MACHINE SOLVER FOR PHYSICS WORD PROBLEMS

Megan Leszczynski & José Moreira
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598 USA
`mel255@cornell.edu`, `jmoreira@us.ibm.com`

ABSTRACT

We build a machine solver for word problems on the physics of a free falling object under constant acceleration of gravity. Each problem consists of a formulation part, describing the setting, and a question part asking for the value of an unknown. Our solver consists of two long short-term memory recurrent neural networks and a numerical integrator. The first neural network (the *labeler*) labels each word of the problem, identifying the physical parameters and the question part of the problem. The second neural network (the *classifier*) identifies what is being asked in the question. Using the information extracted by both networks, the numerical integrator computes the solution. We observe that the classifier is resilient to errors made by the labeler, which does a better job of identifying the physics parameters than the question. Training, validation and test sets of problems are generated from a grammar, with validation and test problems structurally different from the training problems. The overall accuracy of the solver on the test cases is 99.8%.

1 INTRODUCTION

We present a complete system architecture for a machine solver that automatically solves a class of physics word problems, namely classical mechanics of a point particle in free fall. This domain allows us to formulate one dynamical system to which all the physics problems in this domain can be mapped. The dynamical system describes how the state of the particle, defined by its location and velocity, changes over time. Correspondingly, the initial conditions for the dynamical system include the location and velocity of the particle at the time origin.

Given the word problem as input, the solver must first learn to extract the parameters needed to produce the dynamical system and also learn to identify the type of question. Two independently trained recurrent neural networks are used to complete these tasks. The first neural network, referred to as the labeler, learns to find the dynamical system parameters and locate the question within the problem statement. The second neural network, referred to as the classifier, identifies the type of question. Finally, the solver uses a numerical integrator to solve the dynamical system and produce the solution. We use a problem generator in order to produce disjoint datasets as input to the system for training and testing. The generator produces short-answer high school-level physics word problems with mixed units.

After a brief related work section, we provide a more detailed description of the class of physics problems we address. We proceed to describe how the machine solver works and present experimental results. We conclude with a summary of our work and proposals for future works. The appendices contain additional details that did not fit in the body of the paper.

2 RELATED WORK

Automatically solving word problems has been a research interest of the natural language processing community for some time, particularly with math word problems. The main challenge is to develop a semantic representation of the word problem. Kushman et al. (2014) learned to represent mathematical word problem with a system of equations, by aligning words in the word problem to templates. While their technique learns to induce multiple templates and assumes knowledge of numbers and nouns, we assume no knowledge of the words in the text but only map to one template.

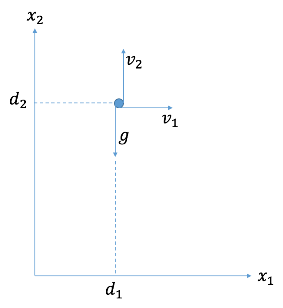
Another study to solve math word problems was done by Hosseini et al. (2014). This study also assumes the ability to identify numbers and nouns in the text and uses a dependency parser to determine relationships between words in the text. Like the other study, this approach generalizes to math word problems that require different equations. Shi et al. (2015) similarly used a parser to solve math word problems. However, their parser maps the word problems to a carefully defined language they created called DOL, from which equations can be derived. Rather than use a parser to break down the word problems, we use neural networks to learn to identify key pieces of information. Our study is the first of our knowledge to apply recurrent neural networks to the task of solving word problems.

We chose to use recurrent neural networks (RNN) for the labeler and the classifier as both of their inputs consist of sequences of words. Recurrent neural networks are commonly used to process sequences, and as a result have found application in natural language processing tasks such as machine translation (Cho et al., 2014b) and speech recognition (Graves et al., 2013). After experimenting with different models, we obtained the most success with Long Short-Term Memory (LSTM) variants of RNNs. For additional discussion on RNNs in general, and LSTMs in particular, we refer the reader to Appendix A.

3 Problem Specification

We consider the following class of physical systems (see Figure 1(a)): In a two-dimensional space, with gravity producing a downward constant acceleration g, there is one particle in free fall. That is, no forces other than gravity are acting on the particle. Movement of the particle starts at time $t = 0$, with an initial position defined by displacements d_1 and d_2 and initial velocity with components v_1 and v_2.

The time behavior of the particle can be represented by the dynamical system shown in Figure 1(b). The state vector $\vec{x}(t) = [x_1(t), x_2(t), \dot{x}_1(t), \dot{x}_2(t)]^T$ consists of two positions and two velocities and its derivative depends only on itself and the acceleration of gravity, as shown in the figure. Combined with the initial condition $\vec{x}(0) = [d_1, d_2, v_1, v_2]^T$, the differential equation produces a unique solution.



(a)

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \ddot{x}_1(t) \\ \ddot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \\ \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} x_1(0) \\ x_2(0) \\ \dot{x}_1(0) \\ \dot{x}_2(0) \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ v_1 \\ v_2 \end{bmatrix}$$

(b)

Figure 1: Physics domain (a): We consider a two-dimensional space with a free falling particle. Displacements d_1 and d_2 define the initial position of the particle, while v_1 and v_2 define its initial velocity. Gravity produces a constant acceleration g pointing straight down. The behavior of the particle is defined by the dynamical system shown in (b).

Our machine solver computes answers to word problems in the domain just described. The word problem must specify, sometimes indirectly, the five parameters of the dynamical system (d_1, d_2, v_1, v_2, and g). It must also include a question that can be answered by computing the time behavior of the system. We discuss how our machine solver works in the next section.

4 Machine Solver

In this section we describe the machine solver, which is composed of two recurrent neural networks and the numerical integrator. The top-level system block diagram is shown in Figure 2.

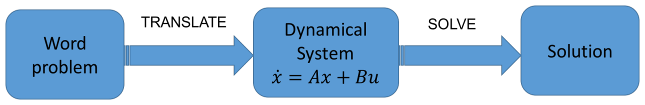


Figure 2: The first step from word problem to dynamical system is accomplished via neural networks. The second step from dynamical system to solution is achieved with a numerical integrator.

4.1 Neural Network Architectures

The data flow through the labeler and classifier neural networks is shown in Figure 3. We used TensorFlow™[1] to develop the neural network models for both labeler and the classifier. TensorFlow is an open source library from Google that allowed us to easily explore different models and training settings with already implemented RNN cells and optimizers (Abadi et al., 2015). We quickly experiment with the provided optimizers to find the optimal optimizer for each network.

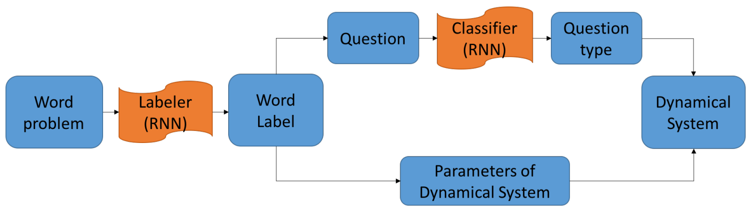


Figure 3: The word problem passes through two RNNs to be transformed into the dynamical system form.

The labeler is an LSTM network with one hidden layer of ten units. Figure 4 shows an example of the data flow through the labeler. The input to the labeler is the full problem statement and the output is a label for each word. The words are input into the labeler via an embedding that is randomly initialized and trained simultaneously with the weights and biases. The weights are also randomly initialized and the biases are initialized to zero. To limit the exploration of the parameter space, we set the dimension of the embedding to equal the number of hidden units.

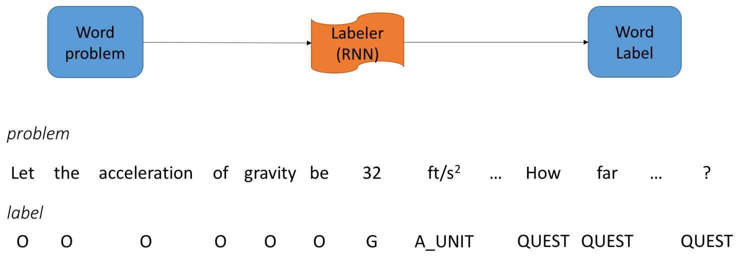

Word problem
Labeler (RNN)
Word Label
problem
Let the acceleration of gravity be 32 ft/s² ... How far ... ?
label
O O O O O O G A_UNIT QUEST QUEST QUEST

Figure 4: Example of input to labeler with expected output. A label is associated with each word, where **O** indicates other, or a word not needed for the dynamical system translation. Input text is shortened for the example.

The chosen RNN model is one that produces an output at each time step and has recurrent connection between hidden units, as described by Goodfellow et al. (2016) in Chapter 10, Figure 10.3. At each step of the input sequence, the RNN receives a word embedding and outputs a label for the word. The label that is outputted at each time step can fall into one of the ten categories shown in Table 1. In addition to tagging words for their relevancy to the dynamical system formulation, we tag the question part of the word problem to pass to the classifier.

We use three measures to assess the performance of the labeler: label accuracy, question accuracy, and overall accuracy. Label accuracy is measured as having matching labels in the predicted and expected (generated) labels, not including the question part of the word problem. Question accuracy is measured as having both the first word of the question and the last word of the question labeled correctly, as label-based post processing to extract the question relies only on these indices. Overall accuracy is measured as meeting both of the label and question accuracy criteria.

[1]TensorFlow is a trademark of Google Inc.

Table 1: Possible output word labels and corresponding dynamical system parameters.

LABEL	DESCRIPTION	
QUEST	Question	
G	Value for gravity	g
A_UNIT	Unit for acceleration (gravity)	g
D_UNIT	Unit for initial height	d_2
HEIGHT	Initial height value or height of each story	d_2
V_UNIT	Unit for velocity	v_1, v_2
V	Initial velocity magnitude	v_1, v_2
THETA	Angle of initial movement	v_1, v_2
STORY	Value for number of stories (if applicable)	d_2
O	Other	

We train the labeler with TensorFlow's Adam Optimizer, an initial learning rate of 0.1, and a mini-batch size of 100 word problems. The Adam Optimizer uses adaptive learning rates and is particularly effective with sparse gradients (Kingma & Ba, 2014). We use early stopping based on a validation accuracy or when the training accuracy stops improving. We chose the network architecture and training settings after performing a limited grid search across the number of layers, number of units per a layer, and learning rate. (See Appendix B.)

After the labeler assigns a label to each word, a post processing step maps the labels to the dynamical system parameters, converting the initial conditions and value of gravity to SI units if necessary.

The classifier is an LSTM network with one hidden layer of 1,000 units. An example of the data flow through the classifier is shown in Figure 5. For the problems in our dataset, the formulation part of the word problem does not provide information necessary to classify the type of question. Moreover, as sequences become longer, the performance of RNNs tend to decrease (Pascanu et al., 2013). Armed with these two observations, we chose to only have the question part of the word problem as the sequence to input into the classifier.

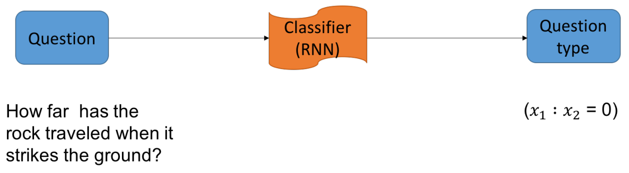


Figure 5: Example of input to classifier with expected output. Symbol x_1 refers to horizontal displacement and symbol x_2 refers to vertical displacement.

As with the labeler, we encode the words of the sequence into word embeddings, matching the dimension of the word embedding to the number of hidden units, and training them with the weights and biases. In this case, a sequence would be one question. Unlike the labeler, there is only one output for each sequence, occurring on the last step of the sequence. For more information see Chapter 10, figure 10.5 of Goodfellow et al. (2016) for an illustration. The singular output is the type of question, which can fall into one of the nine types shown in Table 2.

The classifier is trained with TensorFlow's Gradient Descent Optimizer, an initial learning rate of 0.5, and a mini-batch size of 100 questions. As with the labeler, we performed a grid search to choose these hyperparameters. (See Appendix B.)

4.2 Numerical Integrator

The numerical integrator computes the evolution over time of the dynamical system shown in Figure 1(b). As input it receives the initial conditions, the value of g, and the type of question extracted from the labeler and the classifier. Using SciPy's ordinary differential equation integrator, a table of values representing the system's state to the point that the object hits the ground is iteratively constructed. The numerical solution is refined to a precision of 0.001 (one part in a thousand), based on the type of the question. For example, if the question is about the maximum height, we produce

Table 2: Possible Output Question Types

QUESTION TYPE	DESCRIPTION
(x_1 : max)	Maximum horizontal distance traveled
(speed : max)	Maximum speed obtained
(x_2 : max)	Maximum height obtained
(speed : max height)	Speed at maximum height
(time : max height)	Time that maximum height is reached
(x_1 : x_2=0)	Distance traveled when object hits ground
(time : x_2=0)	Time that object hits ground
(speed : x_2=0)	Speed object is traveling when it hits the ground
(x_1: max height)	Distance object has traveled when it reaches its maximum height

a first instance of the table, find the maximum height in that table, and then search for the maximum around that value with increased precision, repeating until we reach the desired precision. Finally, the question type is used to determine which value from the table to output from the solver. This data flow is shown in Figure 6.

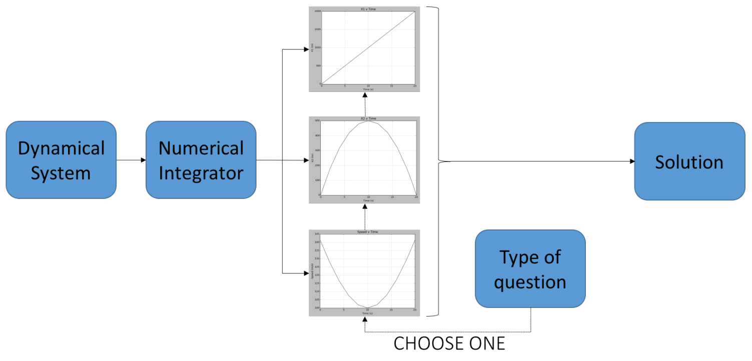


Figure 6: Outputs from the labeler and the classifier feed into the numerical integrator, where the labeler outputs form the dynamical system to integrate and the classifier outputs control the focus and output of the integrator.

4.3 Training, Validation, and Test Sets

We define the word problems with a grammar that is provided in the Appendix. The word problems in the training, validation, and test sets are exclusively made up of problems that follow the specifications laid out by the grammar. The grammar allows for mixed units, meaning that within the same problem, the height may have a metric unit, while the velocity may have a U.S. customary unit. The grammar also permits the initial conditions to be exposed in multiple ways. For instance, a theta value and speed will be provided in some problems, from which the solver would need to calculate the initial vertical velocity using the theta, whereas in other problems no theta value may be provided. Using mixed units and varying numbers of values to provide information about each initial condition allows us to increase the complexity of the problems within the scope of the dynamical system.

The grammar also ensures that the training set is disjoint from the validation and test sets, particularly in structure. Examples of generated problems are shown below in Figure 7. This is vital in assessing the ability of the trained networks to generalize.

We implement the grammar in Python. When a new problem is instantiated, the grammar rules are descended to build up the problem, making random choices when choices are available. Labels for each problem are also automatically generated. The complete generative model is shown in Figure 8. By using a problem generator to build our datasets, we are also free to choose the size of the dataset. Our problem generator is capable of generating ~26,000 different training problems and ~22,000 different test and validation problems.

Assume the acceleration due to gravity is 85 ft/s^2. A ping pong ball is dropped from the top of a 8 story building, where each story is 89 m. What is the maximum speed the ping pong ball obtains?

A chair is launched at a speed of 51 mph and an angle from the horizontal of 28 degrees. Let the acceleration due to gravity on Planet Watson be 98 m/s^2. How much time has passed when it reaches its maximum height?

Figure 7: Examples of generated problems that adhere to the grammar

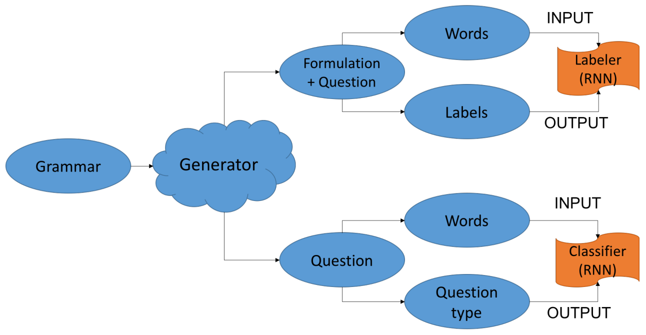


Figure 8: The generative model allows us to generate the input and output for the neural networks without requiring any manual annotation.

5 Experimental Results

5.1 Learning Progress

The datasets consisted of 7,000 word problems for training, 2,000 word problems for validation, and 1,000 word problems for test. The progress of training over time is shown in Figure 9. As can be seen in the left graph, the labeler learns to identify the beginning and end of the question faster than it learns to correctly predict the labels. The overall accuracy of the labeler is both limited by and equivalent to that of the label accuracy. With this particular model of the labeler, there is no problem for which the labeler correctly predicts the non-question labels, but incorrectly locates the question.

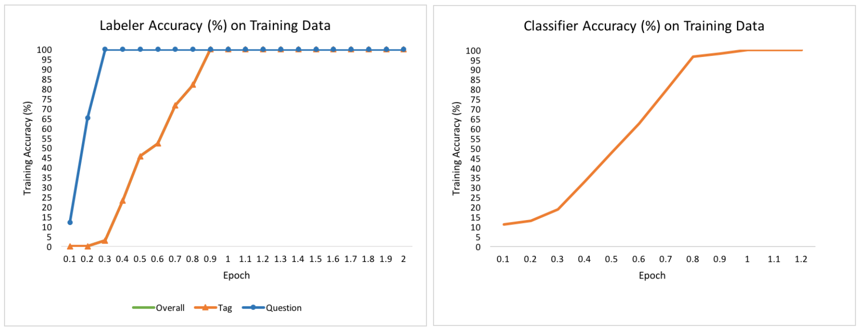


Figure 9: Training accuracy of labeler (left) and classifier (right)

The training accuracy for the label, question, and overall reach 100% for all by the end of the first epoch. The classifier also reaches 100% accuracy on the training set by the end of the first epoch. The epoch is broken down into fractions as the training accuracy is evaluated every seven mini-batches of 100 problems.

The accuracy on the test set after the labeler and classifier have been independently trained are shown in Table 3. The accuracy of the combined RNN system amounts to an overall accuracy of 99.8%. The labeler achieves 100% accuracy on predicting the non-question labels and incurs a small error on predicting the beginning and end of the question. As a result, the question that is extracted based on the labeler's predictions does not always match the true question. However, based on the classifier's accuracy of 99.8%, the classifier is often resilient to the errors that labeler makes in extracting the question. While the labeler incorrectly extracts ninety-one questions, the classifier only incorrectly classifies two questions from a test set of 1,000 word problems. Figure 12 in Appendix C shows examples of the labeler's errors and how the classifier handles them.

We note that for the two wrongly classified cases, both shown in Figure 12, the classification error is the same. That is, a question that should be about the speed of the object when it hits the ground is classified as a question about the maximum speed the object reaches. The numerical answer to the problem is the same for both classes of question. Therefore, even in the case of wrongly classified questions, the system produces the right answer.

The high accuracy of the labeler and classifier are not a total surprise. LSTMs have been shown to be very effective in learning context-free and even context-sensitive languages (Gers & Schmidhuber, 2001; Cleeremans et al., 1989; Rodriguez, 2001), including the ability to generalize and recognize structures not seen before. Our training, validation and test sets are from a regular language, as described in Appendix E, so an LSTM should do well in learning them. In fact, we have seen situations (with the test, validation and test sets all with distinct structures) where the labeler and classifier both achieve perfect accuracy on all test problems. We decided to include the data on the "not so perfect" case because it illustrates some important points (Figure 12).

Table 3: Accuracies shown are on the test set of word problems for the system. The classifier is fed the extracted questions as identified by the labeler. The combined RNN system accuracy is based on the final output of the system having the same dynamical system parameters and question type as the generated output for a word problem.

	Labeler			
Overall	Label	Question	Classifier	Combined RNN System
0.909	1.000	0.909	0.998	0.998

5.2 Analysis of Trained Variables

The trained variables for both models consist of word embeddings for input to the RNN, and weights and biases within the RNN and from the RNN to the final output. We focus our evaluation on the RNN weights, as we believe these are more specific to the our physics problem solver. For an evaluation of the word embeddings, please see Appendix D.

The distributions of weights for the labeler and classifier are shown in figures 10. As the labeler was an LSTM network, there are weights from the input and the previous hidden values to input, forget, and an output gates, as well as to the memory cells. While there appears to be a high concentration of negative weights to the output gate and positive weights to the input gate, this is likely a result of random initialization of the weights as this pattern was not consistently found with other random initializations. The output weights, which go from the output of the LSTM cell's hidden units to the target labels, have a slightly wider range. The few number of zero weights indicates that the majority outputs from the hidden units of the LSTM cell contribute to making the final prediction of the label.

The LSTM weight distribution for the classifier is more uniform and compressed than that of the labeler. We believe this is due to the great increase in parameters since the classifier has 1,000-dimensional embeddings and 1,000 hidden units, leading to 8 million weights (Karpathy et al., 2015). We predict that each piece of information captured by the trained embeddings and hidden units makes a less significant contribution to the final prediction than with the labeler, as indicated by the classifier's smaller weight values. The range of the output values for the output weights similarly contributes to this prediction, with a very small range of weights which are mostly concentrated around zero.

After examining the general distribution of weights, we also wanted to explore potential patterns of specific weights. We chose to explore the heat map of the weights for labeler since there are a magnitude fewer connections, allowing the patterns to be more readily examined. We include the heat map of the weight matrices for the connections between the hidden units of the labeler to the output predictions in Figure 11. Looking at the heat map, hidden units 3 and 8 seem to have a similar weight distribution across the output categories. We also see seemingly logical pairs forming, such

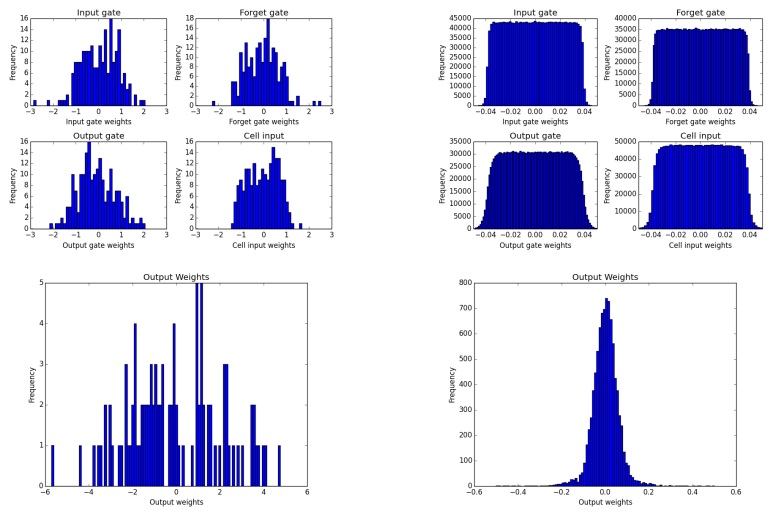


Figure 10: **Top left**: labeler LSTM weight distributions. **Top right**: classifier LSTM weight distributions. **Bottom left**: labeler output weight distributions. **Bottom right**: classifier output weight distributions.

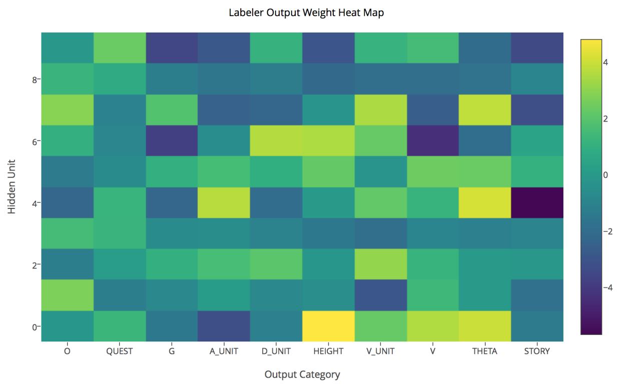


Figure 11: Heat map for labeler weights from LSTM hidden layer to output layer.

as the strong positive weights associated with D_UNIT and HEIGHT for hidden unit 6 and for V and THETA for hidden unit 0. However, there are also features that are challenging to explain, such as the strong positive contribution hidden unit 4 makes to predicting THETA while making an equally strong negative contribution to predicting STORY.

6 Conclusions

We have developed a machine solver for a word problems on the physics of a free falling object in two-dimensional space with constant acceleration of gravity. The solver has three main components. The *labeler* labels each word of the problem to identify the parameters of a canonical dynamical system that describes the time evolution of the object, and the part of the problem that corresponds to the question being asked. The *classifier* classifies the question part. Finally, an integrator is used to solve the dynamical system, producing a numerical answer to the problem.

A grammar-based generator is used to produce the training, validation and test set of problems for the neural networks. The grammar is specified so that the validation and test problems are structurally different from the training problems. We use a total of 10,000 generated problems, partitioned into 7,000 for training, 2,000 for validation and 1,000 for testing.

When measured against the test set of 1,000 problems, the dynamical system parameters are correctly identified in all of them. The question part is precisely identified in 909 cases, but because the classifier can work with partial questions, in the end all but 2 questions are classified correctly. Therefore, the combined accuracy of the two neural networks, for the purpose of solving the physics problems, is 99.8%.

There are several opportunities for future work. First, we would like to investigate more deeply *how* our neural networks work. In particular, what features of the word problem they are identifying and how specific units are responsible for that identification. Second, we could extend our solver by considering more complex physical situations, including additional forces, three-dimensional motion, multiple objects, and so on. We would have to extend our canonical dynamical system to represent those situations and/or use a collection of dynamical systems. We expect that the complexity of the neural networks and the training/validation/test sets will grow accordingly. Finally, the more ambitious goal would be to remove the canonical dynamical system(s) and train the networks to build their own. We believe this would be closer to the way humans solve these physics problems.

REFERENCES

Martín Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. Software available from http://tensorflow.org.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*, 2014a.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014b.

Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. Finite state automata and simple recurrent networks. *Neural Comput.*, 1(3):372–381, September 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.3.372. URL `http://dx.doi.org/10.1162/neco.1989.1.3.372`.

Felix A. Gers and Jürgen Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Trans. Neural Networks*, 12(6):1333–1340, 2001. doi: 10.1109/72.963769. URL `http://dx.doi.org/10.1109/72.963769`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press. Book available from http://www.deeplearningbook.org, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649. IEEE, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, November 1997. ISSN 0899-7667.

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

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Paul Rodriguez. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Comput.*, 13(9):2093–2118, September 2001. ISSN 0899-7667. doi: 10.1162/089976601750399326. URL `http://dx.doi.org/10.1162/089976601750399326`.

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

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

A RECURRENT NEURAL NETWORKS

The labeler and classifier are both recurrent neural networks (RNNs). We provide background information on RNNs in this section, followed by an overview of Long Short-Term Memory (LSTM) networks, which are an advanced type of RNNs and were used to build our networks. A recurrent neural network receives the previous values of the hidden layer as input in addition to the current input values into the network. Thus each hidden unit retains information about the history of the sequence. As explained in Goodfellow et al. (2016), the fundamental behavior of recurrent neural networks can be captured in the following equation:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta),$$

where $h^{(t)}$ represents the state of the RNN unit at time t, $x^{(t)}$ represents the current input, and θ represents the weights and biases. The function f is usually hyperbolic tangent (Karpathy et al., 2015). It is important to note that the weights and biases are reused across time. Thus, while an RNN with one hidden layer can be unfolded in time to having many layers, the weights and biases between each of the unfolded layers are shared.

A limitation of the basic recurrent neural network described above is that it cannot retain information over long sequences. If a key piece of information for predicting an output at the end of a long sequence occurs at the very beginning of the sequence, the basic recurrent neural network will likely fail as a result of training difficulties. A popular solution for this limitation is the Long Short-Term Memory (LSTM) - essentially a highly capable, more complex type of recurrent neural network (Hochreiter & Schmidhuber, 1997). An LSTM is composed of a memory cell, and input, output, and forget gates that determine how to modify and reveal the contents of memory cell. Each of these gates has its own set of weights and biases that are connected to the inputs. Therefore the number of weights within a layer of an LSTM is quadrupled from that of a basic recurrent neural network to $2n \times 4n$, where n is the number of hidden units in the layer and assumes each layer has the same number of units. $2n$ is from the input being a concatenation of the output from the previous hidden layer (in time) with the current input, as occurs for all RNNs, and the $4n$ is for the connections to each of the three gates as well as to the memory cell input. More specifically, the equations for the LSTM are as follows (Graves, 2013); (Zaremba et al., 2014):

$$\begin{pmatrix} i \\ f \\ o \\ j \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot j$$

$$h_t^l = o \odot \tanh(c_t^l)$$

As both of our neural network models have only one hidden layer, h_t^{l-1} merely refers to the current input. $T_{2n,4n}$ refers to the weight and bias transformation $Wx+b$ applied to the concatenated hidden layer inputs. The hyperbolic tangent and sigmoid functions are applied element-wise. The variables i, f, o, and j refer to the input gate, forget gate, output gate, and cell input, respectively.

Another potential solution to the inability of the basic recurrent neural network to capture long-term dependencies is the Gated Recurrent Unit (GRU) (Cho et al., 2014a), however, we had the most success with the LSTM for our specific labeler and classifier tasks.

B Choosing the right RNN configuration

We selected the models for our RNNs by performing a grid search over the learning rate, the number of units, and the number of layers. The results of the grid search for the the labeler recurrent network are shown in Table 4 and the results for the classifier network are shown in Table 5. For each RNN, we chose the most efficient model, in that it requires the least space and obtains the greatest accuracy with the lowest training time.

Interestingly, for the classifier, we see that models with two or three layers and lower learning rates achieve an equivalent accuracy as the one-layer model. However, they are inferior to the one layer model in that the multi-layer models require more space and usually require longer to train.

Table 4: The chosen RNN network for the labeler has one layer of ten units with a learning rate of 0.1. The notation $x/y/z$ means x for overall accuracy, y for label accuracy, and z for question accuracy, where accuracy is given as a proportion of correct predictions over total predictions. All results shown use TensorFlow's Adam Optimizer and LSTM cell.

		Learning Rate		
Layers	Units	0.01	0.1	0.5
1	10	0.197/1.000/0.197	**0.911/1.000/0.911**	0.001/0.110/0.032
	100	0.850/1.000/0.850	0.763/0.932/0.814	0.196/0.207/0.587
	1000	0.048/0.281/0.525	0.882/0.907/0.955	0.225/0.230/0.975
2	10	0.000/0.000/0.000	0.037/0.099/0.048	0.005/0.009/0.354
	100	0.096/0.337/0.096	0.000/0.000/0.000	0.000/0.000/0.000
	1000	0.000/0.000/0.000	0.000/0.000/0.000	0.000/0.000/0.000
3	10	0.000/0.000/0.015	0.021/0.132/0.059	0.000/0.000/0.000
	100	0.076/0.442/0.091	0.000/0.000/0.000	0.000/0.000/0.000
	1000	0.000/0.000/0.000	0.000/0.000/0.000	0.000/0.000/0.000

Table 5: The chosen network for the labeler has one layer of 1,000 units. The values shown are accuracies given as a proportion of the number of correctly predicted classifications over total classifications. All results use TensorFlow's Gradient Descent Optimizer and LSTM cell.

		Learning Rate		
Layers	Units	0.01	0.1	0.5
1	10	0.193	0.486	0.830
	100	0.774	0.801	0.889
	1000	0.980	0.997	**1.000**
2	10	0.163	0.424	0.637
	100	0.833	0.875	0.819
	1000	**1.000**	**1.000**	0.724
3	10	0.297	0.656	0.482
	100	0.867	0.907	0.539
	1000	**1.000**	**1.000**	0.695

C Error Handling

This section is included to illustrate examples of the the labeler network incorrectly extracting the question. In each of these cases, the classifier receives as input the labeler's incorrect output. The classifier's handling of these errors is shown in Figure 12.

(1) Labeler input: Let the acceleration due to gravity on Planet Watson be 65 ft/s^2. A ping pong ball is released from the top of a 3 story building, where each story is 79 m. What is the maximum speed the ping pong ball obtains?

Labeler output / classifier input: What is the maximum speed the
Classifier output: (speed : max)
Expected output: (speed : max)

(2) Labeler input: Assume the acceleration due to gravity is 49 m/s^2. A ping pong ball is launched at a speed of 35 m/s and an elevation of 88 degrees. What is the magnitude of the velocity of the ping pong ball just before it touches the ground?

Labeler output / classifier input: What is the magnitude of the velocity of the
Classifier output: (speed : max)
Expected output: (speed : x_2=0)

(3) Labeler input: Let the acceleration due to gravity on Planet Watson be 71 ft/s^2. A ping pong ball is thrown at a speed of 53 mph and an elevation of 52 degrees. What is the magnitude of the velocity of the ping pong ball just before it touches the ground?

Labeler output / classifier input: What is the magnitude of the velocity of the
Classifier output: (speed : max)
Expected output: (speed : x_2=0)

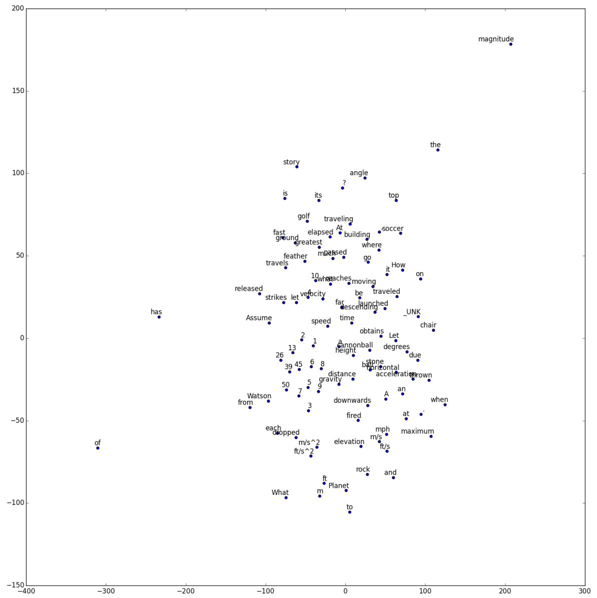
Figure 12: Examples of incorrectly extracted questions from the labeler and the classifier's response to them. In all three cases, the question is cut short. The classifier still makes the correct the classification for the first case, but fails for the second and third cases.

D Word Embeddings

To input the words into both RNNs, the words were first encoded as word embeddings. Word embeddings map words to a multi-dimensional space, providing the words with numerical representations which expose relationships between words. The final embeddings for the labeler network are 10-dimensional, and the embeddings for the classifier network are 1,000-dimensional. Rather than use Word2Vec, we chose to train the embeddings simultaneously with the weights and biases. We were interested in seeing if embeddings trained for a particular task could capture intuitive word features, as can often be seen with embeddings trained with Word2Vec (Mikolov et al., 2013).

In order to explore the results of the trained embeddings, we used scikit-learn's implementation of t-SNE to map the high-dimensional embeddings down to two dimensions (van der Maaten & Hinton, 2008). The results from t-SNE are shown in Figure 13. Words appear exactly as they appear in the word problem, and no stemmers are used.

The embeddings from the labeler network seem more intuitive, as numbers and similar units, such as "m/s", "mph", and "ft/s", are mapped to similar regions. We had hypothesized that the embedding may capture some word function related to the task the embeddings were being trained to perform. However, the objects seem to be distributed throughout the space and have no easily distinguishable pattern, despite playing a similar functional role in each word problem. It is even more difficult to discern any patterns from the embeddings from the classifier network. We do see that words such as "traveling", "traveled", and "travels" map near each other, as well as question words "What" and "How". We predict that the limited vocabulary in the question space of only forty words may con-



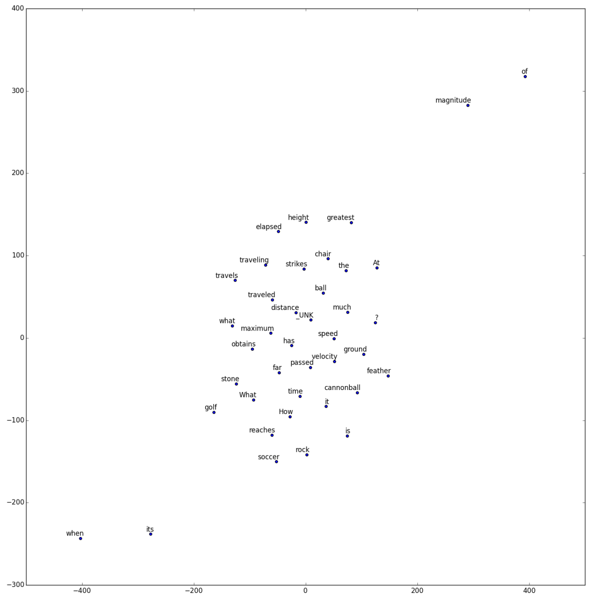


Figure 13: **Top:** The embeddings from the labeler network for the top 100 most frequent words in the word problems. **Bottom:** The embeddings from the classifier network for all words in the questions.

tribute to these more perplexing results by reducing the effectiveness of which t-SNE can determine the similarity between words.

E Word Problem Grammar

We used the grammar below to generate our word problems with a problem generator.

Notation: “object” is used as a parameter in order to enforce consistency between parts of the problem. Within a word problem, the same object must appear wherever an object symbol occurs. As used in the question part of the grammar, “x1” indicates horizontal displacement and “x2” indicates vertical displacement. When used with numbers, “...” indicates the sequence of numbers continues in between the bars.

⟨*word_problem(object)*⟩ ::= ⟨*formulation(object)*⟩ ⟨*question(object)*⟩

⟨*object*⟩ ::= ⟨*training_object*⟩ | ⟨*val_test_object*⟩

⟨*training_object*⟩ ::= `golf ball` | `stone` | `chair` | `feather` | `soccer ball` | `rock` | `cannonball`

⟨*val_test_object*⟩ ::= `pebble` | `ping pong ball` | `vacuum` | `tennis ball` | `basketball` | `hat`

⟨*formulation(object)*⟩ ::= ⟨*training_formulation(object)*⟩ | ⟨*val_test_formulation(object)*⟩

⟨*training_formulation(object)*⟩ ::= `A` object `is` ⟨*action*⟩. ⟨*Assumption*⟩.

⟨*val_test_formulation(object)*⟩ ::= ⟨*Assumption*⟩. `A` object `is` ⟨*action*⟩.

⟨*assumption*⟩ ::= `Let the acceleration due to gravity on Planet Watson be` ⟨*acceleration*⟩ .
| `Assume the acceleration due to gravity is` ⟨*acceleration*⟩ .

⟨*acceleration*⟩ ::= ⟨*accel_value*⟩ ⟨*accel_unit*⟩

⟨*accel_value*⟩ ::= `1` | `2` | `3` | ... | `100`

⟨*accel_unit*⟩ ::= m/s^2 | ft/s^2

⟨*action*⟩ ::= ⟨*moving*⟩ | ⟨*stationary*⟩

⟨*moving*⟩ ::= ⟨*descent*⟩ | ⟨*projectile*⟩

⟨*descent*⟩ ::= `descending at a speed of` ⟨*speed*⟩
| `moving downwards at a speed of` ⟨*speed*⟩

⟨*projectile*⟩ ::= ⟨*proj_verb*⟩ `at a speed of` ⟨*speed*⟩ `and an` ⟨*angle_word*⟩ `of` ⟨*angle*⟩ `degrees`

⟨*proj_verb*⟩ ::= `thrown` | `fired` | `launched`

⟨*speed*⟩ ::= ⟨*speed_value*⟩ ⟨*speed_unit*⟩

⟨*speed_value*⟩ ::= `0` | `1` | `2` | ... | `99`

⟨*speed_unit*⟩ ::= m/s | ft/s | mph

⟨*angle_word*⟩ ::= `elevation` | `angle from the horizontal`

⟨*angle*⟩ ::= `1` | `2` | `3` | ... | `89`

⟨*stationary*⟩ ::= ⟨*stat_verb*⟩ `from` ⟨*location*⟩

⟨*stat_verb*⟩ ::= `released` | `dropped` | `let go`

⟨*location*⟩	::=	⟨*height*⟩
	|	`the top of a` ⟨*num_stories*⟩ `story building, where each story is` ⟨*height*⟩
⟨*height*⟩	::=	⟨*height_value*⟩ ⟨*height_unit*⟩
⟨*height_value*⟩	::=	⟨*training_height*⟩ | ⟨*validation_height*⟩ | ⟨*test_height*⟩
⟨*training_height*⟩	::=	`1` | `2` | `3` | ... | `50`
⟨*validation_height*⟩	::=	`51` | `52` | `53` | ... | `76`
⟨*test_height*⟩	::=	`77` | `78` | `79` | ... | `99`
⟨*height_unit*⟩	::=	m | ft
⟨*num_stories*⟩	::=	`1` | `2` | `3` | ... | `10`
⟨*question(object)*⟩	::=	⟨*max_question(object)*⟩ | ⟨*prop_cond_question(object)*⟩
⟨*max_question(object)*⟩	::=	⟨*max_x1(object)*⟩ | ⟨*max_x2(object)*⟩ | ⟨*max_speed(object)*⟩
⟨*max_x1(object)*⟩	::=	`What is the maximum distance the` object `travels`
	|	`What is the greatest distance the` object `travels`
⟨*max_x2(object)*⟩	::=	⟨*training_max_x2(object)*⟩ | ⟨*val_test_max_x2(object)*⟩
⟨*training_max_x2(object)*⟩	::=	`What is the maximum height the` object `reaches?`
⟨*val_test_max_x2(object)*⟩	::=	`What is the greatest height the` object `reaches?`
⟨*max_speed(object)*⟩	::=	`What is the maximum speed the` object `obtains?`
	|	`What is the greatest speed the` object `obtains?`
⟨*prop_cond_question(object)*⟩	::=	⟨*proposition(object)*⟩ ⟨*condition*⟩
⟨*proposition(object)*⟩	::=	⟨*unk_time*⟩ | ⟨*unk_x1(object)*⟩ | ⟨*unk_speed(object)*⟩
⟨*unk_time*⟩	::=	`How much time has elapsed`
	|	`How much time has passed`
⟨*unk_x1(object)*⟩	::=	`How far has the` object `traveled`
	|	`What distance has the` object `traveled`
⟨*unk_speed(object)*⟩	::=	`How fast is the` object `traveling`
	|	`At what speed is the` object `traveling`
	|	`What is the magnitude of the velocity of the` object
⟨*condition*⟩	::=	⟨*distance_condition*⟩ | `when it reaches its maximum height?`
⟨*distance_condition*⟩	::=	⟨*training_distance_condition*⟩ | ⟨*val_test_distance_condition*⟩
⟨*training_distance_condition*⟩	::=	`when it reaches the ground?`
	|	`when it strikes the ground?`
⟨*val_test_distance_condition*⟩	::=	`just before it touches the ground?`
	|	`as it hits the ground?`

Whenever the grammar dictates a choice of construct (for example, when selecting the object of a word problem), a uniform random number generator is used to select one of the valid constructs. Therefore, the frequency of a particular form in the training, validation and test sets ultimately

Table 6: Occurrence counts for different objects in word problems.

(a) training set

object	count
golf ball	1052
stone	1007
chair	987
feather	1020
soccer ball	965
rock	989
cannonball	980

(b) validation set

object	count
pebble	336
ping pong ball	342
vacuum	316
tennis ball	355
basketball	325
hat	326

(c) test set

object	count
pebble	156
ping pong ball	159
vacuum	165
tennis ball	163
basketball	178
hat	179

Table 7: Occurrence counts for different question types.

(a) training set

class	count
(x_1 : max)	1163
(speed : max)	1157
(x_2 : max)	1120
(speed : max height)	610
(time : max height)	602
(x_1 : x_2=0)	598
(time : x_2=0)	596
(speed : x_2=0)	585
(x_1 : max height)	569

(b) validation set

class	count
(x_1 : max)	326
(speed : max)	349
(x_2 : max)	325
(speed : max height)	160
(time : max height)	158
(x_1 : x_2=0)	160
(time : x_2=0)	194
(speed : x_2=0)	180
(x_1 : max height)	148

(c) test set

class	count
(x_1 : max)	168
(speed : max)	180
(x_2 : max)	166
(speed : max height)	64
(time : max height)	92
(x_1 : x_2=0)	88
(time : x_2=0)	75
(speed : x_2=0)	77
(x_1 : max height)	90

depend on how many random choices are necessary to produce that form and how many variations there are in each choice.

Table 6 illustrates the simple case of occurrence counts of the different objects in our word problems. The training set uses seven different objects, while the validation and test sets use six objects. Not surprisingly, each object in the training set appears in approximately $1/7$ of the total number of problems in that set. Meanwhile, each object in the validation and test sets appears in approximately $1/5$ of the total number of problems in those sets.

A more interesting situation is illuatrated in Table 7 for the occurrence counts of question types. As shown in Table 2, there are nine different question types. However, the grammar works by first choosing one of two *groups* of questions: either *max*-type questions (the first three in Table 2) or *conditional*-type questions (the last six in Table 2). Within each group, there is equal probability for each question type. Consequently, as Table 7 shows, each of the max-type questions is approximately twice as common as each of the conditional-type questions.