LOADING DEEP NETWORKS IS HARD

J. Šíma
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Praha 8, CZ
e-mail: sima@uivt.cas.cz

Abstract

The loading problem formulated by J.S. Judd seems to be a relevant model for supervised connectionist learning of the feedforward networks from the complexity point of view. It is known that loading general network architectures is NP-complete (intractable) when the (training) tasks are also general. Many strong restrictions on architectural design and/or on the tasks do not help to avoid the intractibility of loading. J.S. Judd concentrated on the width expanding architectures with constant depth and found a polynomial time algorithm for loading restricted shallow architectures. He suppressed the effect of depth on loading complexity and left as an open prototypical computational problem the loading of easy regular triangular architectures that might capture the crux of depth difficulties. We have proven this problem to be NP-complete. This result does not give much hope for the existence of an efficient algorithm for loading of the deep networks.

1. The Relevance of the Loading Problem

The research of the neural network models for the computational exploitation has been rapidly developed in the last years. This fact is confirmed by the existence of many journals and conferences concerning this topic. One of the basic reasons for this exciting activity in this area is the successful practical use of the learning algorithm back propagation (Rumelhart, et al. 1986) for the multi-layered neural network, and it is not by chance that this is the most widely applied neural network architecture. The defect of this algorithm, likewise of many others, is that it is very time consuming and that learning bigger networks can be intractable. The effort to speed up this algorithm has brought only limited results. Difficult problems cannot be solved easily. The task of theoretical computer scientists is to discover intrinsic causes of these difficulties. The interpretation and the explanation of such negative results is an ungrateful an unpopular task but on the other hand it is very needed. It can help to understand better the learning problem and to propose new techniques and methods for its' solution.

Rather than testing improved neural learning algorithms on XOR or parity tasks, the adequate tool for exploring the above-mentioned issues is the theory of complexity. From this point of view the practically solvable problems are those for which there is a polynomial time solving algorithm. On the other hand, so-called NP-complete problems are generally considered to be practically unsolvable. This approach leads to Valiant's theory of the learnable (Blumer, et al. 1986; Kearns, et al. 1987; Pitt and Valiant 1988; Svaytser 1989; Valiant 1984, 1985; Wiedermann 1991). This theory takes into consideration the complexity issues of learning and generalization from examples, from the general artificial intelligence point of view.

Our contribution issues from J.S. Judd's work (1990), who formulated the loading problem as a computational decision problem, to be a relevant model for supervised connectionist learning of the feedforward neural networks from the complexity point of view. J.S. Judd presented many trustworthy arguments for the convenience of this model. The polemic of the relevance of the loading problem in the book review by E.B. Baum (1991) is also very inspiring. In our opinion his most relevant argument is that in contradistinction to the loading problem, in application of neural networks we typically have control over the net we are loading. Thus, we can choose one that makes our problem easy. If we want to formalize the classical connectionist learning from the complexity point of view we must somehow qualify the set of possible network architectures from which one is chosen to be loaded. In the opposite case, when the network architecture can be arbitrary, then it would be very simple to create sufficiently large architecture with an appropriate configuration (synaptic weights) that performs the required task. But in this case the generalization does not make good sense, and the classical von Neuman's computer architecture is more convenient for the memorization task. So, we can generalize the loading problem for the set of possible network architectures from which one is chosen to be loaded. But the original loading problem is a special case of this generalized loading problem for the one-element set of the network architectures. Therefore the generalized loading problem must be at least as hard as the original one. This consideration weakens E.B. Baum's argument.

J.S. Judd proved the general loading problem to be NP-complete and added many negative corollaries concerning the special cases of this problem, with strong restrictions on architectural design and/or on the (training) tasks that do not help to avoid the intractibility of loading. Then he concentrated on the width expanding network architectures with constant depth and found a polynomial time algorithm for loading strongly restricted shallow architectures. His algorithm suppressed the effect of depth on loading complexity using shallow architectures that have a bounded size (by a constant) of the set of all partial configurations (i.e. assignments of functions to each computational node in the network that can potentially affect the given output). The possibility of there being an efficient method to search for correct partial configuration was completely ignored. In spite of this J.S. Judd stated the importance of the "depth issue" for the loading complexity. He offered an open prototypical computational problem of the loading of easy regular triangular architectures that might capture the crux of depth difficulties. We prove this problem to be NP-complete. This result does not give much hope for the existence of an efficient algorithm for loading of the deep networks.

2. The Loading Problem for Triangular Architectures

Here we remember the definition of the **loading problem** for **triangular architectures** by J.S. Judd. The **instance** of this problem (i.e. the input for the loading problem) consists of an integer d and a task T. The integer gives the depth (and width) of an architecture depicted in figure 1.

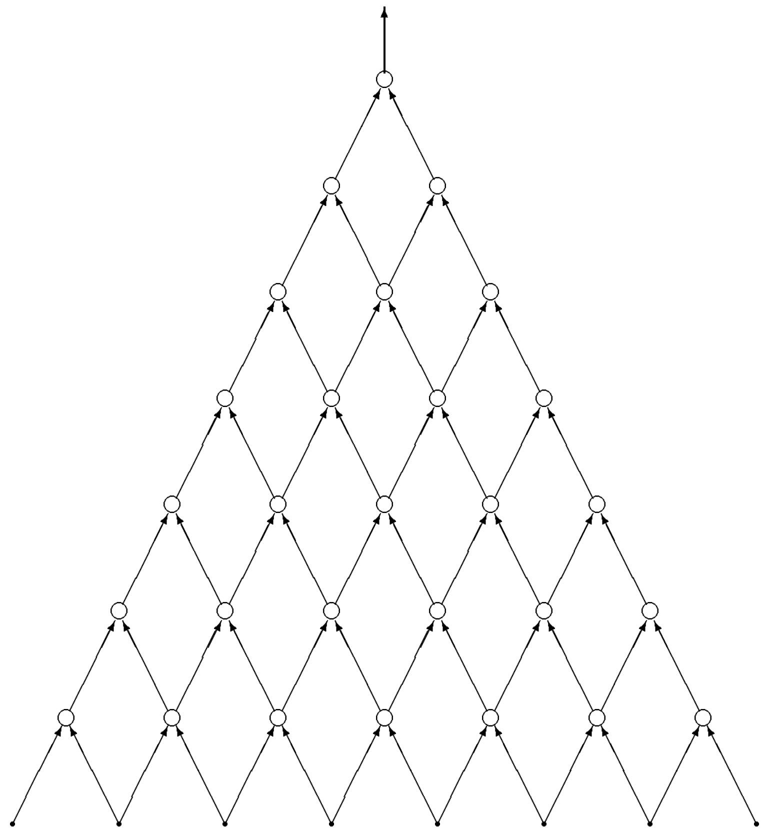

Figure 1: An example of the triangular architecture (d=7).

The nodes are arranged in a triangle with each layer having one fewer nodes than the previous layer until the last layer has one node in it. Each node receives an input from both of the two below it. The **task** is a collection of items, each with $d+1$ stimulus binary bits and 1 response binary bit (i.e. $T = \{(\sigma_i, \rho_i)/\sigma_i \in \{0,1\}^{d+1}, \ \rho_i \in \{0,1\}, i = 1, \ldots, \tau\}$).

The question of the decision form of the loading problem (i.e. the yes/no output of the loading algorithm) follows: Is there a configuration for this architecture that performs the given task? The **configuration** of a network is a list of functions corresponding one to one with the set of nodes of the net, meaning that the node computes the corresponding function. The configured architecture **performs** the task if the network function, composed of the node functions according to the configuration, computes for every item stimulus of the task the output value which agrees with the response of this item. In our case we say that the instance (d, T) is **performable** if there is a relevant configuration.

The node functions are taken from the so-called **node function set**. Typically, connectionists have used the set of linearly separable functions. Due to the simplicity we consider for example NAOFns = {OR, NAND} as the node function set. Then we generalize our result for non-trivial subset of symmetric and monotone functions. More exact, we consider the node function sets $\mathcal{F}$ that satisfy the following conditions:

(i) $\{\text{OR, NAND}\} \subseteq \mathcal{F}$ or $\{\text{AND, NOR}\} \subseteq \mathcal{F}$

(ii) $\mathcal{F} \subseteq \{0, 1, \text{OR, AND, NOR, NAND}\}$.

In our case for two-input gates it hardly matters. Moreover, J.S. Judd proved many theorems indicating that the difficulty in the loading problem has very little to do with the choice of node function sets (e.g. the loading problem is NP-complete also for the real-valued logistic-linear functions used in the back propagation learning algorithm). Meanwhile, our result, like the Blum and Rivest's one (1992), depends on this choice and it should be generalized for more complex node function sets.

3. The Intractibility of Loading Triangular Architectures

In this section we will prove the main result implying the intractibility of loading triangular architectures. For this purpose we will employ the technique of polynomial time reduction from satisfiability problem (Balcázar, et al. 1988). An instance (Z, C) of the general **satisfiability problem** (SAT) is an expression in Boolean variables from Z given in conjunctive normal form (i.e. a conjunction of disjunctions) in which all disjunctions from C (called **clauses**) consist of literals. A **literal** is a logical variable or its negation. With respect to this we will distinguish the positive and the negative literal. Without loss of generality any clause will not contain two literals derived from the same variable. We will use the **modified satisfiability problem** (MSAT) of which any instance (Z, C) satisfies the following condition. We consider a fixed ordering of the set of variables Z according to which the literals in clauses are ordered. Then the positive and negative literals alternate periodically in the clauses of an instance (Z, C) of MSAT. It means that the positive literal always follows after the negative one and conversely. The instance (Z, C) is said to be **satisfiable** if the variables can all be given values such that the whole logical expression is true. First, we prove the following lemma concerning the complexity of the decision problem whether the instance (Z, C) of MSAT is satisfiable.

Lemma

MSAT is NP-complete.

Proof:

We outline the basic idea of the proof. For every instance (Z, C) of SAT we construct in a polynomial time the instance (Z', C') of MSAT so that both logical expressions are equivalent. This means that (Z, C) is satisfiable iff (Z', C') is satisfiable.

Suppose that there is a clause from C in which the positive literal α_j follows after the positive literal $\alpha_i, i < j$ (similarly for negative literals), i.e. $\ldots \alpha_i \vee \alpha_j \ldots$. In this case we add a new variable α_k to Z. We insert α_k in the given ordering of Z between the variables derived from these literals, i.e. $i < k < j$. We put the negative literal $\neg\alpha_k$ derived from this new variable into the considered clause, i.e. $\ldots \alpha_i \vee \neg\alpha_k \vee \alpha_j \ldots$. In order to preserve the satisfiability of the newly created logical expression we must add one more clause to C which contains only one positive literal α_k derived from the new variable. This is sufficient to prevent the influence of this variable on the original clause. Repeating this process we obtain the equivalent instance (Z', C') of MSAT from the instance of SAT in a polynomial time. □

Now, we are ready to prove the following theorem:

Theorem

The loading problem for triangular architectures (LTA) is NP-complete for the node function set NAOFs = {OR, NAND}.

Proof:

First, we prove this loading problem to be NP-hard by polynomial time reduction from MSAT which is NP-complete according to the previous lemma. Let (Z, C) be an instance of MSAT where $Z = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the set of n Boolean variables and $C = \{C_1, C_2, \ldots, C_k\}$ is the set of k clauses. For any clause $C_j \in C$ we will use the following simplified notation omiting the index j of the clause

$$C_j = \{L(\alpha_{i_1}), L(\alpha_{i_2}), \ldots, L(\alpha_{i_m})\}$$

where $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ and $L(\alpha_{i_p})$, $p = 1, \ldots m$, is the literal derived from the variable α_{i_p} (i.e. α_{i_p} or $\neg\alpha_{i_p}$). We remember that for any possible p one from the literals $L(\alpha_{i_p}), L(\alpha_{i_{p+1}})$ is positive and the other one is negative because it is a MSAT instance. For (Z, C) to be satisfiable, there must be an assignment $\Pi : Z \longrightarrow \{0,1\}$ such that at least one literal in each clause has value 1.

For any instance (Z, C) of MSAT we construct the corresponding instance (d, T) for LTA in a polynomial time with respect to the size of (Z, C) so that (Z, C) is satisfiable iff (d, T) is performable. The formal construction follows:
Put $d = 2n - 1$ and $T = T_0 \cup T_1$ where

$$\begin{aligned} T_0 &= \{([10]^n, 0)\}, \\ T_1 &= \{(\sigma_j, 1)/\sigma_j \in \{0,1\}^{d+1}, j = 1, \ldots k\}. \end{aligned}$$

The item stimuli σ_j of the partial task T_1 are constructed gradually according to the form of the clause $C_j \in C$. Due to the notation simplicity we again omit the index j of the clause.

I) a) $L(\alpha_{i_1}) = \alpha_{i_1}$
$\sigma^0 = [10]^{i_1 - 1}$

b) $L(\alpha_{i_1}) = \neg\alpha_{i_1}$
$\sigma^0 = [01]^{i_1 - 1}$

II) $p = 1, 2, \ldots, m-1$

a) $L(\alpha_{i_p}) = \alpha_{i_p}$ (i.e. $L(\alpha_{i_{p+1}}) = \neg\alpha_{i_{p+1}}$)
$\sigma^p = \sigma^{p-1} \cdot \mathbf{11}[01]^{i_{p+1} - i_p - 1}$

b) $L(\alpha_{i_p}) = \neg\alpha_{i_p}$ (i.e. $L(\alpha_{i_{p+1}}) = \alpha_{i_{p+1}}$)
$\sigma^p = \sigma^{p-1} \cdot \mathbf{00}[10]^{i_{p+1} - i_p - 1}$

III) a) $L(\alpha_{i_m}) = \alpha_{i_m}$
$\sigma_j = \sigma^{m-1} \cdot \mathbf{11}[01]^{n - i_m}$

b) $L(\alpha_{i_m}) = \neg\alpha_{i_m}$
$\sigma_j = \sigma^{m-1} \cdot \mathbf{00}[10]^{n - i_m}$

The task size is $|T| = k + 1$ so this construction can be done in a polynomial time with respect to the size of the instance (Z, C).

In order to make the preceding formal notation clearer we present an easy example of the above-mentioned construction. Let us consider the instance (Z, C) of MSAT where

$$\begin{aligned} &Z = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}, \quad C = \{C_1, C_2\}, \\ &C_1 = \{\alpha_1, \neg\alpha_3\}, \quad C_2 = \{\neg\alpha_2, \alpha_3, \neg\alpha_4\} \end{aligned}$$

so $n = 4, k = 2$. Then the corresponding instance (d, T) of LTA will take the following form:

$$\begin{array}{lcl} d & = & 7, \\ T_0 & : & (10101010, 0), \\ T_1 & : & (\mathbf{11}010\mathbf{00}10, 1) \\ & & (01\mathbf{001100}, 1). \end{array}$$

The semantics of this construction will be clear from the following. We will denote by

$$f_1, g_1, f_2, g_2, \ldots, f_{n-1}, g_{n-1}, f_n \in \mathrm{NAOFns} = \{\mathrm{OR},\ \mathrm{NAND}\}$$

the relevant node functions for the first-layer nodes in the triangular architecture determined by $d = 2n - 1$. For $t = (s, r) \in T$ denote by $v(t) \in \{0, 1\}^d$

$$v(t) = (f_1(s_1, s_2), g_1(s_2, s_3), \ldots, g_{n-1}(s_{d-1}, s_d), f_n(s_d, s_{d+1}))$$

the vector composed of the first-layer node function values and let $v_0 = v(t),\ t \in T_0$.

First, we prove that if (d, T) is performable then (Z, C) is satisfiable. Assume that the triangular architecture determined by d performs the task $T = T_0 \cup T_1$. It means that there is a relevant configuration for this architecture, especially the node functions for the first-layer nodes. It must hold $v_0 \neq v(t)$ for all $t \in T_1$ because the relevant resulting responses of the item of T_0 and the items of T_1 are different. We know that for both functions $f \in \mathrm{NAOFns} = \{\mathrm{OR},\ \mathrm{NAND}\}$ $f(0, 1)$ is equal to $f(1, 0)$. Therefore by comparing T_0 and T_1 we can see that the node functions $g_i,\ i = 1, \ldots, n-1$ do not help to differentiate between v_0 and $v(t),\ t \in T_1$. Coming back to the above-mentioned example of the construction we ask when $v_0 \neq v(t)$ for all $t \in T_1$? It is just in the case when the following condition is satisfied:

$$(f_1(1,0) \neq f_1(1,1) \ \text{ or } \ f_3(1,0) \neq f_3(0,0))$$

$$\text{and}$$

$$(f_2(1,0) \neq f_2(0,0) \ \text{ or } \ f_3(1,0) \neq f_3(1,1) \ \text{ or } \ f_4(1,0) \neq f_4(0,0)),$$

i.e.

$$(f_1 \equiv \ \mathrm{NAND} \text{ or } \ f_3 \not\equiv \ \mathrm{NAND})$$

$$\text{and}$$

$$(f_2 \not\equiv \ \mathrm{NAND} \text{ or } \ f_3 \equiv \ \mathrm{NAND} \text{ or } \ f_4 \not\equiv \ \mathrm{NAND}).$$

This corresponds with the semantics of the original instance (Z, C) of MSAT when α_i means $f_i \equiv$ NAND. Generally, we define an assignment $\Pi : Z \longrightarrow \{0, 1\}$

$$\begin{array}{rcll} \Pi(\alpha_i) & = & 1 & \text{if } f_i \equiv \text{NAND} \\ & & 0 & \text{otherwise,} \qquad i = 1, \ldots, n \end{array}$$

which confirms the satisfiability of (Z, C).

Conversely, we prove that if (Z, C) is satisfiable then (d, T) is performable. Assume that the instance (Z, C) is satisfiable which is confirmed by an assignment $\Pi : Z \longrightarrow \{0, 1\}$. We choose the following first-layer node functions

$$\begin{array}{rlll} f_i \equiv & \text{NAND} & \text{if} & \Pi(\alpha_i) = 1 \\ & \text{OR} & \text{if} & \Pi(\alpha_i) = 0, \quad i = 1, \ldots, n, \\ & & & \\ g_i \equiv & \text{NAND} & i = 1, .., n-1. & \end{array}$$

All second-layer node functions be NAND functions and all node functions in remaining layers be OR functions. We check this configuration of the triangular architecture determined by d to perform the task $T = T_0 \cup T_1$. Using the preceding consideration we obtain that $v_0 \neq v(t)$ for all $t \in T_1$ because (Z, C) is satisfiable. In this case we know that $v_0 = 1^d$ and the vector composed of the second-layer node function values is 0^{d-1} because of the NAND functions so the relevant resulting response for $t \in T_0$ will be 0 due to the OR functions in remaining layers. Because v_0 differs from $v(t)$, $t \in T_1$, that contains at least one 0 the resulting response for $t \in T_1$ will be 1. Then (d, T) is performable.

This completes the proof that LTA is NP-hard. Finally, it must be demonstrated that LTA $\in NP$, i.e. that there is a nondeterministic machine that can decide the LTA problem in time polynomial in the length of (d, T). Writing down a complete configuration of NAOFns takes at most one bit for each node in the triangular architecture determined by d. Whether the configuration is correct can be checked by evaluating each node function once for each item in T; this takes $O(d(d+1)/2 \cdot |T|)$, which is polynomial as desired. This completes the proof of the whole theorem. □

Corollary

LTA is NP-complete also for the node function sets $\mathcal{F}$ of symmetric and monotone functions (i.e. $\mathcal{F} \subseteq \{0, 1, \text{OR}, \text{ AND}, \text{ NOR}, \text{ NAND}\}$) containing at least $\{\text{OR}, \text{ NAND}\}$ or $\{\text{AND}, \text{ NOR}\}$ functions.

Proof:

We outline the generalization of the proof of the preceding theorem. In the case when $\{\text{AND}, \text{ NOR}\} \subseteq \mathcal{F}$ we modify the construction of the instance (d, T) in the way that we exchange the resulting response bits of the partial tasks T_0, T_1 (i.e. $T_0 = \{([10]^n, 1)\}$, $T_1 = \{(\sigma_j, 0)/\sigma_j \in \{0, 1\}^{d+1}, j = 1, \ldots, k\}$). In this case we replace NAND by AND and OR by NOR in the rest of the proof.

The part of the proof that (Z,C) is satisfiable if (d,T) is performable requires the node function set $\mathcal{F}$ containing only symmetric ($f(0,1) = f(1,0)$, $f \in \mathcal{F}$) and monotone ($f(0,0) \neq f(1,1)$ for non-constant $f \in \mathcal{F}$) functions. For such a general $\mathcal{F}$ we define the assignment $\Pi : Z \longrightarrow \{0,1\}$

$$\begin{array}{rcll} \Pi(\alpha_i) & = & 1 & \text{if } f_i \equiv \text{NAND or } f_i \equiv \text{AND} \\ & & 0 & \text{otherwise}, \qquad\qquad\qquad i = 1,\ldots,n. \end{array} \qquad \square$$

References

Baum, E.B. 1991. Book Reviews. IEEE Transactions on Neural Networks, 2, 181 - 182.

Balcázar, J.L., Díaz, J., Garabarró, J. 1988. Structural Complexity I. Springer-Verlag Berlin Heidelberg.

Blum, A., Rivest, R.L. 1992. Training a 3-Node Neural Network is NP-Complete. Neural Networks, 5, 117 - 127.

Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M. 1986. Classifying Learnable Geometric Concepts with the Vapnik-Chervonenkis Dimension. In Proc. of the 18$^{\text{th}}$ Annual STOC, New York, 273 - 282.

Judd, J.S. 1990. Neural Network Design and the Complexity of Lerning. The MIT Press, Cambridge.

Kearns, M., Li, M., Pitt, L., Valiant, L.G. 1987. On the Learnability of Boolean Formulae. In Proc. of the 19$^{\text{th}}$ STOC, 285 - 295.

Pitt, L., Valiant, L.G. 1988. Computational Limitations on Learning from Examples. Journal of ACM, 35, 965 - 984.

Rumelhart, D.E., Hinton, G.E., Williams, R.J. 1986. Learning Representations by Back-Propagating Errors. Nature, 323, 533 - 536.

Svaytser, H. 1989. Even Simple Neural Nets Cannot Be Trained Reliably with a Polynomial Number of Examples. In Proc. of IJCNN, Washington, II, 141 - 145.

Valiant, L.G. 1984. A Theory of the Learnable. Communications of the ACM, 27, 1134 - 1142.

Valiant, L.G. 1985. Learning Disjunctions of Cojunctions. In Proc. of the 9$^{\text{th}}$ IJCAI, Los Angeles, 560 - 566.

Wiedermann, J. 1991. Some Afterthoughts on Computational Learning Theory. In Proc. of SOFSEM, Jasná pod Chopkom (Slovakia), 271 - 274.