یک پیاده سازی موازی برای الگوریتم بقا

بهبود مشعوفی
دانشگاه ارومیه-گروه برق
b.mashoufi@mail.urmia.ac.ir

محمد رضا میبدی
دانشگاه صنعتی امیرکبیر-دانشکده کامپیوتر
meybodi@ce.aut.ac.ir

سید احمد معتمدی
دانشگاه صنعتی امیرکبیر-دانشکده برق
motamedi@aut.ac.ir

چکیده: مهمترین محدودیت در تحقیقات مربوط به شبکه های عصبی، زمان بالای آموزش آنها میباشد. یک راه حل برای کاهش زمان آموزش، استفاده از نقاط توازی موجود در شبکه و نگاشت آن بر روی یک کامپیوتر موازی است. یکی از روشهای نگاشت، روش افراز نرون میباشد. در این روش داده های زیادی بین پردازشگرها مبادله شده و پیچیدگی ارتباطی بالا میباشد. لذا زمان زیادی صرف مبادله اطلاعات شده و زمان آموزش افزایش می یابد. هزینه ارتباطات متناسب با تعداد نرونهاست. اگر تعداد نرونها خیلی کم باشد. با اینکه شبکه دارای هزینه ارتباطی پائینی خواهد بود ولی قادر به یادگیری مسئله نخواهد بود. از طرف دیگر شبکه های با تعداد بالای نرون، دچار *Overfitting* شده و قدرت تعمیم پائینی خواهد داشت علاوه بر این آموزش اینگونه شبکه ها مستلزم هزینه ارتباطی بالائی خواهد بود. لذا نیاز به الگوریتمهای داریم که بتوان تعداد بهینه نرونها را تعیین کرد. یکی از الگوریتمهای موجود، الگوریتم بقا نرون میباشد. در این مقاله یک الگوریتم موازی تحت عنوان الگوریتم بقا نرون موازی ارایه میگردد. با استفاده از الگوریتم پیشنهادی میتوان هزینه ارتباطی بین پردازشگرها را کاهش داده در نتیجه زمان آموزش را تقلیل داد. الگوریتم مذکور بر روی کاربرد بازشناسی واجهای فارسی اعمال شد. نتایج شبیه سازیها نشان میدهد الگوریتم پیشنهادی از سرعت بالایی نسبت به روش افراز نرون برخوردار میباشد.

واژه های کلیدی- شبکه های عصبی، پردازش موازی، اتوماتانهای یادگیر، هزینه ارتباطات، یادگیری ساختار

۱- مقدمه

مهمترین محدودیت در تحقیقات مربوط به شبکه های عصبی زمان بالای آموزش آنها میباشد. برای انواع استاندارد شبکه ها، سخت افزارهای خاص سریعی وجود دارد. اما اغلب این سخت افزارها از انعطاف لازم برای کارهای تحقیقاتی برخوردار نمیباشند. در مقابل کامپیوترهای با کاربرد عام دارای انعطاف پذیری بالایی بوده ولی زمانهای آموزش برای شبکه های بزرگ با تعداد زیاد الگوهای آموزش، با استفاده از این کامپیوترها بسیار بالا میباشد. استفاده از سوپر کامپیوتر ها میتواند زمان آموزش را بمیزان قابل توجهی کاهش دهد، اما این راه حل بدلیل بالا بودن قیمت، هزینه نگهداری و توان مصرفی و همچنین حجیم بودن آنها همیشه رضایت بخش نبوده و استفاده از سوپر کامپیوترها برای آموزش شبکه ها، راه حل بهینه ای نمیباشد. در بسیاری از مراکز تحقیقاتی از نقاط توازی موجود در شبکه های عصبی استفاده کرده و آنرا بر روی یک کامپیوتر موازی نگاشت میکنند. با استفاده از روشهای موازی میتوان شبکه های بزرگ با تعداد زیاد داده های آموزش را با سرعت بالایی تعلیم داد. الگوریتم پس انتشار خطا دارای چندین نقطه توازی میباشد که عبارتند از: توازی نرون و توازی داده های آموزش. میتوان با دو روش مختلف، الگوریتم پس انتشار خطا را بر روی یک کامپیوتر موازی نگاشت کرد. این روشها عبارتند از: روش افراز داده [۱][۲] و روش افراز نرون [۳][۴] . در روش افراز نرون، نرونهای هر لایه بطور مساوی بین پروسسورها تقسیم میشود. در این روش در طول پروسه آموزش داده های زیادی بین پردازشگرها مبادله شده و لذا پیچیدگی ارتباطی بالا میباشد. در نتیجه زمان زیادی صرف مبادله اطلاعات شده و زمان آموزش افزایش می یابد. هزینه ارتباطات متناسب با تعداد نرونهای لایه های مخفی بوده و با کاهش تعداد نرونها میتوان هزینه ارتباطات را کاهش داد. اگر تعداد نرونهای لایه مخفی خیلی کم باشد. با اینکه شبکه دارای هزینه ارتباطی پائینی خواهد بود ولی قادر به یادگیری مسئله نخواهد بود. از طرف دیگر شبکه های با تعداد بالای نرونهای لایه مخفی، دچار Overfitting شده و قدرت تعمیم پائینی خواهد داشت علاوه بر این آموزش اینگونه شبکه ها با استفاده از کامپیوترهای موازی مستلزم هزینه ارتباطی بالائی خواهد بود. لذا نیاز به الگوریتمهای داریم که بتوان تعداد بهینه نرونهای لایه مخفی را تعیین کرد. الگوریتمهائی را که تاکنون توسط افراد مختلف برای این منظور ارائه شده است میتوان به پنج گروه زیرتقسیم کرد. الگوریتمهای هرس[۵] ، الگوریتمهای سازنده[۶] ، الگوریتمهای ترکیبی[۷] ، الگوریتمهای تکاملی[۸] و الگوریتمهای بر اساس اتوماتانهای یادگیر [۹،۱۰،۱۱،۱۲،۱۳،۱۴]. توسط میبدی و بیگی الگوریتمهایی بر اساس اتوماتانهای یادگیر برای دستیابی به ساختارهای بهینه با پیچیدگی آموزش کم و قدرت تعمیم بالاارائه گردیده است[۱۲،۱۳،۱۴]. در اولین الگوریتم ارائه شده تحت عنوان الگوریتم بقانرون NSA[1] از یک اتوماتان یادگیر مهاجرت اشیا بعنوان یک ابزار جستجوی عمومی استفاده شده است. این الگوریتم حین آموزش، ساختار مناسبی برای شبکهٔ عصبی سه لایه از حیث پائین بودن پیچیدگی آموزش و قدرت تعمیم بالا ارایه میکند. اتوماتانهای یادگیر علاوه بر تعیین ساختاربهینه برای شبکه های

[1] Neuron Survival Algorithm

عصبی،برای تطبیق پارامترهای شبکه های عصبی نیز مورد استفاده قرارگرفته است[۱۵،۱۶] .در این مقاله یک الگوریتم موازی تحت عنوان الگوریتم بقا نرون موازی PNSA[۲] ارایه میگردد. با ارائه این الگوریتم میتوان نه تنها با انطباق مدل با پیچیدگی مسئله، قدرت تعمیم شبکه را افزایش داد بلکه با کاهش پیچیدگی شبکه تواماً هزینه محاسباتی هر پردازشگر وهزینه ارتباطی بین پردازشگر ها را کاهش داده در نتیجه زمان آموزش را تقلیل داد. بخش های بعدی مقاله بصورت زیر سازماندهی شده است. در بخش ۲ الگوریتم یادگیری پس انتشار خطا و اتوماتانهای یادگیر توضیح داده میشود. موازی سازی در بخش ۳ آورده شده است. در بخش ۴ الگوریتم NSA را توضیح داده ایم. الگوریتم پیشنهادی PNSA در بخش ۵ مطرح میشود. در بخش ۶ نتایج شبیه سازیها و در بخش پایانی نتیجه گیری آورده شده است.

۲- الگوریتم پس انتشار خطاواتوماتانهای یادگیر

الگوریتم پس انتشار: الگوریتم BP[3]یک روش سیستماتیک برای آموزش شبکه های عصبی چند لایه میباشد[۱۷].الگوریتمBP، دومسیر محاسباتی دارد. مسیر رفت و مسیر برگشت.

مسیر رفت: این مسیر با معادلات زیر توصیف می شود:

$$\underline{a}^0 = \underline{p}(k)$$
$$\underline{a}^{l+1}(k) = \underline{F}^{l+1}\left(W^{l+1}(k)\underline{a}^l + \underline{b}^{l+1}(k)\right), \quad l = 0,1,...,L-1$$
$$\underline{a} = \underline{a}^L(k)$$

در این مسیر توابع محرک، روی تمامی نرونها عمل می کند، یعنی:

$$\underline{F}^{l+1}(\underline{n}(k)) = \left[f^{l+1}\left(n_1(k)\right)...f^{l+1}\left(n_{s_{l+1}}(k)\right)\right]^T$$

مسیر برگشت: در این مسیر بردارهای حساسیت از لایه آخر به لایه اول برگشت داده می شوند. معادلات زیر، دینامیک مسیر برگشت را بیان میکند.

$$\underline{\delta}^L(k) = -2\dot{F}^L(\underline{n})\underline{e}(k)$$
$$\underline{\delta}^l(k) = \dot{F}^l(\underline{n}^l)(W^{l+1})^T\underline{\delta}^{l+1}, \quad l = L-1,...,1$$
$$\underline{e}(k) = \underline{t}(k) - \underline{a}(k)$$

در مسیر برگشت ابتدا بردار خطا محاسبه شده سپس بردار خطا از سمت راست به چپ و از لایه آخر به لایهٔ اول توزیع شده و با الگوریتم بازگشتی گرادیان محلی نرون به نرون محاسبه میشود.

تنظیم پارامترها: در این مرحله ماتریسهای وزن و بردارهای بایاس شبکه بصورت زیر تنظیم می شوند.

$$W^l(k+1) = W^l(k) - \alpha\underline{\delta}^l(k)\left(\underline{a}^{l-1}(k)\right)^T$$
$$b^l(k+1) = b^l(k) - \alpha\underline{\delta}^l(k), \qquad l = 1,2,...,L$$

توقف: اگر میانگین مربعات خطا در هر e poch کمتر از مقدار از پیش تعیین شده ای باشد الگوریتم BP متوقف می شود.

اتوماتان یادگیر:اتوماتان یادگیر [۱۸] یک مدل انتزاعی است که میتواند تعداد محدودی اقدام را انجام دهد. هر اقدام انتخاب شده توسط محیط ارزیابی شده و پاسخی به اتوماتای یادگیر داده میشود. اتوماتای یادگیر از این پاسخ استفاده نموده و اقدام خود را برای مرحله بعد انتخاب میکند. اتوماتانهای یادگیر را می توان به دو گروه اصلی اتوماتان یادگیر با ساختار ثابت و اتوماتان یادگیر با ساختار متغیر تقسیم کرد. اگر احتمال انتقال از یک حالت به حالت دیگر و احتمالهای اقدام و حالت ثابت باشند، اتوماتان با ساختار ثابت، در غیر این صورت اتوماتان با ساختار متغیر نامیده می شود. اتوماتانهای Tsetline، Krinsky، Krylov واتوماتان مهاجرت اشیاء مثالهایی از اتوماتان با ساختار ثابت میباشند.

۳- موازی سازی

عملکرد آموزش شبکه های عصبی را میتوان با اجرای قسمتهای مختلف الگوریتم اموزش بطور موازی بهبود بخشید. آموزش شبکه های عصبی با تنظیم وزنها در طول تعدادی حلقه های متداخل بصورت زیر صورت میگیرد.

For each training example

For each layer

For each node in the layer

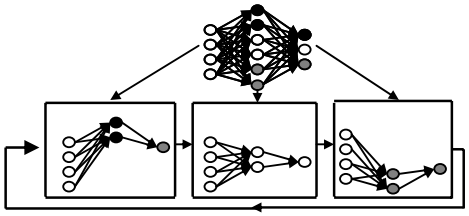
هرکدام از این عملیات را میتوان بصورت موازی، سریال و یا ترکیبی از این دو انجام داد. الگوریتم پس انتشار خطا دارای چندین نقطه توازی مانند توازی نرون و توازی داده های آموزش می باشد. با استفاده از توازی موجود در الگوریتم پس انتشار خطا میتوان به دو روش مختلف افراز داده و افراز نرون الگوریتم پس انتشار خطا را بر روی یک سخت افزار موازی نگاشت کرد. در ادامه روش افراز نرون را توضیح میدهیم.

۳-۱- روش افـــــراز نرون NP[4]

در این حالت نرونهای هر لایه بطور مساوی بین پروسسورها تقسیم میشود. در هر تکرار، خروجی یک لایه خاص محاسبه می شود. تمامی پردازشگرها دارای یک کپی محلی از کل بردار ورودی لایه بوده

[2] Parallel Neuron Survival Algorithm
[3] BackPropagation
[4] Neuron Partioning

و بخشی از بردار خروجی خود را محاسبه میکنند. پردازشگر مدیر بردارهای جزئی را جمع کرده، پس از ایجاد یک بردار کامل یک کپی از آن را بین پردازشگرها توزیع میکند. این بردار بعنوان ورودی برای لایهٔ بعد مورد استفاده قرار می گیرد. مسیر برگشت نیز مشابه مسیر رفت میباشد.



شکل۱: نحوه توزیع نرونهای لایه های مختلف بین پردازشگرها

ملاحظه می کنیم در این روش حجم بالایی از اطلاعات بین پردازشگرها مبادله شده و هزینه ارتباطات بالا میباشد. این مسئله میتواند باعث کاهش Speed Up و در نتیجه افزایش زمان آموزش شبکه عصبی گردد. نحوه توزیع نرونها بین پردازشگرها در شکل ۱ نشان داده شده است.

۳-۲- PC_Cluster و MPI[۵]

باتوجه به افزایش روز افزون قدرت و کاهش قیمت کامپیوتر های شخصی و همچنین رشد و توسعه شبکه های کامپیوتری، در سالهای اخیر تلاش گسترده ای برای ساخت سیستمهای پردازش موازی مبتنی بر کامپیوترهای شخصی صورت گرفته است. این سیستمها بدلیل استفاده از سخت افزارهای آماده و ارزان قیمت، دارای هزینه به کارائی بهتری نسبت به ابر کامپیوترها میباشند[۱۹] در مرکز تحقیقات برق و الکترونیک دانشگاه صنعتی امیر کبیر، سیستم PC_Cluster به منظور دستیابی به سرعت پردازش بالا طراحی و ساخته شده است. این سیستم متشکل از ۳۲ عدد پردازنده بعنوان واحدهای پردازشگر میباشد. ارتباط سیستم با کاربر توسط یک کامپیوتر بعنوان مدیر سیستم برقرار میشود. برنامه های کاربردی نوشته شده به زبانC که با استاندارد MPI تهیه شده باشند بر روی سیستم قابل اجرا میباشد. در این مقاله از این سیستم بعنوان یک ماشین موازی برای شبیه سازی شبکه های عصبی استفاده کرده ایم.

۴- الگوریتم بقا نرون

در این الگوریتم از یک اتوماتان مهاجرت اشیاء برای تعیین تعداد نرونهای لایه مخفی استفاده شده است. وظیفه این اتوماتان تقسیم بندی نرونها به دو گروه مناسب و نامناسب می باشد. این اتوماتان به صورت شش تایی $\langle \underline{\alpha}, \underline{H}, \underline{\Phi}, \underline{\beta}, \underline{F}, \underline{G}, \rangle$ نشان داده میشود. که در آن $\underline{\alpha} = \{\alpha_1, \alpha_2\}$ اقدامهای اتوماتان یادگیر میباشد. این اتوماتان دارای دو خروجی میباشد. خروجی شماره یک خروجی مناسب یا نرونهای روشن میباشد. خروجی شماره ۲ خروجی نامناسب یا نرونهای خاموش میباشد. $\underline{H} = \{H_1, H_2, ..., H_n\}$ واحدهای مخفی در خروجی اتوماتان میباشند. اگر واحد H_i در خروجی یک ظاهر شود این واحد روشن بوده درغیر اینصورت خاموش خواهدبود.

$\underline{\Phi} = \{\Phi_1, \Phi_2, ..., \Phi_{2N}\}$ حالتهای اتوماتان بوده و N عمق حافظه می باشد. حالتهای اتوماتان به دو گروه $\{\Phi_1, \Phi_2, ..., \Phi_N\}$ و $\{\Phi_{N+1}, \Phi_2, ..., \Phi_{2N}\}$ تقسیم می شوند. بر این اساس واحدهای روشن با مجموعه $ON = \{H_i \mid 1 \le State(H_i) \le N\}$ و واحدهای خاموش با مجموعه $ON = \{H_i \mid N+1 \le State(H_i) \le 2N\}$ نشان داده میشوند. $\underline{\beta} = \{0,1\}$ ورودیهای اتوماتان میباشد. در این مجموعه ۱ جریمه و . پاداش را نشان میدهد. $F : \Phi \times \beta \to \Phi$ تابع نگاشت حالتها می باشد. این تابع باتوجه به حالت فعلی و ورودی، حالت بعدی را تعیین میکند. $G : \Phi \to \alpha$ تابع نگاشت خروجی میباشد. اگر واحدی در حالت Φ_1 باشد، آن واحد مناسب‌ترین واحد بوده و بیشترین اهمیت را داراست. اگر واحدی در حالت Φ_N باشد دارای کمترین اهیمت خواهد بود. اگر حالت واحدی به مجموعه $\{\Phi_{N+1}, \Phi_2, ..., \Phi_{2N}\}$ متعلق باشد، آن واحد خاموش خواهد بود. اگر واحد خاموش در وضعیت Φ_{N+1} قرار داشته باشد دارای بیشترین اهمیت بوده و اگر در وضعیت Φ_{2N} باشد دارای کمترین اهمیت میباشد. نحوه عملکرد الگوریتم NSA به این صورت میباشد. در ابتدا تمامی نرونها روشن بوده و در وضعیت Φ_1 قرار داشته و در آموزش شرکت میکنند. واحدهائیکه دارای عملکرد مناسب نیستند جریمه شده و واحدهای با عملکرد مناسب پاداش داده میشوند. واحدهائیکه در مورد آنها نمی توان تصمیم گیری کرد نه جریمه شده و نه پاداش داده میشوند. برای ارزیابی عملکرد یک واحد، از متوسط انرژی استفاده شده توسط آن، استفاده می کنیم. نحوه تغییر فعالیت یک واحد، انرژی مصرف شده واحد نامیده شده و توسط دو قانون زیر بیان میشود.اگر برای تمامی الگوهای ورودی، مقدار فعالیت واحد تغییرات زیادی داشته باشد در اینصورت واحد دارای عملکرد خوب میباشد و اگر برای تمامی الگوهای ورودی مقدار فعالیت واحد دارای تغییرات کمی باشد واحد دارای عملکرد خوب نیست.

تشخیص نحوه عملکرد واحد روشن: اگر فعالیت واحدی برای تمامی الگوها از یک مقدار آستانه کمتر باشد واحد بد و اگر از یک مقدار آستانه بیشتر باشد واحد خوب نامیده میشود. برای تعیین مقادیر آستانه، ابتدا واریانس مقدار فعالیت واحد برای تمامی الگوهای آموزش بصورت زیر محاسبه میشود.

$$\delta_l = \sqrt{\frac{\sum_{k=j}^{P} \left(|U_{lk}| - \mu_l\right)^2}{P}} \qquad l \in ON$$

که در آن، U_{lk}، فعالیت واحد شماره۱ برای الگوی شماره K و P تعداد الگوهای آموزش میباشد. μ_l مقدار متوسط فعالیت واحد

۶) Message Passing Interface

شماره ۱ بوده که بصورت زیر تعریف میشود.

$$\mu_l = \frac{\sum_{k=1}^{p} |U_{lk}|}{P} \qquad l \in ON$$

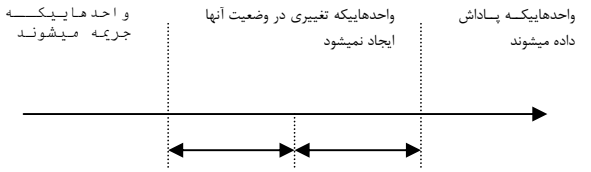
پس از محاسبه واریانس واحدهای روشن همچنانکه در شکل ۲ نشان داده شده است، واحدهای روشنی که واریانس فعالیتهای آن کمتر از یک مقدار آستانه میباشد جریمه شده و واحدهایی که مقدار فعالیت آنها بزرگتر از یک مقدار آستانه دیگر میباشد پاداش می بینند. واحدهای روشنی که واریانس فعالیت آنها بین دو مقدار آستانه قرار می گیرند نه جریمه شده و نه پاداش می بینند. مقدار M_{ON} که مقدار متوسط واریانسهای واحدهای روشن میباشد بصورت زیر محاسبه می شود.

$$M_{ON} = \frac{\sum_{k \in ON} \delta_k}{|ON|}$$

پهنای X_{ON} بصورت زیر محاسبه می شود.

$$X_{ON} = \lambda_{ON} \frac{|ON| + |OFF|}{|ON|} \times \frac{Max(\delta_{ON})}{Min(\delta_{ON})}$$

در معادله بالا ثابت λ_{ON} ضریب پهنای روشنی نامیده میشود. مقدار آستانه پائین $M_{ON} - X_{ON}$ و مقدار آستانه بالا $M_{ON} + X_{ON}$ میباشد.



$$M_{ON} - X_{ON} \quad M_{ON} \quad M_{ON} + X_{ON}$$

شکل ۱: مقادیر استانه ای واحدهای روشن

نحوه تمایز بین واحدهای خاموش: واحدهای خاموش در آموزش شبکه شرکت نمی کنند. در این حالت ما از گذشته این واحدها استفاده می کنیم. فعالیت یک واحد خاموش برای یک الگو براساس آخرین مقدار فعالیت این واحد در زمان روشن بودن برای آن الگو محاسبه می شود. اگر یک واحد برای مدت زمان زیادی خاموش باشد ارزش فعالیت واحد کاهش می یابد. بنابراین فعالیت واحد بصورت زیر حساب میشود.

$$U_{lk}(n+1) = e^{-\lambda_d |U_{lk}(n)|}$$

در معادله بالا ثابت λ_d، ضریب کاهش فعالیت نامیده شده و n زمان را نشان میدهد. بنابراین مقدار فعالیت یک واحد خاموش بتدریج کاهش می یابد. واریانس واحدهای خاموش بصورت زیر محاسبه می شود.

$$\sigma_l = \sqrt{\frac{\sum (|U_{lk} - \mu_l|)^2}{P}} \qquad l \in OFF$$

که U_{lk} مقدار فعالیت واحد شماره ۱ برای الگوی شماره K بوده و μ_l مقدار متوسط فعالیت واحد خاموش است که بصورت زیر بیان می شود.

$$\mu_l = \frac{\sum_{k=l}^{P} |U_{lk}|}{P} \qquad l \in OFF$$

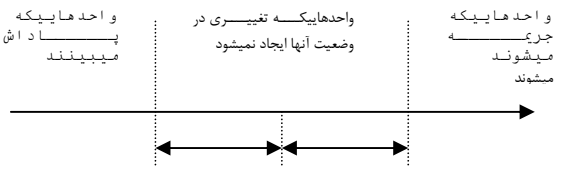
پس از محاسبه واریانس واحدهای خاموش، واحدهائیکه واریانس فعالیت آنها از یک مقدار آستانه کمتر است پاداش دیده و واحدهائیکه واریانس فعالیت آنها بیشتر از یک مقدار آستانه دیگر می باشد جریمه می شوند. واحدهائیکه واریانس فعالیت آنها بین این دو مقدار آستانه میباشد نه جریمه شده و نه پاداش داده میشوند. مقدار M_{OFF} که مقدار متوسط واریانس واحدهای خاموش میباشد بصورت زیر محاسبه میشود.

$$M_{OFF} = \frac{\sum_{k \in OFF} \delta_k}{|OFF|}$$

پهنای X_{OFF} بصورت زیر محاسبه میشود.

$$X_{OFF} = \lambda_{OFF} \frac{|OFF| + |ON|}{|OFF|} \times \frac{Max(\delta_{OFF})}{Min(\delta_{OFF})}$$

در معادله بالا ثابت λ_{OFF} ضریب پهنای خاموشی نامیده میشود. مقدار آستانه پائین $M_{OFF} - X_{OFF}$ و مقدار آستانه بالا $M_{OFF} + X_{OFF}$ میباشد.



$$M_{OFF} - X_{OFF} \quad M_{OFF} \quad M_{OFF} + X_{OFF}$$

شکل ۳: مقادیر آستانه ای واحدهای خاموش

۵- الگوریتم بقا نرون موازی PNSA

الگوریتم NSA یک الگوریتم سریال میباشد. در این بخش بمنظور کاهش پیچیدگی ارتباطی در آموزش شبکه ها با استفاده از PC_Cluster، یک الگوریتم موازی تحت عنوان الگوریتم بقا نرون موازی(PNSA)ارایه میگردد. عملکرد الگوریتم پیشنهادی به شرح زیرمیباشد.

ابتدا پردازشگر مدیر با اجرای تابع ScatterNetAmongProcesses، ماتریسهای وزن و بردارهای بایاس لایه های مختلف را بین پردازشگرها تقسیم میکند.

سپس با اجرای تابع SendPatternsToAll() یک نسخه ازکل الگوهای آموزش را برای تمامی پردازشگرها ارسال میکند. در مرحله بعد مسیر رفت الگوریتم BP، بطور موازی توسط پردازشگرها اجرا شده و هر پردازشگر خطای جزیی مربوط به خود را محاسبه کرده و آنرا برای پردازشگر مدیر ارسال میکند.پردازشگرمدیر خطاهای جزیی را دریافت کرده خطای کل را محاسبه میکند. اگر خطای کل از یک مقدار آستانه کمتر بود دستور اتمام را برای پردازشگرها ارسال کرده و برنامه متوقف میشود در غیر اینصورت الگوریتم BP بطور موازی M بار(M پریود ارزیابی نرونها میباشد) توسط پردازشگرها اجرا میشود. پس از این مرحله هر کدام از پردازشگرها با اجرای توابعON_Node_ActivityTest وOFF_Node_ActivityTest نحوه عملکرد نرونهای روشن و خاموش خود را مطابق الگوریتم بقا ارزیابی کرده و بر اساس آن حالت نرونها را تغییر میدهند. علاوه بر این تابع ON_Node_ActivityTest تعداد نرونهایی راکه باید حذف شوند را مشخص میکند. اگر نیازی به اضافه کردن نرونهای جدید یا حذف نرونهای بی اهمیت باشد در اینصورت پردازشگر مدیر ماتریسهای وزن و بردارهای بایاس لایه اول را از پردازشگرها جمع آوری کرده پس از اضافه کردن نرونهای مورد نیاز و یا حذف نرونهای بی اهمیت، شبکه را مجددا بین پردازشگرها توزیع میکند. این روند تا رسیدن به خطای هدف ادامه مییابد. شکل ۵ الگوریتم PNSA را که به زبان MPI نوشته شده نشان میدهد.

```
For all PE_cnt where 0 ≤ PE_cnt ≤ No_processes
  If ( process_rank = 0 ) // Master processor Does the
  following operations
    ScatterNetAmongProcesses (w,b)
    SendPatternsToAll ()
    For epoch_cnt ← 0 to epoch_requested do
       Totalerror ← AccumulateErrorFromAll ()
       If ( Totalerror ≤ eg ) // eg is error goal
         SendToSlavesFinishMessage (Yes)
         Break
       Else
         SendToSlavesFinishMessage (No)
         If ( epoch_cnt / evaluation_period = k; k  N)
           Act_ON ← ON_Node_ActivityTest() //returns 0
           if there is no need to prune in hidden layers.
           Act_OFF ← OFF_Node_ActivityTest() //returns
           0 if there is no need to add any neuron in hidden
           layers.
           If (Act_OFF ≠ 0)
               SendCommandToAllSlaves (Construct)
               GatherFirstLayerFromAllSlaves()
               AddApropriateNo.OfNeursToFirstLayer()
               ScatterFirstLayerAmongSlaves()
           Else
               If (Act_ON ≠ 0)
               SendCommandToAllSlaves(Do pruning)
               GatherFirstLayerFromAllSlaves ()
               PrunLayer ()
               ScatterFirstLayerAmongSlaves ()
           Else
                SendCommandToAllSlaves (Go on)
           Endif
         Endif
       Endif
       Previouserror ← Totalerror
    Endif
  Endfor
( w,b ) ← GatherNetFromAllSlaves ()
```

```
Else // Slave processors do the following operations
( w_i , b_i ) ← ReceiveNet ()
ReceivePatterns ()
For epoch_cnt ← 0 to epoch_requested do
 {              // ForwardNet ()
   Output_i [layer0] ← LayerFun.(Activ.fun.( w_i [layer0]
   *InputPattn+ b_i [layer 0]))
   For i ← 1 to No.OfLayers do
    Output[layeri-1] ← AllToAllBrodcastAmngSlaves
    ( Output_i [layer i-1] )
    Output_i [layer      i] ← LayerFun.(Activationfun.( w_i
   [layer i] * Output[layer i-1] + b_i [layer i]) )
   Endfor
}
 CalculateError ()
 SendErrorToMaster ()
 Finish ← ReceiveFinishMessage ()
 If ( Finish = Yes )
   Break
 Endif
 If (epoch_cnt / evaluation_period = k; k  N)
   ON_Node_ActivityTest ()
   OFF_Node_ActivityTest ()
   Command ← ReceiveCommandFromMaster ()
   If (Command = Construct)
    SendFirstLayerToMaster()
    ReceiveFirstLayerFromMaster()
   Else
    If (Command = Do pruning routine)
     SendFirstLayerToMaster ()
     ReceiveFirstLayerFromMaster ()
    Endif
 Endif
 For block ← 0 to No_Block do
   {                    // ForwardNet ()
     Output_i [layer0] ← LayerFun.(Activationfun.
   ( w_i [layer 0] * InputPattern + b_i [layer 0]) )
      For i ← 1 to NumberOfLayers do
        Output[layeri-
        1] ← AllToAllBrdcastAmngSlaves( Output_i
        [layer i-1] )
        Output_i [layeri] ← LayerFun.(Activationfun.(
        w_i [layer i] * Output[layer i-1] + b_i [layer i]) )
        Endfor
    }
    {     // BackwardNet ()
      E_i ← CalculateError ()
      Delta_i [No.OfLayers-
      1] ← F• (Layer[No.OfLayer -1]).* E_i //Delta
       of output layer
       For I ← NumberOfLayer-2 to 0 do
        Delta_i [layer i] ← F• (Layer[layer i]).*(
        Trnspose( w_i [layer i+1])* Delta_i [layer i+1] )
        If( process_rank = 1)
          WholeDelta[layer
          i] ← Gather&SumDeltaFromAll
          Slaves ()
          ScatterDeltaAmongAllSlaves
          (WholeDelta[layer i])
         Else
          SendToFirstSlave ( Delta_i [layer i])
          Delta_i [layeri] ← ReceivFromFirstSlave ()
    Endif
```

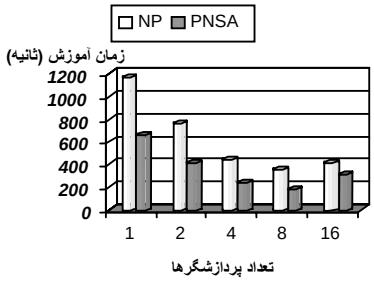
```
        Endfor
        Δ w_i [layer0] ← alfa*(1-m)* Delta_i [layer
        0]*Transpose(InputPattern) + m*Δ w_i [layer 0]
        w_i [layer 0] ← w_i [layer 0]+Δ w_i [layer 0]
        For i ← 1to NumberOfLayers do
          Δ w_i [layeri] ← alfa*(1-m)* Delta_i [layer
          i]*Transpose(Output[layeri-
          1])+m*Δ w_i [layer i]
          w_i [layer i] ← w_i [layer i]+Δ w_i [layer i]
        Endfor
          Δ b_i [layer0] ← alfa*(1- m)* Delta_i [layer
          0]*Transpose(1L) + m*Δ b_i [layer 0]
          b_i [layer0] ← b_i [layer 0]+Δ b_i [layer 0]
          For i ← 1 to NumberOfLayers do
            Δ b_i [layer i] ← alfa*(1-m)* Delta_i
            [layer i]*Transpose(1L) + m*Δ b_i [layer i]
            b_i [layer i] ← b_i [layer i]+ Δ b_i [layer i]
          Endfor
        }
      Endfor
    Endfor
  SendToMaster ( w_i , b_i )
 Endif
Endfor
```

شکل ۵: الگوریتم PNSA

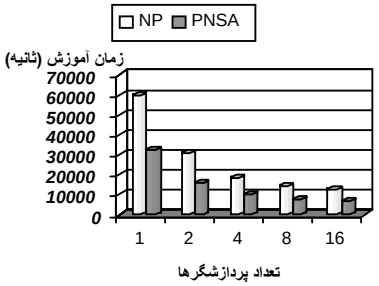
۶- نتایج شبیه سازیها

به منظور مقایسه عملکرد الگوریتم PNSA، با الگوریتم NP آزمایشهایی بر روی کاربرد باز شناسی واجهای فارسی صورت گرفته است. در آزمایشهای مختلف از دو نوع شبکه با ابعاد مختلف و تعداد الگوهای آموزش متفاوت استفاده شده است.

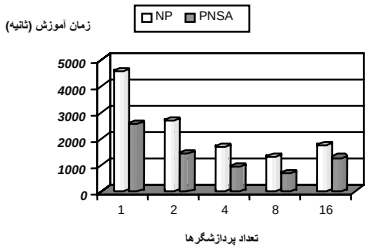
مشخصات شبکه های مورد استفاده:در آزمایش اول از یک شبکه عصبی کوچک سه لایه با ۷۵ نرون در لایه ورودی و ۶ نرون در لایه خروجی استفاده کرده ایم. آزمایش برای چهار شبکه مختلف با تعداد نرونهای مخفی مختلف تکرار شده است. شبکه اول دارای ۱۶ نرون مخفی، شبکه دوم ۳۲ نرون مخفی، شبکه سوم ۶۴ و بالاخره شبکه چهارم ۱۲۸ نرون مخفی میباشد. تعداد کل الگوهای آموزش، ۳۲۷۷۶ الگو بوده و برای آموزش شبکه کوچک از روش خوشه ای استفاده شده و تعداد بلوکهای داده ها ۲۰۰ میباشد. شبیه سازی بعدی بر روی یک شبکه عصبی بزرگ سه لایه با ۷۵ نرون در لایه ورودی و ۳۲ نرون در لایه مخفی صورت گرفته است. آزمایش برای دو شبکه مختلف با ۶۴ و ۱۲۸ نرون مخفی تکرار شده است. تعداد الگوهای آموزشی در مورد شبکه بزرگ ۹۶۶۲۴ الگو میباشد. از روش آموزش دسته ای استفاده کرده و تعداد epochs نیز ۱۰۰ میباشد. در تمامی شبیه سازیها، تابع سیگوئید غیر خطی بعنوان توابع فعالیت لایه خروجی و مخفی مورد استفاده قرار گرفته است. نرخ یادگیری ۰/۰۱ و ضریب ممنتم ۰/۹۸ در نظر گرفته شده است.



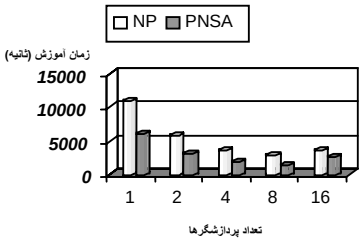
شکل ۶: مقایسه زمان آموزش برای شبکه کوچک با ابعاد ۶×۱۶ ×۷۵



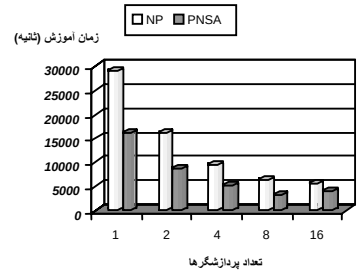
شکل ۷: مقایسه زمان آموزش برای شبکه کوچک با ابعاد ۶×۳۲×۷۵



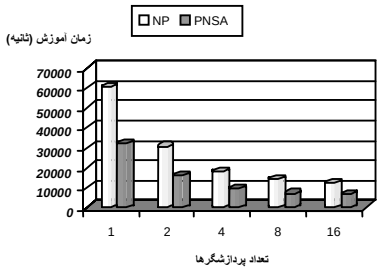
شکل ۸: مقایسه زمان آموزش برای شبکه کوچک با ابعاد ۶×۶۴×۷۵



شکل ۹: مقایسه زمان آموزش برای شبکه کوچک با ابعاد ۶×۱۲۸×۷۵



شکل ۱۰: مقایسه زمان آموزش برای شبکه بزرگ با ابعاد ۳۲×۶۴ ×۷۵



شکل ۱۱: مقایسه زمان آموزش برای شبکه بزرگ با ابعاد ۳۲×۱۲۸ ×۷۵

شرایط شبیه سازی برای الگوریتم NP: از روش افراز نرون برای نگاشت شبکه عصبی بر روی PC_Cluster استفاده کرده ایم.

شرایط شبیه سازی برای الگوریتم PNSA : پرید ارزیابی ۱۰، ضریب پهنای روشنی ۵ ، ضریب پهنای خاموشی ۱۵ و عمق حافظه اتوماتان ۴ انتخاب شده است. شکلهای ۶ الی ۱۱ زمان آموزش را بر حسب ثانیه نشان میدهند. با توجه به نمودارهای شکلهای ۶ الی ۱۱ میتوان نتیجه گیری زیر را بعمل آورد.

نتیجه: الگوریتم PNSA، برای هر دو شبکه کوچک و بزرگ و به ازای تعداد مختلف نرونهای لایه مخفی، دارای زمان آموزش کمتری نسبت به الگوریتم NP می باشد. در الگوریتم پیشنهادی بدلیل اینکه پردازشگرها در حین فرایند آموزش عملکرد نرونهای خود را ارزیابی کرده و نرونهای با اهمیت کم را خاموش میکنند. این امر سبب کاهش پیچیدگی شبکه و در نتیجه کاهش بار محاسباتی هر یک از پردازشگرها می گردد. علاوه بر این با توجه به اینکه در طول آموزش مقادیر خروجی نرونهای پردازشگرها باهم مبادله میگردد. لذا هر گونه کاهش در تعداد نرونهای پردازشگرها می تواند بار ارتباطی بین پردازشگرها نیز کاهش دهد. در مجموع با کاهش تعداد نرونهای پردازشگرها بار محاسباتی و ارتباطی کاهش یافته در نتیجه باعث کاهش زمان آموزش میگردد.

۷- نتیجه‌گیری

در این مقاله یک الگوریتم یادگیری ساختار موازی تحت عنوان PNSA، ارائه گردید. از سیستم کامپیوتر موازی PC_Cluster بعنوان یک ماشین موازی برای شبیه سازی شبکه های عصبی استفاده کرده و با استفاده از استاندارد MPI برنامه هایی به زبان C برای الگوریتمهای NP و الگوریتم پیشنهادی PNSA نوشته و بر روی این کامپیوتر موازی اجرا کردیم. نتایج شبیه سازیها بر روی کاربرد بازشناسی واجهای فارسی نشان داد الگوریتم PNSA با کاهش دادن پیچیدگی شبکه، هزینه ارتباطی بین پردازشگر ها را کاهش داده در نتیجه زمان آموزش را تقلیل میدهد. لذا الگوریتم PNSA دارای سرعت بیشتری نسبت به الگوریتم NP میباشد.

سپاسگزاری

از مرکز تحقیقات برق و الکترونیک دانشگاه صنعتی امیر کبیر به خصوص آقای مهندس آرش جلال زاده که در استفاده از سیستم PC_Cluster ما را یاری دادند همچنین از آقای دکتر سید محمد احدی که دادگان واجهای فارسی را در اختیار ما قرار دادند، متشکریم.

مراجع

[1] Shou King Foo, P. Saratchandran and N. Sundararajan, " Parallel implementation of backpropagation neural networks on a heterogenous array of transputers,"IEEE Transactions on Systems, Man, and Cybernetics,pp. 118-126, 1997.

[2] B. Girau and H.Paugam-Moisy, "Load sharing in the Training Set Partition Algorithm for Parallel Neural Learning,"in IEEE Parallel Processing Symposium, pp. 586-591, 1995.

[3] Youngsik Kim, Mi - Jung Noh, Tack - Don Han, Shin - Dug Kim, Sung-Bong Yang, "Memory-based Processor Array for Artificial Neural Networks,"in IEEE Inter. Conf. On Neural Networks, pp. 969-974, 1997.

[4] Sherif Kassem Fathy and Mostafa Mahmoud Syiam, "A Parallel Design and Implementation For Backpropagation Neural Network Using MIMD Architecture, "in IEEE Inter. Conf. On Neural Networks, "Vol. 2, pp. 1361-1366, 1996.

[5] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks ," IEEE Trans. Neural Networks, vol. 1, n o. 2, pp. 239-242, 1990.

[6] Steven Young and Tom Downs, "CARVE-A Constructive Algorithms for Real-Valued Examples," IEEE Trans. Neural Networks Networks, vol. 9, no. 6, pp. 1180-1190, NOV. 1998.

[7] Nabhan T. M. and Zomaya A. Y., "Toward Neural Networks Structure for Function Approximation," Neural Networks, vol. 7, no.1, pp. 89-99, 1993.

[8] J. D. Schaffer, D. Whitely and L. J. Eshelman, "Combination of genetic algorithms and neural networks: A Survey of the state of the art," IEEE Proc. COGANN-92, pp. 1-37, 1992.

[9] B. Mashoufi, M. R. Meybodi, S. A. Motamedi and M. B. Menhaj, "Neural networks Engineering Using Learning Automata: Introducing an Adaptive Algorithm for determining number of hidden Layer neurons for three layer neural networks", Proc. Of ICEE- 2001, The 9th – Iranian Conference on Electrical Engineering, pp. 27:1 27:14, Power and Water Institute of Technology, Tehran, Iran, 2001.

[10] B. Mashoufi, M. R. Meybodi, S. A. Motamedi and M. B. Menhaj, "Introducing New Learning Automata Based Algorithms for Determining Number of Input Weights of Hidden Neurons for Three Layer Neural Networks", Amirkabir Journal of Science and Technology, pp. 286-319.

[11] B. Mashoufi, M. R. Meybodi, S. A. Motamedi and M. B. Menhaj, "Adaptive Survival Algorithm", IUST-International Journal of Engineering Science, Iran University of Science and Technology, Vol. 15, No. 3, 2004, pp. 1-16.

[12] Beigy. H and Meybodi, M. R. "A Learning Automata based algorithm for determination of optimal number of hidden units in three layers feedforward neural networks" Journal of Amirkabir, Tehran, Iran, to appear.

[13] Meybodi, M. R. and Beigy. H. (1999), "Optimization of Neural Networks Using Learning Automata," Proc. Of 4th Annual Int. Computer Society of Iran Computer Conf. CSICC-98, Tehran, Iran, pp. 417-428, Iran (In Persian).

[14] Meybodi, M. R. and Beigy. H" Neural Networks Engineering Using Learning Automata: Determination of desired size for three layer feedforward Neural Networks" Technical Reports, Computer Eng. Dept. Amirkabir University of Technology, Tehran, Iran, 1999.

[15] B. Mashoufi, M. B. Menhaj, S. A. Motamedi and M. R. Meybodi, "Introducing a novel learning automata based method for adapting VLR learning algorithm parameters for learning MLP neural networks", Amirkabir journal of science and technology Vol. 13, No. 51, pp. 398-412, summer 2002.

[16] B. Mashoufi, M. B. Menhaj, S. A. Motamedi and M. R. Meybodi, "Introducing an Adaptive VLRBP Algorithm Using Learning Automata for Multilayer Neural Network", IEICE Transactions on Information and Systems, Vol. E86-D, No. 3, March 2003.

[17] Simon Haykin, Neural Networks a comprehensive Foundation, McMaster University, Canada, 1994

[18] K. S. Narendra and M. A. L. Thathachar, Learning Automata: An Introduction, Prentice Hall, Englewood cliffs, 1989.

[19] M.S. Warren, D.J. Becker, M.P. Goda, J.K. Salmon, and T. Sterling." Parallel Supercomputing with Commodity Components."In H.R. Arabnia, editor, Proceedings of the International conference on parallel Distributed Proccesing Techniques and Applications (PDPTA'97), pp. 1372-1381, 1997.