موضوع پروژه : کشف الگوریتم ضرب ماتریس سریعتر با استفاده از بهینه سازی

در مقاله ی آلفاتنسور ، نویسندگان این مقاله الگوریتمی را برای کشف الگوریتم های ضرب ماتریس سریعتر با استفاده از روش deep reinforcement learning ارائه دادند.

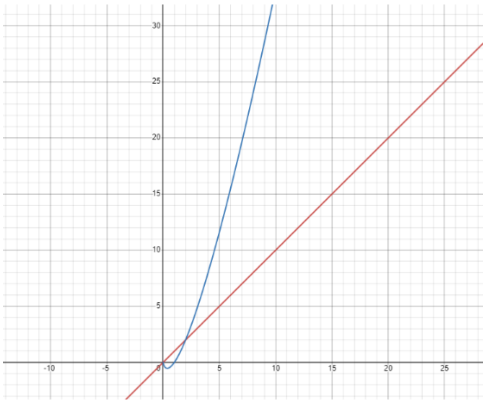
در ابتدا باید منظور خود را از الگوریتم ضرب ماتریس سریعتر بیان کنیم. در این مقاله به طور تئوری فرض شده است که الگوریتمی که تعداد ضرب اسکالر کمتری داشته باشد (حتی اگر تعداد جمع بیشتری داشته باشد) ، الگوریتم سریعتری است. (به علت پیچیدگی زمانی بالای ضرب نسبت به جمع در سطح سخت افزار)

علت :

$N : number\ of\ digits$

$Addition\ of\ two\ N\ digit\ numbers:\ O(N)\ (if\ parallel\ \rightarrow\ O(1))$

$Multiplication\ of\ two\ N\ digit\ numbers:\ O(N \times \log(N))$



توضیح بیشتر در این دو لینک :

https://iq.opengenus.org/time-complexity-of-addition/

https://iq.opengenus.org/time-complexity-of-multiplication/

اکنون فرض کنید می خواهیم یک ماتریس ۲ در ۲ را در یک ماتریس ۲ در ۲ دیگر ضرب کنیم. الگوریتم بدیهی و ساده ای که به ذهنمان می رسد به شکل زیر است :

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix}$$

$$c_1 = a_1 \times b_1 + a_2 \times b_3 \qquad c_2 = a_1 \times b_2 + a_2 \times b_4$$

$$c_3 = a_3 \times b_1 + a_4 \times b_3 \qquad c_4 = a_3 \times b_2 + a_4 \times b_4$$

اما والکر استراسن در سال ۱۹۶۹ نشان داد که این الگوریتم سریعترین الگوریتم ممکن برای ضرب یک ماتریس ۲ در ۲ در یک ماتریس ۲ در ۲ نیست و الگوریتم زیر را که فقط ۷ ضرب اسکالر دارد، ارائه داد :

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix}$$

$$\begin{aligned}
m_1 &= (a_1 + a_4) \times (b_1 + b_4) \\
m_2 &= (a_3 + a_4) \times (b_1) \\
m_3 &= (a_1) \times (b_2 - b_4) \\
m_4 &= (a_4) \times (b_3 - b_1) \\
m_5 &= (a_1 + a_2) \times (b_4) \\
m_6 &= (a_3 - a_1) \times (b_1 + b_2) \\
m_7 &= (a_2 - a_4) \times (b_3 + b_4)
\end{aligned}
\longrightarrow
\begin{aligned}
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 = a_1 b_2 - a_1 b_4 + a_1 b_4 + a_2 b_4 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}$$

همانطور که می بینید در این الگوریتم ابتدا ۷ متغیر کمکی تعریف می شود که هر کدام ضرب دو پرانتز هستند که پرانتز های سمت چپ درایه های ماتریس اول و پرانتز های سمت راست درایه های ماتریس دوم هستند.

در نهایت برای درایه های ماتریس خروجی این متغیر های کمکی را به شکلی که در بالا می بینید جمع و منها کرده و اگر متغیر های کمکی را باز کنیم خواهیم دید که به درستی جواب خروجی این الگوریتم با جواب خروجی درست یا همان الگوریتم ساده و بدیهی برابر است.

مقاله ی آلفاتنسور در ابتدا این ایده ی استراسن را برای ضرب ماتریس های با ابعاد دلخواه گسترش می دهد و آن را با ماتریس ها مدل سازی می کند. در ادامه مدل سازی آنها شرح داده می شود :

در این روش سه ماتریس U و V و W را در نظر بگیرید :

$$
\begin{aligned}
m_1 &= (a_1 + a_4)(b_1 + b_4) \\
m_2 &= (a_3 + a_4)\, b_1 \\
m_3 &= a_1\, (b_2 - b_4) \\
m_4 &= a_4\, (b_3 - b_1) \\
m_5 &= (a_1 + a_2)\, b_4 \\
m_6 &= (a_3 - a_1)(b_1 + b_2) \\
m_7 &= (a_2 - a_4)(b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}
$$

$$
\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}
$$

$$
\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}
$$

$$
\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}
$$

در واقع فرض کنید هر کدام از درایه های ماتریس های U و V و W برای ما مجهول هستند و ما می خواهیم آنها را پیدا کنیم. الگوریتم استراسن طبق این مدل سازی، ماتریس های U و V و W اش به شکل فوق پر می شود. هر ستون از U ، نمایانگر ضرایب پشت درایه های مربوط به ماتریس اول در متغیر کمکی متناظر با آن ستون هستند. برای مثال ستون دوم U می گوید که برای متغیر کمی دوم (m2) ، از ماتریس اول، دو درایه ی اول را ضربدر صفر کن و دو درایه ی سوم و چهارم را ضربدر یک کن و با یکدیگر جمع کن.

همچنین هر ستون از V ، نمایانگر ضرایب پشت درایه های مربوط به ماتریس دوم در متغیر کمکی متناظر با آن ستون هستند.

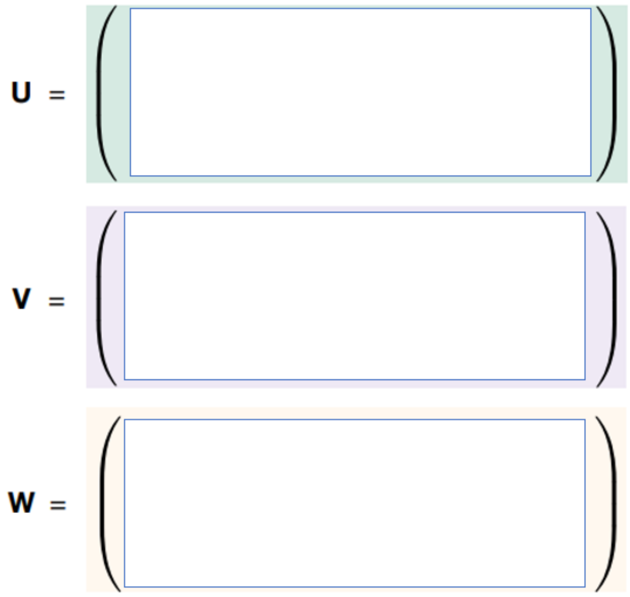
هر سطر از ماتریس W ، نمایانگر ضرایب پشت متغیر های کمکی متناظر با درایه ی خروجی آن سطر هستند.

برای مثال سطر دوم ماتریس W می گوید برای درایه ی خروجی دوم ، فقط متغیر کمکی سوم و پنجم را با هم جمع کن.

نویسندگان مقاله ی آلفاتنسور پس از ارائه ی این مدل سازی، آن را به صورت یک بازی در نظر گرفتند که در آن عامل هوش مصنوعی به دنبال پیدا کردن ستون های u و v و w ای است که با کمترین تعداد متغیر کمکی ، به الگوریتمی درست برسد. آنها با استفاده از deep reinforcement learning این کار را انجام دادند.

اما پروژه ای که بنده در نظر گرفتم این است که ما با استفاده از مدل سازی ای که در بهینه سازی خواندیم، پیدا کردن این ماتریس U و V و W را با حل چند معادله چند مجهول انجام دهیم.

فرض می کنیم الگوریتمی با ۷ ضرب اسکالر برای ضرب ماتریس ۲در۲ در یک ماتریس ۲در۲ وجود دارد.



همانطور که در شکل فوق می بینید ما 4*7 + 4*7 + 4*7 = 84 مجهول داریم.

از طرف دیگر 4*4*4=64 معادله نیز داریم.

هر معادله بیانگر این است که ضریب پشت ai*bi برای یک درایه ی خروجی برابر با ضریب پشت آن در الگوریتم درست باشد.

در صفحه ی بعد یک معادله از ۶۴ معادله شرح داده می شود:

$$
\begin{aligned}
m_1 &= (a_1 + a_4)(b_1 + b_4)\\
m_2 &= (a_3 + a_4)\, b_1\\
m_3 &= a_1\,(b_2 - b_4)\\
m_4 &= a_4\,(b_3 - b_1)\\
m_5 &= (a_1 + a_2)\, b_4\\
m_6 &= (a_3 - a_1)(b_1 + b_2)\\
m_7 &= (a_2 - a_4)(b_3 + b_4)\\
c_1 &= m_1 + m_4 - m_5 + m_7\\
c_2 &= m_3 + m_5\\
c_3 &= m_2 + m_4\\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}
$$

$$
\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & 0 & 0 & 0 & 1 & 0\\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}
$$

$$
\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 1\\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}
$$

$$
\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1\\ 0 & 0 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}
$$

می دانیم که c1=a1b1+a2b3 . بنابراین ضریب پشت a1b1 و a2b3 باید یک باشد و تمام جفت انتخاب های دیگر باید ضریب صفر داشته باشند.

اکنون برای مثال ضریب پشت a1b1 را چک میکنیم. طبق شکل فوق می دانیم که ضریب پشت a1b1 به دست آمده از الگوریتم ما، برای درایه ی خروجی c1 برابر است با :

$$U_{11}V_{11}W_{11} + U_{12}V_{12}W_{12} + U_{13}V_{13}W_{13} + U_{13}V_{13}W_{13} + U_{14}V_{14}W_{14} + U_{15}V_{15}W_{15} + U_{16}V_{16}W_{16} + U_{17}V_{17}W_{17}$$

و الگوریتم ما وقتی درست است که این مقدار برابر با یک شود بنابراین یکی از معادله های ما این است :

$$U_{11}V_{11}W_{11} + U_{12}V_{12}W_{12} + U_{13}V_{13}W_{13} + U_{13}V_{13}W_{13} + U_{14}V_{14}W_{14} + U_{15}V_{15}W_{15} + U_{16}V_{16}W_{16} + U_{17}V_{17}W_{17} = 1$$

درنهایت مدل سازی مسئله ی بهینه سازی ما به شکل زیر خواهد شد :

$$\min z = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\left(\sum_{t=1}^{7} U_{it}V_{jt}W_{kt} - q^*\right)^2$$

$$s.t.$$

$$-1 \le U_{ij} \le 1 \qquad \forall\, i \in \{1,2,3,4\}, j \in \{1,2,3,4,5,6,7\}$$

$$-1 \le V_{ij} \le 1 \qquad \forall\, i \in \{1,2,3,4\}, j \in \{1,2,3,4,5,6,7\}$$

$$-1 \le W_{ij} \le 1 \qquad \forall\, i \in \{1,2,3,4\}, j \in \{1,2,3,4,5,6,7\}$$

* : q یک مقدار ثابت است که اگر i,j,k برابر با یکی از ۸ حالت (1,1,1 – 2,3,1 – 1,2,2 – 2,4,2 – 3,1,3 – 4,3,3 – 3,2,4 – 4,4,4) باشند مقدار آن یک و در غیر این صورت مقدار آن صفر می باشد.

پیاده سازی این پروژه در فایل discovering_faster_matrix_multiplication_with_optimization.ipynb می باشد. (ترجیحاً در گوگل کولب اجرا کنید و حتماً فایل ipopt-linux64.zip در کنار کد باشد.)

منابع :

Fawzi, A., Balog, M., Huang, A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 610, 47–53 (2022). https://doi.org/10.1038/s41586-022-05172-4

DeepMind’s blog post: https://www.deepmind.com/blog/discovering-novel-algorithms-with-alphatensor

Yannic Kilcher explanation on Youtube: https://youtu.be/3N3Bl5AA5QU