտնել քաղաքականություն, որը առավելագույնի է հասցնում ակնկալվող կուտակային պարգևը երկարաժամկետ հեռանկարում: Քաղաքականության գրադիենտ մեթոդները օգտագործում են գրադիենտ անկում, որպեսզի պարբերաբար թարմացնեն քաղաքականության պարամետրերը՝ առավելագույնի հասցնելու ակնկալվող կուտակային պարգևը: Քաղաքականությունը սովորաբար ներկայացված է որպես նեյրոնային ցանց, որտեղ մուտքային տվյալը վիճակն է, իսկ ելքը՝ հավանականության բաշխումը հնարավոր գործողությունների վրա: Օբյեկտիվ ֆունկցիայի գրադիենտը հաշվարկվում է քաղաքականության պարամետրերի նկատմամբ, և պարամետրերը թարմացվում են գրադիենտի ուղղությամբ շարժվելու համար: Քաղաքականության գրադիենտի հանրաճանաչ ալգորիթմը REINFORCE ալգորիթմն է, որը Մոնտե Կառլոյի ալգորիթմ է, որը գնահատում է ակնկալվող պարգևատրման գրադիենտը՝ օգտագործելով ընթացիկ քաղաքականության նմուշները: Մեկ այլ հանրաճանաչ ալգորիթմ է Actor-Critic ալգորիթմը, որը համատեղում է քաղաքականության գրադիենտը արժեքային ֆունկցիայի գնահատման հետ: Actor-Critic ալգորիթմը օգտագործում է նեյրոնային ցանց՝ քաղաքականությունը ներկայացնելու համար, և մեկ այլ նեյրոնային ցանց՝ արժեքային ֆունկցիան ներկայացնելու համար: Քաղաքականության ցանցը թարմացվում է՝ օգտագործելով քաղաքականության գրադիենտը, մինչդեռ արժեքային ցանցը թարմացվում է՝ օգտագործելով ժամանակային տարբերության (TD) սխալը, որը կանխատեսված և իրական պարգևների տարբերությունն է:

Քաղաքականության գրադիենտ մեթոդները մի քանի առավելություններ ունեն արժեքի վրա հիմնված մեթոդների նկատմամբ, ինչպիսիք են Q-learning-ը, ներառյալ՝ ստոխաստիկ քաղաքականություններ սովորելու ունակությունը, շարունակական գործողությունների տարածքների մշակումը և ավելի լավ կոնվերգենցիայի հատկությունները:

* **Actor - Critic:** Actor - Critic ալգորիթմները ամրապնդող ուսուցման ալգորիթմի տեսակ են, որոնք համատեղում են ինչպես քաղաքականության վրա հիմնված, այնպես էլ արժեքի վրա հիմնված մեթոդների առավելությունները: Այս ալգորիթմը ներառում է երկու նեյրոնային ցանց՝ actor ցանց և critic ​​ցանց:

Actor ցանցը պատասխանատու է շրջակա միջավայրի ներկա վիճակի հիման վրա գործողությունների ընտրության համար: Այն օգտագործում է քաղաքականության գրադիենտ մեթոդը՝ իր պարամետրերը թարմացնելու և իր կատարողականությունը բարելավելու համար: Critic ​​ցանցը, մյուս կողմից, գնահատում է ներկա վիճակի և գործողությունների զույգի արժեքը: Այն օգտագործում է ժամանակային տարբերությունների ուսուցման մեթոդը՝ իր պարամետրերը թարմացնելու և շրջակա միջավայրից ստացված արձագանքներից սովորելու համար: Actor - Critic ալգորիթմն օգտագործում է քննադատական ​​ցանցը տվյալ վիճակում գործողության արժեքը գնահատելու համար, այնուհետև օգտագործում է այս գնահատումը actor ցանցի քաղաքականությունը թարմացնելու համար: Սա թույլ է տալիս ալգորիթմին հավասարակշռել հետախուզումն ու շահագործումը ավելի արդյունավետ ձևով և բարելավել գործակալի ընդհանուր կատարողականի աստիճանը:

***2․2 RL մեթոդների կիրառման հնարավորությունը մշակվող ծրագրային հավելվածում։***

Մշակվող ծրագրային հավելվածը նպատակ ունի ավտոմատացնել կոդի օպտիմալացման գործընթացը՝ օգտագործելով առաջադեմ մեքենայական ուսուցման մեթոդներ։ Կոդի օպտիմալացման ավանդական մեթոդները, որոնք հաճախ հիմնվում են նախապես սահմանված կանոնների և հաստատուն տրամաբանության վրա, սահմանափակ են իրենց կիրառման շրջանակով և հաճախ չեն կարողանում հարմարվել փոփոխվող իրավիճակներին կամ նոր խնդիրներին։ Այս խնդիրները հաղթահարելու նպատակով, ծրագրային հավելվածում նախատեսվում է կիրառել **ամրապնդման ուսուցման (Reinforcement Learning, RL)** մեթոդները, որոնք ունակ են ինքնուրույն սովորել, հարմարվել տարբեր իրավիճակներին և բարելավել իրենց աշխատանքը՝ հիմնվելով փորձից ստացված արդյունքների վրա։ Մշակվող հավելվածը պետք է իրաանացնի հետևյալ հիմնական քայլերը.

1. Դասակարգել մուտքային կոդերի հատվածները՝ հիմնվելով նրանց կառուցվածքային և կատարողական հատկանիշների վրա։
2. Խմբավորել այդ հատվածները համանման հատկանիշներով կլաստերների մեջ։
3. Յուրաքանչյուր կլաստերի համար RL մեթոդների միջոցով ընտրել օպտիմալացման այն ռազմավարությունները, որոնք առավելագույնս բարելավում են կատարողականի չափանիշները։
4. Հետագայում՝ նոր մուտքային կոդի դեպքում, դասակարգել այն համապատասխան խմբի մեջ և կիրառել համապատասխան օպտիմալացման մեթոդները։

Այս մոտեցումը հնարավորություն է տալիս ոչ միայն հասնել կոդի օպտիմալացման ավտոմատացման, այլև ապահովել, որ օպտիմալացման գործընթացը լինի առավել ճշգրիտ, արդյունավետ և համապատասխան տվյալ խնդրի պահանջներին։

**Ամրապնդման ուսուցումը (RL)**, ի տարբերություն դասական մեքենայական ուսուցման մեթոդների, աշխատում է դինամիկ միջավայրում, որտեղ գործակալը սովորում է որոշումներ ընդունել՝ ելնելով կատարած գործողությունների արդյունքներից։ Մշակվելիք հավելվածում RL մեթոդների կիրառումը թույլ կտա ոչ միայն կատարել գործողություններ կոդի օպտիմալացման համար, այլ նաև սովորել, թե ինչպես կազմակերպել այդ գործողությունների հաջորդականությունը՝ հասնելու առավելագույն արդյունավետության։ RL-ի օգտագործման հիմնական բաղադրիչները մշակվելիք համակարգում՝

1. Վիճակի ներկայացում (State Representation): Կոդի օպտիմալացման համակարգում RL գործակալը պետք է իմանա, թե ինչ վիճակում է գտնվում մուտքային կոդը։ Վիճակը ներկայացվում է որպես վեկտոր, որը պարունակում է կոդի կառուցվածքային, կատարողական և այլ կարևոր հատկանիշների վերաբերյալ տեղեկատվություն։ Օրինակ՝
   * Կառուցվածքային հատկանիշներ․ տողերի քանակը, ցիկլերի քանակը, պայմանական արտահայտությունների թիվը, ֆունկցիաների կամ մեթոդների քանակը։
   * Կատարողական հատկանիշներ․ կոդի գործարկման ժամանակը (runtime), հիշողության օգտագործման ծավալը և CPU-ի ծանրաբեռնվածության մակարդակը։
   * Լրացուցիչ հատկանիշներ․ գրադարաններից կամ երրորդ կողմի գործիքներից կախվածություններ։

Այս տեղեկատվությունը թույլ է տալիս գործակալին ձևավորել հստակ պատկերացում այն մասին, թե ինչ իրավիճակում է գտնվում կոդը, ինչն էլ հիմք է դառնում օպտիմալացման ռազմավարության ընտրության համար։

1. Գործողությունների տարածք (Action Space): RL գործակալը օպտիմալացման գործընթացում պետք է ընտրի, թե որ գործողությունը կատարի տվյալ պահին։ Մեր համակարգում գործողությունների տարածքը ներառում է մի շարք օպտիմալացման տեխնիկաներ, որոնք կարող են կիրառվել կոդի վրա՝ օրինակ՝
   * Շերտերի կրճատում (Layer Pruning): Oգտագործվում է TensorFlow մոդելներում՝ ավելորդ շերտերը հեռացնելու և մոդելի հաշվարկային բարդությունը նվազեցնելու համար։
   * Քանակականացում (Quantization): Տվյալների ճշգրտության նվազեցում, որը թույլ է տալիս արագացնել հաշվարկները՝ պահպանելով կատարման ընդունելի ճշգրտությունը։
   * Ցիկլերի ապահոսք (Loop Unrolling): Ցիկլերի վերաձևակերպման տեխնիկա՝ նվազեցնելու կրկնվող գործողությունների ծախսերը։
   * Հիշողության նախաբեռնում (Memory Prefetching): Հիշողության հասանելիության կարգաբերում, որը նվազեցնում է հիշողության սպասման ժամանակը և բարելավում է ընդհանուր կատարողականը։
2. Պարգևատրման մեխանիզմ (Reward Function): RL գործակալի ուսուցման գործընթացը հիմնված է պարգևատրման մեխանիզմի վրա։ Գործակալը ստանում է դրական կամ բացասական պարգև՝ ելնելով իր գործողությունների ազդեցությունից։
   * Եթե օպտիմալացման գործողությունը բարելավում է կոդի կատարողականը՝ նվազեցնելով գործարկման ժամանակը կամ հիշողության ծախսը, գործակալը ստանում է դրական պարգև։
   * Եթե գործողությունը հակառակը՝ վատթարացնում է կոդի կատարողականը կամ մեծացնում է ռեսուրսների ծախսը, գործակալը ստանում է բացասական պարգև։
3. Հերթական որոշումների ընդունում (Sequential Decision Making): Կոդի օպտիմալացման գործընթացը հաճախ պահանջում է մի քանի գործողությունների համակցված կիրառություն։ Օրինակ, միայն շերտերի կրճատումը բավարար չէ՝ օպտիմալացման լավագույն արդյունքին հասնելու համար։ RL գործակալը սովորում է, թե ինչ հաջորդականությամբ պետք է կիրառել գործողությունները։ Օրինակ՝ նախ կարելի է իրականացնել ցիկլերի ապահոսք, ապա հիշողության նախաբեռնում։

RL գործակալը ուսուցանվում է տվյալների մեծ հավաքածուի հիման վրա, որը պարունակում է տարբեր կառուցվածքային առանձնահատկություններով և կատարողական խնդիրներով կոդի հատվածներ։ Ուսուցման գործընթացը ներառում է հետևյալ քայլերը.

1. Տվյալների հավաքագրում և նախապատրաստում: Համակարգը հավաքագրում է կոդերի լայն ընտրանի, որոնք ներկայացնում են տարբեր կառուցվածքներ և խնդիրներ։ Յուրաքանչյուր կոդի հատված գնահատվում է՝ արտացոլելու համար նրա կատարողականի նախնական վիճակը։
2. Հայտությունների արդյունահանում (Feature Extraction): Յուրաքանչյուր կոդի հատվածի համար առանձնացվում են վերոնշյալ հատկանիշները՝ ապահովելով RL գործակալի համար անհրաժեշտ տեղեկատվությունը։
3. Գործակալի փոխազդեցություն (Agent Interaction): RL գործակալը պատահական կամ փորձարկված ռազմավարություններով կիրառում է օպտիմալացման գործողություններ։
4. Կատարողականի գնահատում (Performance Evaluation): Յուրաքանչյուր գործողությունից հետո համակարգը նորից գնահատում է կոդի կատարողականը։
5. Քաղաքականության թարմացում (Policy Update): Գործակալը ուսուցման ընթացքում մշտապես թարմացնում է իր ռազմավարությունը՝ ավելի լավ որոշումներ ընդունելու համար։

RL մեթոդների ինտեգրումը կոդի օպտիմալացման գործընթացին խոստումնալից մոտեցում է, որը հնարավորություն է տալիս լուծել բարդ խնդիրներ։ Մշակվելիք ծրագրային հավելվածը, հենվելով RL մեթոդների վրա, կկարողանա ոչ միայն ավտոմատացնել օպտիմալացման գործընթացը, այլև հասնել շարունակական կատարելագործման՝ ապահովելով հարմարավետություն և արդյունավետություն ծրագրային ապահովման մշակման տարբեր փուլերում։

**Գլուխ 3**

**Ծրագրային հավելվածի նախագծում և մշակում։**

**3․1 Հավելվածի բնույթը, աշխատանքի սկզունքը։**

Ծրագրային ապահովման մշակման ոլորտում հավելվածները հետզհետե դառնում են ավելի բարդ և մասշտաբային, ուստի կտրուկ աճում է նաև բարձր արդյունավետությամբ աշխատող ծրագրային ապահովման պահանջարկը: Կոդերի ավանդական մեթոդներով օպտիմալացումը, թեև արդյունավետ է, սակայն աշխատատար է և էապես կախված է նախագծողների հնարավոր սխալներից, հաճախ չհամապատասխանելով ժամանակակից ծրագրային ապահովմանը ներկայացվող պահանջներին: Հենց այստեղ է ի հայտ գալիս մեքենայական ուսուցման (ML) և մասնավորապես ամրապնդման ուսուցման (RL) մեթոդների կիրառման անհրաժեշտությունը, որն առաջարկում է բավականին լավ մոտեցում՝ ավտոմատացնելու և բարելավելու կոդի օպտիմալացման գործընթացը [4]:

Մշակվող ծրագրային հավելվածի հիմնական նպատակն է հնարավորինս ավտոմատացնել և կատարելագործել կոդի օպտիմալացման գործընթացը՝ օգտագործելով մեքենայական ուսուցման, մասնավորապես ամրապնդման ուսուցման մոտեցումները։ Մշակվող ծրագրային հավելվածը համակարգված կերպով կդասակարգի և կօպտիմալացնի հավաքագրված կոդի հատվածները, դրանով իսկ բարելավելով ծրագրի աշխատանքը և արդյունավետությունը՝ նվազեցնելով օգտագործվող ռեսուրսները ու ծրագրի կատարման ժամանակը:

Մեքենայական ուսուցման (ML) և մասնավորապես ամրապնդման ուսուցման (RL) մեթոդները հնարավորություն են տալիս ավտոմատացնել կոդի օպտիմալացման գործընթացը՝ նվազեցնելով ծրագրավորողների աշխատանքային ծանրաբեռնվածությունը և ապահովելով ավելի ճշգրիտ ու օպտիմալ արդյունքներ։ Ամրապնդման ուսուցումը հիմնվում է ինտերակտիվ և ինքնակատարելագործվող գործընթացների վրա, որտեղ համակարգը սովորում է որոշում ընդունել՝ ելնելով նախորդ փորձի և ստացված արդյունքների վրա ունեցած ազդեցությունից։ Մշակվող ծրագրային հավելվածի հիմնական նպատակը հենց այս մեթոդները կիրառելն է՝ կոդի օպտիմալացման գործընթացը ոչ միայն ավտոմատացնելու, այլև կատարելագործելու համար։ Հավելվածը կդառնա մի գործիք, որն ունակ կլինի համակարգված մոտեցմամբ վերլուծել ծրագրային կոդի հատվածները, դասակարգել դրանք ըստ հատկանիշների և ընտրել օպտիմալացման ամենահարմար մեթոդները՝ հաշվի առնելով կոդի բնույթը, կառուցվածքը և պահանջները:

Ծրագրի հիմնական առանձնահատկությունը կայանում է նրանում, որ այն ունակ է մշակելու կոդի օպտիմալացման գործընթացի ամբողջական շղթա։ Նախ, հավելվածը վերլուծում է մուտքագրված կոդը՝ հայտնաբերելով այնպիսի հատկանիշներ, ինչպիսիք են ֆունկցիաների բարդությունը, ցիկլերի խորությունը կամ տվյալների կառավարման մեխանիզմները։ Ապա, ելնելով այս վերլուծությունից, այն դասակարգում է կոդի հատվածները տարբեր խմբերի, ինչը թույլ է տալիս առավել նպատակաուղղված և արդյունավետ լուծումներ գտնել: Հետագա փուլում հավելվածը կիրառում է ամրապնդման ուսուցման մեթոդներ, որոնք հնարավորություն են տալիս մշակել և կատարելագործել օպտիմալացման ռազմավարությունները՝ ելնելով ոչ միայն կոդի տվյալների բազայից, այլև իրական գործողությունների ազդեցություններից:

Հավելվածի կառուցվածքը ներառում է մի քանի կարևոր մոդուլներ, որոնք միասին ապահովում են նրա ամբողջական և արդյունավետ աշխատանքը։ Մուտքային կոդի վերլուծության մոդուլը զբաղվում է տվյալների արտածմամբ և նախապատրաստմամբ, որին հաջորդում է դասակարգման և կլաստերացման փուլը։ Այս փուլը կարևոր է, քանի որ կոդի տարբեր տեսակներ պահանջում են օպտիմալացման տարբեր ռազմավարություններ։ Օրինակ, ցիկլերով ծանրաբեռնված կոդը կարող է օպտիմալացվել ապահոսքի (loop unrolling) կամ բեռնաթափման (offloading) մեթոդներով, իսկ հիշողության ինտենսիվ կոդը՝ տվյալների քանակականացման (quantization) տեխնիկայով։ Կարևոր է նաև այն հանգամանքը, որ հավելվածը ինքնակատարելագործվող է։ RL մեթոդների շնորհիվ այն ժամանակի ընթացքում սովորում է իր իսկ գործած սխալներից և հաջողություններից՝ ստեղծելով ավելի ճշգրիտ և արդյունավետ օպտիմալացման համակարգ։ Այս մոտեցումը հնարավորություն է տալիս հավելվածին հարմարվել նոր մարտահրավերներին և ծրագրավորման նոր պահանջներին՝ առանց մշտական թարմացումների։

Մշակվող հավելվածը մեծ նշանակություն ունի ոչ միայն տեխնիկական, այլև գործնական տեսանկյունից։ Այն առաջարկում է արդյունավետ լուծում, որը կարող է զգալիորեն նվազեցնել ծրագրավորողների աշխատանքային ծանրաբեռնվածությունը։ Նման հավելվածը թույլ է տալիս խնայել ոչ միայն ժամանակ, այլև ռեսուրսներ՝ միաժամանակ բարձրացնելով ծրագրերի որակը։ Այսպիսի լուծումը հատկապես կարևոր է մեծածավալ ծրագրային նախագծերի դեպքում, որտեղ ամենափոքր օպտիմալացումը կարող է բերել զգալի փոփոխությունների՝ ինչպես ծրագրի կատարման արագության, այնպես էլ գործարկման ծախսերի տեսանկյունից։

Հավելվածի աշխատանքի սկզբունքը հիմնված է տվյալների վերլուծության, դասակարգման, օպտիմալացման ռազմավարությունների մշակման և դրանց կիրառման հաջորդական գործընթացի վրա։ Այս սկզբունքը իրագործվում է մի քանի փոխկապակցված մոդուլների միջոցով, որոնք միասին ապահովում են հավելվածի ամբողջական աշխատանքը։

#### ****Նախնական ուսուցման փուլ:****

Հավելվածի ամբողջական աշխատանքը հնարավոր է դառնում նախնական ուսուցման գործընթացի միջոցով։ Այս փուլում համակարգը կուտակում է փորձառություն՝ ուսումնասիրելով հավաքագրված կոդի բազան, որի նպատակն է հասկանալ տարբեր տեսակի կոդերի օպտիմալացման արդյունավետ մեթոդները։ Նախնական ուսուցման համար պահանջվում է․

* **Կոդերի մեծ բազա**, որը պետք է ներառի տարբեր բնույթի ծրագրեր և կոդի հատվածներ։ Այս բազայում պետք է ներառվեն պարզ կոդեր, բարդ ֆունկցիաներ, ցիկլերով հագեցած հատվածներ և հիշողության ինտենսիվ ծրագրեր։
* **Հատկանիշների ուսումնասիրություն**, որտեղ յուրաքանչյուր կոդի առանձնահատկությունները վերլուծվում են։ Այս հատկությունները ներառում են կոդի կառուցվածքը, ֆունկցիոնալ բարդությունը, հիշողության և հաշվարկային ռեսուրսների պահանջները։
* **Օպտիմալացման մեթոդների փորձարկում**, որտեղ յուրաքանչյուր կոդի վրա փորձարկվում են տարբեր ռազմավարություններ։ RL մեթոդները հնարավորություն են տալիս համակարգին կատարել փորձարկումներ՝ գնահատելով, թե որ մեթոդներն են ապահովում առավելագույն արդյունավետություն։

Այս ուսուցման գործընթացի արդյունքում համակարգը ստեղծում է օպտիմալացման որոշումների բազա, որտեղ գրանցված են կոդի տարբեր տեսակների և դրանց օպտիմալացման ռազմավարությունների միջև կապերը։ Այս բազան կարևոր է հետագա փուլերում, երբ համակարգը կկայացնի որոշումներ նոր մուտքագրված կոդերի օպտիմալացման վերաբերյալ։ Հավելվածի աշխատանքի այս փուլում իրականացվում է ծրագրային կոդի հավաքագրում և դրա նախնական վերլուծություն։ Օգտագործողը ներմուծում է այն կոդը, որը պետք է օպտիմալացվի։ Այս փուլում հավելվածը կատարում է կոդի կառուցվածքային վերլուծություն՝ հայտնաբերելով այնպիսի հատկանիշներ, ինչպիսիք են․

* **Տողերի քանակը**, որը ցուցադրում է կոդի ծավալը։
* **Ցիկլերի և ֆունկցիաների բարդությունը**, ինչը ազդում է կատարման արագության վրա։
* **Հիշողության կառավարման մեխանիզմները**, որոնք կարևոր են ռեսուրսների արդյունավետ օգտագործման համար։

Այս վերլուծությունը թույլ է տալիս համակարգին առաջին քայլով հասկանալ կոդի տեսակը, կառուցվածքը և հնարավոր օպտիմալացման պահանջները։

#### ****Կոդի դասակարգում և կլաստերավորում։****

Հաջորդ փուլում հավելվածը, օգտագործելով մեքենայական ուսուցման մեթոդներ, դասակարգում է հավաքագրված կոդերը։ Սա ենթադրում է կոդի հատվածների բաժանում տարբեր խմբերի՝ ելնելով դրանց կառուցվածքային և ֆունկցիոնալ հատկանիշներից։ Օրինակ, հավելվածը կարող է տարբերակել․

* Ցիկլերով հագեցած կոդը։
* Հիշողության ինտենսիվ կոդը։
* Ցանցային ինտենսիվ (network-intensive) ֆունկցիաները։

Այնուհետև կիրառվում են կլաստերավորման ալգորիթմներ, որոնք համախմբում են նման հատկություններ ունեցող կոդերը խմբերում։ Սա կարևոր քայլ է, քանի որ տարբեր խմբերի կոդերի համար կիրառվում են տարբեր օպտիմալացման ռազմավարություններ։

#### ****Օպտիմալացման ռազմավարությունների ընտրություն։****

Կոդի վերլուծությունից և կլաստերավորումից հետո հավելվածը ընտրում է օպտիմալացման ամենահարմար ռազմավարությունը։ Այս փուլում ամրապնդման ուսուցումը (RL) հանդիսանում է առանցքային բաղադրիչ։ RL մեթոդները թույլ են տալիս հավելվածին ինքնուրույն փորձարկել տարբեր ռազմավարություններ՝ ուսումնասիրելով դրանց ազդեցությունը կոդի կատարման վրա։  
Օրինակ, RL գործակալը կարող է փորձարկել տարբեր մեթոդներ, ինչպիսիք են՝

* **Ապահոսքը (loop unrolling)**՝ նվազեցնելու կրկնվող հաշվարկների քանակը։
* **Տվյալների քանակականացումը (quantization)**՝ հիշողության օգտագործումը նվազեցնելու համար։
* **Տվյալների բեռնաթափումը (offloading)**՝ ծանր հաշվարկները տեղափոխելով ավելի հզոր միջավայրեր։

Յուրաքանչյուր փորձից հետո RL գործակալը գնահատում է ստացված արդյունքները, սովորում նախորդ գործողություններից և պարգևատրվում կամ պատժվում՝ ելնելով կոդի կատարողականության փոփոխություններից։

#### ****Նոր կոդի օպտիմալացում։****

Հավելվածի աշխատանքը շարունակվում է նոր մուտքային կոդի հետ։ Ներմուծելով նոր կոդ, համակարգը այն վերլուծում է, դասակարգում և ընտրում համապատասխան օպտիմալացման ռազմավարությունը։ Այս ավտոմատացված գործընթացն ապահովում է արագ և ճշգրիտ օպտիմալացում՝ առանց հավելյալ ծրագրավորման։

#### ****Ինքնակատարելագործում։****

Հավելվածի հիմնական առավելություններից մեկն այն է, որ այն ժամանակի ընթացքում ինքնակատարելագործվում է։ Օգտագործելով RL մեթոդները՝ հավելվածը կուտակում է փորձառություն, և յուրաքանչյուր հաջորդ գործառույթ ավելի արդյունավետ է դառնում։ Սա հնարավորություն է տալիս համակարգին հարմարվել նոր մարտահրավերներին և մշտապես կատարելագործել իր օպտիմալացման մեթոդները։

Հավելվածի աշխատանքի արդյունքում ծրագրավորողները հնարավորություն կունենան խնայել զգալի ժամանակ և ռեսուրսներ, քանի որ օպտիմալացման գործընթացը իրականացվում է ավտոմատացված կերպով։ Այն թույլ կտա մշակել ավելի արագ և արդյունավետ ծրագրեր՝ նվազեցնելով հիշողության օգտագործումը, կրճատելով կատարման ժամանակը և ապահովելով ռեսուրսների ավելի խնայող օգտագործում։

***3․2 Հավելվածի նախագծման և մշակման ընթացքը։***

Հավելվածի մշակման պրոցեսը ներառում է մի քանի հիմնական փուլեր, որոնք պետք է հաջորդաբար իրականացվեն՝ ապահովելով հավելվածի ճիշտ աշխատանքն ու արդյունավետությունը։ Ստորև ներկայացվում է յուրաքանչյուր քայլը, ինչպես նաև՝ համապատասխան կոդային օրինակները։

Ներկայացնենք կատարված հետազոտության և մշակվող հավելվածի նախագծման ընթացքում իրականացված քայլերը։

1․ Տեսական ուսումնասիրություն․ ուսումնասիրվել են ամրապնդման ուսուցման (RL) տեսական հիմունքները, այսինքն՝ թե ինչպես է համակարգը (գործակալը, agent) սովորում շրջակա միջավայրից ստացած տվյալների և փորձի հիման վրա:

2․ Կոդի օպտիմալացման հասկացությունների ուսումնասիրություն․ ուսումնասիրվել են RL-ի այն մեթոդները և մոտեցումները, որոնք հնարավոր է կիրառել կոդի օպտիմալացման համար:

3․ Կոդի օպտիմալացվող պարամետրերի սահմանում․ կազմվել է օպտիմալացման ենթարկվող հիմնական պարամետրերի ցանկը, ինչպիսիք են ծրագրի կատարման արագությունը, օգտագործված հիշողության ծավալը, օգտագործվող այլ ռեսուրսների ծավալը:

4․ Կոդի օպտիմալացման համար RL-ի հաջող կիրառելիության օրինակների վերլուծություն․ ուսումնասիրվել են հետազոտական նախագծեր, որտեղ RL-ը հաջողությամբ կիրառվել է օպտիմալացման համար։

5․ Օպտիմալացման գործիքների վերլուծություն․ կատարվել է, ինչպես RL-ի, այնպես էլ ավանդական մեթոդներով իրականացվող օպտիմալացման գործիքակազմի վերլուծություն, որոշվել են RL-ի հետ համատեղելի այն գործիքները, որոնք կարող են ինտեգրվել նախագծում։

6․ Օպտիմալացման համար տվյալների հավաքագրում․ հավաքագրվել են համապատասխան տվյալներ(ծրագրային կոդերի հատվածներ), որոնք կենթարկվեն օպտիմալացման, ընդ որում կոդերի հատվածները ընդգրկում են օպտիմալացման համար կիրառելի սցենարներ:

7․ RL գործակալի մշակում, ընտրություն․ ընտրվել է մշակվող հավելվածի համար RL-ի առավել հարմար ալգորիթմ՝ Q-learning - ը, որը լավագույնս համապատասխանում է օպտիմալացման նպատակներին:

8. Օգտագործման միջավայրի մշակում․ մշակվել է միջավայր, որտեղ գործակալը փորձարկում է օպտիմալացման տարբեր ռազմավարություններ և սովորում դրա հիման վրա:

9․ Ուսուցման փուլ․ իրականացվել է գործակալի վերապատրաստում՝ RL-ի ընտրված ալգորիթմի և հավաքագրված տվյալների հիման վրա:

10․ Օպտիմալացման արդյունքների վերլուծություն․ հաշվարկվել և վերլուծվել են RL գործակալի գործողություններից ստացված օպտիմալացման արդյունքները, որոնք համեմատվել են կոդերի սկզբնական բնութագրիչների հետ։

11․ Փորձարկման արդյունքների ամփոփում․ ներկայացվել են բարելավման ցուցանիշները և համակարգի օպտիմալացման հետ կապված վերլուծությունները ներառող վերջնական արդյունքները:

Ուստի կատարած քայլերի միջոցով ամբողջությամբ մշակվեց առաջադրված ծրագրային հավելվածը։

**Գրականության ցանկ**

1. John Hearty, Advanced Machine Learning with Python, 2016.
2. Alex Smola and S.V.N. Vishwanathan, Introduction to Machine Learning, 2010.
3. Tom M. Mitchell, Machine Learning, 1997.
4. Geoff Hulten, Building Intelligent Systems - A Guide to Machine Learning Engineering, 2018.
5. Jason Bell, Machine Learning - Hands-On for Developers and Technical Professionals, 2014.
6. [Kamel Abdous](https://www.researchgate.net/profile/Kamel-Abdous-2?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19), Automatic Code Optimization With Machine Learning And Combinatorial Optimization, 2020.
7. Yacine Hakimi, Riyadh Baghdadi, Yacine Challal, A Hybrid Machine Learning Model for Code Optimization, 2023.
8. Zheng Wang and Michael O’Boyle, Machine Learning in Compiler Optimisation, 2018.
9. Lovekush Chaurasia, Amol Borse, Optimization for Machine Learning Models, 2020.
10. <https://research.google/blog/mlgo-a-machine-learning-framework-for-compiler-optimization/>
11. <https://medium.com/@koushikkushal95/optimization-algorithms-in-machine-learning-a-comprehensive-guide-to-understand-the-concept-and-3db1df7a2f59>