SIMPLE DEDUCTIVE REASONING TESTS AND DATA SETS FOR EXPOSING LIMITATION OF TODAY'S DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning for Deductive Reasoning is an open problem not yet explicitly called out in the machine learning world today. Deductive reasoning involves storing facts in memory and generation of newer facts over time. The concept of memory in deduction systems is fundamentally different from the representation of weights in a deep neural network. A majority of the machine learning models are inductive reasoning models including state of the art deep neural networks which are effectively tensor interpolation based models. The architecture of the underlying neuron today, can only enable it to perform simple arithmetic or a comparison test against a constant. A baby step towards realization of memory is through recurrent neural networks and its variants, however the formal representation is not sufficient enough to capture complexities. A success story of having deductive reasoning in the inductive reasoning based machine learning is the feature engineering step, however deep neural networks have tall claims to do away with this step. In this context, it is important to clarify what is and what is not possible with inductive machine learning approaches through rigorous studies and present to the scientific community in the form of data sets. In this paper, we have created 10 data sets for - (a) selection (3 data sets) - minimum, maximum and top 2nd element in an array of numbers; (b) matching (3 data sets) - duplicate detection, counting and histogram learning; (c) divisibility tests (2 data sets) - divisability of two numbers and divisability by 3; (d) representation (2 data sets) - binary representation and parity. Though extremely simple in terms of feature engineering, in all of these tests, simple deep neural networks, random forest and recurrent neural networks have failed with very low accuracies. This clearly calls out representational limitations of the deep neural networks when it comes to memory. We propose these tests as the Devil's Ten Tests on top of which newer methods can be introduced for Learning Models for Deductive Reasoning.

1 INTRODUCTION

Deductive reasoning is a branch of artificial intelligence where inferences are represented as assertions or facts over data (Khemani, 2013). Starting with a set of given facts, the system combines facts based on rules to generate newer facts and update the knowledge store. On the other hand machine learning algorithms employ induction based approaches which are predominantly pattern mapping methods (McClelland et al., 1986) Fundamentally, in a pipeline of operations, the vectors are are arithmetically combined, scaled up or scaled down and mapped to the target vector of interest. However for the sake of convenience and interpretation, a vector is typically represented as a tensor, although the ideas of vector transformations remain the same, as a pipeline of tensor transformations. Today deep neural networks are performing tasks almost at the level of human or even better in computer vision and other forms of data (Bengio et al., 2017)).

However, today a popular false opinion has formed in a typical engineer using deep neural networks that feature engineering is no longer required and they can accomplish any type of pattern mapping problem. While several common problems are indeed accomplished, the problem statements are fundamentally of inductive reasoning or interpolation type in nature. The deep neural networks with today's notion of a neuron are not suitable for deductive reasoning. In most popular architectures

S.No	Data set (N=100000, K=50)	Algorithm invocation
1	MIN	SEL(”min”, N, K)
2	MAX	SEL(”max”, N, K)
3	TOP2	SEL(”top 2”, N, K)
4	BINARY	REP(”parity”,N,K)
5	SORT	SORT(N,K)
6	COUNT	MAT(”count”,N,K)
7	DIV	DIV(”divisible”,N,K)
8	DIV3	DIV(”divisible_by_3”,N,K)
9	PARITY	REP(”parity”,N,K)
10	DUP	MAT(”duplicate”,N,K)

Table 1: Data set name and algorithm that generated the data set are shown here.

and practices, a neuron used in the state of the art (as of this article), is only capable of doing only simple arithmetic operations such as a sigmoid or a tanh transformation or comparison against a constant value as in case of a rectified linear unit. Such a representation is insufficient to capture the notion of knowledge store in a deductive reasoning methodology.

A knowledge store in a deductive reasoning methodology is a storage for facts which are modified over time. For instance, counting number of ones in a binary representation of the input, the current count is a fact and as the algorithm iterates over input, the count value gets updated. Most of the iterations over input, can be represented in a declarative style as first order logic statements such as prolog (Bratko, 2001). The weight space representation of a deep neural network does not capture the facts, unification and inference mechanism as required by a deductive reasoning methodology. The earlier version of machine learning formulations involving *feature engineering*, accommodated for deductive reasoning in the form of outputs of deduction centered programs which are added as features. However, there is an on going myth that deep neural networks will do away with feature engineering that needs to be addressed in a more specific and detailed manner. A baby step in the direction of storage of past information in data is a recurrent neural network formulation and its several variants (Mikolov et al., 2011). However to the best of our knowledge and exploration, today there is no RNN formulation which is meant to learn facts, unification and deductive inferences.

It is also important at this stage of the state of the art, to clearly call out limitations of the existing deep neural networks and machine learning formulations on what is and what is not possible in a more specific way. As machine learning systems operate over data sets, the first step in demonstrating what is not possible is via simple data sets for pattern mapping. We have created some very simple data sets where a single engineered feature is sufficient enough to capture the pattern, however deep networks fail to capture the deduction patterns. The data sets are for - (a) selection (3 data sets) - minimum, maximum and top 2nd element in an array of numbers; (b) matching (3 data sets) - duplicate detection, counting and histogram learning; (c) divisibility tests (2 data sets) - divisability of two numbers and divisability by 3; (d) representation (2 data sets) - binary representation and parity. We demonstrate that in all of these data sets, the deep neural networks fail with very low accuracies. The experiments clearly call out newer methods for *learning deduction patterns* in input data.

2 METHODS

In order to present how deductive reasoning is fundamentally different from interpolation based reasoning, we have created ten data sets which are described in this section. Data sets are generated by invoking the algorithm as shown in the (Table 1).

2.1 DATA SETS FOR SELECTION PROBLEMS

Identification of maximum or minimum element in an array of numbers, requires facts to be recorded in a storage, infer on top of them in the current iteration and update the storage with newer facts. The data set consists of $D = (x_i, y_i)$ tuples $i \in [1..N]$ where $N = 100000$. Each $x_i \in R^K$ is a K

dimensional vector where $K = 50$ in our case and y_i contains the value of an element of interest in the array, i.e. $y_i = max(x_{i,1}, \ldots, x_{i,K})$.

The data set is generated as shown in the schematic (Algorithm 1).

Algorithm 1 Selection Data Set Generation Algorithm - SEL(T,N,K)

Require: T, N, K
  $(\forall i \in [1..N]), (\forall j \in [1..K]) x_{i,j} = RAND()$ //RAND() function generates a random number
  //T: Type of selection, N: Number of elements in data set, K: Dimensionality of each point
  if T is "max" **then**
    $(\forall i \in [1..N]) : y_i = max(x_i)$
  else if T is "min" **then**
    $(\forall i \in [1..N]) : y_i = min(x_i)$
  else if T is "top 2" **then**
    $(\forall i \in [1..N]) : y_i = max(set(x_i) - \{max(x_i)\})$
  end if

2.2 Data Sets for Matching Problems

The data set is almost same as for the selection problem, however here, duplicate detection and histogram learning are carried out. This requires the numbers to be remembered and compared against other elements. The Duplicate data set contains of $D = (x_i, y_i)$ tuples $i \in [1..N]$ where $N = 100000$. Each $x_i \in R^K$ is a K dimensional vector where $K = 50$ in our case and y_i contains either 1 or 0. 1 if a element $j \in x_i$ is present more than once.
The Histogram data set contains of of $D = (x_i, y_i)$ tuples $i \in [1..N]$ where $N = 100000$. Each $x_i \in R^K$ is a K dimensional vector where $K = 50$ in our case and y_i contains the count of elements which lies between 1 and 10. The data set is generated as shown in the schematic (Algorithm 2).

Algorithm 2 Matching Data Set Generation Algorithm - MAT(T,N,K)

Require: T,N,K
  $(\forall i \in [1..N]), (\forall j \in [1..K]) x_{i,j} = RAND()$ //RAND() function generates a random number
  //T:Type of Matching, N: Number of elements in data set, K: Dimensionality of each point
  if T is "duplicate" **then**
    $(\forall i \in [1..N]), y_i = 0, (\forall j, k \in [1..K], j \neq k \ \& \ \& \ x_i[j] == x_i[k]) : y_i = 1$
    for $(\forall i \in [1..N])$ **do**
      //check if a element is present more than one time
      $y_i = 0$
      for $\forall j \in [1..K]$ **do**
        for $(\forall k \in [1..K])$ **do**
          $(j \neq k \ \& \ \& \ x_{i,j} == x_{i,k})$
          $\rightarrow y_i = 1$
        end for
      end for
    end for
  end if
  if T is "count" **then**
    for $(\forall i \in [1..N])$ **do**
      //count number of vector elements ¡= 10
      for $(j \in [1..K])$ **do**
        $x_{i,j} <= 10 \rightarrow y_i = y_i + 1$
      end for
    end for
  end if

2.3 DATA SETS FOR DIVISIBILITY PROBLEMS

For divisibility tests the facts are to be recorded in storage and new element is processed based on this fact and after processing, update the storage with the new fact. For example if we want to perform divisibility test if 1342 is divisible by 11. At first 13 will be evaluated against 11 it will give a remainder 2 which is new fact that we need to store and move on to next element and that's 4. Combining this element with our fact we should get 24 and that will be evaluated against 11 and will give remainder or new fact as 2. And this keeps on going until no element is left.
The Divisibility data set contains of $D = (x_i, y_i)$ tuples $i \in [1..N]$ where $N = 100000$. Each x_i contains a tuple of two elements $x_i = a, b$ and y_i contains either 0 and 1. 1, if $(a \mod b == 0)$ and 0, if $(a \mod b \neq 0)$.

The data set is generated as shown in the schematic (Algorithm 3).

Algorithm 3 Divisibility tests Data Set Generation Algorithm - DIV(T,N)

Require: T,N
  if T is "divisible" **then**
    for $(\forall i \in [1..N])$ **do**
      $(x_i = [RAND(), RAND()])$
      $y_i = 0$
      $(x_{i,0} \mod x_{i,1} == 0) \rightarrow y_i = 1$
    end for
  end if
  if T is "divisibility_by_3" **then**
    for $(\forall i \in [1..N]), x_i = RAND()$ **do**
      $y_i = 0$
      $(x_i \mod 3 == 0) \rightarrow y_i = 1$
    end for
  end if

2.4 DATA SETS FOR REPRESENTATION PROBLEMS

This contains two data sets $i)$ parity data set and $ii)$ binary representation data set. The parity data set contains of $D = (x_i, y_i)$ tuples where $x_i \in R$ and y_i denotes the parity of the x_i. Parity of a number is based on the number of 1 present in it's binary representation. If number of 1 is even then parity of that will be 0. And if the number of 1 is odd then parity of that will be 1. For example if we take 17. It's binary representation is 10001. It contains 2 one's in its binary representation so parity of 17 is 0. The binary representation data set contains of $D = (x_i, y_i)$ tuples where $x_i \in R$ and $y_i \in 0, 1^K$ where K is 20 in our case denotes its binary representation. For example if x_i is 20 and K is 7 then y_i will be 0010100.

The data set is generated as shown in the schematic (Algorithm 4).

2.5 DATA SET FOR SORTING PROBLEM

This data set consists of x_i as a list of numbers, mapped to its sorted equivalent y_i. The idea is sorting is a very prominent pattern to humans, however basic neurons are not able to capture all of the the element to element relationships. The algorithm for sorting is shown in (Algorithm 5). For example, if $x_i = [1, 3, 2, 1, -9]$ then sorted $y_i = [-9, 1, 1, 2, 3]$.

2.6 TRAIN AND TEST DATA SET PARTITIONS

The machine learning model is tested based on a bit lenient train and test partitions of 90% and 10% respectively. The training data size has been increased so as to allow the classifier to make use of more number of patterns as the data sets are tough for any induction based classification formulation today.

Algorithm 4 Representation Data set Generator- REP(T,N,K)

Require: T,N,K
  if T is "binary" **then**
    for $\forall i \in [1..N]$ **do**
      $x_i = RAND()$ //single number
      y_i is binary representation of x_i in K bits.
    end for
  end if
  if T is "parity" **then**
    for $(\forall i \in [1..N])$ **do**
      $x_i = RAND()$
      s =binary representation of x_i in K bits.
      r =count number of 1 in s.
      $y_i = 0$
      $(r \mod 2 \neq 0) \rightarrow y_i = 1$
    end for
  end if

Algorithm 5 Sorting Test Data Set Generation Algorithm - SORT(N,K)

  for $\forall i \in [1..N]$ **do**
    $\forall j \in [1..K] : x_{i,j} = RAND()$
    $y_i = SORT(x_i)$ //Sorted version of x_i in ascending order
    $(\forall j, k) : (j < k) \rightarrow y_{i,j} < y_{i,k}$
  end for

3 RESULTS

The performance of machine learning models on each of the data sets is computed in two forms - regression and classification. For those problems where the final output is a numeric quantity, the problem is posed as regression and Root Mean Squared Deviation (RMSD) values are reported. The problems posed as regression are selection problems, binary representation of a number, sorting and count of numbers between 0 and 10 and presented in (Table 2). For the problems where the output is a decision, they are posed as classification problem in our setting. The problems posed as classification are divisibility of two numbers, divisibility by 3, parity and detection of duplicate and presented in the (Table 3).

Except for the parity problem where a random forest has performed with $> 92\%$ accuracy, in rest of the places the three methods chosen for evaluation have all failed to give 100% accuracies although the data sets are quite simple.

It is worth discussing, the parity problem in more detail on which random forest appeared to have worked (the * marked observation), however we can readily observe that anything less than 100% would not be of any practical use and not reliable. Consider a binary representation of input of length K elements. The total number of variations of the input is 2^K. In this case, the input dimensionality is $K = 50$, which corresponds to a data set of 2^50 elements amounting to a total storage cost

S.No	Dataset Name	RMSD of RF	RMSD of NN	RMSD of RNN
1	MIN	7.2788	28.2572	20.378
2	MAX	7.2788	28.398	24.397
3	TOP2	3.912	28.455	62.81
4	BINARY	∞	∞	∞
5	SORT	∞	∞	∞
6	COUNT	10.90	28.50	4.4312

Table 2: Performance of data sets on RNN, NN and RF based on regression formulation. Root Mean Squared Deviation (RMSD) values are computed on the output predictions.

S.No	Dataset Name	Accuracy of RF	Accuracy of NN	Accuracy of RNN
7	DIV	0.691	0.5133	0.5015
8	DIV3	0.6668	0.6566	0.6668
9	PARITY	0.926*	0.6836	0.5279
10	DUP	0.4962	0.67	0.5427

Table 3: Performance of data sets on RNN, NN and RF based on classification formulation. Accuracy values are reported on the output predictions.

of $50 \times 4 \times 2^{50} = 200$ *peta bytes*. When the input dimensionality becomes 1000, which is not uncommon, the storage requirement for data grows to $1000 \times 4 \times 2^{1000}$ which is an astronomical number. A decision tree to successfully learn with 100% certainty requires all input patterns to be available in training. The size of the tree in terms of the number of nodes is $2^{K+1} - 1$. Storing a tree having $2^{51} - 1$ nodes has practical limitations for the problem it is trying to address, *as simple as parity* computation. However *a simple for loop* through the data requires only one pass through the input and requires only *a constant space storage* of a couple of floating point variables and predicts with 100% certainty. Having an additional feature in the input, feature engineered to compute parity (i.e. $K + 1$ dimensional input), would reduce the amount of training data requirement to just $K + 1$ points to result in 100% accuracy of any standard classifier.

4 Conclusions

We have presented here an issue with limitations of the present day deep neural networks to address deduction based inferences. Today deep neural networks are positioned to do away with feature engineering however we have demonstrated that this is not true. We have called out that deep neural networks are mainly vector arithmetic and interpolation based models with a simple neuron at the core to perform arithmetic operations or comparison against a constant number. The notion of knowledge representation as storage of facts and deductive reasoning is not captured by the weight space representation of neural networks. We have demonstrated failure of typical deep neural networks, random forest and recurrent neural network formulations on data sets which require deduction reasoning based features. The data sets are for - (a) selection (3 data sets) - minimum, maximum and top 2nd element in an array of numbers; (b) matching (3 data sets) - duplicate detection, counting and histogram learning; (c) divisibility tests (2 data sets) - divisibility of two numbers and divisibility by 3; (d) representation (2 data sets) - binary representation and parity. We also discussed the parity problem and impracticality of storage and computation to achieve full accuracy by a strong classifier such as a decision tree unless a simple computation friendly feature is engineered. We have presented a case for novel models for *learning deductive reasoning from examples.* These novel models can be a combination of deduction and induction systems operating together as a model on top of an evolutionary computational framework.

References

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Massachusetts, USA:, 2017.

Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.

Deepak Khemani. *A first course in artificial intelligence*. McGraw-Hill Education, 2013.

James L McClelland, David E Rumelhart, PDP Research Group, et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5528–5531. IEEE, 2011.