



طراحی شبکه بهینه‌ساز خودکار الهام گرفته از

الگوریتم بهینه‌سازی BFGS با حافظه محدود

محمد اعتصام، اشکان صادقی لطف‌آبادی و سید کمال‌الدین غیائی شیرازی*

^۱گروه مهندسی کامپیوتر، دانشگاه فردوسی مشهد، مشهد، ایران

چکیده:

امروزه برخلاف توسعه مدل‌های یادگیری ماشین برای استخراج ویژگی‌ها به صورت خودکار، هنوز الگوریتم‌های بهینه‌سازی به صورت دستی طراحی می‌شوند. یکی از اهداف فرایادگیری (meta-learning)، خودکار کردن فرایند بهینه‌سازی است. الگوریتم‌های بهینه‌سازی دستی مبتنی بر بردار گرادیان تنها براساس عملیات ضرب داخلی، ضرب اسکالر و جمع برداری بر روی بردارهای ورودی نوشته می‌شوند؛ بنابراین می‌توان گفت که این الگوریتم‌ها در فضای هیلبرت بعد مسئله بهینه‌سازی اجرا می‌شوند. ما نیز قصد داریم با ایده گرفتن از این مطلب، فضایی برای یادگیری ورودی‌ها ایجاد کنیم که مستقل از ابعاد ورودی باشد. بدین منظور با ایده گرفتن از الگوریتم BFGS با حافظه محدود (L-BFGS) و همچنین سلول LSTM یک ساختار جدید با نام Hilbert LSTM (HLSTM) معرفی می‌کنیم که فرایند یادگیری در آن مستقل از ابعاد ورودی انجام می‌شود. به عبارتی الگوریتم یادگیری در فضای هیلبرت مسئله بهینه‌سازی اجرا می‌شود. برای رسیدن به این هدف از لایه ضرایب خطی استفاده می‌کنیم که ترکیب خطی بردارهای ورودی را محاسبه می‌کند و ضرایب این ترکیب خطی، با کمک ضرب داخلی بردارهای ورودی به دست می‌آید. آزمایش‌های ما نشان می‌دهند که نتایج به دست آمده به وسیله بهینه‌ساز ارائه شده، به مراتب بهتر از نتایج الگوریتم‌های بهینه‌سازی دستی است.

واژگان کلیدی: Hilbert LSTM، L-BFGS، فرایادگیری، بهینه‌سازی خودکار

Designing L-BFGS inspired automatic optimizer network

Mohammad Etesam, Ashkan Sadeghi-Lotfabadi, Kamaledin Ghiasi-Shirazi*

¹Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

Abstract:

The optimization of deep neural networks based on mini-batches is an active area of research, and the improvements in this field have a great impact on the success of using deep neural networks in practical problems with large data. In the last decade, algorithms such as RMSProp, AdaGrad, and Adam have been devised for the optimization of neural networks on mini-batches, having a great impact on making the training of neural networks easier. The common feature of all these methods is that they are applied to each dimension of the gradient vector separately, and therefore the optimization of each parameter of the neural network is done independent of other parameters. Next, researchers tried to learn these algorithms automatically and devised methods for learning to optimize, which is a type of meta-learning. Optimization learning algorithms use an optimizer network to obtain the optimization direction at each step. Therefore, when these methods are used to optimize a neural network, we have an optimizee network whose parameters we want to learn and an optimizer network whose parameters are meta-learned.

* Corresponding author

* نویسنده عهده‌دار مکاتبات

سال ۱۴۰۳ شماره ۱ پیاپی ۵۹

• تاریخ ارسال مقاله: ۱۳۹۹/۲/۲۱ • تاریخ پذیرش: ۱۴۰۲/۱۱/۲۸ • تاریخ انتشار: ۱۴۰۳/۵/۱۰ • نوع مطالعه: بنیادی

The optimizer network receives the previous parameters and their gradients and suggests a new direction to optimize the optimizee network. Similar to the optimization algorithms based on mini-batches, all the methods devised so far for learning optimization also have this point in common that they apply the optimization method independently to each parameter. The fact that the gradient direction is not a suitable direction for optimization is also accepted in mathematical optimization algorithms, and usually, if there is no computational issue, the gradient direction is corrected by using the inverse of the Hessian matrix or its approximation by Newton or quasi-Newton methods. Therefore, we see that neural network optimization algorithms and non-linear mathematical optimization algorithms are common in not using the gradient direction.

However, in mathematical optimization methods, the proposed vector for optimization is obtained by vector operations on the previous points and gradient vectors, and an optimization algorithm never performs an operation on the elements of a vector independently. In a more detailed look, we can say that the mathematical optimization is performed in a Hilbert space, which is equal to the number of parameters of the optimization problem, and optimization direction is calculated only by using vector addition, scalar multiplication in a vector, internal multiplication between vectors and applying scalar functions on scalar values. These operations are exactly the operations that are performed in a Hilbert space.

Limiting operations on vector to vector addition, scalar multiplication in a vector, and internal multiplication of vectors is very important in reaching an optimization algorithm independent of the number of dimensions. For example, if the optimizer neural network has among its parameters a weight vector that is multiplied by the vectors of the optimization problem, then the optimizer becomes dependent on the dimensions of the problem and a dimension-independent optimization algorithm is no longer obtained. In this way, it is necessary to properly redesign the optimizer neural network components in such a way that their operations are exclusive to the allowed operations in the Hilbert space.

Here, we consider M. Andrychowicz et. al. (in: NeurIPS 2016. Learning to learn by gradient descent by gradient descent) method as the base method, where the optimizer is an LSTM network. In this paper we propose a version of the LSTM in which all computations are restricted to allowed operations in Hilbert space. We call this network Hilbert LSTM and we design our optimizer network based on it. In this way, we design a model for the optimizer network, which, like mathematical optimization methods, firstly considers the relationship between different dimensions of the space and secondly, it does not depend on the dimensions of the space. Unlike the previous methods that only used the parameters and gradients of the previous step, inspired by limited-memory BFGS, we use the parameters and gradients of m previous steps, where m refers to the limited memory size. Our results and tests show that the independence of the dimensions in the calculations in our optimizer makes the generalization power of our method to different dimensions more than other methods.

Keywords: Hilbert LSTM, LSTM, L-BFGS, meta-learning, automatic optimization

کلاسیک این اشکال را با تغییر اندازه گام گردیان به کمک ماتریس Hessian که شامل مشتقات جزئی مرتبه دوم است، حل می‌کنند. علاوه بر این راه‌های دیگری نظیر استفاده از ماتریس Gauss-Newton تعمیم‌یافته یا ماتریس اطلاعات Fisher نیز وجود دارد [۱].

اغلب طراحی روش‌های بهینه‌سازی و قواعد به‌هنگام‌سازی، با توجه به نوع مسأله و حوزه کاری انجام می‌شود. برای مثال در حوزه یادگیری عمیق توجه به نکاتی نظیر ابعاد بالا و غیر محدب‌بودن مسائل نقش مهمی در طراحی روش‌های بهینه‌سازی دارد. روش‌هایی مانند momentum [۲]، Rprop [۳]، Adagrad [۴]، RMSprop [۵] و ADAM [۶] با توجه به این نکات طراحی شدند [۷]. در مقابل برخی روش‌ها که روی تنگی تمرکز داشتند، رویه متفاوتی را در پیش گرفتند [۸، ۱]. طراحی الگوریتم‌های بهینه‌سازی با توجه به حوزه فعالیت، باعث می‌شود که این الگوریتم‌ها در خارج از آن حوزه عملکرد خوبی نداشته باشند. همچنین براساس نظریه ارائه‌شده "هیچ ناهاری مجانی نیست" در [۹] می‌توان گفت که در

۱- مقدمه

اغلب مسائل یادگیری ماشین را می‌توان به شکل بهینه‌سازی یک تابع هدف $f(\theta)$ روی دامنه $\theta \in \Theta$ تعریف کرد. در این صورت هدف، یافتن پارمتر بهینه است، به‌طوری‌که $f(\theta)$ کمینه شود:

$$\theta^* = \arg \min_{\theta \in \Theta} f(\theta) \quad (۱)$$

روش‌های بسیاری برای بهینه‌کردن این عبارت وجود دارد؛ اما روش استاندارد برای بهینه‌کردن یک تابع مشتق‌پذیر، استفاده از روش‌های مبتنی بر کاهش گرادیان است که با کمک قانون (۲) پارامترها را به‌هنگام می‌کنند.

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \quad (۲)$$

روش کاهش گرادیان فقط از گرادیان استفاده می‌کند و اطلاعات مرتبه دوم را نادیده می‌گیرد. روش‌های بهینه‌سازی

بهینه‌سازی ترکیبی، هیچ الگوریتمی لزوماً بهتر از استراتژی تصادفی عمل نمی‌کند.

هدف فرایادگیری ارائه روشی است تا مسأله طراحی الگوریتم بهینه‌سازی، به یک مسأله یادگیری ماشین تبدیل شود. در روش‌های کلاسیک، بسته به نوع مسأله، ویژگی‌های روش بهینه‌سازی به صورت دستی تعیین می‌شود؛ ولی در فرایادگیری هدف یادگیری ویژگی‌ها با کمک یادگیری ماشین و طراحی یک روش بهینه‌سازی با کمک آن است.

در یادگیری، وقتی به دنبال آموزش یک مدل هستیم، از مجموعه‌ای از داده‌ها استفاده می‌کنیم. در حوزه فرایادگیری این داده‌ها همان مثال‌هایی از مسائل مختلف است. این مثال‌ها با ایجاد قوانین و محدودیت‌هایی روی پارامترها منجر به آموزش مدل می‌شوند؛ سپس باید کارایی مدل در نقاطی بررسی شود که هنوز مشاهده نشده‌اند که این به مفهوم تعمیم^۱ است. در واقع تعمیم یعنی قدرت پیش‌بینی رفتار تابع در نقاط جدید. از آن‌جا که در فرایادگیری داده‌ها همان نمونه مسائل هستند، می‌توان گفت که تعمیم در این حوزه، به مفهوم انتقال یادگیری^۲ بین مسائل مختلف است. ما از قدرت شبکه‌های عمیق بازگشتی برای یادگیری اطلاعات و انتقال و تعمیم این دانش بین مسائل مختلف استفاده می‌کنیم.

رویکردهای مختلفی در حوزه فرایادگیری وجود دارد؛ اما یکی از مهم‌ترین آن‌ها استفاده از دو شبکه بهینه‌ساز و بهینه شونده است [۱]. طبق این روش، شبکه بهینه‌ساز، اندازه گام به‌هنگام‌سازی را برای پارامترهای بهینه‌شونده تعیین می‌کند. این رویکرد مبنای کار ما برای ارائه روشی جدید بود. در مورد این رویکرد و دیگر روش‌هایی که تاکنون ارائه شده در بخش بعد توضیح خواهیم داد.

ما نیز در روش خود از دو شبکه بهینه‌ساز و بهینه‌شونده استفاده می‌کنیم. برای طراحی شبکه بهینه‌ساز از الگوریتم L-BFGS [۱۰] ایده می‌گیریم تا قواعد یادگیری را استخراج کنیم. در طراحی ما یادگیری، مستقل از ابعاد ورودی انجام می‌شود و این امر از آنجا حاصل می‌شود که ما از فضای ضرب داخلی ورودی‌ها استفاده می‌کنیم. الگوریتم‌های بهینه‌سازی مبتنی بر بردار گرادیان در هر لحظه بردار گرادیان را دریافت می‌کنند و با نگهداری اطلاعات مکان‌ها و گرادیان‌های گذشته، تغییر مکان فعلی را به دست می‌آورند. ورودی و خروجی این الگوریتم‌ها، بردارهایی با ابعاد مسأله بهینه‌سازی است. همچنین این الگوریتم‌ها تنها مبتنی بر عملیات ضرب اسکالر، ضرب برداری و جمع برداری بر روی این بردارها هستند؛ بنابراین می‌توان گفت که این الگوریتم‌ها در فضای هیلبرت بعد مسأله بهینه‌سازی اجرا می‌شوند. ما نیز قصد داریم با ایده گرفتن از این مطلب، فضایی برای یادگیری ورودی‌ها ایجاد کنیم که مستقل از ابعاد ورودی باشد.

به‌طور مشخص این کار را با کمک ساختاری جدید به نام لایه ضرایب خطی انجام می‌دهیم؛ سپس از این ساختار در طراحی یک سلول جدید با نام HLSTM^۳ استفاده می‌کنیم. درواقع HLSTM حاصل قراردادن لایه ضرایب خطی در سلول حافظه کوتاه‌مدت بلند (LSTM^۴) است؛ بنابراین یادگیری HLSTM به گونه‌ای است که مستقل از ابعاد ورودی‌های آن است.

در ادامه مقاله ابتدا در بخش ۲ مروری بر کارهای پیشین خواهیم داشت؛ سپس در بخش ۳ روش ارائه شده خود را معرفی می‌کنیم. در بخش ۴ به ارزیابی روش ارائه شده و مقایسه آن با دیگر روش‌های بهینه‌سازی می‌پردازیم. در نهایت در بخش ۵ جمع‌بندی و نتیجه‌گیری مقاله را خواهیم داشت و کارهای قابل انجام را در آینده بررسی می‌کنیم.

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \quad (3)$$

۲- مروری بر کارهای پیشین

ایده فرایادگیری یا یادگرفتن یادگیری^۵ از گذشته [۱۱، ۱۲] تا به امروز [۱۳-۱۷] یکی از حوزه‌های مهم در یادگیری ماشین بوده است. یکی از بحث‌های مهم که در همین اواخر مطرح شده این است که می‌توان فرایادگیری را به عنوان یک بلوک ساختمانی مهم در هوش مصنوعی در نظر گرفت [۱۸]. یکی از انواع تعمیم، یادگیری چندوظیفه‌ای^۶ [۱۹] است. به‌طور کلی می‌توان گفت که در این روش، یادگیری در دو مقیاس زمانی مختلف انجام می‌شود: یادگیری سریع در درون هر وظیفه^۷ و یادگیری آهسته‌تر در بین وظایف مختلف. به عبارتی یادگیری دوم از نوع فرایادگیری است که سعی در انتقال دانش بین مسائل مختلف دارد.

رویکرد ارائه شده در [۱۲، ۲۰، ۲۱] متداول‌ترین نگرش نسبت به فرایادگیری است که معتقد است شبکه‌ها می‌توانند خودشان وزن‌شان را تغییر دهند و به‌هنگام کنند. چنین سیستمی به‌صورت ابتدا تا انتها مشتق‌پذیر است و این امر باعث می‌شود که شبکه و الگوریتم یادگیری به‌صورت پیوسته به‌وسیله کاهش گرادیان قابل آموزش باشند. علاوه‌براین، کارهایی برای تغییر روش جستجو از کاهش گرادیان به دیگر روش‌ها نیز انجام شده است [۱، ۲۲]. برای مثال در [۲۳] از یادگیری تقویتی برای تعیین طول گام استفاده شده است.

برخی مقالات نیز روش‌هایی ارائه کرده‌اند تا بدون پس‌انتشار خطا بتوان پارامترها را به‌هنگام کرد [۲۴، ۲۵]. همچنین نشان داده شد که شبکه‌های عصبی بازگشتی با وزن ثابت می‌توانند رفتار پویا داشته باشند، بدون این‌که نیاز باشد وزن‌شان را تغییر دهیم [۲۶، ۲۷]. در نهایت مقالات [۲۸، ۲۹] نشان دادند که خروجی پس‌انتشار خطای یک شبکه را می‌توان

³ Hilbert LSTM

⁴ Long Short-Term Memory

⁵ learning to learn

⁶ multi-task learning

⁷ task

¹ generalization

² transfer learning

به عنوان ورودی یک شبکه دیگر استفاده کرد و هر دو شبکه را به صورت پیوسته آموزش داد. این راه کار در مقاله [۱] نیز مورد استفاده قرار گرفته است و یک شبکه به عنوان بهینه ساز و دیگری به عنوان بهینه شونده معرفی شده است. در ادامه این روش را دقیق تر بررسی می کنیم.

۱-۱- استفاده از دو شبکه بهینه ساز و

بهینه شونده

یک رویکرد در فرایادگیری، استفاده از دو ساختار بهینه ساز و بهینه شونده است. به این صورت که بهینه ساز، قانونی برای به هنگام سازی وزن های بهینه شونده یاد می گیرد. در مقاله [۱] از این ساختار استفاده شده که بهینه ساز از نوع شبکه LSTM [۳۰] است. طبق این مقاله یک بهینه شونده با پارامتر θ و یک قانون بهینه سازی (g) تعریف می شود. این قانون به وسیله یک بهینه ساز با پارامتر ϕ تعریف می شود. در نتیجه قانون کلی برای به هنگام سازی پارامتر θ به شکل معادله (۳) به دست می آید.

برای مدل سازی g از یک شبکه عصبی بازگشتی از نوع LSTM استفاده می شود که همان شبکه بهینه ساز است. فرایند یادگیری و تعامل دو ساختار بهینه شونده و بهینه ساز در (شکل ۲-۴) آمده است. بهینه ساز طراحی شده نقش تعیین قانون g برای به هنگام سازی پارامترهای بهینه شونده را برعهده دارد. درواقع طول گام به هنگام سازی برای پارامترهای بهینه شونده، به وسیله بهینه ساز تعیین می شوند. بهینه شونده نیز خطا را به بهینه ساز بازمی گرداند. این فرایند به صورت تکرار شونده انجام می شود تا خطای نهایی خروجی بهینه شونده کاهش یابد. پارامترهای شبکه بهینه ساز نیز به وسیله کاهش گرادینان به هنگام می شوند.

در [۱] برای آموزش بهینه ساز از تابع هزینه (۴) استفاده می شود:

$$L(\phi) = E_f[f(\theta^*(f, \phi))] \quad (4)$$

که در اینجا E به مفهوم امید ریاضی است و پارامتر بهینه θ^* نیز تابعی از پارامتر ϕ است. قانون بهینه سازی پارامتر θ یا همان طول گام به هنگام سازی، طبق آنچه در مقدمه بیان کردیم با g_t نام گذاری شده است.

تابع هزینه تعریف شده در معادله (۴) فقط به مقدار نهایی پارامتر بستگی دارد؛ اما برای آموزش شبکه بهینه ساز بهتر است تابع هزینه به تمام مسیر بهینه سازی وابسته باشد؛ بنابراین تابع هزینه جدیدی تعریف می کنند:

$$L(\phi) = E_f \left[\sum_{t=1}^T \omega_t f(\theta_t) \right] \quad (5)$$

در این تابع هزینه ω_t وزنی اختیاری است و هر عدد بزرگتر از صفری می تواند باشد. درواقع طبق تئوری، فقط زمانی می توان نسبت به مدل ارزیابی داشت که مسیر تا انتها طی شده باشد. یعنی ω_t باید فقط به ازای $t = T$ یک باشد و در بقیه

لحظات صفر باشد. ولی طبق آزمایش های انجام شده در [۱] این کار منجر به این می شود که الگوریتم پس انتشار خطا به خوبی عمل نکند و به عنوان راه حل پیشنهاد شده است تا ω_t در تمام لحظات برابر یک باشد. پارامتر θ با قانون $\theta_{t+1} = \theta_t + g_t$ به روزرسانی می شود که g_t خروجی شبکه عصبی بازگشتی (m) است:

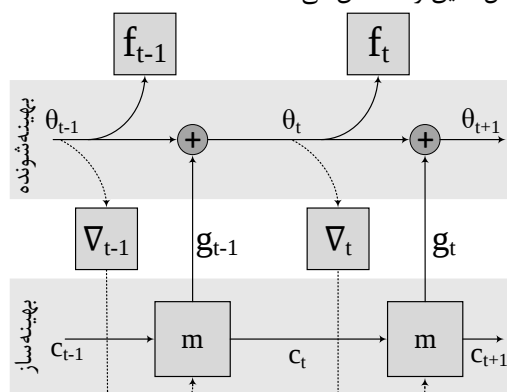
$$\begin{bmatrix} g_t \\ c_{t+1} \end{bmatrix} = m(\nabla_t, c_t, \phi) \quad (6)$$

همچنین برای بهینه سازی تابع هزینه (۶) روی پارامتر ϕ از الگوریتم کاهش گرادینان استفاده می کنند. جزییات بهینه ساز و نحوه تعامل آن با بهینه شونده، در (شکل ۱-۱) قابل مشاهده است. گرادینان فقط از خطوط توپر عبور می کند و از خطوط نقطه چین عبور نمی کند. نادیده گرفتن گرادینان در خطوط نقطه چین بیان گر این مطلب است که گرادینان بهینه شونده به پارامترهای بهینه ساز وابسته نیست. به عبارت دیگر یعنی $\partial \nabla_t / \partial \phi = 0$ و این فرض باعث می شود تا نیازی به محاسبه مشتق دوم f نباشد.

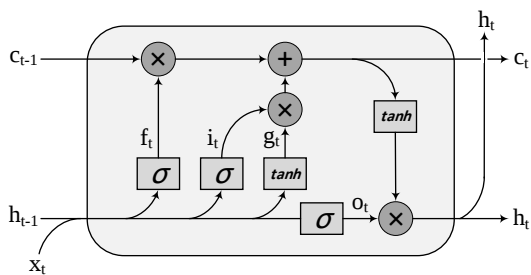
در بخش ۳ خواهیم دید که مدل ارائه شده به وسیله ما به صورت مختصات محور عمل نمی کند و ارتباطات ابعاد را نیز در نظر می گیرد. همچنین می تواند ورودی هایی با ابعاد متفاوت را بهینه کند.

۱-۲- حافظه کوتاه مدت بلند (LSTM)

LSTM [۳۰] نوعی شبکه عصبی بازگشتی است و همانطور که در بخش قبلی دیدیم برای طراحی بهینه ساز از آن استفاده شده است. (شکل ۴-۴) ساختار یک سلول LSTM را نشان می دهد. C حالت سلول را در هر لحظه نشان می دهد. h و x به ترتیب حالت مخفی و ورودی هستند. σ گیت با تابع فعالیت سیگموئید را نشان می دهد. $\tanh$ نشان دهنده ی تابع فعالیت تانژانت هایپربولیک است. f ، i و o به ترتیب خروجی گیت فراموشی، گیت ورودی و گیت خروجی است. پارامتر زمان نیز با اندیس t مشخص شده است. روابط زیر مقدار متغیرها را به شکل دقیق تر مشخص می کنند.



(شکل ۱): تعامل شبکه بهینه ساز و بهینه شونده در [۱]
(Figure -2): interaction of optimizer network and optimizee in [1]



(شکل - ۴): سلول LSTM
(Figure- 4): An LSTM cell

۱-۳ الگوریتم L-BFGS

در روش BFGS کلاسیک، برای یافتن کمینه تابعی مانند $F(\theta)$ از تقریب ماتریس Hessian (H) استفاده می‌شود. برای کاربردهای در مقیاس بزرگ لازم است از نسخه حافظه محدود استفاده کرد که روی تعداد زیادی متغیر قابل اجرا است که به آن روش L-BFGS می‌گویند. L-BFGS تاریخچه‌ای از متغیرها را در بازه زمانی l لحظه قبل ذخیره می‌کند که به l اندازه حافظه می‌گویند؛ سپس از یک تخمین اولیه برای H استفاده می‌کند که به‌طور معمول ماتریس همانی است. در ادامه به‌صورت مرحله‌به‌مرحله با شروع از لحظه $t - l$ ماتریس H به‌هنگام سازی می‌شود. دلیل این امر این است که دیگر نیازی به ذخیره ماتریس H نیست و تنها ضرب‌های میانی ایجادشده باید ذخیره شود. این الگوریتم تعدادی بردار را به عنوان ورودی می‌پذیرد. این بردارها شامل بردار q (که برابر منفی گرادیان کل ویژگی‌های بهینه‌شونده است) و بردارهای y_i و s_i هستند:

$$y_k = \nabla F(\theta_{(t+1)}) - \nabla F(\theta_{(t)}) \quad (13)$$

$$s_k = \theta_{(t+1)} - \theta_{(t)}$$

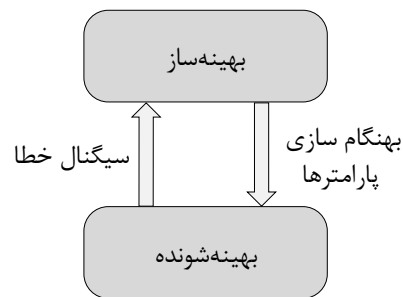
در نتیجه تعداد بردارهای ورودی برابر $2 * l + 1$ است. خروجی این الگوریتم بردار r است که معادل با ضرب تخمین ماتریس H در گرادیان تابع F در لحظه t است:

$$r = H_k \nabla F(\theta_{(t)}) \quad (14)$$

در نهایت با کمک این تخمین پارامترها به‌هنگام می‌شوند.

۲- روش ارائه‌شده

همان‌طور که گفتیم الگوریتم‌های بهینه‌سازی مبتنی بر بردار گرادیان (همانند کاهش گرادیان یا BFGS) در هر لحظه بردار گرادیان را دریافت می‌کنند و با نگهداری اطلاعات مکان‌های گذشته و گرادیان‌های گذشته، تغییر مکان فعلی را به دست می‌آورند. ورودی این الگوریتم‌ها و خروجی آنها بردارهایی با ابعاد مسأله بهینه‌سازی است و این الگوریتم‌ها تنها مبتنی بر عملیات ضرب داخلی، ضرب اسکالر و جمع برداری بر روی این بردارها نوشته می‌شوند؛ بنابراین می‌توان گفت که این الگوریتم‌ها در فضای هیلبرت بعد مسأله بهینه‌سازی اجرا می‌شوند. ما نیز قصد داریم با ایده‌گرفتن از این مطلب، فضایی

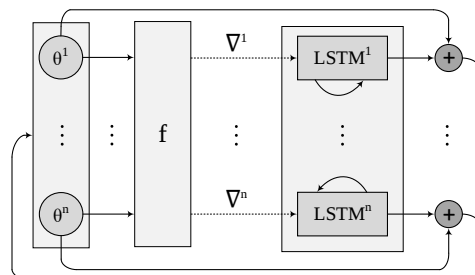


(شکل - ۲): شبکه بهینه‌ساز نحوه بهنگام سازی پارامترهای

بهینه‌شونده را تعیین می‌کند.

Figure (1): optimizer network determines the update rule for the optimizee parameters

همچنین برای کاهش بار محاسبات، مشابه آنچه در الگوریتم‌های ADAM و RMSprop وجود دارد، شبکه بهینه‌ساز به‌صورت مختصات‌محور^۸ عمل می‌کند. یعنی هر بُعد از پارامترهای تابع هدف، به‌صورت جداگانه و مستقل از ابعاد دیگر به‌هنگام می‌شود. (شکل - ۳) بیان‌گر این توضیحات است.



(شکل - ۳): عملکرد مختصات‌محور روی پارامترها

(Figure -3): coordinatewise operation on the parameters

$$i_t = \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \quad (7)$$

$$f_t = \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \quad (8)$$

$$g_t = \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \quad (9)$$

$$o_t = \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \quad (10)$$

$$c_t = f_t \times c_{t-1} + i_t \times g_t \quad (11)$$

$$h_t = o_t \times \tanh(c_t) \quad (12)$$

در این روابط W پارامتر شبکه است و اندیس آن نشان‌دهنده‌ی این است که پارامتر مربوط به کدام گیت و متغیر است. همچنین به دلیل خاصیت وزن اشتراکی^۹ در طول زمان، پارامتر W نیاز به اندیس t ندارد.

با در نظرگیری ساختار اولیه LSTM ما به دنبال اعمال تغییراتی هستیم تا مدل HLSTM را ارائه کنیم.

⁸ coordinatewise

⁹ weight sharing

ورودی و خروجی‌های این لایه در **Error! Reference source not found.** قابل مشاهده است.

در این لایه هر یک از بردارهای خروجی ترکیبی خطی از بردارهای ورودی هستند. برای اینکه از تعدادی بردار، ترکیب خطی آن‌ها را بسازیم به تعدادی ضریب نیاز داریم. این ضرایب، میزان دخیل بودن هر بردار ورودی را برای هر بردار خروجی تعیین می‌کند. زیرساختاری یا واحدی که ضرایب را تولید می‌کند بخش تولید ضرایب می‌نامیم. ساده‌ترین راه برای تولید ضرایب این است که ضرب داخلی بردارهای ورودی را به‌صورت دو به دو محاسبه کرده سپس آن‌ها را به یک شبکه عصبی بدهیم. این کار باعث می‌شود که یادگیری شبکه عصبی، مستقل از ابعاد بردارها باشد؛ چون از ضرب داخلی بردارها استفاده می‌کند و مستقیماً خود بردارها را به عنوان ورودی نمی‌گیرد.

در (شکل - ۵) جزئیات لایه ضرایب خطی را نشان داده‌ایم. برای مثال در این شکل مشخص است که چگونه n بردار ورودی دریافت و p بردار خروجی حاصل می‌شود. فرض کنید ساختار ما دارای n ورودی است. این ورودی‌ها به بخش تولید ضرایب داده می‌شوند. این بخش n عدد اسکالر تولید می‌کند که هر یک از این n عدد متناظر با یک بردار ورودی است و در آن بردار ضرب می‌شود. در نهایت بردارهای حاصل با هم جمع می‌شوند و یک خروجی را می‌سازند. برای تولید خروجی‌های بیشتر همین روال با ضرایب جدید انجام می‌شود. بنابراین اگر n بردار ورودی داشته باشیم و تعداد خروجی‌ها هم p باشد، بخش تولید ضرایب $n \times p$ مقدار اسکالر تولید می‌کند.

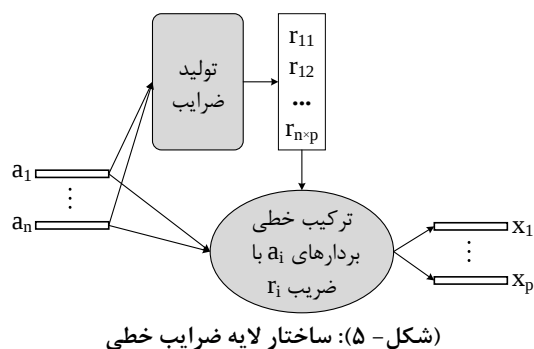
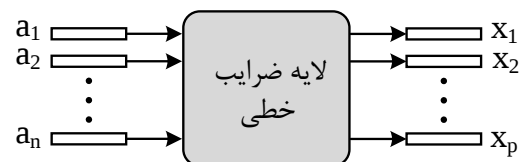


Figure (6): structure of linear coefficients layer

همانطور که گفتیم بخش تولید ضرایب، تعدادی بردار را به عنوان ورودی می‌گیرد و تعدادی اسکالر متناسب با تعداد بردارهای خروجی تولید می‌کند. این بخش خود شامل دو قسمت است. قسمت اول از بردارهای ورودی نمونه برداری می‌کند. قسمت دوم نمونه‌ها را به یک شبکه عصبی می‌دهد که این شبکه خروجی نهایی را که همان اعداد اسکالر است تولید می‌کند. این قسمت‌ها در **Error! Reference source not found.** نشان داده شده‌اند.

برای یادگیری ورودی‌ها ایجاد کنیم که مستقل از ابعاد ورودی باشد.

ما برای پیاده‌سازی روش خود از دو شبکه بهینه‌ساز و بهینه‌شونده استفاده می‌کنیم. شبکه بهینه‌ساز نقش تعیین گام به‌هنگام‌سازی برای پارامترهای بهینه‌شونده را بر عهده دارد. برای ساخت بهینه‌ساز از سلول LSTM استفاده می‌کنیم که ایده طراحی آن را از LSTM گرفته‌ایم. LSTM برخلاف LSTM می‌تواند با ورودی‌هایی با اندازه‌های مختلف کار کند و آن‌ها را یاد بگیرد. درواقع یادگیری ما مستقل از ابعاد است. دلیل این استقلال، این است که ما شبکه خود را به‌طورمستقیم روی داده‌ها آموزش نمی‌دهیم و ساختاری ارائه می‌کنیم تا روی ضرب داخلی ورودی‌ها آموزش ببیند. این ساختار، لایه ضرایب خطی نام دارد که در **Error! Reference source not found.** آمده است. در بخش ۳-۱ این لایه و اجزای آن را به‌صورت کامل توضیح می‌دهیم. در این لایه برای ترکیب ورودی‌ها از ضرب داخلی استفاده می‌کنیم. این ساختار الهام‌گرفته از الگوریتم L-BFGS است.



(شکل - ۵): لایه ضرایب خطی ارائه شده و ورودی‌ها و خروجی‌های این لایه
(Figure- 5): proposed linear coefficients layer and the inputs and outputs of this layer

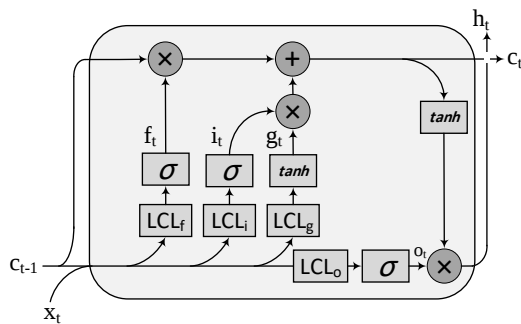
در ادامه ابتدا به ارتباط روش خود با L-BFGS می‌پردازیم و ساختار لایه ضرایب خطی را معرفی می‌کنیم. پس از آن سلول LSTM را معرفی می‌کنیم. همچنین معماری‌های مختلفی که می‌توان برای LSTM در نظر گرفت را بررسی می‌کنیم. سپس بهینه‌ساز را طراحی می‌کنیم که مبتنی بر LSTM است.

۲-۱- ساختار لایه ضرایب خطی

در بخش ۳-۲ الگوریتم L-BFGS را توضیح دادیم. تعامل به‌صورت دسته‌ای L-BFGS با ورودی‌ها، باعث شد تا به فکر ارائه بهینه‌سازی خودکار باشیم تا علاوه بر عمل کردن به‌صورت دسته‌ای روی ورودی‌ها، قادر به یادگیری قوانین بهینه‌سازی نیز باشد. برای این منظور نیاز به ساختاری است که بتواند صرف نظر از ابعاد بردارها، تعدادی بردار را به عنوان ورودی بگیرد. لایه ضرایب خطی تعدادی بردار را به عنوان ورودی می‌گیرد و در خروجی نیز تعدادی بردار برمی‌گرداند. تعداد بردارهای ورودی و خروجی مشخص است، ولی تعداد ویژگی‌های هریک از بردارهای ورودی و خروجی نامشخص اما یکسان است.

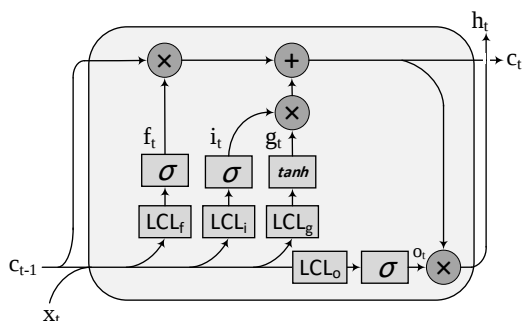
در همان لحظه (x_t) است. این بخش را در HLSTM با عنوان LCL_g نام‌گذاری می‌کنیم که در (شکل - ۶) قابل مشاهده است. خروجی این لایه هم به یک غیر خطی ساز $\tanh$ داده می‌شود تا خروجی g_t را تولید کند.

گیت‌ها مشابه LSTM با σ تعیین شده‌اند. اما در HLSTM برای تولید ورودی گیت‌ها از لایه ضرایب خطی استفاده می‌کنیم. این لایه‌ها ورودی گیت‌های فراموشی، ورودی و خروجی را می‌سازند و به ترتیب LCL_f ، LCL_i و LCL_o نام دارند. این بخش‌ها در (شکل - ۶) قابل مشاهده هستند. همانطور که مشخص است تمام بخش‌هایی که نیاز به یادگیری دارند، مستقیماً از بردارهای ورودی به سلول استفاده نمی‌کنند و از ضرب داخلی آن‌ها استفاده می‌کنند.



(شکل - ۶): ساختار اول سلول HLSTM
(Figure -8): first structure of HLSTM cell

برای تولید h_t خروجی را از یک تابع فعالیت $\tanh$ گذارنده‌ایم. اگر روی یک بردار بدون اینکه تبدیل روی آن صورت گیرد یک غیرخطی‌ساز اعمال شود، کمی مبهم به نظر می‌رسد و منجر به تخریب داده‌ها می‌شود. مثلاً در کاربرد طراحی بهینه‌ساز، اعمال غیرخطی‌ساز در میانه گراف محاسباتی، منجر به از بین رفتن اطلاعات می‌شود. برای رفع این مشکل دو راه وجود دارد. راه اول این است که در ورودی سلول HLSTM به جای h_{t-1} از c_{t-1} استفاده کنیم. در این صورت غیرخطی‌ساز $\tanh$ نقشی در میانه گراف محاسباتی ندارد و فقط در خروجی نهایی شرکت می‌کند. HLSTM (شکل - ۶) با این روش طراحی شده است. اما راه دیگر این است که کلاً بخش مربوط به غیرخطی‌ساز $\tanh$ در خروجی را حذف کنیم. ساختار HLSTM طراحی شده با این روش در (شکل - ۷) نشان داده شده است.



(شکل - ۷): ساختار بخش تولید ضرایب
(Figure-7): structure of coefficient generation unit

اما مشکلی که در استفاده از ضرب داخلی وجود دارد این است که وابسته به طول بردارها است. بنابراین باید هنجارسازی مناسبی صورت گیرد تا وابستگی ضرب داخلی را به تعداد ویژگی‌های بردارها کم کند. برای این کار ما حاصل ضرب داخلی بردارها را بر حاصل جمع نرم ۲ آن‌ها می‌کنیم:

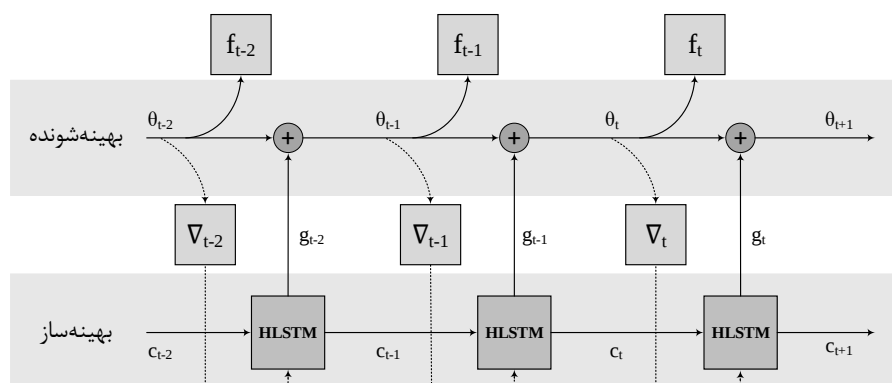
$$\frac{(a_i^T a_j)}{(\|a_i\|^2 + \|a_j\|^2)} \quad (15)$$

اعداد تولید شده در بخش نمونه‌برداری به یک شبکه عصبی داده می‌شوند. می‌توان از شبکه‌های مختلفی استفاده کرد ولی در اینجا ما از شبکه خطی تک لایه‌ای دارای بایاس استفاده می‌کنیم. خروجی شبکه عصبی همان ضرایب لازم برای ترکیب خطی بردارهای ورودی است؛ درواقع ما در اینجا ضرایب ترکیب خطی بردارهای ورودی را از طریق یک شبکه عصبی یاد گرفتیم که خود مستقل از ابعاد داده‌های ورودی آموزش دیده است.

۲-۲- ساختار HLSTM

در بخش قبل لایه ضرایب خطی را توضیح دادیم. حال می‌خواهیم با ایده گرفتن از سلول LSTM و همچنین ترکیب آن با لایه ضرایب خطی یک ساختار بازگشتی مبتنی بر بردار طراحی کنیم. این ساختار صرف نظر از ابعاد بردار ورودی، ویژگی‌ها را یاد می‌گیرد.

همانطور که در (شکل - ۴) مشاهده می‌شود، در قسمتی از سلول LSTM یک شبکه با تابع فعالیت $\tanh$ خروجی g_t را تولید می‌کند. ما برای طراحی HLSTM، بخش خطی این قسمت را با لایه ضرایب خطی جایگزین می‌کنیم. تعداد بردارهای ورودی به لایه ضرایب خطی برابر با مجموع تعداد بردارهای حالت مخفی لحظه قبل (h_{t-1}) و تعداد بردارهای ورودی



(شکل - ۸): ساختار شبکه بهینه ساز و بهینه شونده و نحوه تعامل آن ها با یکدیگر
(Figure -10): the structure of optimizer network and optimizee and interaction between them

بهینه ساز طراحی شده توسط ما برای یادگیری ورودی از فضای ضرب داخلی استفاده می کند، در نتیجه می تواند با ورودی های با ابعاد متنوع و مختلف هم کار کند. همچنین برخلاف بسیاری از روش های بهینه سازی نظیر ADAM، RMSprop و [۱] که به صورت مختصات محور روی ابعاد عمل می کنند، مدل ما ارتباطات بین ابعاد مختلف ورودی را در نظر می گیرد. در بخش بعدی علاوه بر مقایسه روش خود با الگوریتم های بهینه سازی دیگر، معماری های مختلفی از بهینه ساز خود را نیز با هم مقایسه می کنیم.

۳- آزمایش ها و نتایج

در این بخش به آزمایش معماری های مختلف بهینه ساز ارائه شده و همچنین مقایسه آن با دیگر الگوریتم های بهینه سازی می پردازیم. برای این کار مجموعه ای از توابع درجه دو مصنوعی مطابق معادله (۱۷) را تولید می کنیم و مدل های مختلف را برای کمینه کردن آن ها آموزش می دهیم.

$$f(\theta) = \|W\theta - y\|_2^2 \quad (17)$$

مقادیر این متغیرها با نمونه برداری از توزیع گوسی مستقل با توزیع یکسان به دست می آیند. الگوریتم ها روی مجموعه ای تصادفی از این توابع درجه دو آموزش می بینند و سپس در مرحله بعد روی مجموعه جدیدی اعتبارسنجی^{۱۰} و آزمون^{۱۱} می شوند.

در تمام آزمایش ها از یک بهینه ساز دولایه استفاده می کنیم. در هر لایه با توجه به معماری، ساختار اول سلول HLSTM یا ساختار دوم آن را به کار می گیریم. برای آموزش بهینه ساز نیز از الگوریتم ADAM استفاده می کنیم. در پایان

۳-۲- طراحی شبکه بهینه ساز با کمک HLSTM

همان طور که توضیح داده شد، ما از لایه ضرایب خطی در ساخت سلول HLSTM استفاده کردیم و اکنون می توانیم با کمک این سلول، بهینه ساز را طراحی کنیم. پارامتر شبکه بهینه ساز را ϕ و پارامتر بهینه شونده را θ می نامیم. حال با کمک بهینه ساز، طول گام به هنگام سازی g_t را برای پارامترهای شبکه بهینه شونده را تعیین می کنیم. برای آموزش بهینه ساز از تابع هزینه (۱۶) استفاده می کنیم:

$$L(\phi) = E_f \left[\sum_{t=1}^T \omega_t f(\theta_t) \right], \quad (16)$$

$$\theta_{t+1} = \theta_t + g_t,$$

$$[g_t]_{c_{t+1}} = HLSTM(\nabla_t, c_t, \phi)$$

همان طور که مشخص است، در هر گام پارامتر θ با توجه به g_t به هنگام می شود که g_t خروجی شبکه HLSTM است. با مقایسه معادله (۱۶) با معادلات (۵) و (۶) مشاهده می شود که ما ساختار HLSTM را جایگزین شبکه عصبی بازگشتی سابق کرده ایم.

برای به هنگام سازی پارامتر شبکه بهینه ساز از الگوریتم کاهش گرادینان استفاده می کنیم. ساختار طراحی شده و نحوه تعامل شبکه بهینه ساز و بهینه شونده در (شکل - ۸) قابل مشاهده است. در مدل ما نیز مشابه [۱] گرادینان فقط از خطوط توپر عبور می کند و از خطوط نقطه چین عبور نمی کند. عدم بازگرداندن گرادینان از خطوط نقطه چین به این معنا است که گرادینان بهینه شونده به پارامترهای بهینه ساز وابسته نیست؛ همچنین این فرض منجر به این می شود که نیازی به محاسبه مشتق دوم f نباشد. در طراحی بهینه ساز می توان از هر دو ساختار HLSTM طراحی شده استفاده کرد.

¹⁰ validation
¹¹ test

هر دور^{۱۲}، پارامترهای بهینه‌ساز ثابت می‌شوند و مدل اعتبارسنجی می‌شود. در انتهای آموزش بهترین مدل انتخاب می‌شود و آزمون براساس آن مدل انجام می‌شود. ارزیابی براساس میانگین نتایج به‌دست‌آمده روی صد مسأله درجه دوم جدید انجام می‌شود.

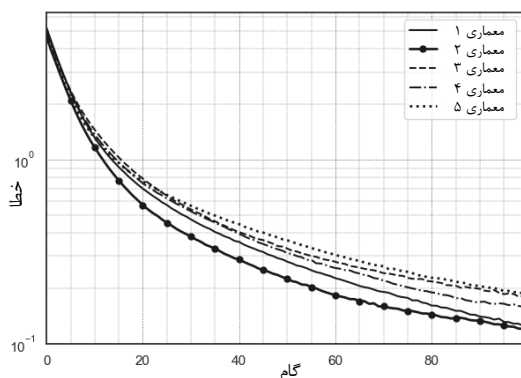
(جدول ۱-): پارامترهای معماری‌های مختلف ارائه شده

Table (1): parameters of different proposed architectures

پارامترها	معماری ۱	معماری ۲	معماری ۳	معماری ۴	معماری ۵
لایه ۱	ساختار دوم HLSTM	ساختار دوم HLSTM	ساختار دوم HLSTM	ساختار دوم HLSTM	ساختار دوم HLSTM
لایه ۲	ساختار دوم HLSTM	ساختار اول HLSTM	ساختار دوم HLSTM	ساختار اول HLSTM	ساختار اول HLSTM
گام بهینه‌سازی	۱۰۰	۱۰۰	۱۰۰	۱۰۰	۱۰۰
دور	۵۰	۵۰	۱۰۰۰	۱۰۰۰	۵۰
تکرار	۲۰	۲۰	۱	۱	۲۰
نرخ یادگیری	۰/۰۰۳	۰/۰۰۳	۰/۰۰۳	۰/۰۰۳	۰/۰۰۳
تعداد نورون مخفی	۱	۱	۱	۱	۵

کوچکتری نسبت به معماری ۱ صورت می‌پذیرد. تعداد نورون‌های مخفی برای هر لایه هم برابر ۱ است.

- معماری ۴: مانند معماری ۲، لایه اول از ساختار دوم سلول HLSTM استفاده می‌کند و لایه دوم از ساختار اول استفاده می‌کند. تعداد دور و تکرار مشابه معماری ۳ به ترتیب برابر ۱۰۰۰ و ۱ است. تعداد نورون‌های مخفی برای هر لایه هم برابر ۱ است
- معماری ۵: این معماری مشابه معماری ۲ است با این تفاوت که تعداد نورون‌های مخفی برای هر لایه ۵ است. دقت کنید که در تمام معماری‌ها قبلی این تعداد برابر ۱ بود.



(شکل - ۹): نتایج آزمایشات انجام شده روی معماری‌های مختلف ارائه شده روی مسائل درجه ۲ با اندازه بعد ۵
(Figure -11): the result of experiments on different architectures of proposed method on 5-dimensional quadratic functions

خلاصه معماری‌های بالا با جزئیات کامل در (جدول) آمده است. در این جدول، گام بهینه‌سازی، طول زمانی تخصیص داده شده به مدل برای یادگیری و ارزیابی را تعیین می‌کند. نرخ یادگیری نیز برای تمام معماری‌ها برابر ۰/۰۰۳ در نظر گرفته شده است. ما نرخ یادگیری‌ها مختلفی را برای معماری‌ها بررسی کردیم و با روش جستجوی تصادفی به این نرخ رسیدیم. همچنین تعداد نورون‌های مخفی برای هر لایه در جدول

برای پیاده‌سازی مدل خود و دیگر الگوریتم‌ها از زبان برنامه‌نویسی Python و به‌طور خاص از ابزار Pytorch استفاده می‌کنیم. پارامترهای دقیق هر آزمایش در هر بخش به‌طور دقیق ذکر شده‌اند.

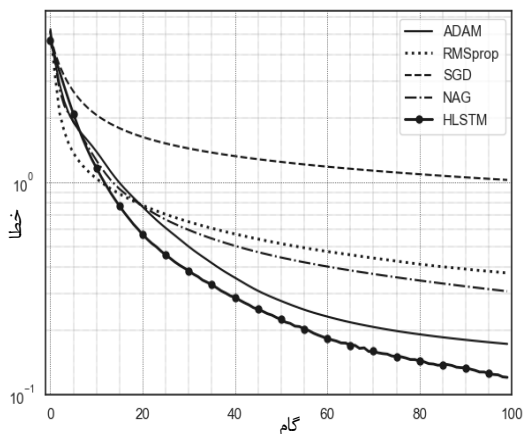
۱-۳- مقایسه معماری‌های مختلف بهینه‌ساز ارائه شده

در این بخش معماری‌های مختلفی از بهینه‌ساز ارائه شده را طراحی و با هم مقایسه می‌کنیم. لازم به یادآوری است که در تمام معماری‌ها از دو لایه HLSTM برای بهینه‌ساز استفاده می‌شود. در انجام آزمایش‌ها با دو مفهوم دور و تکرار^{۱۳} سروکار داریم. اگر تعداد دور برابر n و تعداد تکرار برابر m باشد، در مجموع $n \times m$ دفعه آموزش انجام می‌گیرد. به این صورت که در هر دور تعداد m مسئله درجه دو تصادفی طراحی و برای آموزش استفاده می‌شود. پس از این m دفعه ما به انتهای دور می‌رسیم و مدل آموزش دیده‌شده را اعتبارسنجی می‌کنیم. در ادامه معماری‌ها را معرفی می‌کنیم:

- معماری ۱: هر دو لایه این معماری از ساختار دوم سلول HLSTM ((شکل - ۷)) استفاده می‌کنند. تعداد دورها برابر ۵۰ و تکرارها برابر ۲۰ است. همچنین تعداد نورون‌های مخفی برای هر لایه ۱ در نظر می‌گیریم.
- معماری ۲: لایه اول این معماری از ساختار دوم سلول HLSTM استفاده می‌کند. اما لایه دوم از ساختار اول ((شکل - ۶)) استفاده می‌کند. تعداد دورها برابر ۵۰ و تکرارها برابر ۲۰ است و تعداد نورون‌های مخفی برای هر لایه هم برابر ۱ می‌باشد.
- معماری ۳: مانند معماری ۱ هر دو لایه این معماری از ساختار دوم سلول HLSTM ((شکل - ۷)) استفاده می‌کنند. اما در اینجا تعداد دورها برابر ۱۰۰۰ و تعداد تکرارها برابر ۱ است. یعنی اعتبارسنجی در طی گام‌های

¹³ iteration

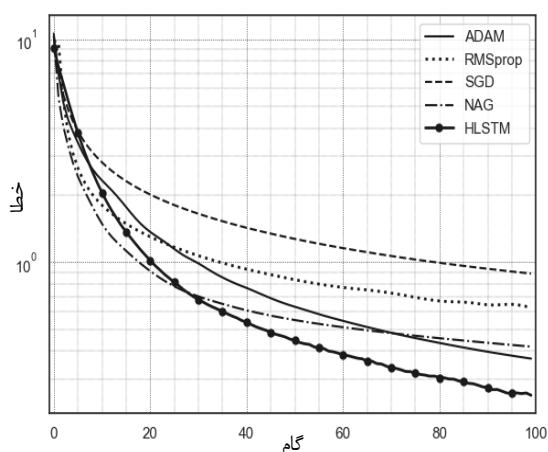
عمل کرده است. همپنین روش‌های NAG و ADAM هم نسبت به دیگر روش‌های کلاسیک عملکرد بهتری داشته‌اند.



شکل (۱۰): مقایسه روش ارائه شده (HLSTM) با الگوریتم‌های

بهینه‌سازی دستی روی مسائل درجه ۲ با اندازه بعد ۵

Figure (13): comparison of proposed method with hand-designed optimization algorithms on 5-dimensional quadratic functions



شکل (۱۱): مقایسه روش ارائه شده (HLSTM) با

الگوریتم‌های بهینه‌سازی دستی روی مسائل درجه ۲ با اندازه بعد ۱۰

Figure (14): comparison of proposed method with hand-designed optimization algorithms on 10-dimensional quadratic functions

۳-۳- مقایسه بهینه‌ساز ارائه‌شده با

بهینه‌ساز مبتنی بر LSTM

در این بخش می‌خواهیم بهینه‌ساز ارائه‌شده خود را، که در طراحی آن از HLSTM استفاده کردیم، با یک بهینه‌ساز دیگر [۱] که مبتنی بر LSTM است مقایسه کنیم. در روش [۱] تعداد سلول‌های مخفی در هر لایه ۲۰ در نظر گرفته شده است. اندازه توابع درجه دو که در اینجا قصد استفاده از آنها را داریم ۵ است. یعنی W ماتریسی ۵ در ۵ و b برداری با طول ۵ است. تعداد دوره‌های در نظر گرفته‌شده برای آموزش مدل ما ۵۰ و برای مدل دیگر صد است. نتیجه Error! Reference

مشخص شده است. ابتدا نتایج را روی توابع درجه دو با اندازه بعد ۵ بررسی می‌کنیم. یعنی W ماتریسی ۵ در ۵ و b برداری با طول ۵ است. برای ارزیابی صد مسئله درجه دو جدید با همین اندازه از توزیع یاد شده تولید می‌کنیم؛ سپس میانگین نتایج هر مدل را در طول گام‌های بهینه‌سازی این مسائل نشان می‌دهیم. (شکل - ۹) این نتایج را نشان می‌دهد. طبق نتایج معماری ۲ بهترین نتیجه را داده و پس از آن معماری ۱ با اختلاف اندکی نتیجه خوبی داده است؛ اما معماری ۳ نتیجه خوبی ندارد و این نشان می‌دهد ارزیابی مدل در پس از هر بار حل مسئله درجه دو راهکار چندان مناسبی نیست. معماری ۵ نیز نتیجه ضعیف‌تری نسبت به دیگر معماری‌ها دارد و نشان می‌دهد افزایش بردارهای مخفی تأثیری در بهبود نتایج ندارد و حتی موجب بدتر شدن آن نیز می‌شود.

حال قصد داریم آزمایش را با توابع درجه دو با اندازه ۱۰ انجام دهیم. با توجه به اینکه معماری ۱ و ۲ با اختلاف نسبت به دیگر معماری‌ها نتایج بهتری گرفته‌اند، فقط روی این دو معماری آزمایش‌ها را انجام می‌دهیم. نتایج در Error! Reference source not found. (۲) نتایج بهتری ارائه داده است و این نشان می‌دهد استفاده از HLSTM نوع اول در لایه دوم منجر به نتایج بهتر می‌شود. با مقایسه (شکل - ۹) و Error! Reference source not found. مشاهده می‌شود که مسئله با اندازه ۱۰ به مراتب سخت‌تر از مسئله با اندازه ۵ است و میزان خطای مدل در مسئله بزرگتر، بیشتر است.

با توجه به اینکه نتایج معماری ۲ بهتر از دیگر معماری‌ها بود. در ادامه برای مقایسه روش خود با دیگر روش‌ها از این معماری استفاده می‌کنیم.

۲-۳- مقایسه بهینه‌ساز ارائه‌شده با

الگوریتم‌های بهینه‌سازی دستی

در این بخش قصد داریم بهینه‌ساز ارائه شده (مبتنی بر معماری ۲) را با دیگر روش‌های بهینه‌سازی کلاسیک مقایسه کنیم. الگوریتم‌هایی که برای مقایسه انتخاب کردیم، عبارتند از: ADAM, RMSprop, SGD و NAG. برای هر روش بهترین نرخ یادگیری را در نظر می‌گیریم به‌طوری که کمترین خطا حاصل شود. این نرخ برای الگوریتم‌ها به ترتیب برابر ۰/۰۳، ۰/۰۱ و ۰/۰۱ است. اندازه بعد تابع درجه دو در اینجا برابر ۵ و گام بهینه‌سازی و دور برای روش‌های کلاسیک هر دو برابر صد است. برای روش خودمان دور را همان پنجاه قرار می‌دهیم. نتایج به‌دست‌آمده در شکل (۱۰) مشاهده می‌شوند. روش ارائه‌شده ما که با نام HLSTM در شکل مشخص شده، با اختلاف زیادی نسبت به دیگر روش‌ها بهتر عمل کرده و خطا را کاهش داده است. در بین روش‌های کلاسیک نیز ADAM و پس از آن NAG نتایج بهتری گرفته‌اند.

در آزمایشی دیگر قصد داریم روش‌های قبلی را این بار روی مسئله درجه دو با اندازه ۱۰ بررسی کنیم. شکل (۱۱) نتایج را نشان می‌دهد. در اینجا نیز روش ما بهتر از دیگر روش‌ها

۴- نتیجه‌گیری و کارهای آینده

ما در اینجا یک بهینه‌ساز خودکار طراحی کردیم تا با کمک آن بتوانیم طول گام بهینه‌سازی برای پارامترهای یک شبکه دیگر را تعیین کنیم. در واقع ما مسأله بهینه‌سازی را به عنوان یک مسأله یادگیری ماشین در نظر گرفتیم. برای طراحی بهینه‌ساز از الگوریتم L-BFGS و ساختار سلول LSTM ایده گرفتیم. ابتدا لایه ضرایب خطی را معرفی کردیم. سپس با کمک آن سلول HLSTM را طراحی کردیم. در ادامه با کمک HLSTM، شبکه بهینه‌ساز را طراحی کردیم. طراحی ما به گونه‌ای بود که می‌توانست با بردارهایی با طول متغیر کار کند نحوه بهینه‌سازی را مستقل از ابعاد داده یاد بگیرد. نتایج به‌دست آمده نشان‌دهنده‌ی موثر بودن راهکار ما بود.

یکی از کارهایی که می‌توان در ادامه انجام داد، افزایش سرعت اجرا و یادگیری بهینه‌ساز است. ساختار بازگشتی بهینه‌ساز باعث افزایش زمان اجرا نسبت به الگوریتم‌های کلاسیک می‌شد. با بهینه‌کردن زمان اجرا می‌توان از بهینه‌ساز ارائه شده در حل مسائل پیچیده‌تر و یادگیری داده‌های واقعی استفاده کرد. علاوه‌بر این می‌توان سیر تکاملی LSTM را برای HLSTM نیز در نظر گرفت. برای مثال می‌توان HLSTM دو طرفه یا چند جهته طراحی کرد.

۵- منابع

- [1] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in Advances in neural information processing systems, 2016, pp. 3981-3989.
- [2] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$," in Doklady an ussr, 1983, vol. 269, pp. 543-547.
- [3] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in IEEE international conference on neural networks, 1993: IEEE, pp. 586-591.
- [4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Journal of machine learning research, vol. 12, no. Jul, pp. 2121-2159, 2011.
- [5] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," COURSE: Neural networks for machine learning, vol. 4, no. 2, pp. 26-31, 2012.
- [6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

- [7] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in International conference on machine learning, 2015, pp. 2408-2417.
- [8] D. L. Donoho, "Compressed sensing," IEEE Transactions on information theory, vol. 52, no. 4, pp. 1289-1306, 2006.
- [9] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE transactions on evolutionary computation, vol. 1, no. 1, pp. 67-82, 1997.
- [10] N. J. W. SJ, "Numerical optimization: Springer Science+ Business Media," ed: LLC, 2006.
- [11] S. Thrun and L. Pratt, "Learning to learn: Introduction and overview," in Learning to learn: Springer, 1998, pp. 3-17.
- [12] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook," Technische Universität München, 1987.
- [13] X. Chen et al., "Symbolic discovery of optimization algorithms," Advances in Neural Information Processing Systems, vol. 36, 2024.
- [14] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," IEEE transactions on pattern analysis and machine intelligence, vol. 44, no. 9, pp. 5149-5169, 2021.
- [15] E. Gärtner, L. Metz, M. Andriluka, C. D. Freeman, and C. Sminchisescu, "Transformer-based learned optimization," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 11970-11979.
- [16] L. S. Metz, N. Maheswaranathan, C. D. Freeman, B. Poole, and J. N. Sohl-Dickstein, "Training neural networks using learned optimizers," ed: Google Patents, 2022.
- [17] E. D. Cubuk, L. S. Metz, S. S. Schoenholz, and A. A. Merchant, "Optimization using learned neural network optimizers," ed: Google Patents, 2022.
- [18] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," Behavioral and brain sciences, vol. 40, 2017.
- [19] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in International conference on machine learning, 2016, pp. 1842-1850.
- [20] J. Schmidhuber, "A neural network that embeds its own meta-levels," in IEEE International Conference on Neural Networks, 1993: IEEE, pp. 407-412.
- [21] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," Neural Computation, vol. 4, no. 1, pp. 131-139, 1992.



اشکان صادقی لطف‌آبادی مدرک کارشناسی خود را در رشته مهندسی کامپیوتر-سخت‌افزار در سال ۱۳۹۳ از دانشگاه فردوسی مشهد و سپس در سال ۱۳۹۶ مدرک کارشناسی ارشد خود را در رشته کامپیوتر-هوش مصنوعی و رباتیک از همان دانشگاه دریافت کرد. ایشان هم اکنون دانشجوی دکترای دانشگاه فردوسی مشهد در رشته کامپیوتر-هوش مصنوعی و رباتیک است. زمینه‌های پژوهشی مورد علاقه ایشان شبکه‌های عصبی مصنوعی، یادگیری عمیق و مدل‌های دنباله‌ای است. نشانی رایانامه ایشان عبارت است از:

sadeghia@mail.um.ac.ir



سید کمال الدین غیائی شیرازی در سال ۱۳۷۶ دیپلم ریاضی خود را از دبیرستان البرز تهران و پس از طی دوره پیش-دانشگاهی در مدرسه شهید مطهری در سال ۱۳۸۱ مدرک کارشناسی خود را از دانشگاه شهید بهشتی تهران در رشته مهندسی نرم‌افزار کامپیوتر دریافت کرد. ایشان سپس در سال ۱۳۸۳ مدرک کارشناسی ارشد خود را از دانشگاه صنعتی شریف و در سال ۱۳۸۹ مدرک دکترای تخصصی خود را از دانشگاه صنعتی امیرکبیر در گرایش هوش مصنوعی رشته مهندسی کامپیوتر دریافت کرد. وی از سال ۱۳۹۲ تاکنون به‌عنوان هیئت علمی گرایش هوش مصنوعی گروه مهندسی کامپیوتر دانشکده مهندسی دانشگاه فردوسی مشهد مشغول به خدمت است. زمینه‌های پژوهشی ایشان شبکه‌های عصبی، یادگیری عمیق، روش‌های هسته، مدل‌های گرافی احتمالاتی، یادگیری ماشین و شناسایی آماری الگو است. نشانی رایانامه ایشان عبارت است از:

k.ghiasi@um.ac.ir

- [22] J. Schmidhuber, J. Zhao, and M. Wiering, "Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement," *Machine Learning*, vol. 28, no. 1, pp. 105-130, 1997.
- [23] C. Daniel, J. Taylor, and S. Nowozin, "Learning step size controllers for robust neural network training," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [24] T. P. Runarsson and M. T. Jonsson, "Evolution and design of distributed learning rules," in *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No. 00, 2000: IEEE*, pp. 59-63.
- [25] Y. Bengio, S. Bengio, and J. Cloutier, "Learning a synaptic learning rule: Université de Montréal," *Département d'informatique et de recherche opérationnelle*, 1990.
- [26] N. E. Cotter and P. R. Conwell, "Fixed-weight networks can learn," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990: IEEE, pp. 553-559.
- [27] A. S. Younger, P. R. Conwell, and N. E. Cotter, "Fixed-weight on-line learning," *IEEE Transactions on Neural Networks*, vol. 10, no. 2, pp. 272-283, 1999.
- [28] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *International Conference on Artificial Neural Networks*, 2001: Springer, pp. 87-94.
- [29] A. S. Younger, S. Hochreiter, and P. R. Conwell, "Meta-learning with backpropagation," in *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, 2001, vol. 3: IEEE.
- [30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.



محمد اعتصام در سال ۱۳۹۴ مدرک کارشناسی خود را از دانشگاه فردوسی مشهد در رشته مهندسی نرم‌افزار کامپیوتر و همچنین در سال ۱۳۹۸ مدرک کارشناسی ارشد خود را از دانشگاه فردوسی مشهد در گرایش هوش مصنوعی رشته مهندسی کامپیوتر دریافت کرد. برخی زمینه‌های پژوهشی مورد علاقه ایشان شبکه‌های عصبی و یادگیری عمیق است.

نشانی رایانامه ایشان عبارت است از:

etesam@mail.um.ac.ir

