دانشکده صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

گزارش درس روش پژوهش و ارائه

بررسی الگوریتم تکامل عصبی عمیق در آموزش شبکه های عصبی

نگارش
کیوان ایپچی حق

استاد مشاور
دکتر رضا صفا بخش

بهار ۱۴۰۱

سپاس گزاری

در ابتدا لازم میدانم که از استاد گرانقدر دکتر رضا صفا بخش نهایت تشکر را داشته باشم که مطالب بسیار مفیدی را به من آموختند و تهیه این گزارش بدون راهنمایی های ایشان امکان پذیر نبود.

کیوان ایپچی حق

بهار ۱۴۰۱

چکیده

نیاز به سیستم های خبره به سرعت در مسائل مختلف حال گسترش است. امروزه بیشتر از قبل در مسائل روزمره به هوش مصنوعی وابسته هستیم و با افزایش تقاضا و همچنین افزایش تنوع مسئله های مطرح شده، نیاز به سیستم هایی داریم که با خطای کمتر و در زمان کمتر خروجی مطلوب تری به ما بدهند.

در گذشته شبکه های عصبی طراحی شده بیشتر بر روی بازی ها و بر اساس میزان دستیابی به هدف از پیش تعریف شده آزمایش می شدند. با گذشت زمان شبکه های پیشرفته تری طراحی شدند که قادر هستند خود به تعریف درستی از هدف برسند و دخالت انسان کمتر و کمتر شد. از طرفی روش های مدرنی بر روی کار آمد که قادر بود در بسیاری از شرایط از روش های سنتی پیشی بگیرند.

در ابتدا سعی بر شناخت الگوریتم های پایه یادگیری عمیق داشته و بعد از معرفی چندین ویژگی قابل تامل این الگوریتم با چندین روش مطرح آن آشنا می شویم. سپس الگوریتم *NEAT* را معرفی و جنبه های مختلف آنرا برسی می کنیم. در فصل بعد با استفاده از آموخته های دو فصل پیشین ساختار های مختلف این الگوریتم را برسی کرده و به نتایج جالبی می رسیم.

واژه های کلیدی:

استراتژی های تکاملی، الگوریتم ژنتیک، تابع هدف، الگوریتم جستوجو تازگی، الگوریتم *NEAT*، شبیه سازی دوپا، مسئله مارپیچ

فهرست مطالب

فهرست اشکال

عنوان صفحه

فصل اول

مقدمه

۱- مقدمه

در زمینه یادگیری عمیق، شبکه های عصبی عمیق (*DNN*) با لایه های بسیار زیاد و میلیون ها نورون و اتصالات اکنون به طور معمول از طریق شیب نزول تصادفی[۱] آموزش داده می شوند. با این حال، در این مقاله برسی خواهیم کرد که تکامل عصبی عمیق[۲] که در آن شبکه های عصبی از طریق الگوریتم های استراتژی تکاملی بهینه سازی می شوند هم روشی مؤثر برای آموزش شبکه های عصبی عمیق در حل مسائل یادگیری تقویتی[۳] کاربرد دارد.

اوبر[۴] – شرکت حمل نقل آمریکایی - حوزه های زیادی دارد که در آنها یادگیری ماشینی می تواند به عملکرد سیستم ها کمک های بسزایی کند. توسعه طیف گسترده ای از رویکردهای یادگیری قدرتمند که شامل تکامل عصبی می شود، به این شرکت و دیگر شرکت ها کمک می کند تا به مأموریت آنها در راستای توسعه راه حل های حمل و نقل ایمن تر و مطمئن تر دست یابد.

در این مقاله ابتدا به برسی استراتژی های تکاملی پرداخته و ویژگی های کلیدی آن که موجب ایجاد تمایز در مقابل الگوریتم ها و استراتژی های دیگر شده را مورد مطالعه قرار می دهیم. سپس نشان می دهیم که با استفاده از یک الگوریتم ژنتیک[۵] ساده می توان شبکه های عصبی عمیق را بهینه کرد که باعث می شود این الگوریتم را به عنوان یک جایگزین رقابتی برای روش های مبتنی بر گرادیان در نظر گرفت که برای مسائلی مانند یادگیری بازی های آتاری، حرکت شبیه سازی شده دوپا یا مسئله مارپیچ فریبنده که عموما با یادگیری تقویتی حل می شوند، استفاده کرد. سپس الگوریتم جدیدی با نام *NEAT*[۶] معرفی شده و ویژگی های آن مطرح می شود که با استفاده از مفاهیم الگوریتم ژنتیک و حتی جستوجو تازگی[۷] کار می کند. در نهایت آموخته های فصل های پیشین را مانند تکه های پازل کنار هم گذاشته تا در آخرین فصل طی آزمایش های مختلف، این الگوریتم ها را در مسائل مختلف با یکدیگر مقایسه کرده و نتایج قابل ملاحظه ای بدست می آوریم.

---

[1] Stochastic Gradient Decent (SGD)
[2] Neuroevolution
[3] Reinforcement Learning (RL)
[4] Uber Technologies Inc.
[5] Genetic Algorithm (GA)
[6] Neuroevolution of augmenting topologies (NEAT)
[7] Novelty Search (NS)

فصل دوم

استراتژی های تکاملی

۲- استراتژی های تکاملی[8]

استراتژی های تکاملی یک کلاس از الگوریتم های بهینه سازی در سری الگوریتم های جست و جوی اکتشافی است که از تکامل طبیعی الهام گرفته شده و به آن ها همانند شبکه های عصبی، به صورت یک جعبه سیاه نگاه می شود. این استراتژی ها متعلق به الگوریتم های تکاملی بوده که از جهش، ترکیب مجدد و انتخاب کاندید ها برای چرخه بعدی که اصطلاحا به آن نسل هم گفته می شود، استفاده می کنند. برای درک بهتر این موضوع شبه کد کلی این استراتژی ها آورده شده است:

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

شکل ۱- شبه کد استراتژی های تکاملی

الگوریتم های داخل این کلاس از نظر نحوه نمایش جمعیت، اعمال جهش و ترکیب با هم تفاوت هایی دارند. از معروف ترین اعضای این کلاس می توان به استراتژی تکامل انطباق ماتریس کوواریانس[9] و الگوریتم ژنتیک[10] اشاره کرد. در ابتدا به ویژگی های مهم این استراتژی ها که موجب تمایز آنها با الگوریتم های دیگر شده می پردازیم:

۳-۱-۱ ویژگی ها

۲-۱-۱ مقیاس پذیری و موازی سازی

از مزایای قابل توجه این کلاس از استراتژی ها، مقیاس پذیر بودن آنها در استفاده از تعداد زیادی کارگران به صورت موازی است. زیرا هر کارگر قادر است بدون وابستگی به دیگر کارگران کار خود را انجام داده و در نهایت خروجی مطلوب را باز گرداند؛ زیرا اگر آشفتگی دانه های تصادفی[11] را از پیش بین کارگران همگان سازی کرده باشیم تا تغییر آشفتگی به صورت موازی و باهم باشد، هر کارگر تنها با فرستادن و دریافت یک عدد می تواند پارامتر های خود را بروز رسانی کند که باعث از بین رفتن بیشتر وابستگی بین کارگران می شود. این ویژگی در تمایز با روش های گرادیان است که در آن کارگران باید به همواره با یکدیگر در ارتباط باشند زیرا خروجی کارگران به یکدیگر وابسته اند.

در شکل زیر، شبه کدی را برای پیاده سازی استراتژی های تکاملی موازی سازی شده را میبینیم که به استفاده از دانه های تصادفی مشترک استفاده میکنند:

[8] Evolution Strategies (ES)
[9] Covariance Matrix adaptation Evolution Strategy (CMA-ES)
[10] Genetic Algorithm (GA)
[11] Random Seeds

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

شکل ۲- شبه کد استراتژی های تکاملی موازی سازی شده

در عمل، نمونه گیری را به این صورت اجرا می کنیم که هر کارگر ابتدا جمعیت مرتب شده ای را با استفاده از نویز گاؤسی[۱۲] (توزیع نرمال) واحد بین تمام کارگران ساخته و سپس پارامتر ها را با اضافه کردن تصادفی نویز در هر تکرار، آشفته می کنیم. اگرچه این بدان معنی است که اغتشاشات در سراسر تکرارها کاملاً مستقل از یکدیگر نیستند، مشکل خاصی در عملکرد الگوریتم بوجود نمی آید. (آشفتگی ها لزوما مستقل از یکدیگر نیستند زیرا می توان دو کارگری داشت که آشفتگی یکسانی از فضای حالت داشته باشند اما این احتمال در فضای نمونه به اندازه بزرگ به صفر میل می کند)

۲-۲-۲ پارامتر سازی شبکه

همانطور که در بخش قبلی اشاره شد، استراتژی های تکاملی با استفاده از اغتشاش پارامتر ها عمل می کنند. زمانی این استراتژی عملکرد قابل قبولی خواهد داشت که بخشی از جمعیت نسبت به بقیه عملکرد بهتری از خود نشان دهند؛ پس حیاتی است که توزیع نرمال به صورت تصادفی، پارامتر ها و در نتیجه جمعیت بهتری برای نسل های آینده تولید کند.

نرمال سازی دسته مجازی[۱۳] یک تکنیک استفاده شده برای آموزش در این استراتژی است که همانند نرمال سازی عادی دسته ای عمل میکند. در این تکنیک یک دسته برای شروع آموزش انتخاب شده و در طی یک آموزش ثابت می ماند. این عمل سبب می شود که مدل در مراحل اولیه آموزش نسبت به تغییرات کوچک و ناگهانی واکنش بیشتری بدهد و به همین علت نیاز است حجم جمعیت در هر دسته را به اندازه کافی بزرگ باشد تا مانع تاثیر زیاد تغییرات کوچک اطلاعات بر عملکرد کلی مدل شویم. این تکنیک برای مدل های پیچیده و گسترده هزینه زیادی به همراه دارد، اما در مدل های نسبتا ساده این هزینه قابل چشم پوشی است و باعث بهبود فرایند یادگیری مدل می شود.

الگوریتم ژنتیک که کمی جلو تر به بررسی دقیق تر آن می پردازیم، به طور سنتی هر عضو جمعیت را به صورت بردار θ ذخیره می کند که باعث می شود از نظر حافظه بهینه نباشد. از طرفی مقیاس پذیری پایینی دارد و اندازه شبکه عصبی را به صورت

[12] Gaussian Distribution (Normal Distribution)
[13] Virtual Batch Normalization

چشمگیری افزایش میدهد. برای حل این مشکل از روش جدیدی استفاده می کنیم که در آن هر بردار پارامتر را با استفاده از دانه اولیه[۱۴] و یک لیست از دانه های تصادفی که دنباله ای از جهش هایی میسازند که منجر به ساخته شدن هر θ می شود.

$$\theta^n = F(\theta^{n-1}, T_n) = \theta^{n-1} + \sigma\varepsilon(T_n)$$

شکل ۳- فرمول رمز گزاری جدید برای اعضای جمعیت

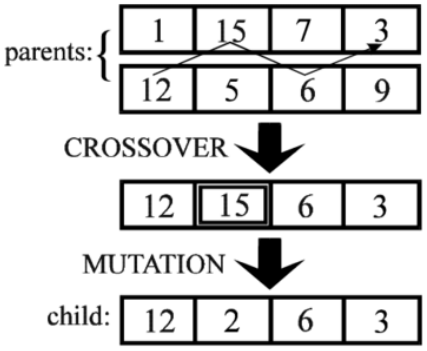
در این فرمول T_n مجموعه ای از دانه های هر جهش بوده و $\varepsilon(T_n) = N(0,1)$ یک تولید کننده اعداد تصادفی با توزیع گاؤسی است. با استفاده از این تکنیک میتوان شبکه عصبی با بیش از ۴ میلیون پارامتر را در تنها چندین هزار بایت ذخیره کرد. برای افزایش اهمیت این موضوع می توان به این نکته اشاره کرد که آخرین مدل GPT-3 دارای ۱۷۵ میلیارد پارامتر است که حدوداً ۴۵ ترا بایت حجم میگیرد.

۲-۲ روش ها

۲-۲-۲ الگوریتم ژنتیک[۱۵]

هر چرخه از این الگوریتم ابتدا با تشکیل جمعیت شروع شده و پس از یک جهش تصادفی که برای طبیعی نگه داشتن روند تکامل حیاتی است، مقدار تابع هدف (*fitness*) که عملا امتیاز اعضای جدید است، ارزیابی میشود. در نهایت بخشی از جمعیت که بالاترین امتیاز را دارند ترکیب شده تا جمعیت جدیدی ساخته شود. این روند تا جایی ادامه دارد که به طور کامل جمعیت تکامل یافته و به هدف نهایی خود برسند. حال با ثابت نگه داشتن روند کلی و تغییر دادن کارکرد اجزا آن میتوان نمایش های مختلفی بدست آورد.

شکل ۴ مثال ساده ای از روند این چرخه را برای دو والد انتخابی از چرخه پیشین نشان میدهد. هر عضو این جمعیت ژن خود را دارد که توسط تابع *crossover* ترکیب شده و در نهایت تعدادی ژن در مرحله *mutation* به تصادف جهش پیدا می کنند. جهش عضو مهمی از این چرخه است زیرا از به دام افتادن الگوریتم در نقاط بحرانی محلی جلوگیری میکند.



شکل ۴- فرایند تکرار شونده تولید فرزند توسط ۲ والد در الگوریتم ژنتیک

در ابتدا با استفاده از یک الگوریتم ژنتیک بسیار ساده آزمایشی را پیش برده تا نحوه عملکرد یک الگوریتم ژنتیک بدون گرادیان را مورد مطالعه قرار دهیم. در این الگوریتم برای تابع *crossover* از روش *truncation-selection* استفاده کردیم که در آن درصدی از جمعیت با بالاترین امتیاز بعد از اعمال جهش تصادفی با الگوی نویز گاؤسی برای تولید فرزند انتخاب میشوند.

[14] Initialization Seed
[15] Genetic Algorithm

برای درک عمیق تر دلیل انقلابی بودن این الگوریتم در زمان خود، یک مسئله بهینه سازی مطرح میکنیم: تصور کنید در این مسئله ۶ پارامتر داریم که هر یک از آنها می توانند ۶ حالت مختلف داشته باشند. اگر فرض کنیم ارزیابی هر ترکیب ۶تایی پارامتر ها و محاسبه امتیاز این ترکیب یک دقیقه زمان نیاز داشته باشد، با روش ***brute-force*** به $۶^۶$ دقیقه (معادل ۳۲ روز) زمان نیاز داریم؛ اما اگر روشی داشتیم که در هر نسل فقط ۶۰ عضو برتر را آزمایش می کرد و ۶ نسل داشتیم، به ۶۰۰ دقیقه (معادل ۱۰ ساعت) نیاز خواهیم داشت.

درست است که این یک تخمین تقریبی است و همیشه ۶۰ عضو برتر انتخاب نمی شوند، ولی با این وجود می توانید از هزینه های سربار ای که می توان از آنها جلوگیری کرد شهود خوبی پیدا کنید. دقت این الگوریتم به تابع هدف، تعداد نسل و پارامتر های دیگر بستگی دارد اما با استناد به نتایج به دست آمده می توان آن را یک الگوریتم اکتشافی موفق بنامیم.

بنابراین الگوریتم به گونه ای طراحی شده است که برای یک تابع هدف از پیش تعریف شده و تعداد نسل هایی که باید تکرار شوند اجرا شود و جمعیت هر نسل نیز مشخص شده است. بنابراین اعضای با امتیاز کمتر در هر نسل از بین می روند، اما آیا این تنها روشی است که می توانیم با استفاده از آن در زمان کمتری به بهترین راه حل های ممکن برسیم؟ آیا لزومی دارد اعضای ناکارآمد جمعیت از بین بروند؟ آیا دخالت انسان در تنظیم توابع هدف می تواند بر عملکرد الگوریتم ها تاثیر منفی بگذارد؟ در ادامه الگوریتم جایگزین دیگری را برسی میکنیم که بخشی از مشکلات مطرح شده را حل می کند:

۲-۲-۲ جست و جو تازگی[16]

هدف نهایی از هوشمند سازی الگوریتم ها آن است که پارامتر هایی مانند آنچه در الگوریتم ژنتیک به صورت دستی تنظیم می کردیم نیز توسط خود الگوریتم و با توجه به شرایط تنظیم شوند یا به عبارتی پویا باشند. الگوریتم جست و جو تازگی سعی بر حل این مشکل دارد به گونه ای که این الگوریتم به پیشرفتی که توسط اهداف یا عملکرد تعریف شده است پاداش نمی دهد، بلکه به متفاوت بودن پاداش می دهد. به عبارتی تکامل آن بدون هدف والا است.

هدف از طراحی این الگوریتم آن است که رشد پیچیدگی در تکامل طبیعی، محصول جانبی بهینه سازی تابع هدف نیست، بلکه نتیجه تمایل بی پایان آن برای کشف راه های مختلف برای رویارویی با چالش های زندگی است. این ویژگی منجر به کشف دائمی تازگی و افزایش پیچیدگی مدل می شود.

برای بهتر نشان دادن این موضوع و مقایسه دو الگوریتم مطرح شده – الگوریتم ژنتیک و الگوریتم جست و جو تازگی – می توان آزمایش شبیه سازی حرکت دوپا را مثال زد که در آن تابع هدف الگوریتم ژنتیک ممکن است خم شدن دوپا را در نسل اول به عنوان یک حرکت نامطلوب از بین ببرد، اما الگوریتم جست و جو تازگی به این تفاوت و خلاقیت در هر نسل بعدی تا زمانی که پایداری با مقدار خمش مناسب حاصل شود، پاداش خواهد داد.

[16] Novelty Search

فصل سوم

تکامل عصبی عمیق

۳- تکامل عصبی عمیق[۱۷]

استراتژی تکامل عصبی عمیق راه کار جایگزین برای آموزش، بهبود و تکامل شبکه های عصبی است که اولین و معروف ترین الگوریتم آن ***NEAT***، این موضوع را به عنوان یک رویکرد قابل اجرا نشان داد.

این الگوریتم که فقط از شبکه نود[۱۸] های متراکم تشکیل می شود، قابلیت آن را می دهد که بتوانیم از الگوریتم های مختلف مانند الگوریتم ژنتیک و الگوریتم جست و جو تازگی و همچنین از مفاهیم مختلف مانند تکامل و یا ***Gradient Decent*** و ***Backpropagation*** استفاده کنیم و یا حتی از ترکیب همه این مفاهیم. مقالات اخیر حتی راه هایی را برای استفاده از الگوریتم های ***NEAT*** و ***NEAT-Like*** برای ساختار شبکه عصبی تکامل یافته و سپس استفاده از ***Gradient Decent*** و ***Backpropagation*** برای بهینه سازی این شبکه ها برجسته کرده اند، منطقه ای که در آینده نزدیک بی اندازه مرتبط و مهم خواهد شد.

این الگوریتم راه حل مناسبی برای مشکلات زیادی که پیش از آن وجود داشت ارائه داد که به مهم ترین آنها می پردازیم:

۳-۱ رمز گزاری

در زیست شناسی ما یک ژنوتیپ [۱۹]و یک فنوتیپ [۲۰]داریم. ژنوتیپ بازنمایی ژنتیکی یک موجود و فنوتیپ نمایش فیزیکی واقعی موجود است. استراتژی ها و الگوریتم های تکاملی همیشه آینه زیست شناسی هستند و تکامل عصبی هم از این نظر تفاوتی ندارد. برای رمز گزاری ژن های طراحی شده مسئله دو روش مستقیم و غیر مستقیم داریم:

۳-۱-۱ روش مستقیم

یک رمز گذاری مستقیم به صراحت همه چیز را در مورد یک فرد مشخص می کند. اگر یک شبکه عصبی داشته باشیم، به این معنی است که هر ژن مستقیماً به گره، اتصال یا ویژگی شبکه مرتبط می شود. این می تواند رمز گذاری ۱ و ۰ باشد، رمز گذاری گراف (به هم پیوستن گره های مختلف با اتصالات وزنی) و یا حتی روشی پیچیده تر باشد. نکته آن است که همیشه ارتباط مستقیمی بین ژنوتیپ و فنوتیپ وجود خواهد داشت که بسیار واضح و خواندنی است.

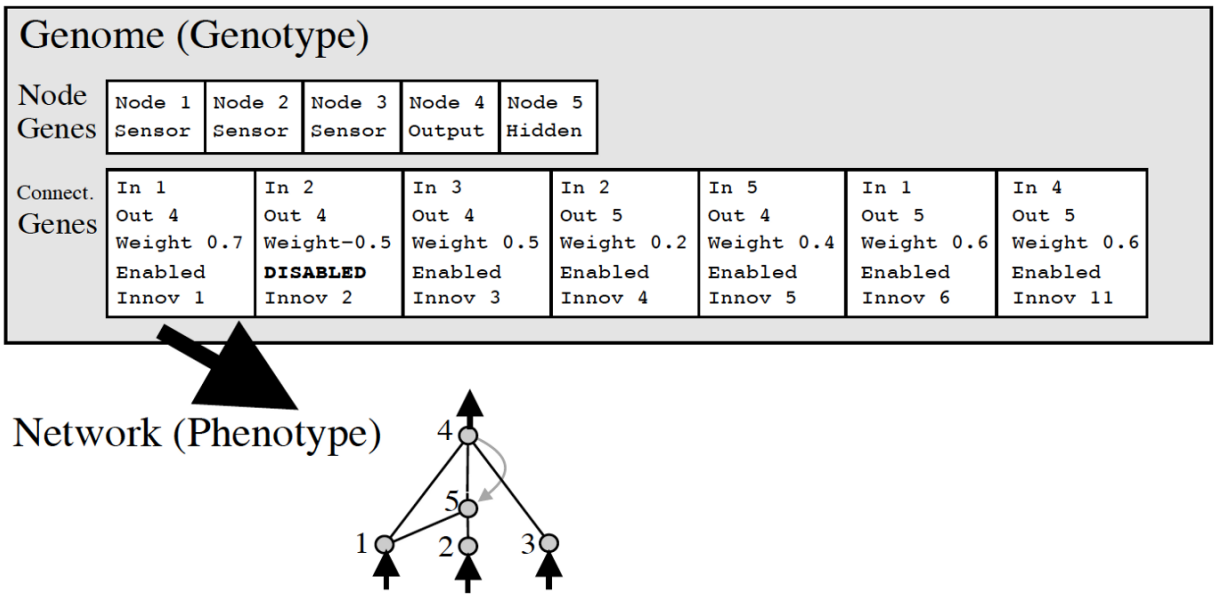
۳-۱-۲ روش غیر مستقیم

رمز گذاری غیر مستقیم دقیقا برعکس روش مستقیم است. به جای آنکه مستقیماً مشخص شود یک ساختار چگونه تعریف شود، رمز گذاری های غیرمستقیم تمایل دارند قوانین یا پارامترهای فرآیندها را برای ایجاد یک ژن مشخص کنند. در نتیجه، رمز گذاری های غیر مستقیم بسیار فشرده تر هستند. از طرف دیگر، تنظیم قوانین برای یک رمز گذاری غیرمستقیم می تواند منجر به یک سو گیری شدید در فضای جستجو شود. بنابراین، ایجاد یک رمز گذاری غیرمستقیم بدون دانش اساسی در مورد نحوه استفاده از رمز گذاری بسیار دشوارتر است.

به دلیل سادگی و طبیعی تر شدن شرایط مسئله، الگوریتم ***NEAT*** از روش مستقیم استفاده می کند تا در عین پیچیدگی بتوان درک شفافی از آن داشت. در شکل زیر ساختار ژنوتیپ ها و فنوتیپ ها مشاهده می شود:

---

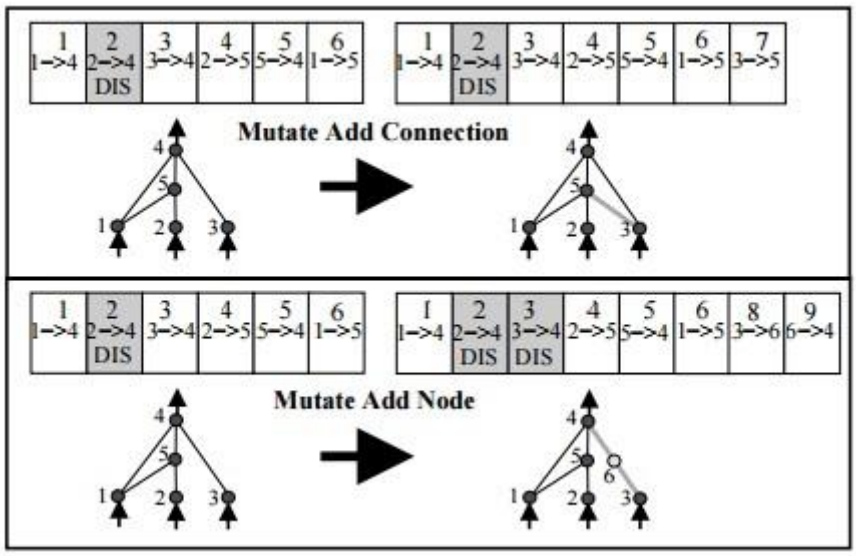
[17] Deep Neuroevolution
[18] Node
[19] Genotype
[20] Phenotype



شکل ۵- ساختار ژنوتیپ ها و فنوتیپ ها

۴-۳ جهش

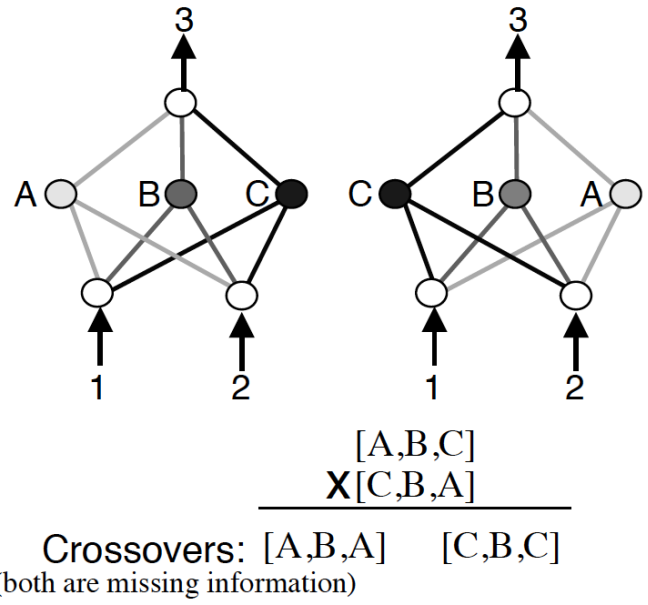
در این الگوریتم، جهش می تواند اتصالات موجود را تغییر دهد یا ساختار جدیدی را به شبکه اضافه و حتی تغییر دهد. در صورتی که یک اتصال جدید بین گره شروع و پایان یا به عبارتی اول یا آخر رشته ژن ها اضافه شود، به طور تصادفی وزنی به آن اختصاص داده می شود. همچنین اگر یک گره جدید در جایی وسط رشته اضافه شود، بین دو گره که قبلا متصل شده اند قرار می گیرد و اتصال قبلی غیرفعال می شود (اگرچه هنوز در آن ژن وجود خارجی دارد). گره شروع قبلی با وزن اتصال قدیمی به گره جدید و گره جدید با وزن ۱ به گره انتهایی قبلی مرتبط می شود.



شکل ۶- روش های مختلف برای جهش در نقاط مختلف رشته ژن ها

۴-۳ معاونت های رقابتی[۲۱]

از مشکلات الگوریتم ژنتیک تعریف تابع *crossover* است به گونه ای که ایده آن است که نه تنها ترکیب کورکورانه لزوما منجر به پیشرفت شبکه نشده، بلکه می تواند به مرور زمان شبکه ای غیر قابل اعتماد و غیر کاربردی تشکیل دهد. همانگونه که در شکل زیر قابل مشاهده است، اگر دو شبکه به گره های مرکزی وابسته باشند که هر دو از شبکه دوباره ترکیب شوند، مشکل بزرگی داریم که منجر به از دست رفتن اطلاعات می شود و مقاله های الگوریتم *NEAT* آنرا "معاونت های رقابتی" می نامند.

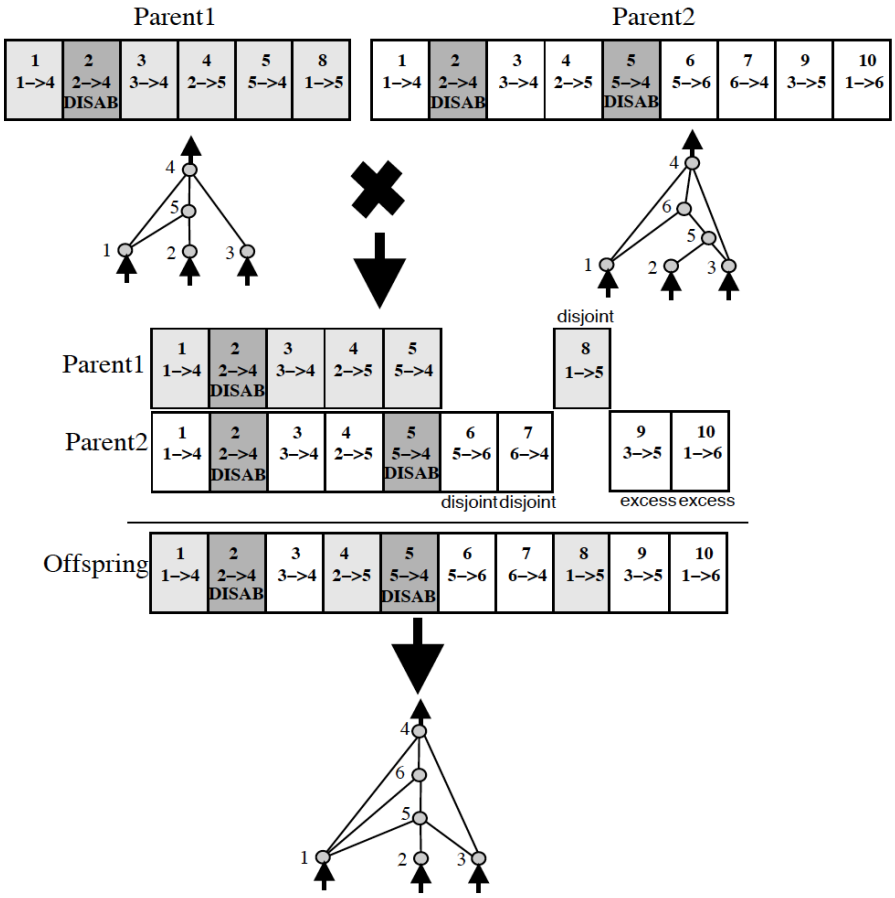


شکل ۷- از دست رفتن اطلاعات در دو شبکه که به گره مرکزی وابسته هستند

از طرفی، ژن ها می توانند اندازه های مختلفی داشته باشند. چگونه ژن هایی را که از نظر اندازه با هم سازگار نیستند را تراز کنیم؟ در زیست شناسی، از طریق ایده ای به نام همسانی به این امر رسیدگی می شود. همسانی، همراستا سازی کروموزومها بر اساس ژن های مطابق برای یک صفت خاص است. هنگامی که این اتفاق می افتد، تابع *crossover* می تواند با احتمال خطای بسیار کمتری نسبت به زمانی که کروموزومها کورکورانه با هم مخلوط شده باشند، اتفاق بیفتد.

الگوریتم *NEAT* با استفاده از نشانه های تاریخی با این مشکل مقابله می کند. با علامت گذاری تکامل های جدید با یک عدد تاریخی، در هنگام ترکیب دو والد، می توان این کار را با شانس بسیار کمتری برای ایجاد فرزند های غیر عملیاتی انجام داد. هر ژن را می توان همراستا و (به طور بالقوه) ترکیب کرد. هر بار که یک گره جدید یا نوع جدیدی از اتصال رخ می دهد، یک علامت گذاری تاریخی اختصاص داده می شود که اجازه می دهد در هنگام پرورش دو والد همسویی آسان تری داشته باشیم. شکل زیر توصیف بهتری از این موضوع دارد:

[21] Competing Conventions



شکل ۸- استفاده از نشانه های تاریخی برای همسو سازی آسان تر والد ها

۴-۳ گونه سازی[۲۲]

ایده جالبی در توسعه الگوریتم *NEAT* مطرح شد که بیشتر تحولات جدید، موارد خوبی نیستند. در واقع، افزودن یک فرزند جدید - که می تواند گره یا اتصال جدیدی باشد - قبل از بهینه سازی ژن ها، اغلب منجر به عملکرد ضعیف تر فرزند می شود. این امر ساختارهای جدید را در معرض بحران از بین رفتن قرار می دهد و نیازمند روشی هستیم تا از ساختار های تازه شکل گرفته (و در حال شکل گرفتن) محافظت کنیم تا به آنها اجازه دهیم قبل از حذف کامل آنها از جمعیت، بهینه سازی شوند. *NEAT* مفهوم گونه سازی را پیشنهاد می کند.

گونه سازی به سادگی جمعیت را بر اساس شباهت ژن ها و اتصالات به چندین گونه تقسیم می کند. عملکردی برای تصمیم گیری در مورد چگونگی گونه سازی در مقاله های معتبر ارائه شده است، اما بخش مهمی که باید به آن توجه کرد آن است که افراد در یک جمعیت فقط باید با افراد دیگر در همان گونه مشابه رقابت کنند. این مفهوم ایجاد و بهینه سازی ساختار جدید را با خطر کمتری برای حذف شدن پیش از توسعه و بهینه سازی فراهم می کند.

[22] Speciation

فصل چهارم

استفاده از الگوریتم *NEAT* در حل مسائل

۴- استفاده از الگوریتم *NEAT* در حل مسائل

در فصل دوم با استراتژی های تکاملی و سپس با الگوریتم های موجود نظیر الگوریتم ژنتیک و جست و جو تازگی آشنا شدیم. در فصل سوم با الگوریتم *NEAT* آشنا و ویژگی های انقلابی که آنرا متمایز می کند آشنا شدیم. حال می توان با استفاده از آموخته های این دو فصل مسئله های مختلفی را حل کنیم.

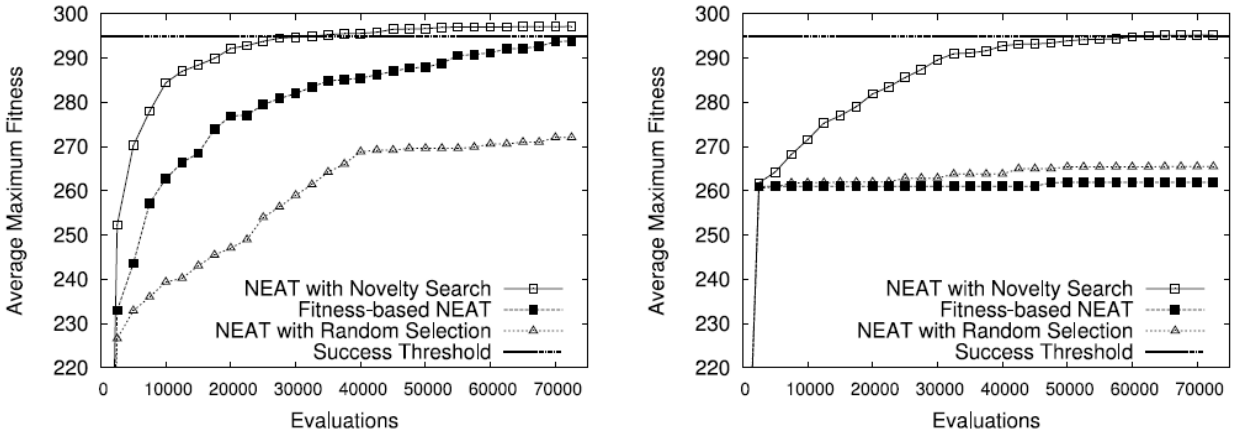
آزمایش های گوناگونی با استفاده از ترکیب های مختلف الگوریتم NEAT در مسائل واقعی برای مقایسه بهترین ساختار برای الگوریتم *NEAT* و انتخاب بهترین ساختار طراحی و انجام شده است. در آنها بخصوص از الگوریتم های زیر استفاده شده است:

- الگوریتم *NEAT* با استفاده از جست و جو تازگی
- الگوریتم *NEAT* با استفاده از الگوریتم ژنتیک
- الگوریتم *NEAT* با استفاده از انتخاب تصادفی

که در ادامه به چندین آزمایش بارز انجام شده توسط این الگوریتم ها می پردازیم:

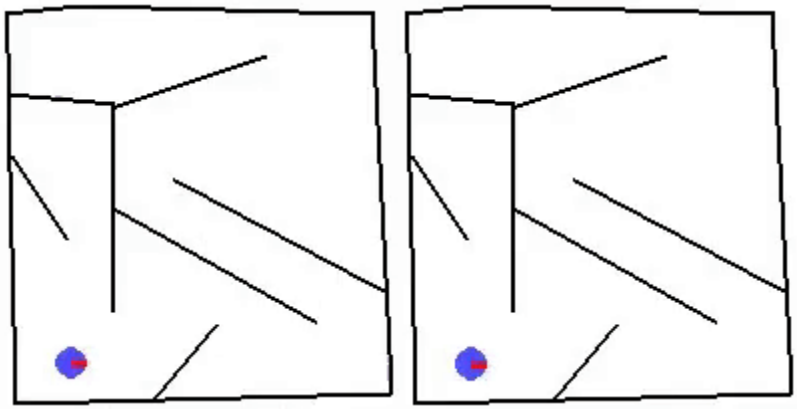
۴-۱ شبکه مارپیچ فریبنده

این آزمایش بر روی ترکیب های مختلف الگوریتم *NEAT* در دو نقشه مختلف انجام شد:



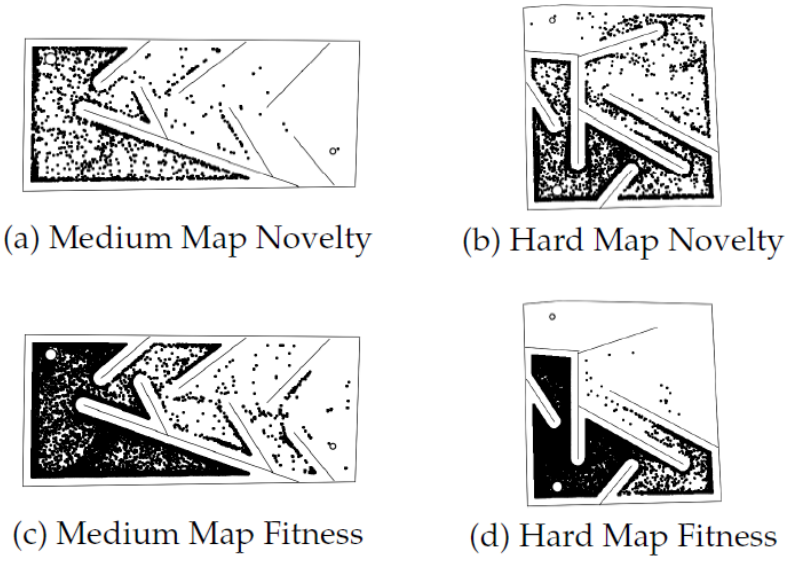
شکل ۹- مقایسه الگوریتم ها در حل مسئله مارپیچ

مطابق انتظار برای مسائلی از این قبیل استفاده از مفهوم جست و جو تازگی در الگوریتم *NEAT* تاثیر بسزایی در یافتن مسیر دارد که باعث شده در کمترین زمان (تعداد نسل) مسئله حل شود و سخت شدن نقشه آزمایش تاثیر زیادی بر روی عملکرد این الگوریتم ندارد. اما برای الگوریتم های دیگر بزرگ شدن نقشه به طور قابل توجهی عملکرد الگوریتم های دیگر را کاهش می دهد به طوری که دیگر قادر به رسیدن به مقصد نیستند.



شکل ۱۰- مقایسه ین دو الگوریتم در شبیه سازی واقعی. در سمت راست الگوریتم با تابع هدف و در سمت چپ با جستوجو تازگی قرار دارند

این آزمایش ۴۰ بار صورت گرفت و الگوریتم *NEAT* با جست و جو تازگی در ۳۹ بار توانست به مقصد برسد، در صورتی که الگوریتم های دیگر کمتر از ۱۰٪ مواقع موفق به حل مسئله و یافتن مسیر شدند. شبیه سازی مسیر های طی شده در الگوریتم ها را مشاهده می کنید:



شکل ۱۱- شکل های مارپیچ بعد از ۲۵۰ هزار نسل

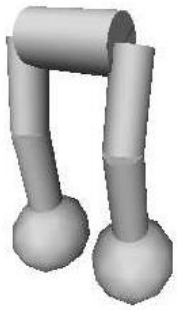
۲-۴ آزمایش دوپا

در حالی که دامنه مسئله مارپیچ نشان می دهد که جستجو برای تازگی با آزمودن مسیر های جدید می تواند سریع تر به مقصد برسد، یک سوال منطقی این است که آیا جستجوی جدید صرفا برای دامنه مسئله پیشین مناسب است و یا آیا می توان آن را با موفقیت در حوزه های دشوارتر آزمایش کرد؟

بنابراین نیاز به آزمایش جستجوی جدید بر روی یک مشکل شناخته شده با دشواری قابل توجه نسبت به مسئله مارپیچ داریم. شهود پشت چنین تلاشی این است که روش جستجوی جدید ممکن است در چنین حوزه ای نیز موفق باشد زیرا از آنجایی که

جستجوی جدید هدف را نادیده گرفته و به ساختار های خلاقانه بهای بیشتری می دهد، ممکن است بتواند در حوزه های دشوار به سادگی با جستجوی تازگی موفق شود.

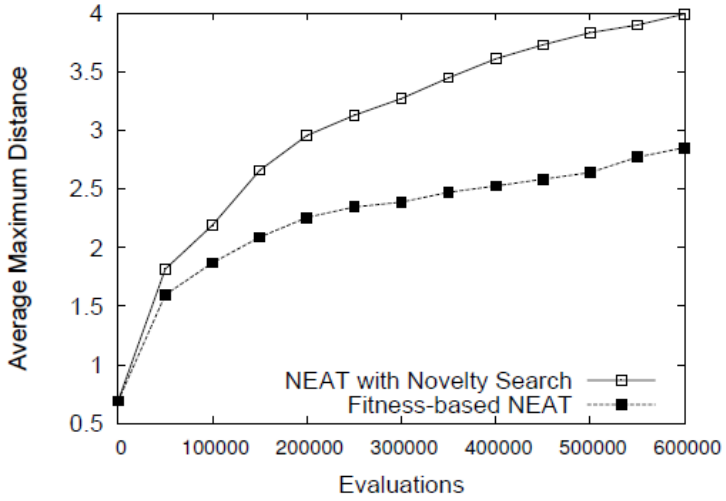
آزمایش دوپا به صورت زیر عمل می کند. یک ربات دوپا برای مدت زمان ثابتی (که در این شبیه سازی ۱۵ ثانیه در نظر گرفته شده) توسط یک شبکه عصبی کنترل می شود و سپس موفقیت شبکه برای ایستاده نگه داشتن دوپا ارزیابی می شود. از طرفی هدف آزمایش آن است که ربات بیشترین فاصله ممکن را از محل شروع بدون افتادن حرکت کند.



شکل ۱۲- یک دوپا که توسط یک شبکه عصبی قادر به ایستادن است

شبکه عصبی که دوپا را کنترل می کند تنها دو ورودی دارد که حسگر های تماسی هستند که به هر پا سیگنال می دهند که آیا با زمین تماس دارد یا خیر. پراکندگی ورودی مشکل را دشوارتر می کند زیرا شبکه هیچ اطلاعاتی در مورد جهت ربات یا زوایای جاری اتصالات آن ندارد.

بعد از ۵۰ تست متوالی شبکه دوپا با استفاده از مفهوم جست و جو تازگی به طور میانگین در هر ۱۵ ثانیه حدودا ۱۳.۷ متر حرکت می کند؛ این در حالی است با استفاده از تابع هدف حدوداً ۶.۷ متر حرکت داشتیم. نتایج در شکل زیر قابل مشاهده است:



شکل ۱۳- مقایسه عملکرد هر دو الگوریتم در ۵۰ تست متوالی

برسی الگوریتم تکامل عصبی عمیق در آموزش شبکه های عصبی

فصل پنجم

نتیجه گیری و پیشنهاد ها

۵- نتیجه گیری و پیشنهاد ها

در بخش پایانی گزارش، جمع بندی و مروری بر سیر مطالب عنوان شده در گزارش خواهیم داشت. همچنین نتایج حاصل را بیان کرده و پیشنهاد هایی در راستای ادامه کار ارائه می دهیم.

۵-۱ نتیجه گیری

در این گزارش با هدف تمرکز بر روی ارائه روش نوینی برای آموزش شبکه های عصبی که بیشتر از روش های سنتی بر روی طبیعی بودن تکامل تاکید داشته، در فصل اول سعی کردیم مفاهیم استراتژی های تکاملی را بازگو کنیم. سپس با مطالعه چندین ویژگی بارز از جمله موازی سازی، مقیاس پذیری و پارامتر سازی بر انعطاف پذیری این استراتژی ها تاکید کردیم. در نهایت با معرفی دو الگوریتم ژنتیک و جست و جو تازگی به جمع بندی خوبی درباره این استراتژی حل مسئله رسیدیم.

در فصل دوم با معرفی الگوریتم ***NEAT*** به عنوان الگوریتم جایگزین روش سنتی ***SGD*** برای آموزش شبکه های عصبی و بیان ویژگی های مهم آن از جمله نحوه رمز گزاری آن، نحوه ایجاد جهش و اعمال معاونت های رقابتی و در نهایت گونه سازی سعی بر گسترش این موضوع، بیان قدرت و انعطاف پذیری این الگوریتم داشتیم.

در فصل سوم با نشان دادن چندین آزمایش و شبیه سازی های صورت گرفته در این زمینه مانند مسئله مارپیچ فریبنده و شبیه سازی دوپا به اهمیت استفاده از الگوریتم های جست و جو تازگی و الگوریتم هایی که انعطاف پذیری بالایی برای یادگیری روند های مختلف داشته پرداخته و سعی بر کم تر کردن نقش انسان در تنظیم کردن پارامتر ها و پر رنگ کردن نقش خود مدل برای انجام این کار داشتیم.

احتمالا مهم ترین درسی که از این گزارش و برسی شیوه های مختلف حل مسئله می گیریم نقش خلاقیت در خلق مدل های هوشمند تر است و اجبار به کمتر کردن نقش انسان در کنترل این مدل ها است. به گونه ای که مدل ها قابلیت وقف دادن خود با شرایط و یادگیری حالت های مختلف را داشته باشند.

۵-۲ پیشنهاد ها

روش های متعددی برای آموزش شبکه های عصبی وجود دارند که نه تنها سریع تر و بهینه تر هستند بلکه انعطاف پذیری بیشتری دارند. ما در این مقاله چندین الگوریتم را مورد مطالعه قرار داده و در نهایت ترکیب کردیم اما نباید فراموش کرد که این حوزه پژوهش بسیار فعال و امیدوار کننده است. از طرفی همچنان چالش های متعددی وجود دارند که الگوریتم های مطرح شده در این گزارش لزوما کارآمد ترین روش برای مواجهه با آنها نیستند.

یکی از مهم ترین دستاورد های چند سال اخیر، استفاده از روش ***Ensemble*** برای ترکیب مدل ها و خروجی آنها است. در این روش که زیر شاخه های زیادی دارد مدل ها (و گاه اطلاعات) به شیوه های گوناگونی با یکدیگر ترکیب شده تا خروجی بهتری دهد. طبق مقالات معتبر، استفاده از این روش به تنهایی میتواند دقت مدل های یادگیری ماشین را بین ۵-۱۰ درصد افزایش دهد. استفاده از این روش برای آموزش شبکه های عصبی عمیق حوزه ای نوپا بوده و متخصصین همچنان در حال جمع آموری اطلاعات و آزمایش مدل های مختلف هستند.

به عنوان چشم انداز این گزارش می توان نگاهی به استفاده از شیوه تازه مطرح شده با الگوریتم های اصلی گزارش داشت و امید است که بتوان با کمک آن مدل ها را بهبود داد.

مراجع و منابع

[1] F.P Such, V. Madhavan, E. Conti, j. Lehman, K. O. Stanley and J. Clune, “Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” *Uber AI Labs*., vol. 3, pp. 2-5, Apr. 2018.

[2] T. Salimans, J. Ho, Xi Chen, S. Sidor and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” OpenAI., vol. 2, pp. 2-6, Sep. 2017.

[3] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *The MIT Press*., vol. 10, pp 4-13, June. 2002.

[4] J. Lehman and K. O. Stanley, “Abandoning Objectives: Evolution Through the Search for Novelty Alone,” *PubMed*., vol 2, pp. 9-32, June 2011.

[5] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, “Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients, “Uber *AI Labs*. Vol. 3, pp. 1-10, May 2018.

[6] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. O. Stanley, J. Clune, “Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents,” *Uber AI Labs*. Vol 3, pp. 2-10, Oct 2018.

[7] Medium (2022, May 1). *On Genetic Algorithms: Why Novelty Search is important* [On-line]. Available: https://medium.com/@nniranjhana/on-genetic-algorithms-why-novelty-search-is-important-6d2879d3ed81