Deep-learned asymptotically optimal matrix product using approximate bilinear algorithms

Shrohan Mohapatra
Department of Physics,
University of Massachusetts, Amherst

August 2, 2020

Abstract

This article introduces a novel neural network framework for the approximate bilinear algorithm that scales asymptotically with the input size. The network actually learns from the deviations from the expected form of the identity and in the progress of learning the matrix multiplication exponent gradually decreases via a probabilistic trail. The probability density function associated with the learning algorithm decides the time the algorithm takes to converge and arrive at an appreciable form of the identity. Consequently, one can also accelerate commercially accepted context-free parsing by the reduction to Boolean matrix multiplication and then using the proposed scheme.

1 Introduction

Artificial neural networks have attracted research in terms of their use in both supervised and unsupervised learning algorithms that cater to applications such as function approximation [1, 2, 3], pattern and sequence recognition [4, 5], and several other commercial ones [6, 7, 8, 9]. There has been some recent attempts to use novel neural infrastructures in parsing [10, 14] and natural language processing [11, 12, 13] which organise context-free parsers and syntax trees in a mathematically reminiscent way of recurrent neural networks with an attempt to increase performance. Traditional generalised context-free parsing methodologies such as Earley parser, CYK algorithm etc. [15] that run in cubic running time naturally invited some intuitions into reduction of parsing to problem of Boolean matrix multiplication [34, 36] that naturally accelerates the complexity to the order of the input size exponentiated to the *matrix multiplication exponent*. There has been an evolution of attempts by algebraic complexity theorists to bring down the exponent to the conjecture [16, 17, 18, 19, 20, 21, 22, 23, 24] that makes matrix multiplication as fast as the input size. The mathematical base behind the algorithm design of matrix multiplication is explained in section 2. With all the above, there comes a natural question of whether one can use

neural networks to reduce the matrix multiplication exponent, which in turn would optimise linked processes in parsing and natural processing. This is just a well-framed example of how fitting the notion of universal approximation using neural nets into the approximate bilinear matrix multiplication algorithm would have appreciable consequences into related theoretical ventures as well. This article presents a rather naïve beginning of a paradigm of using unsupervised learning to automate the optimisation of the matrix multiplication, instead of just raising the tensors already known to be useful to higher exponents leading to larger complexities in just solving the problem itself [19, 23, 24]. The rest of the article is organised as follows: section 2 discusses the formal background to some of the essential definitions borrowed from algebraic complexity theory, and also sets up a convenient graph-theoretic notion of neural networks; section 3 explains the neural network and the learning algorithm adopted to greedily converge to the fastest possible optimality in a probabilistic fashion and section 4 presents the proof of concept of its correctness and convergence. Section 5 concludes the article.

2 Background

We begin with the basic definition of the matrix multiplication: let $\mathbb{F}$ be the field, and let $A = A_{ij} \in \mathbb{F}^{n \times n}$ and $B = B_{ij} \in \mathbb{F}^{n \times n}$ be two $n \times n$ matrices. The matrix product $C = AB = (AB)_{ij} = C_{ij} \in \mathbb{F}^{n \times n}$ is defined as

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

If we use this definition itself as a method to compute the matrix product, for an $n \times n$ matrix, we need n multiplications and $(n-1)$ additions for each of the n^2 elements of the answer. This way, one needs $n^2 \cdot n = n^3$ multiplications and $n^2 \cdot (n-1) = n^3 - n^2$ additions. An equivalent recursive definition of the matrix product is given in the following way where the matrices are broken into 4 2×2 matrices.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{12}B_{22} + A_{11}B_{12} \\ A_{21}B_{11} + A_{22}B_{21} & A_{22}B_{22} + A_{21}B_{12} \end{pmatrix}$$

where A_{ij}, B_{ij}, $1 \leq i, j \leq 2$ are $\frac{n}{2} \times \frac{n}{2}$ matrices. Let $M_R(n)$ be the number of multiplications required in this recursive definition and $P_R(n)$ the number of additions required to add two $n \times n$ matrices. Consequently, we arrive at the recurrence relations

$$M_R(n) = 8M_R\left(\frac{n}{2}\right) + k_M, P_R(n) = 4P_R\left(\frac{n}{2}\right) + k_P$$

where k_M and k_P are the constants that represent the time taken for multiplication and addition respectively by the implementation of the algorithm. Solving for the recurrences yields

$$M_R(n) = k_M\left(\frac{6n^3-1}{7}\right), P_R(n) = k_P\left(\frac{2n^2-1}{3}\right)$$

In totality, even using the master theorem for such divide-and-conquer recurrence relations, the overall complexity of this "naïve schoolbook" algorithm is $O(n^3)$. Strassen [16] showed that one can modify the divide-and-conquer mechanism; he introduced the following seven matrices

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

The elements of the matrix product are retrieved from the above seven matrices as follows.

$$A_{11}B_{11} + A_{12}B_{21} = M_1 + M_4 - M_5 + M_7$$
$$= (A_{11}+A_{22})(B_{11}+B_{22})+A_{22}(B_{21}-B_{11})-(A_{11}+A_{12})B_{22}+(A_{12}-A_{22})(B_{21}+B_{22})$$
$$A_{12}B_{22} + A_{11}B_{12} = M_3 + M_5 = A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22}$$
$$A_{21}B_{11} + A_{22}B_{21} = M_2 + M_4 = (A_{21} + A_{22})B_{11} + A_{22}(B_{21} - B_{11})$$
$$A_{22}B_{22} + A_{21}B_{12} = M_1 - M_2 + M_3 + M_6$$
$$= (A_{11}+A_{22})(B_{11}+B_{22})-(A_{21}+A_{22})B_{11}+A_{11}(B_{12}-B_{22})+(A_{21}-A_{11})(B_{11}+B_{12})$$

This way, since this recursion takes 7 recursions instead of 8, the overall time complexity $T_{st}(n)$ is given as, (k_{st} is a constant representing the unit operations)

$$T_{st}(n) = 7T_{st}\left(\frac{n}{2}\right) + k_{st}$$

which can be solved to obtain,

$$T_{st}(n) = k_{st}\left(\frac{5n^{2.80735}-1}{6}\right) = O(n^{2.80735})$$

Strassen's algorithm is an example of a bilinear algorithm. The generalised definition of the bilinear algorithm [17] includes two steps: for an integer t, there are t matrices which are the products of the linear combination of A_{ij}

and B_{ij} such that the resultant elements of the product matrix are the linear combinations of those t elements. The integer t for an $n \times n$ matrix is known as the *bilinear complexity*. It can be easily proven [17] that if the bilinear complexity of an $n \times n$ matrix is r then the matrix multiplication exponent satisfies the bound $\omega \leq O(log_n t)$. Now the definitions and the generic framework for the matrix multiplication I use from here on have been directly borrowed from [17]. Let $\mathbb{F}$ be a field and U, V, W be some vector spaces. Let the spans of vector spaces U, V and W be $\{x_1, x_2, x_3, \cdots, x_{dim(U)}\}$, $\{y_1, y_2, y_3, \cdots, y_{dim(V)}\}$ and $\{z_1, z_2, z_3, \cdots, z_{dim(W)}\}$. A tensor T over (U, V, W) is defined as a linear combination

$$T = \sum_{u=1}^{dim(U)} \sum_{v=1}^{dim(V)} \sum_{w=1}^{dim(W)} t^{uvw} \cdot x_u \otimes y_v \otimes z_w$$

where $t^{uvw} \in \mathbb{F}$. We define the *rank* of a tensor T over (U, V, W), denoted $R(T)$, where U, V, W are vector spaces, $m = dim(U)$, $n = dim(V)$, $p = dim(W)$, as the minimal integer w_r such that

$$T = \sum_{u=1}^{m}\sum_{v=1}^{n}\sum_{w=1}^{p} t^{uvw} \cdot x_u \otimes y_v \otimes z_w = \sum_{s=1}^{w_r} \left[\left(\sum_{u=1}^{m} \alpha^{su} \cdot x_u \right) \otimes \left(\sum_{v=1}^{n} \beta^{sv} \cdot y_v \right) \otimes \left(\sum_{w=1}^{p} \gamma^{sw} \cdot z_w \right) \right]$$

where $\alpha^{su}, \beta^{sv}, \gamma^{sw} \in \mathbb{F}$. Here, sometimes T is denoted as $T = (\langle m, n, p \rangle)$ and the rank is denoted as $R(\langle m, n, p \rangle)$. Again it has been proven that if $r_1, r_2, r_3, r_4 \in \mathbb{Z}$, and $R(\langle r_1, r_2, r_3 \rangle) \leq r_4$, then the matrix multiplication exponent ω satisfies $(r_1 r_2 r_3)^{\frac{\omega}{3}} \leq r_4$. The above formalism represents a generalised *exact* bilinear algorithm. An indeterminate notion of *approximation* is adopted to encompass faster regimes of asymptotically faster algorithms, using an indeterminate λ and the ring of polynomials $\mathbb{F}[\lambda]$ with coefficients in the field $\mathbb{F}$. We define the *border rank* of the tensor $(U, V, W) = (\langle m, n, p \rangle)$, $m = dim(U), n = dim(V), p = dim(W)$, denoted as $\underline{R}(\langle m, n, p \rangle)$, as the minimal integer w_{br} such that there exists an integer $c \geq 0$ and a tensor T_{new} satisfying,

$$\lambda^c \sum_{u=1}^{m}\sum_{v=1}^{n}\sum_{w=1}^{p} t^{uvw} \cdot x_u \otimes y_v \otimes z_w = \sum_{s=1}^{w_{br}} \left[\left(\sum_{u=1}^{m} \alpha^{su} \cdot x_u \right) \otimes \right.$$

$$\left. \left(\sum_{v=1}^{n} \beta^{sv} \cdot y_v \right) \otimes \left(\sum_{w=1}^{p} \gamma^{sw} \cdot z_w \right) \right] + \lambda^{c+1} T_{new} \tag{1}$$

where $\alpha^{su}, \beta^{sv}, \gamma^{sw} \in \mathbb{F}[\lambda]$. Two important statements made in [17] about border ranks of tensors are that: there exists a constant j such that $R(T) \leq j \times \underline{R}(T)$ and for any 4 integers $r_1, r_2, r_3, r_4 \in \mathbb{Z}$ such that $\underline{R}(\langle r_1, r_2, r_3 \rangle) \leq r_4$, then the matrix multiplication exponent ω satisfies $(r_1 r_2 r_3)^{\frac{\omega}{3}} \leq r_4$. Schönhage's generalisation of the above is framed as the *asymptotic sum inequality*: For $k, t \in \mathbb{Z}, k, t \geq 0$, $m_1, \cdots, m_k, n_1, \cdots, n_k, p_1, \cdots, p_k \in \mathbb{Z}, \geq 0$. If

$$\underline{R}\left(\otimes_{i=1}^{k} \langle m_i, n_i, p_i \rangle \right) \leq t$$

then,

$$\sum_{i=1}^{k}(m_i n_i p_i)^{\frac{\omega}{3}} \leq t$$

Some more recent approaches in literature progressively use the above formalisms [18, 19, 20, 21, 22, 23, 24] to approach the lowest limit $\omega = 2$ by taking the Coppersmith-Winograd tensor and its higher powers. The current formal bound is steady at $\omega \leq 2.3728639$ [23]. The algorithm discussed in section 3 uses the approximate bilinear algorithms defining border ranks, where the deep neural net "learns" the approximation and improves upon the coefficients, more precisely, zeroing them one by one until we approach the asymptotic limit. Before that, we will need to revisit how we formalise neural networks.

In this article, the neural networks will be defined as an abstract graphical model. We begin with some basic definitions from graph theory [25]. A *graph* is an ordered 2-tuple $G = (V, E)$ where V is a set known as the set of *vertices* and E is the set of *edges* defined in the following way.

$$E = \{(x, y) \mid (x, y) \in V \times V \wedge x \neq y\}$$

A *directed graph* is a graph $G_{dir} = (V, E)$, the elements $(x, y) \in E$ are ordered pairs, where $x, y \in V$. A *weighted graph* is a graph whose vertices or edges have assigned weights. An independent set is a set of vertices in a graph, no two of which have an edge in between them. A k-partite graph is a graph whose vertices can be partitioned into k different independent sets. This number k is known as the chromatic number of the graph. Now I am defining a deep neural network (DNN) of depth k as a k-partite weighted directed graph $N = (V_N, E_N)$ with the following features:

1. Each vertex $v \in V_N$ has an associated set IS_v, which I call the *inward set*, as the set $IS_v = \{\, c \in V_N \mid (c, v) \in E_N \,\}$.

2. Also, every vertex $v \in V_N$ is associated with a time-varying constant $x_v(t) \in \mathbb{F}$, $\mathbb{F}$ being a field, a time-invariant *activation function* $K_v : \mathbb{F}^{|IS_v|} \to \mathbb{F}$, and a constant b_v, the *bias*.

3. Each edge $(u, v) \in E_N$ is associated with a time-varying weight $p_{uv}(t) \in \mathbb{F}$ which varies with time as per a chosen *learning algorithm* A.

4. At a time instance t, the constant with the vertex v, $x_v(t)$ is computed as,

$$x_v(t) = K_v\left(\sum_{w \in IS_v} x_w(t) p_{uw}(t) + b_v\right)$$

5. The learning algorithm A compares the *computed* and the *actual* outputs and obtains $p_{uv}(t+1)$ from $p_{uv}(t)$ for each $(u, v) \in E_N$.

3 Deep learning algorithm for matrix product

The equation 1 that explains the basic structure of an approximate bilinear algorithm for matrix products can be modified in the following form;

$$\sum_{u=1}^{m}\sum_{v=1}^{n}\sum_{w=1}^{p}(\lambda^{3M}\xi_{uvw})\cdot x_u\otimes y_v\otimes z_w=\sum_{u=1}^{m}\sum_{v=1}^{n}\sum_{w=1}^{p}x_u\otimes y_v\otimes z_w$$
$$\Bigg(\sum_{r=0}^{3M}\lambda^r\Bigg(\sum_{l=0}^{M}\sum_{q=0}^{M}\Bigg(\sum_{s=1}^{w_R}A_{(r-l-q)su}B_{lsv}C_{qsw}\Bigg)\Bigg)-\lambda^{3M-1}t_{uvw}\Bigg)$$

Using the uniqueness of the components and the basis elements of the tensor products of the vector spaces, we can readily conclude that

$$\lambda^{3M}\xi_{uvw}=\sum_{r=0}^{3M}\lambda^r\Bigg(\sum_{l=0}^{M}\sum_{q=0}^{M}\Bigg(\sum_{s=1}^{w_R}A_{(r-l-q)su}B_{lsv}C_{qsw}\Bigg)\Bigg)-\lambda^{3M-1}t_{uvw},$$

$$\forall 1\leq u\leq m, 1\leq v\leq n, 1\leq w\leq p \tag{2}$$

where $c=3M-1$, $\alpha^{su}(\lambda)=\sum_{r=0}^{3M}A_{rsu}\lambda^r$, $\beta^{sv}(\lambda)=\sum_{r=0}^{3M}B_{rsv}\lambda^r$, and $\gamma^{sw}(\lambda)=\sum_{r=0}^{3M}C_{rsw}\lambda^r$ in the equation 1 introduced in section 2. Using the structure above, I present the following graph-theoretic construction of the neural network, i.e. the graph $N=(V_N,E_N)$ to emulate the functionality of equation 2 consisting of different types of vertices with different activation functions. The notation used here is discussed towards the end of section 2. The set of vertices V_N is partitioned into the following subcategories, classified based on their activation functions:

1. $Y_N=\{v_1\}\cup\{y_{uvw},1\leq u\leq m,1\leq v\leq n,1\leq w\leq p\}$ is the set consisting of the entry vertices of "logarithmic activations", where the vertex v_1 considers the value of the indeterminate λ as the input $x_{v_1}(t)=\lambda$, and the vertices y_{uvw} consider the values of the elements of the tensor t_{uvw} as the respective inputs $x_{y_{uvw}}(t)=t_{uvw}$. The activation functions $K_{v_1}:\mathbb{R}\to\mathbb{R}$ and $K_{y_{uvw}}:\mathbb{R}\to\mathbb{R},\forall 1\leq u\leq m,1\leq v\leq n,1\leq w\leq p$ are all the same, $K_{v_1}(x)=K_{y_{uvw}}(x)=log(x)$.

2. $S_N=\{e_0,e_1,\cdots,e_M,e_{3M+1}\}\cup\{e_{uvw},1\leq u\leq m,1\leq v\leq n,1\leq w\leq p\}$ is the set consisting of the second layer neurons of "exponential activations", where the vertices $e_i,0\leq i\leq M\vee i=3M+1$ produces the literal monomials λ^i directed by the weights of the edges and the vertices $e_{uvw},1\leq u\leq m,1\leq v\leq n,1\leq w\leq p$ reverts back to the tensor elements t_{uvw}. The activation functions $K_{e_i}:\mathbb{R}\to\mathbb{R},0\leq i\leq M\vee i=3M+1$ and $K_{e_{uvw}}:\mathbb{R}\to\mathbb{R},\forall 1\leq u\leq m,1\leq v\leq n,1\leq w\leq p$ are all the same, $K_{e_i}(x)=K_{e_{uvw}}(x)=exp(x)$.

3. $L_N = \{\Xi_{su}, 1 \leq s \leq w_R, 1 \leq u \leq m\} \cup \{\Psi_{sv}, 1 \leq s \leq w_R, 1 \leq v \leq n\} \cup \{\Phi_{sw}, 1 \leq s \leq w_R, 1 \leq w \leq p\}$ is the set of vertices with "identity activations". These vertices will produce the individual polynomials to be hunted for as the linearly-combined factors in equation 2. The activation function for each such vertex is the identity function $K_{\Xi_{su}}(x) = K_{\Psi_{sv}}(x) = K_{\Phi_{sw}}(x) = x$. For each u, v and w, $1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p$, there are two additional vertices in V_N; one which assimilates the factorised polynomials $\left(\sum_{k=0}^{M} A_{ksu}\lambda^k\right)\left(\sum_{k=0}^{M} B_{ksv}\lambda^k\right)\left(\sum_{k=0}^{M} C_{ksw}\lambda^k\right)$ and produces the sum $\sum_{s=1}^{w_R}\left(\sum_{k=0}^{M} A_{ksu}\lambda^k\right)\left(\sum_{k=0}^{M} B_{ksv}\lambda^k\right)\left(\sum_{k=0}^{M} C_{ksw}\lambda^k\right)$; the other subtracts the sum $\sum_{u,v,w=1,1,1}^{m,n,p} \lambda^{3M-1} t_{uvw}$ from the former.

4. $P_N = \{q_{uvw}, 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p\} \cup \{p_{suvw}, 1 \leq s \leq w_R, 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p\}$ is the set of vertices that collect the polynomial factors to multiply them. Vertices q_{uvw} will be responsible for generating the monic terms $\lambda^{3M-1} t_{uvw}$, whereas the vertices p_{suvw} will be responsible for generating the products $\left(\sum_{k=0}^{M} A_{ksu}\lambda^k\right)\left(\sum_{k=0}^{M} B_{ksv}\lambda^k\right)\left(\sum_{k=0}^{M} C_{ksw}\lambda^k\right)$. The activation function for vertices q_{uvw} is a bivariate product $K_{uvw} : \mathbb{R}^2 \to \mathbb{R}, K_{uvw}(x, y) = xy$ and that for vertices p_{suvw} is a trivariate product $K_{uvw} : \mathbb{R}^3 \to \mathbb{R}, K_{uvw}(x, y, z) = xyz$.

Thus the set of vertices is really $V_N = Y_N \cup S_N \cup L_N \cup P_N$. The set of edges E_N contains the following elements.

1. $(v_1, e_k), 0 \leq k \leq M \vee k = 3M + 1$ with the weight k.
2. $(y_{uvw}, e_{uvw}), 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p$ with weight 1.
3. $(e_k, \Xi_{su}), 1 \leq k \leq M, 1 \leq s \leq w_R, 1 \leq u \leq m$ with weights as the coefficients A_{ksu}.
4. $(e_k, \Psi_{sv}), 1 \leq k \leq M, 1 \leq s \leq w_R, 1 \leq v \leq n$ with weights as the coefficients B_{ksv}.
5. $(e_k, \Phi_{sw}), 1 \leq k \leq M, 1 \leq s \leq w_R, 1 \leq w \leq p$ with weights as the coefficients C_{ksw}.
6. Edges of the forms (Ξ_{su}, p_{suvw}), (Ψ_{sv}, p_{suvw}) and (Φ_{sw}, p_{suvw}), $1 \leq s \leq w_R, 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p$, all the weights being unity.
7. Edges that connect all the vertices p_{suvw} to the uvw^{th} linear activation function that produces the sum $\sum_{s=1}^{w_R}\left(\sum_{k=0}^{M} A_{ksu}\lambda^k\right)\left(\sum_{k=0}^{M} B_{ksv}\lambda^k\right)\left(\sum_{k=0}^{M} C_{ksw}\lambda^k\right)$. Each such edge has unit weight.

8. Edges $(e_{uvw}, q_{uvw}), 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p$ of unit weight.

9. Edges $(e_{3M-1}, q_{uvw}), 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p$ of unit weight.

10. Edges that connect the uvw^{th} vertex with linear activation function and the vertex q_{uvw} to the resultant vertex for the uvw^{th} tensorial element.

So for implementing the above neural network for the approximate bilinear algorithm, there are precisely $2w_R(mnp+m+n+p)+2mnp+M+3$ activation nodes and $M+2+mnp+w_R(mnp+mn+np+mp+M(m+n+p))$ edges. For the symmetric case $w_R = n^3, m = n = p$, there are $O(M+n^6)$ nodes and $O(n^6+Mn^4)$ edges. The edges that "learn" are the ones that contain the coefficients A_{ksu}, B_{ksv}, C_{ksw}, which are $O(Mn^4)$ in number. The chromatic number of the above graph is 2, but the depth of the graph is 6. The learning algorithm focuses on the edges containing the coefficients and follows the following probabilistic approach mentioned in the form of pseudocode.

Algorithm: Probabilistic learning algorithm for the neural network
while $TRUE$
  if $w_R = \lceil (mnp)^{\frac{2}{3}} \rceil \ \wedge \ \forall u, v, w, r, 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p,$
    $0 \leq r \leq 3M-2, \sum_{l=0}^{M} \sum_{q=0}^{M} \sum_{s=1}^{w_R} A_{(r-l-q)su} B_{lsv} C_{qsw} == 0$
    $\wedge \ \forall u, v, w, 1 \leq u \leq m, 1 \leq v \leq n, 1 \leq w \leq p,$
    $\sum_{l=0}^{M} \sum_{q=0}^{M} \sum_{s=1}^{w_R} A_{(3M-1-l-q)su} B_{lsv} C_{qsw} == t_{uvw}$
    exit
  else Run one iteration of the aforementioned DNN.
    Choose $\mu_\lambda \sim M(\lambda), M(\lambda)$ with probability distribution function $f(\lambda)$
     such that $\mu_\lambda A_{(r-l-q)w_R u} B_{lw_R v} C_{qw_R w} = A'_{(r-l-q)w_R u} B'_{lw_R v} C'_{qw_R w}$
    where $A'_{kw_R u}, B'_{lw_R v}, C'_{qw_R w}$ are the new coefficients and the weights
     of the edges $(e_k, \Xi_{w_R u}), (e_l, \Phi_{w_R v}),$ and $(e_q, \Psi_{w_R w})$ respectively. The
    coefficients $A'_{kw_R u}, B'_{lw_R v}, C'_{qw_R w}$ would be drawn from the same
     probability distribution $A'_{kw_R u}, B'_{lw_R v}, C'_{qw_R w} \sim M(\lambda)$
    if $\forall u, v, w, r, \mu_\lambda = 0$
      then $w_R = w_R - 1$
       else if $\exists u, v, w, r, \mu_\lambda = 0 \ \wedge \ \exists u', v', w', r', \mu_\lambda \neq 0$ then $\mu_\lambda = 1$

The above neural network is of the same structure as the equation 2, which guarantees the partial correctness, i.e. the neural network actually "learns" from the mistake it makes in terms of the coefficients $A_{(r-l-q)w_R u}$, $B_{lw_R v}$, $C_{qw_R w}$ where the sum needs to go to zero for $0 \leq r \leq 3M-2$ and the tensorial element t_{uvw} for $r = 3M-1$, which is trivially visible from the conventional resemblance of the neural network and the intended bilinear identity of the form of the equation 2. Since the learning algorithm is of the probabilistic nature, as we choose the coefficients from an appreciable distribution, the termination of the algorithm is also probable. A brief but more detailed explanation of the same is explained in section 4.

4 The probabilistic termination of the algorithm

The proof that the deep learning algorithm in section 3, if it terminates, is correct comes from the mathematical structure of the neural network itself. This section discusses the proof of termination using probabilistic tools. For a random variable $X \sim R(\lambda)$ with a probability density function (p.d.f.) $r(x)$, the probability of the event $a \leq X \leq b$ is given as,

$$P(a \leq X \leq b) = \int_{x=a}^{x=b} r(x)dx$$

I would be introducing some more shorthand notations for further definition of the event of interest; $\mu_{\lambda i(r,l,q)suvw}$ denotes the choice of μ_λ for the product of $A_{(r-l-q)w_R u}$, $B_{lw_R v}$ and $C_{qw_R w}$ for the i^{th} *step* or iteration of the learning algorithm. The *convolutional sum* $*_{n=1}^{m} g_n$ is the iterated notation of sequentially convolving g_1, g_2, $\cdots$, g_m. Finally $\mathcal{M}\{f_t(x)\}(s)$ (the 't'-subscript having no notational significance of its own; it's been added to remove ambiguity) denotes the Mellin transform of the function $f_t(x)$ defined as

$$\mathcal{M}\{f_t(x)\}(s) = \int_{x=0}^{x=\infty} x^{s-1} f_t(x)dx$$

But a more compact notation $\mathcal{M}(R)$ would mean the Mellin transform of the random variable R. Clearly, given the choice of the design of the learning algorithm, all the random variables involved, namely A_{ksu}, B_{ksv}, C_{ksw} and $\mu_{\lambda i(r,l,q)suvw}$, are independent identically distributed variables with the p.d.f. $f(x)$. It is well-known and extensively used in literature [26, 27] that the probability density of the sum of two random variables X and Y with pdfs $f_X(x)$ and $f_Y(y)$ respectively, is the convolution $f_X * f_Y(z)$; and that of the product is $\mathcal{M}^{-1}(\mathcal{M}_X \cdot \mathcal{M}_Y)$, the inverse Mellin's transform of the "product of the Mellin's transforms". After k iterations of the probabilistic learning algorithm, the coefficient of λ^r in equation 2, or the output of the activated nodes p_{suvw}, looks like the following, if it were to terminate at $w_R = w_{R_0}$ and

$$0 \leq r \leq 3M-2, \sum_{l=0}^{M}\sum_{q=0}^{M}\sum_{s=1}^{w_{R_0}}\prod_{i=1}^{k} \mu_{\lambda i(r,l,q)suvw} A_{(r-l-q)su} B_{lsv} C_{qsw} = 0$$

$$\sum_{l=0}^{M}\sum_{q=0}^{M}\sum_{s=1}^{w_{R_0}}\prod_{i=1}^{k} \mu_{\lambda i(3M-1,l,q)suvw} A_{(3M-1-l-q)su} B_{lsv} C_{qsw} = t_{uvw}$$

Well this event is as equably probable as the following p.d.f.

$$f_{kw_R ruvw} = *_{l=0}^{M} *_{q=0}^{M} *_{s=1}^{w_{R_0}} \mathcal{M}^{-1}\Bigg(\Big(\prod_{i=1}^{k} \mathcal{M}(\mu_{\lambda i(r,l,q)suvw})\Big) \mathcal{M}(A_{(r-l-q)su}) \mathcal{M}(B_{lsv}) \mathcal{M}(C_{qsw}) \Bigg)$$

Using the parameters of the probability density function, one can tweek into the profile by taking a limit where the function behaves more like a delta function at a point closer to zero for $0 \leq r \leq 3M-2$ and the tensor element for $r = 3M-1$. For example, if I consider the log-normal distribution,

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} exp\bigg(- \frac{(log(x) - \mu)^2}{2\sigma^2} \bigg)$$

It is well-known that the product of log-normal distributions is also a log-normal one [28]. There's been extensive numerical study on the convolutions of more than one log-normal distribution [29, 30, 31], where the cumulative density function can be made steeper at the point of interest by a careful choice of parameters, which will increase the certainty of the termination of the algorithm at $w_R = \lceil (mnp)^{\frac{2}{3}} \rceil$ and at any finite centralised value of k. This makes it very conclusive that the termination of the deep learning algorithm really depends on the probability distribution function. The example of log-normal is a rather more reliable one, with partial reliability on the symbolic (in the sense of the product of log-normal distributions) and numerical simplicity (in terms of their convolutions). More precisely, the convolution of multiple log-normal distributions is associated with a cumulative density function that can approach the limit of the step function stepping up on the point of interest. This step-like cumulative density function corresponds to the delta function-like p.d.f. which is really appreciable. This delta function-like behaviour of the probability density function suggests the *certainty of the algorithm*. There are other analytically involved examples, such as normal distributions (the product of two normal variables yields a behaviour isomorphic to a well-scaled modified Bessel's functions [32]); and Cauchy distributions [33], which intuitively grows in complexity with increasing value of k and M. This poses some degree of risk on the convergence or the "learning" time of the algorithm that comes along with the choice of the p.d.f. These analyses can pave ways for future work into the development of the asymptotically optimal matrix multiplication algorithm.

As an instance, I am considering the multiple convolutions of the product of independent identically distributed random variables that follow Mellin-transformed algebra. Figure 1 shows for the normal distribution $X \sim N(\mu, \sigma^2)$, $f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$. I am considering a simple case $\mu = 0$ where the random variable $X^n, n \in \mathbb{Z}$ with the probability density function, as in Meijer-G function,

$$f_{X^n}(x) = \frac{1}{2^{\frac{3n-2}{2}} \pi^{\frac{n}{2}} \sigma^n} G^{n0}_{0n} \bigg(\frac{x}{2^{\frac{n}{2}} \sigma^n}, \frac{1}{2} \bigg| \begin{matrix} \{\} \\ \{0, 0, \cdots (\text{n times}), \cdots 0\} \end{matrix} \bigg)$$

So the above function and its subsequent integer multiples $k \in \mathbb{N}$ of the random variable X^n, i.e. kX^n have been plotted in figure 1 in the form of cumulative distribution function. Figures 2 and 3 show for the log-normal distribution $log(X_1) \sim N(\mu, \sigma^2)$, $f_{X_1}(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} exp(-\frac{(log(x)-\mu)^2}{2\sigma^2})$. Again I am considering $\mu = 0$ and consequently the higher powers of log-normal variable X_1, i.e.

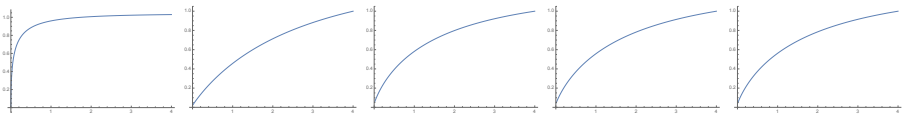

Figure 1: The cumulative distribution functions of the distribution of an exponentiated normally distributed random variable and its multiples. The leftmost plot shows the "normal variable" raised to the fifth power, following a Meijer-G function based distribution. The consequent plots as we go to the right show the consequent multiples (convolutions) of the exponentiated distribution. The random variable we begin with here is of mean 0 and variance 1.

$X_1^n, n \in \mathbb{N}$, $log(X_1) \sim N(0, \sigma)$. The distribution $log(X_1^n) \sim N(0, n\sigma^2)$ is also log-normal. The convolution of the log-normal distribution has been plotted in figure 2 with $\sigma = 1$, and figure 3 with $\sigma = \frac{1}{1000}$. Figure 4 shows for the Cauchy distribution $X_3 \sim Cauchy(x_0, \gamma)$, $f_{X_3}(x) = \frac{1}{\pi\gamma\left(1+\left(\frac{x-x_0}{\gamma}\right)^2\right)}$. Now an example of the random variable X_3^4 has the probability distribution function,

$$f_{X_3^4}(x) = \frac{1}{96\pi^4}\sum_{n=0}^{4}\frac{(-1)^n\big(log(x) - log((x_0+i\gamma)^n(x_0-i\gamma)^{4-n})\big)}{x-(x_0+i\gamma)^n(x_0-i\gamma)^{4-n}}\big[4\pi^2 + \big(log(x) - log((x_0+i\gamma)^n(x_0-i\gamma)^{4-n})\big)^2\big]$$

Now this function and its self-convolutions have been plotted in figure 4. Now each of these convolutions shown up in figures 1, 2, 3, 4 seem to show some tendencies towards a uniform distribution, which is not desirable because the final convergence of the algorithm now becomes all the more of a Markov chain. Instead, if I tend to restrict myself to a choice of the log-normal distribution but with a careful selection of mean and variance, then it can be demonstrated that the convergence can be well-tailored as per choice with some practical trade-offs. Again let the coefficients A_{ksu}, B_{ksv}, C_{ksw} and $\mu_{\lambda_{i(r,l,q)suvw}}$ be independent identically log-normal distributed (IILND) variables. As a testbed example, let there be k such IILND variables with mean $\frac{1}{k}log(x_0)$ and numerically small variance $\sigma \to 0$, such that

$$f(x) = \lim_{\sigma\to 0}\frac{1}{x\sqrt{2\pi\sigma^2}}exp\bigg(-\frac{1}{2\sigma^2}\big(log(x) - \frac{1}{k}log(x_0)\big)^2\bigg) = \delta(x - x_0^{\frac{1}{k}})\Theta(x_0^{\frac{1}{k}})$$

, k^{th} power of which leads to the limiting distribution function $\delta(x - x_0)\Theta(x_0)$. This shows that the choice of parameters in the probability density function plays a pivotal role in the convergence of the learning algorithm.

5 Conclusion

In this article, we pick up on the essential mathematical structure of the approximate bilinear algorithms and make an attempt to fit in the structure of

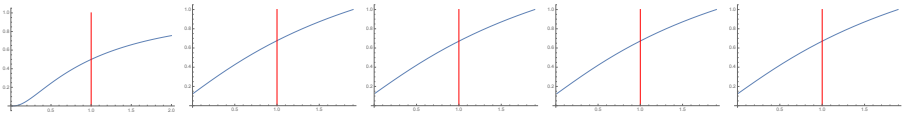

Figure 2: The cumulative distribution functions of the distribution of an exponentiated log-normally distributed random variable and its multiples. The leftmost plot shows the "log-normal variable" raised to the fifth power, following a Gauss error function based distribution. The consequent plots as we go to the right show the consequent multiples (convolutions) of the exponentiated distribution. The random variable we begin with here is of mean 0 and variance 1.

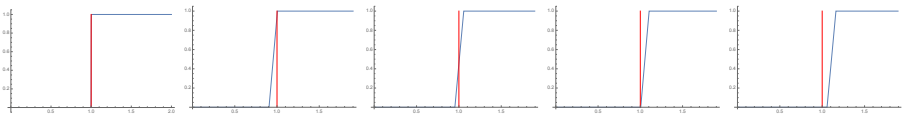

Figure 3: The cumulative distribution functions of the distribution of an exponentiated log-normally distributed random variable and its multiples. The leftmost plot shows the "log-normal variable" raised to the fifth power, following a Gauss error function based distribution. The consequent plots as we go to the right show the consequent multiples (convolutions) of the exponentiated distribution. The random variable we begin with here is of mean 0 and variance $\frac{1}{1000}$.

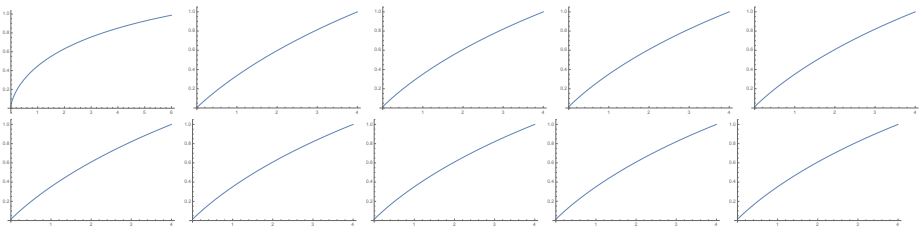

Figure 4: The cumulative distribution functions of the distribution of an exponentiated Cauchy distributed random variable and its multiples. The leftmost plot shows the "Cauchy variable" raised to the fourth power, following a polylog function based distribution. The consequent plots as we go to the right and to the next row, show the consequent multiples (convolutions) of the exponentiated distribution. The random variable we begin with here is of mean 0 and the half-width at the half-maximum is set at 4.

the neural network, and present a way to automate the seeking of the asymptotic optimisation of the matrix multiplication problem to the lowest possible complexity bound which has been conjectured for long. We also open to probabilistic determinacy of the algorithm with the randomisation of the inputs and the weights of the neural networks. There has been an example of a distribution that approximately gets to the delta-function like behaviour in the probability density, and on the contrary, there can be examples that can be symbolically intractable and cannot be immediately guaranteed, by theory, for the termination of the algorithm. This validation of the termination of the algorithm intuitively depends on the choice of the distribution of the random coefficients of the polynomials of the coefficient, the granularities of which is a subject of future work. The approximate bilinear algorithm with which we have started may have different border ranks for *sparse tensors*, for instance, the matrices having a significant number of zero elements, and interestingly, the proposed approach can adapt itself to this very special interesting case. This scheme promises a convenient automation of matrix multiplication algorithm design that has consequences and applications in algebraic complexity theory [17], parsing [34, 35, 36], hence compiler design, programming language theory and natural language processing [37, 38], and many more.

References

[1] Hanin, B., Sellke, M., "Approximating Continuous Functions by ReLU Nets of Minimal Width", arXiv:1710.11278 (2017)

[2] Yarotsky, D., "Error bounds for approximations with deep ReLU networks", Neural Networks : The official journal of the International Neural Network Society, 94 (2017): 103-114

[3] Park, Sejun et al, "Minimum Width for Universal Approximation", arXiv:2006.08859 (2020)

[4] French, J., "The time traveller's CAPM", Investment Analysts Journal, 46 (2): 8196 (2016)

[5] Choy, Christopher B., et al. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction", European conference on computer vision, Springer, Cham, (2016)

[6] Ganesan, N. "Application of Neural Networks in Diagnosing Cancer Disease Using Demographic Data", International Journal of Computer Applications, 1 (26): 8197, (2010)

[7] Nabian, M. A. and Meidani, H., "Deep Learning for Accelerated Seismic Reliability Analysis of Transportation Networks", Computer Aided Civil and Infrastructure Engineering, 33: 443-458, (2018)

[8] Hoskins, J.C., Himmelblau, D.M., "Process control via artificial neural networks and reinforcement learning", Computers & Chemical Engineering, 16 (4): 241251, (1992)

[9] Zissis, D., "A cloud based architecture capable of perceiving and predicting multiple vessel behaviour", Applied Soft Computing, 35: 652661, (2015)

[10] Zheng, X., Peng, H., Chen, Y., Zhang, P., Zhang, W., "Character-based parsing with convolutional neural network", In Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI15), AAAI Press, pp. 10541060, (2015)

[11] Shen, Y., Lin, Z., Huang, C-W, Courville, A., "Neural Language Modeling by Jointly Learning Syntax and Lexicon", International Conference on Learning Representations (ICLR), (2018)

[12] Wang, P., Qian, Y., Soong, F.K., He, L., Zhao, H., "Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network", arXiv:1510.06168, (2015)

[13] Steingrimsson, Steinppor et al., Augmenting a BiLSTM tagger with a Morphological Lexicon and a Lexical Category Identification Step, RANLP (2019)

[14] Chen, D., Manning, C., "A Fast and Accurate Dependency Parser using Neural Networks", Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, pp. 740750, (2014)

[15] Hopcroft, J.E., Ullman, J.D., "Introduction to Automata Theory, Languages, and Computation", Reading/MA: Addison-Wesley, ISBN 978-0-201-02988-8, p.145, (1979)

[16] Strassen, V., "Gaussian elimination is not optimal", Numer. Math. 13, 354356 (1969)

[17] Le Gall, François, "Algebraic Complexity Theory and Matrix Multiplication", Association for Computing Machinery, Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, (2014)

[18] Bini, D., Capovani, M., Romani, F., Lotti, G., "O(n^2.7799) complexity for $n \times n$ approximate matrix multiplication", Information Processing Letters 8, 5 (1979), 234235

[19] Coppersmith, D., Winograd, S., "Matrix multiplication via arithmetic progressions", Journal of Symbolic Computation 9, 3 (1990), 251280

[20] Pan, V.Y., "Field extension and triangular aggregating, uniting and canceling for the acceleration of matrix multiplications", In Proceedings of the 20th Annual Symposium on Foundations of Computer Science (1979), pp. 2838

[21] Schönhage, A., "Partial and total matrix multiplication", SIAM Journal on Computing, 10, 3 (1981), 434455

[22] Stothers, A., "On the Complexity of Matrix Multiplication", PhD thesis, University of Edinburgh, (2010)

[23] Le Gall, F., "Powers of tensors and fast matrix multiplication", In Proceedings of the 39th Interna- tional Symposium on Symbolic and Algebraic Computation (2014), 296303

[24] Vassilevska Williams, V., "Multiplying matrices faster than Coppersmith-Winograd", In Proceedings of the 44th Symposium on Theory of Computing (2012), pp. 887898

[25] West, D. B., "Introduction to Graph Theory", 2^{nd} edition, Prentice Hall, (2000)

[26] Grimmett, G., Stirzaker, D., "Probability and random processes", Vol. 80, Oxford university press, (2001)

[27] Epstein, Benjamin, "Some Applications of the Mellin Transform in Statistics", Ann. Math. Statist, 19 (1948), no. 3, pp. 370-379

[28] David E. Burmaster, Delores A. Hull, "Using lognormal distributions and lognormal probability plots in probabilistic risk assessments", Human and Ecological Risk Assessment: An International Journal, 3:2, 235-255, (1997)

[29] Beaulieu, N., Abu-Dayya, A.A., McLane, P.J., "On approximating the distribution of a sum of independent lognormal random variables", WESCANEX 93 'Communications, Computers and Power in the Modern Environment' Conference Proceedings., IEEE, pp. 72-79, (1993)

[30] Fenton, L., "The sum of lognormal probability distributions in scatter transmission systems", IRE Transactions on Communications Systems, volume 8, issue 1, pp. 57 - 67, (1960)

[31] Chakrabarti, N. B., "A Note on a Sum of Lognormals", arXiv:1606.07300, (2016)

[32] Seijas-Macías, A., "An approach to distribution of the product of two normal variables", Discussiones Mathematicae, Probability and Statistics, 32(2012), pp. 8799

[33] Springer, M. D., Thompson, W. E., "The Distribution of Products of Independent Random Variables", SIAM Journal on Applied Mathematics, Vol. 14, No. 3 (1966), pp. 511-526

[34] Lee, L., "Fast Context-Free Grammar Parsing Requires Fast Boolean Matrix Multiplication", Journal of the ACM 49(1), pp. 1-15, (2002), arXiv:cs/0112018

[35] Cohen, S.B., Gildea, D., "Parsing Linear Context-Free Rewriting Systems with Fast Matrix Multiplication", (2015), arXiv:1504.08342

[36] Valiant, L. G., General Context-Free Recognition in Less than Cubic Time, J. Comput. Syst. Sci., 10 (1975): pp. 308-315.

[37] Manning, C.D., Schütze, H., "Foundations of Statistical Natural Language Processing", MIT Press, ISBN 978-0-262-13360-9, (1999)

[38] Frost, R., Hafiz, R., Callaghan, P. (2007) "Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars", 10th International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE, Pages: 109 - 120, (2007), Prague