MathDQN: Solving Arithmetic Word Problems via Deep Reinforcement Learning

Lei Wang [#], Dongxiang Zhang [#*], Lianli Gao [#], Jingkuan Song [#], Long Guo [†], Heng Tao Shen [#]
[#] Center for Future Media and School of Computer Science & Engineering, UESTC, China
[†] Key Lab of High Confidence Software Technologies (MOE), Peking University, China
demolei@outlook.com {zhangdo,lianli.gao}@uestc.edu.cn jingkuan.song@gmail.com
guolong@pku.edu.cn shenhengtao@hotmail.com

Abstract

Designing an automatic solver for math word problems has been considered as a crucial step towards general AI, with the ability of natural language understanding and logical inference. The state-of-the-art performance was achieved by enumerating all the possible expressions from the quantities in the text and customizing a scoring function to identify the one with the maximum probability. However, it incurs exponential search space with the number of quantities and beam search has to be applied to trade accuracy for efficiency.

In this paper, we make the first attempt of applying deep reinforcement learning to solve arithmetic word problems. The motivation is that deep Q-network has witnessed success in solving various problems with big search space and achieves promising performance in terms of both accuracy and running time. To fit the math problem scenario, we propose our MathDQN that is customized from the general deep reinforcement learning framework. Technically, we design the states, actions, reward function, together with a feed-forward neural network as the deep Q-network. Extensive experimental results validate our superiority over state-of-the-art methods. Our MathDQN yields remarkable improvement on most of datasets and boosts the average precision among all the benchmark datasets by 15%.

Introduction

Automatically solving math word problems (MWP) has a long history dating back to 1960s (Bobrow 1964) and still continues to attract researchers' attention in recent years. It requires mapping the human-readable words into machine-understandable logic forms, followed by an inference procedure, and cannot be simply solved by pattern matching or end-to-end classification techniques. Thus, designing an automatic MWP solver, with semantic understanding and inference capability, has been considered as a crucial step towards general AI.

As an early attempt, ARIS (Hosseini *et al.* 2014) considers that verbs play an important role for operator calssification. It uses syntatic analysis to identify relevant entities and their values and extracts verb-related features for SVM-based categorization. An alternative template-based method was proposed in (Mitra and Baral 2016). It defines three types of formulas for common scenarios in Add-Sub problems and maps the identified variables and their attributes to the formulas. Unfortunately, the generality of these two methods is rather limited as they only support two types of operators (i.e., addition and subtraction). The tag-based approach (Liang *et al.* 2016) maps rules to convert identified variables and values into logic forms, which are further transformed into logic statements for inference. It shows better generality but still requires considerable human intervention in tag annotation and rule generation.

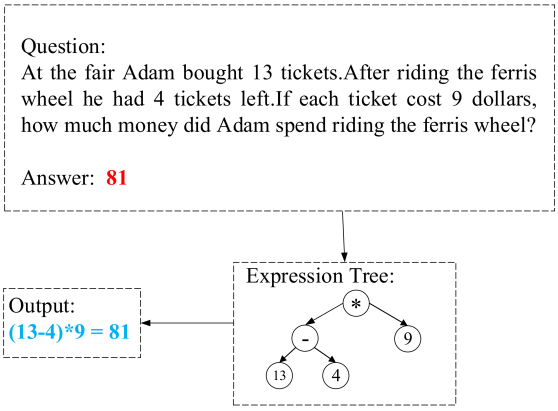

Figure 1: Expression tree for multi-step arithmetic problem.

The state-of-the-art methods for simple arithmetic problems are algorithmic and tree-based. In (Roy and Roth 2015), the multi-step arithmetic problem is decomposed into an expression tree, as illustrated in Figure 1. It builds two types of classifiers, one to identify relevant quantities as leaf nodes and the other for iterative operator selection to construct the tree in a bottom-up manner. In this example, relevant quantities $(13, 4, 9)$ are first identified and the operators in the higher level are determined by an SVM classifier. A scoring function is presented to identify the best expression tree as the result. (Koncel-Kedziorski *et al.* 2015) adopts a more brutal fashion by enumerating all the possible equa-

*Corresponding Author: Dongxiang Zhang

tion trees with integer linear programming. Then, a scoring function is proposed to rank the candidate trees and champion the best one as the answer. Recently, UnitDep (Roy and Roth 2017) augments the scoring function of the expression tree in (Roy and Roth 2015) by further considering the consistence of rate unit associated with the quantities. A new concept named Unit Dependency Graphs (UDGs) was proposed to better capture and reason about units. However, the drawback of these tree-based methods is that the search space grows exponentially with the number of quantities and normally beam search is utilized to trade accuracy for efficiency.

In this paper, we make the first attempt of applying deep reinforcement learning (deep RL) to solve arithmetic word problems. The underlying motivation is that deep Q-network has witnessed success in solving various problems with big search space such as playing text-based games (Narasimhan *et al.* 2015), information extraction (Narasimhan *et al.* 2016), text generation (Guo 2015) and object detection in images (Caicedo and Lazebnik 2015). We consider the arithmetic word problem solver a suitable application for deep RL and propose our MathDQN to leverage the strengths of deep Q-network (DQN). Technically, we tailor the definitions of states, actions, and reward functions which are key components in the reinforcement learning framework. By using a two-layer feed-forward neural network as the deep Q-network to approximate the Q-value function, our framework learns model parameters from the reward feedback of the environment.

To sum up, the main contributions in this work are listed as following:

- To the best of our knowledge, this is the first paper to apply deep reinforcement learning as a general framework to solve math word problems.
- Technically, we customize the principle components in the DQN framework, including states, actions, rewards and network design.
- We conduct experiments on the popular benchmark datasets and the results show that our method is both efficient and accurate. It improves the average precision among all the math problems by 15%.
- We make our implementation code available in Github[1].

Related Work

In this section, we review literature upon automatic algebra word problem solver and present background information on deep reinforcement learning as well as its applications.

Algebra Word Problem Solver

The algebra word problems for primary schools fall into two categories: the basic arithmetic problems with one unknown variable and four types of operators $+$, $-$, $\times$ and $\div$, and the complex equation set problems that involve multiple unknown variables.

[1] https://github.com/uestc-db/DQN_Word_Problem_Solver

Arithmetic Problem Solver ARIS (Hosseini *et al.* 2014) and Formula (Mitra and Baral 2016) are two systems that only support the operators of addition and subtraction. The former uses verb categorization and the latter utilizes formulas defined in advance. To improve the generality, tag-based approach (Liang *et al.* 2016) was proposed with map rules to convert identified variables and values into logic forms, which are further transformed into logic statements for inference. Reasoning techniques are studied in (Shi *et al.* 2015).

(Roy and Roth 2015) is the first algorithmic approach that can handle arithmetic problems with multiple steps and operations, without depending on additional annotations (formulas or tags). It builds an expression tree and trains two classifiers for quantity relevance prediction and operator classification, respectively. A scoring function is proposed to rank the candidate trees and champion the best one as the answer. (Koncel-Kedziorski *et al.* 2015) adopts a more brutal fashion by enumerating all the possible equation trees with integer linear programming. UnitDep (Roy and Roth 2017) is the state-of-the-art solution. It further takes into account the consistence of rate unit associated with the quantities and treats it as a scoring factor. However, the search space of the tree-based methods grows exponentially with the number of quantities. To resolve the issue, beam search can be adopted to reduce running time but in sacrifice of accuracy.

In this paper, our research scope includes how to efficiently and effectively solve the arithmetic problems with deep Q-network. We will explicitly present our MathDQN solution in the next section.

Equation Set Problem Solver The equation set problems are more challenging because they involve multiple unknown variables to resolve. In (Kushman *et al.* 2014), a template-based solution was proposed. Given a corpus of predefined equation sets with unknown slots for variables and numbers, it finds a matching template and infers the unknown slots from text information. In (Zhou *et al.* 2015), an improved method was proposed to reduce the hypothesis space by only enumerating the permutation of number slots. Without any inference capability, these template-based methods are rather rigid and not extensible for complex scenarios. Upadhyay et al. proposed a structured-output learning framework to learn both explicit and implicit signals jointly (Upadhyay *et al.* 2016).

Overall, these methods are tuned for small benchmark datasets to achieve promising results. According to a recent experimental study (Huang *et al.* 2016), their accuracies degrade sharply in a larger and more diverse dataset, named Dolphin, harvested from community question-answering web pages. The findings imply that this line of research still has great room for improvement and calls for more general and robust solutions.

Deep Reinforcement Learning

In reinforcement learning (RL), given a set of internal states $S = s_1, \ldots, s_m$ and actions $A = a_1, \ldots, a_n$, the agent iteratively takes action a at state s and moves to a new state s' until a termination condition is satisfied. This process is guided by certain policies or rules π learned by interacting

with the environment E. Rewards will be given to positive actions and we measure the true value of an action in state s as

$$Q_\pi(s,a) = \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi]$$

where $\gamma \in [0,1]$ is a discounted factor for future rewards. The objective is to maximize the expected sum of future rewards through a sequence of actions.

Q-Learning is model-free and learns an optimal action-value function $Q(s,a)$. However, Q-value function needs to trace all possible state-action pairs, which is prohibitive. Mnih et al. introduced the Deep Q-Network (DQN) (Mnih *et al.* 2015) to approximate the Q-value function with a non-linear multi-layer convolutional network. Given state s, DQN outputs a vector of action values $Q(s,\cdot;\theta)$, where θ are the parameters of the network. For an m-dimensional state space S and an action space A containing n actions, the neural network serves as a function from $\mathbb{R}^m$ to $\mathbb{R}^n$. An experience replay memory is maintained to store previous transitions. In the mini-batch training stage, the observed transitions are sampled uniformly from the memory bank to update the network.

Due to its generality and robustness for sparse datasets, deep reinforcement learning has been successfully applied to solve a wide range of problems, including playing text-based games (Narasimhan *et al.* 2015), information extraction (Narasimhan *et al.* 2016), text generation (Guo 2015) and object detection in images (Caicedo and Lazebnik 2015). In this paper, we make the first attempt to utilize DQN in the problem of algebra problem solver and achieve superior performance over state-of-the-art methods.

Proposed Solution

The proposed framework of deep reinforcement learning is depicted in Figure 2. Given a math word problem, we adopt quantity schema (Roy and Roth 2015) to identify relevant quantities which serve as the bottom layer of expression tree. The irrelevant quantities will be discarded from tree construction. If there are at least three quantities, we propose a **re-order** mechanism to sort the quantities according to their construction order in the expression tree. For example, given an expression $3 + 4 \times 5$, we need to first construct the subtree for $4 \times 5 = 20$. After that, this subtree with intermediate result 20 is further merged with the remaining quantity 3 to generate the final output. Inspired by UnitDep (Roy and Roth 2017), we first determine whether a quantity refers to rate using their proposed rules. For example, in the text "*5 dollars an hour*", quantity 5 is associated with rate "*dollars an hour*". If there are two quantities associated with the same rate, we assign them the highest priority and they will be placed at the beginning of the sorted list. If there are two quantities whose unit is the denominator of a rate associated with the third quantity, we assign higher priority to the first two quantities. For example, in the word problem "*Sam was collecting cans for recycling. On Saturday he filled 3 bags up and on Sunday he filled 4 more bags. each bag had 9 cans in it, how many cans did he pick up total?*", 3 and 4 are associated with unit "bag" and they will be handled earlier than the

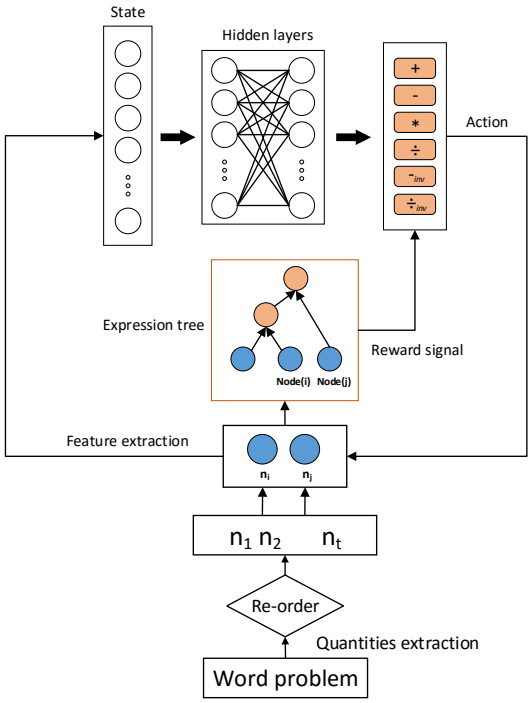

Figure 2: Deep reinforcement learning framework for MWP.

third quantity 9 with rate "cans in each bag". For the remaining cases, we simply sort them by their occurrence positions in the problem text.

Thereafter, we use the sorted quantities as the bottom level and our objective is to build an expression tree using DQN framework. Intuitively, we can represent state in reinforcement learning as the partial tree constructed so far. If the next state or partial tree gets closer to the final "groundtruth" tree, the environment returns positive rewards. Otherwise, negative rewards are returned as punishment. However, it is challenging to vectorize a partial tree with unknown dimensions and send it to a deep Q-network for parameter learning. To resolve the dimension issue, we choose to select a pair of quantities for tree construction, from which we can derive fixed-size features to represent state. For instance, instead of decomposing $(13-4)\times 9$ into $13-4$ followed by 9×9, we wish to select 13 and 4 in the first iteration and determine its operator. In the next step, either pair $(13, 9)$ or $(4, 9)$ may be selected to determine the operator for their lowest common ancestor, i.e., root node of expression tree in this example.

In our deep reinforcement learning framework, we use real-valued vector that constitutes features of selected quantity pair to represent state. Its associated action is to determine a proper operator for these two quantities. In the tree construction, we iteratively select pairs of quantity and its operator through feedback of environment in the form of positive or negative rewards. For DQN, we construct a two-layer feed-forward neural network to calculate the expected Q-value. The parameters of the DQN are learnt using gradient descent of the loss function to reduce the discrepancy between Q-value predicted by DQN and the target optimal Q-

Table 1: Extracted features of quantity pairs for state construction.

Feature	Description
Individual feature	It contains fields indicating the associated verb with the quantity; whether the quantity has a rate unit; whether the rate unit present in the question; and whether any adjective, comparative or adverb word is around the quantity.
Pair feature	It contains fields indicating whether the pair of quantities have the same verb mention; whether the two quantities have the same unit; whether the tokens of the one's unit present in the other's rate unit; whether the value of the former quantity is greater than the value of the other.
Question feature	It contains fields indicating whether any comparison token (e.g., "more", "less" or "than") or rate token (e.g., "each" or "one") presents in the question.

value. In the following, we explicitly explain the states, actions, rewards, and training procedure tailored for the math word problem solver.

States

In previous efforts of deep reinforcement learning, such as information extraction (Narasimhan *et al.* 2016) and text game (He *et al.* 2016), states were represented as real-valued vector with fixed number of dimensions. These vectors, combined with an interaction function, were fed into a neural network to approximate the Q-function in reinforcement learning.

In this paper, we adopt such state construction strategy and concatenate features of selected quantity pair to represent state, which is used as input of a two-layer feed-forward neural network to approximate Q-function. The crafted features are similar to the quantity schema proposed in (Roy and Roth 2015). As shown in Table 1, three types of features for individual quantity, pair quantities, and questions are extracted for state construction. These features can be automatically generated from a given math problem by analyzing its derived parser tree using tools such as Stanford Parser (Socher *et al.* 2013a), and they are helpful in action selection to determine a correct operator node.

In addition, we add two dimensions in the state vector to provide additional clues for action selection. Each dimension is used to indicate whether the associated quantity has appeared in the partial expression tree constructed so far. In other words, the flag values indicate the level of operator node to be determined. If both flags are unset, we are constructing an operator node for two quantities in the leaf level. If one of them is set, we are determining operators in interval tree nodes.

Actions

In each step, the agent takes actions to select two quantities with the maximum expected reward and determine its lowest common ancestor which is essentially an operator. Since we handle simple arithmetic problems, there are four basic types of operators involved: addition $+$, subtraction $-$, multiplication $\times$ and division $\div$.

We also notice that the quantities in the bottom level of expression tree may not follow the same order of their occurrence in the math word problem. Thus, we introduce two new operators: reverse subtraction $-_{inv}$ (i.e., $a -_{inv} b = b - a$) and reverse division $\div_{inv}$ (i.e., $a \div_{inv} b = b \div a$). There is no need to define additional reserve operators for addition and multiplication because $a + b = b + a$ and $a \times b = b \times a$.

Reward Function

In reinforcement learning, the agent receives positive or negative rewards from the environment for each action selected during the training stage. The reward is leveraged in the loss function to calculate the target optimal Q-value, whose discrepancy with the predicted Q-value will be used as feedback to adjust parameters in DQN. Our definition of reward function is straightforward. If the selected operator is correct for its two associated quantities, the environment returns a positive reward. Otherwise, a negative reward is returned as punishment.

Parameter Learning

We use a two-layer feed-forward neural network as the deep Q-network (DQN) to calculate the expected Q-value. The network parameters θ will be learned from the feedback of environment in the form of rewards. We also maintain an experience replay memory $\mathcal{D}$ to store state transitions. To perform update of θ, we sample a batch of transitions (s, a, s', r) at random from $\mathcal{D}$. With the min-batch sampling, the model is updated periodically by minimizing the loss function:

$$L_t(\theta_t) = \mathbb{E}_{s,a}[(y_t - Q(s, a; \theta_t))^2]$$

where $y_t = r + \gamma \max_{a'} Q(s', a'; \theta_{t-1})$ is the target optimal Q-value, which is calculated by summing up the current reward r and the optimal Q-value in the subsequent step. The expectation is over the sampled transitions (s, a, s', r).

The parameters θ of the DQN are learnt using gradient descent of the loss function to close the gap between Q-value predicted by DQN and the target optimal Q-value derived from the Bellman equation:

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}[(y_t - Q(s, a; \theta_t)) \nabla_{\theta_t} Q(s, a; \theta_t)]$$

Training

To sum up, we illustrate the complete training procedure via DQN in Algorithm 1. In the initialization step, we create an experience replay memory $\mathcal{D}$ to store transitions (s, a, r, s') and an action-value function Q with random weight. The whole training dataset is passed through for M epochs. In

each epoch, we traverse the math word problems and extract their quantity schema. The irrelevant quantities are discarded and the remaining are used for expression tree construction. The features used for relevance classifier include fields whether the unit of the quantity appears in the question; whether other quantities have better match with the question, the number of quantities with the best match with the question and certain miscellaneous features.

We adopt ϵ-greedy strategy to obtain trade-off between exploration and exploitation in reinforcement learning. With probability ϵ, a random action is selected for exploration. In our implementation, we follow (Mnih *et al.* 2013) to linearly anneal ϵ (say from 1 to 0.1) with more epochs. However, since the exploration space of expression tree construction is not as huge as video game playing, we gradually reduce ϵ from 0.5 to 0.01 to prefer exploitation. With the selected operator, we obtain reward r_t and new state s_{t+1}. This transition is stored in replay memory $\mathcal{D}$. Note that if $\mathcal{D}$ is full, we only pop transitions with reward $r_t < 0$ to create space for the new transition. This is because the number of negative transitions is much higher than that of positive rewards. When popping a negative reward, the percentage (sampling probability) for both types of transitions always remains the same. After that, we uniformly sample transitions in $\mathcal{D}$ to update parameters in DQN. In particular, we update the optimal target Q-value with the current reward, calculate the loss and use gradient descent to update θ. The training procedure for a math word problem terminates if all the extracted relevant quantities have been selected in the expression tree.

Algorithm 1: Training Procedure via Deep Q-Network

Initialize an experience replay memory $\mathcal{D}$
Initialize an action-value function Q with random weight
for $episode = 1, M$ **do**
  for $w = 1, W$ **do**
    Extract quantity schema from Q_w and remove irrelevant quantities.
    for $t = 1, T$ **do**
      if $random() < \epsilon$ **then**
        Select a random action a_t
      else
        Select $a_t = \max_a Q(\phi(s_t), a; \theta)$
      end if
      Obtain reward r_t and new state s_{t+1} by executing action a_t
      Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
      Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
      if $\phi(s_{j+1})$ is terminal **then**
        $y_j = r_j$
      else
        $y_j = r_j + \gamma \max_{a'} Q(\phi(s_{j+1}), a'; \theta)$
      end if
      Perform gradient descent on the loss $L(\theta)$ based on y_j
      if all the relevant quantities have been selected **then**
        break
      end if
    end for
  end for
end for

Experimental Study

In this section, we evaluate the proposed DQN framework on three publicly available datasets of arithmetic word problems. We evaluate both accuracy and efficiency by comparing with state-of-the-art methods. All the experiments were conducted on the same server, with 4 CPU cores (Intel Xeon CPU E5-2650 with 2:30GHz) and 32GB memory.

Datasets

Since the Dolphin dataset (Huang *et al.* 2016) has not been publicly accessible, we cannot use it to evaluate the generality of our solution. Instead, we still apply the same set of benchmark datasets on arithmetic math problems as used in the state-of-the-art work (Roy and Roth 2015).

1. **AI2** (Hosseini *et al.* 2014). There are 395 single-step or multi-step arithmetic word problems involving only addition and subtraction. Each math problem contains multiple quantities, which may be irrelevant to the solution.
2. **IL** (Roy *et al.* 2015). There are 562 single-step word problems with only one operator, including addition, subtraction, multiplication, and division. Each math problem also contains irrelevant quantities.
3. **CC** (Roy and Roth 2015). There are 600 multi-step problems without irrelevant quantities, harvested from `commoncoresheets`[2]. It involves combination of four types of operators, including (a) addition followed by subtraction; (b) subtraction followed by addition; (c) addition and multiplication; (d) addition and division; (e) subtraction and multiplication; and (f) subtraction and division.
4. **ArithS**. It is a subset of the union of **AI2**, **IL** and **CC**. There are 890 single-step arithmetic problems that involve only one operator.
5. **ArithM**. It is also collected from **AI2** $\bigcup$ **IL** $\bigcup$ **CC**. There are 667 multi-step arithmetic problems that involve at least two operators.

[2] `http://www.commoncoresheets.com/`

Comparison Methods

The comparison methods in this paper include KAZB (Kushman *et al.* 2014), ExpTree (Roy and Roth 2015), ALGES (Koncel-Kedziorski *et al.* 2015) and UnitDep (Roy and Roth 2017). We note that the main differences between UnitDep and ExpTree are that UnitDep introduces the unit rate graph and also takes context into consideration when extracting features. Thus, we use UnitDep-context to denote UnitDep without considering the effect of context feature and UnitDep-rate to denote UnitDep without unit rate graph. For our proposed MathDQN, we also use MathDQN-reorder to denote the method without re-order mechanism.

Parameter Settings

In our DQN model, we set the size of replay memory $\mathcal{D}$ to $15,000$ and the discount factor $\gamma = 0.9$. The DQN model was implemented on top of TensorFlow and the learning rate is set to 0.0001 for RMSProp. To adjust the trade-off between exploration and exploitation, we reduce ϵ in the greedy strategy from 0.5 to 0.01 over $30,000$ epochs. A mini-batch gradient update is executed every one step of expression tree construction and we set the size of mini-batch to 32. The feed-forward neural network contains 2 hidden layers, each with 50 dimensions.

Accuracy Results

We report the precision results for the three benchmarks **AI2**, **IL** and **CC** in Table 2. The last column refers to the average precision among all these math problems.

Table 2: Accuracy on the benchmark datasets.

	AI2	**IL**	**CC**	**Average**
KAZB	62.0	73.7	2.2	43.2
ALGES	52.4	72.9	65.0	64.6
ExpTree	72.0	73.9	45.2	62.3
UnitDep	56.2	71.0	53.5	60.5
UnitDep-context	74.7	**74.8**	47.3	64.2
UnitDep-rate	56.2	70.8	44.3	56.9
MathDQN-Reorder	78.5	73.3	63.7	70.9
MathDQN	**78.5**	73.3	**75.5**	**75.4**

First, we observe that MathDQN achieves the best performance and significantly improves the accuracy in **AI2** and **CC** datasets. Overall, it boosts the average precision by 15 percent on the benchmark datasets. The results validate the effectiveness of our proposed DQN framework.

Second, MathDQN is the only method that performs equally well on all the three datasets. ExpTree and UnitDep-context achieve very promising results in **AI2** and **IL** but cannot handle **CC** effectively. ALGES works well in **IL** and **CC**, but its accuracy in **AI2** is not satisfactory. This finding, to a certain extent, can reflect the generality of the DQN framework.

Third, the unit dependency graph proposed in UnitDep does not take effect in **AI2** because **AI2** only involves operators $+$ and $-$. The context feature proposed in UnitDep is effective in **CC** (increasing the accuracy from 47.3% to 53.5%) but generates negative impact in **AI2** (reducing the accuracy from 74.7% to 56.2%). This shows the limitation of hand-crafted features in UnitDep.

Fourth, the re-order mechanism is effective in **CC**, boosting the accuracy from 63.7% to 75.5%. **AI2** only contains operators of $+$ and $-$ and the order of operators in tree construction is not important because these two operators belong to the same priority. **IL** only contains single-step problems and there is no need for tree construction.

We also conduct a break-down analysis to compare the performance within the single-step (S-STEP) and multi-step (M-STEP) datasets in Table 3. The results show that all the methods demonstrate similar performance in solving arithmetic problems with single step operation, which can be considered as a classification task when relevant quantities are identified. The reason why our method achieves the best performance in **ArithS** (as in Table 3) but not in **IL** (as in Table 2) is that our extracted features work better in problems with operators $+$ and $-$ than problems with $\times$ and $\div$. As the complexity of math problem increases, our DQN framework exhibits significant superiority over its competitors. The results show that our method improves the accuracy over the state-of-the-art by 18.9% in **ArithM** with multi-step math problems.

Table 3: Break-down analysis for single-step and multi-step.

	ArithS	**ArithM**
KAZB	68.6	9.3
ALGES	65.8	63.9
ExpTree	74.0	46.2
UnitDep	66.1	52.9
UnitDep-context	74.89	50.1
UnitDep-rate	66.0	44.7
MathDQN-Reorder	**74.94**	66.4
MathDQN	**74.94**	**76.0**

The evolution of average reward and its associated accuracy are illustrated in Figure 3 and 4 respectively. We can see that initially the rewards are negative for two main reasons. First, we set ϵ a moderate value around 0.5 in the initial stage and the agent has a noticeable probability to select a random action for exploration. Second, the agent has not yet received sufficient training feedback from the environment. With more epochs, the agent learns quickly from the trial-and-error process. We also observe that the reward in AI2 dataset is higher than CC dataset because it only involves two types of operators for action selection.

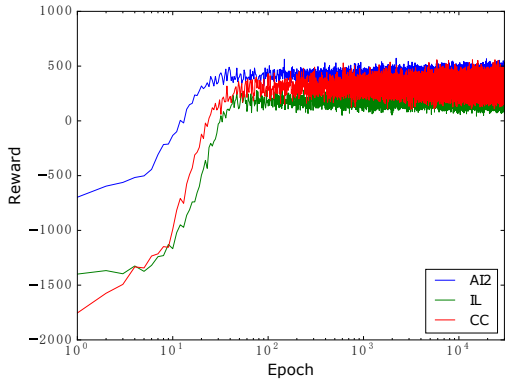

Figure 3: Evolution of average reward on three datasets.

Effect of RL and DQN

We also implement two new baseline methods to better justify the effect of RL and DQN.

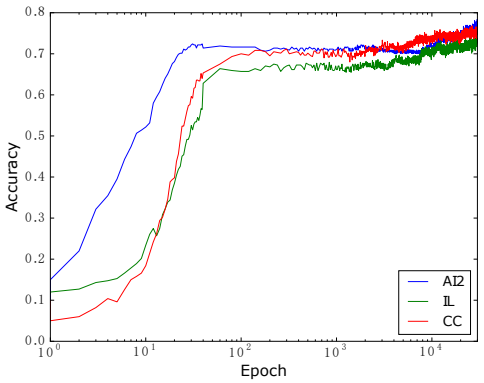

Figure 4: Evolution of accuracy on three datasets.

- **M1:** We replace Q-Learning with Sarsa and retain the two-layer feed-forward neural network. This is used to justify the effect of Q-learning framework.
- **M2:** We simply use a neural-network-based classifier without using the RL framework. Specifically, we train an operator classifier with two-layer feed-forward neural network. Given a math problem with multi operators, we extract each pair of quantities and their operator as the training sample. In the test stage, after re-ordering the leaf nodes in the expression tree, we use the classifier to determine the operator in each step of tree construction. This baseline is used to justify the effect of RL.

Table 4: Evaluating the effect of RL and DQN.

	M1	**M2**	**MathDQN**
AI2	72.8	76.2	78.5
CC	71.8	72.5	75.5

We use AI2 and CC because they contain multi-step problems. The results in Table 4 show that when both using neural network to approximate Q-table, the Q-learning algorithm is better than Sarsa. Moreover, MathDQN is better than neural-network-based classifier, verifying the effectiveness of RL and DQN. To find the reason, we conduct a break-down analysis to examine the accuracy of operator classification in each step. We find that in the first step of operator classification, M2 is even slightly (i.e., 0.2%) higher than MathDQN, but is worse than MathDQN in the subsequent steps. Our explanation is that MathDQN trains the multi-step problem as a whole in the RL framework. Thus, it shows better performance for multi-step problems.

Running Time Results

We also study the efficiency of these math word problem solvers as users expect to obtain the results instantly. The average running time to solve a math problem is reported in Table 5. **CC** contains only multi-step problems and its running time is higher than the other two datasets. ALGES runs the slowest because it uses integer linear programming to enumerate all the possible expression trees. ExpTree and UnitDep adopt beam search to reduce the tree enumeration space and thus improves the efficiency dramatically. MathDQN performs slightly slower than ExpTree in **ArithM** because it requires a two-layer feed-forward neural network for action selection whereas ExpTree simply uses SVM for LCA node classification.

Table 5: Average running time (in millisecond).

	AI2	**IL**	**CC**	**ArithS**	**ArithM**
KAZB	4.6	3.8	25.7	3.8	25.2
ALGES	713.8	308	1308	393	1342
ExpTree	5.8	4.8	6.5	4.9	6.9
UnitDep	8.5	5.9	13.2	5.9	14.1
MathDQN	4.7	3.6	9.1	3.7	9.1

Error Analysis

Finally, we analyze bad cases that our proposed DQN framework cannot solve. The main reasons are summarized in four-folds: 1) missing quantities. It is possible that certain important quantities may be correctly identified by existing syntax parser. For example, quantity "two" in a token "two-bedroom" was not extracted in our experiments. 2) imperfect relevance classifier. We use relevant quantities as the bottom level of the expression tree. Determining whether an extracted quantity is relevant or not is also crucial to the final accuracy. However, it is possible that the classifier cannot identify all the correct quantities. Then, the expression tree would be built on a wrong foundation. 3) limitation of the crafted features. We concatenate features extracted from quantity schema, which may not well handle operators like multiplication and division and leads to incorrect operator classification. 4) limitation of state representation. Ideally, we should model expression tree as a Markov Decision Process and each state is expected to represent a partial tree that has been constructed so far. However, we use features of selected pair of quantities to construct state for reinforcement learning, which can be viewed as an approximate representation.

Conclusion and Future Work

In this paper, we make the first attempt to propose a deep reinforcement learning framework for an automatic arithmetic word problem solver. The experimental results on the benchmark datasets are promising. Our method runs efficiently and achieves superior performance, especially on the more challenging multi-step problems. These findings shed certain light towards a general MWP solver that can evolve without too much human intervention.

In our future work, we will continue the line of research in two main directions. First, we observe that there have been intensive studies about utilizing deep learning models to learn effective and compact representations of words, sentences or paragraphs, such as recusrive autoencoders (Socher *et al.* 2011), recursive neural tensor network (Socher *et al.* 2013b), convolutional neural networks

(Kim 2014), DCNN with dynamic k-max pooling (Kalchbrenner *et al.* 2014), recurrent convolutional neural network (Lai *et al.* 2015) and even character-level deep models (Zhang *et al.* 2015). Since all the existing solvers for math word problems have to carefully craft useful features, it would be an interesting breakthrough to automatically learn effective representation of the syntactic information and thus significantly reduce the human intervention. An initial study in this area can be found in (Wang *et al.* 2017; Huang *et al.* 2017). Second, the state-of-the-art methods for equation set problems are far from satisfactory when experiments were conducted on large-scale diverse datasets. Thus, there still remains a wide gap to bridge and it is worth exploring the generality of deep reinforcement learning on the equation set problems.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China under grants No. 61602087, 61702016, 61632007, the Fundamental Research Funds for the Central Universities under grants No. ZYGX2016J080, ZYGX2014Z007, the China Postdoctoral Science Foundation under Grant No. 2017M610019 and the 111 Project No. B17008.

References

D. Bobrow. Natural language input for a computer problem solving system. In M. Minsky, editor, *Semantic information processing*, pages 146–226. MIT Press, 1964.

Juan C. Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *ICCV*, pages 2488–2496, 2015.

H. Guo. Generating Text with Deep Reinforcement Learning. *ArXiv e-prints*, October 2015.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *ACL*, 2016.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533, 2014.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *ACL*, 2016.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. Learning fine-grained expressions to solve math word problems. In *EMNLP*, pages 816–825, 2017.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL (1)*, pages 655–665. The Association for Computer Linguistics, 2014.

Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, 2014.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *TACL*, 3:585–597, 2015.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, and Yoav Artzi. Learning to automatically solve algebra word problems. In *ACL*, pages 271–281, 2014.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, pages 2267–2273. AAAI Press, 2015.

Chao-Chun Liang, Kuang-Yi Hsu, Chien-Tsung Huang, Chung-Min Li, Shen-Yu Miao, and Keh-Yih Su. A tag-based statistical english math word problem solver with understanding, reasoning and explanation. In *IJCAI*, pages 4254–4255, 2016.

Arindam Mitra and Chitta Baral. Learning to use formulas to solve simple arithmetic problems. In *ACL (1)*. The Association for Computer Linguistics, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, pages 1–11, 2015.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. Improving information extraction by acquiring external evidence with reinforcement learning. In *EMNLP*, pages 2355–2365, 2016.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *EMNLP*, pages 1743–1752, 2015.

Subhro Roy and Dan Roth. Unit dependency graph and its application to arithmetic word problem solving. In *AAAI*, pages 3082–3088. AAAI Press, 2017.

Subhro Roy, Tim Vieira, and Dan Roth. Reasoning about quantities in natural language. *TACL*, 3:1–13, 2015.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*, pages 1132–1142, 2015.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, pages 151–161. ACL, 2011.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *ACL*, pages 455–465, 2013.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642, 2013.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. Learning from explicit and implicit supervision jointly for algebra word problems. In *EMNLP*, pages 297–306. The Association for Computational Linguistics, 2016.

Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *EMNLP*, pages 856–865, 2017.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, pages 649–657, 2015.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. Learn to solve algebra word problems using quadratic programming. In *EMNLP*, pages 817–822, 2015.