Solving Machine Learning Problems

Sunny Tran SUNNYT@MIT.EDU
MIT EECS

Pranav Krishna PKRISHNA@MIT.EDU
MIT EECS

Ishan Pakuwal IPAKUWAL@MIT.EDU
MIT EECS

Prabhakar Kafle PKAFLE@MIT.EDU
MIT EECS

Nikhil Singh NSINGH1@MIT.EDU
MIT Media Lab

Jayson Lynch JAYSONL@MIT.EDU
University of Waterloo

Iddo Drori IDRORI@MIT.EDU
MIT EECS

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Can a machine learn Machine Learning? This work trains a machine learning model to solve machine learning problems from a University undergraduate level course. We generate a new training set of questions and answers consisting of course exercises, homework, and quiz questions from MIT's 6.036 Introduction to Machine Learning course and train a machine learning model to answer these questions. Our system demonstrates an overall accuracy of 96% for open-response questions and 97% for multiple-choice questions, compared with MIT students' average of 93%, achieving grade A performance in the course, all in real-time. Questions cover all 12 topics taught in the course, excluding coding questions or questions with images. Topics include: (i) basic machine learning principles; (ii) perceptrons; (iii) feature extraction and selection; (iv) logistic regression; (v) regression; (vi) neural networks; (vii) advanced neural networks; (viii) convolutional neural networks; (ix) recurrent neural networks; (x) state machines and MDPs; (xi) reinforcement learning; and (xii) decision trees. Our system uses Transformer models within an encoder-decoder architecture with graph and tree representations. An important aspect of our approach is a data-augmentation scheme for generating new example problems. We also train a machine learning model to generate problem hints. Thus, our system automatically generates new questions across topics, answers both open-response questions and multiple-choice questions, classifies problems, and generates problem hints, pushing the envelope of AI for STEM education.

Keywords: Learning to learn; Machine Learning course; Transformers; Graph neural network; Expression trees; Encoder-decoder architecture.

Week	Topic	Exp. Acc. ORQ	Acc. ORQ	Acc. MCQ
1	Basics	1.00	1.00	1.00
2	Perceptrons	0.98	0.98	0.98
3	Features	0.86	0.86	0.89
4	Logistic regression	0.86	0.86	0.90
5	Regression	0.97	0.97	0.97
6	Neural networks I	1.00	1.00	1.00
7	Neural networks II	0.97	0.97	0.98
8	Convolutional neural networks	0.84	0.86	0.89
9	Recurrent neural networks	1.00	1.00	1.00
10	State machines and MDPs	0.94	1.00	1.00
11	Reinforcement learning	1.00	1.00	1.00
12	Decision trees	1.00	1.00	1.00
	Overall average over topics	0.95	0.96	0.97

Table 1: Accuracy achieved using our system for each topic taught in MIT's Introduction to Machine Learning course, 6.036. Our system demonstrates an overall average expression accuracy (percent of correct expressions) of 95% and value accuracy (percent of correct values) of 96% for open response questions (ORQ), and accuracy (percent of correct values) of 97% for multiple-choice questions (MCQ), achieving grade A performance in real-time.

1. Introduction

Significant progress has been made in natural language processing in question answering. The simplest questions to tackle, reading comprehension questions, are handled using large pre-trained Transformer models. More recent developments have allowed models to solve questions that require mathematical reasoning by predicting the associated equation. Unfortunately, such models still fail on questions which involving Linear Algebra and Calculus, such as computing the length of a vector, which are prerequisites for Machine Learning. We create the first Machine Learning model that answers questions about Machine Learning.

1.1. Solving Machine Learning Problems

This work is the first to successfully solve Machine Learning problems (or questions) using Machine Learning. Specifically, our model handles the wide variety of topics, shown in Table 1, covered in MIT's Introduction to Machine Learning course (6.036), except for coding questions and questions that require input images. This includes basic linear algebra, perceptrons, features, logistic regression, regression, neural networks, convolutional neural networks, recurrent neural networks, reinforcement learning, and decision trees with overall grade A performance in real-time. Table 2, along with the first 12 tables in Appendix A of the supplementary materials (SM), show example input questions and output answers generated by our model for each machine learning topic. All of these questions are answered with an open (text) response. We also describe a method for taking advantage of multiple-choice questions, which increases overall accuracy from 94% to 97%.

While previous work has shown the ability of machine learning models to answer questions which require reading comprehension and mathematical reasoning, this work is the first to tackle machine learning problems using expression trees. A motivation for exploring learning machine learning is that STEM subjects have been shown to be more difficult than

other topics for pure Transformer approaches (Hendrycks et al., 2020), and therefore require a rich representation. Recent work (Hendrycks et al., 2021) demonstrates that current state-of-the-art Tranformers fail at few-shot learning these datasets, as well.

Beyond answering questions, our system is also capable of generating hints to help students learn. The ability to automatically generate hints could benefit instructors and students, and suggests the opportunity for a machine learning models in higher education.

1.2. Related Work

1.2.1. Few-Shot Question Answering

Recent work on Transformers and few-shot learning, such as GPT-3 (Brown et al., 2020), demonstrates the results of language models using few-shot learning for question-answering. Current work explores the effectiveness of such models in answering questions from a variety of academic subjects, ranging from the humanities and arts to science, technology, engineering, and math (STEM) subjects (Hendrycks et al., 2020). In this dataset, one of the most difficult topics for Transformers are STEM with topics such as Formal Logic, Abstract Algebra, and High-school Math barely scoring above random chance. Our work goes beyond pure Transformers, to generate question answers using expression trees.

1.2.2. Math Word Problems

A math word problem is defined as a collection of sentences that form a question for which there is a single correct numerical answer. The goal of our work is to construct a model that correctly answers the question. The solution to a math word problem is obtained by applying mathematical operations to numerical values available in the problem, numerical values derived from them, or constants that are associated with the problem text, such as π or e. Math word problems are primarily problems of comprehension, which motivates the use of Transformers. For humans, the performance of solving math word problems is connected with working memory capacity and inhibitory skills. Humans may generate a graph representation for word problems as a heuristic to reduce the problem complexity (Verschaffel et al., 2020). The ability to select among different graph representations and the knowledge about such different representations is referred to as meta-representational competence (Verschaffel et al., 2020). Solving a math word problem may require associating numerical values in the text with entities, using the relationship between multiple entities and numerical values to find a solution. Solutions to math word problems are often in the form of an expression tree. In addition, math word problems have also been included in datasets which are used to benchmark the performances of machine learning models (Hendrycks et al., 2020, 2021).

Given a math word problem, early work (Kushman et al., 2014) learns to select a template equation using a hand-designed set of features and aligns the numerical values to the template equation. Recent work (Xie and Sun, 2019; Wu et al., 2020; Qin et al., 2020) directly generates the expression tree representing an expression which answers the math word problem. Developing the expression tree decoding method further, other work (Zhang et al., 2020) introduces a graph representing the relationships between words and quantities and a graph representing the relationships between different quantities. Graph-to-tree neural networks (Li et al., 2020) consist of a graph encoder and hierarchical tree decoder,

mapping structured inputs to outputs. Recent work on expression pointer Transformers (Kim et al., 2020) generate expression tokens consisting of operators and operands to overcome the fragmentation of expressions and the separation between operands and context. Expression pointer Transformers (Kim et al., 2020) use an efficient ALBERT Transformer with state-of-the-art results. The methods used by these works follow a general approach: encoding the input question, processing the encoded information, decoding the processed encoding, and evaluating the decoded information. Such an approach demonstrates the ability of mathematical reasoning. Our work for solving machine learning problems uses an expression tree representation within an encoder-decoder framework. In contrast with all previous work on solving math word problems, our approach naturally handles questions involving recursion.

1.2.3. Graph Neural Networks

Graph neural networks (GNNs) (Kipf and Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Xu et al., 2019) are commonly used for predicting properties of particular nodes, edges, sub-graphs or the entire graph. In this work, we use GNNs for predicting an entire graph property, which is the encoding of a tree representing an expression.

1.2.4. Neuro-Symbolic Computation (NSCs)

Recent literature on neuro-symbolic computation have shown its strength in aiding the explainability of ML models (d'Avila Garcez et al., 2019; Lamb et al., 2020). Coupled with GNNs, NSCs provide explainable ML models, which is invaluable in research.

1.2.5. Transformers

The development of the Transformer is perhaps the most significant recent breakthrough in natural language processing (Devlin et al., 2019; Shoeybi et al., 2019; Raffel et al., 2020; Brown et al., 2020) and computer vision (Dosovitskiy et al., 2020). Transformers are built from attention mechanisms (Vaswani et al., 2017). An unsupervised corpus of text is transformed into a supervised dataset by defining content-target pairs along the entire text. Transformers are trained with the objectives of predicting masked target words in sentences and predicting whether two sentences follow each other. A language model is first trained to learn an embedding of words or sentences, followed by a map from low dimensional content to target (Mikolov et al., 2013). This embedding is then used on a new, unseen and small dataset in the same low-dimensional space. Our work incorporates Transformers for both augmenting machine learning questions and solving them. Each of our models consists of a Transformer for encoding text. We use the HuggingFace Transformer library (Wolf et al., 2020) and OpenAI GPT-3 (Brown et al., 2020) for comparisons.

1.2.6. Hints

An example of previous work on hint generation uses a factory model (Stamper et al., 2008). A Markov Decision Process (MDP) generator is used, along with a hint provider, to articulate hint in text format. The state space of hints is reduced using abstract syntax trees. A similar approach applies automatic hint generation for programming problems

(Rivers and Koedinger, 2013) using large corpuses of previous students work, which provides information about most common correct solutions.

2. Methods

In this section, we describe our dataset, system architecture, and testing methodology. The key components leading to our success are: (i) a dataset of Machine Learning questions annotated with expression trees representing their solutions; (ii) a data augmentation technique which allows us to automatically generate new, related questions; and (iii) the use of Transformers and graph neural networks to generate expression tress for solving problems, rather than just predicting numerical answers.

2.1. Dataset and Data Augmentation

The questions used for training and testing our model are drawn from MIT's Introduction to Machine Learning course, also known as 6.036, which is available online[1]. Due to the novelty of training a machine learning model to answer Machine Learning questions, we curate a new dataset from 6.036 exercises, homeworks, and quizzes. Example questions and answers are shown in Table 2. Questions involving an image or coding questions are not used in our dataset, and are the limitation of our approach. We present example collected questions in the SM. To prepare the dataset we collect all questions from exercises, homework, and exams for MIT 6.036 Fall 2020. We then remove questions which require graphical input or non-numerical output. Next, we generate expression trees based on problem solutions, so that we have the problem, expression tree, and solution templates. Finally, we automatically generate additional problems from the question templates.

Expression Trees We further enhance this dataset by including not only answers to the question, but also expression trees representing how the answer is calculated from information in the question. We find that a relatively small set of expression trees is sufficient to cover most questions. Nodes in the tree are either quantities that appear in the question, special constants, or mathematical operations. The allowed set of operations are $\{+, -, /, \times, \max, \log, \exp\}$. The SM consist of a comprehensive set of expression trees our system automatically generates from machine learning questions.

Data Augmentation After collecting questions from 6.036 homework assignments, exercises, and quizzes, each question is augmented by paraphrasing the question text. This results in five variants per question. Each variant is then augmented by automatically plugging in 100 different sets of values for variables. Overall, we curate a dataset consisting of 14,000 augmented questions. Even though there are only 28 canonical expression tree structures, this represents a large space after augmentation and scales up. We provide an example in the SM which shows an example of questions generated by augmentation: (i) question; (ii) template; (iii) paraphrased template; (iv) paraphrased question (1 out of 5 questions); (v) augmentation by plugging in examples values (1 of 100 questions) into each paraphrased template; (vi) final expression; and (vii) answer.

1. https://introml.odl.mit.edu

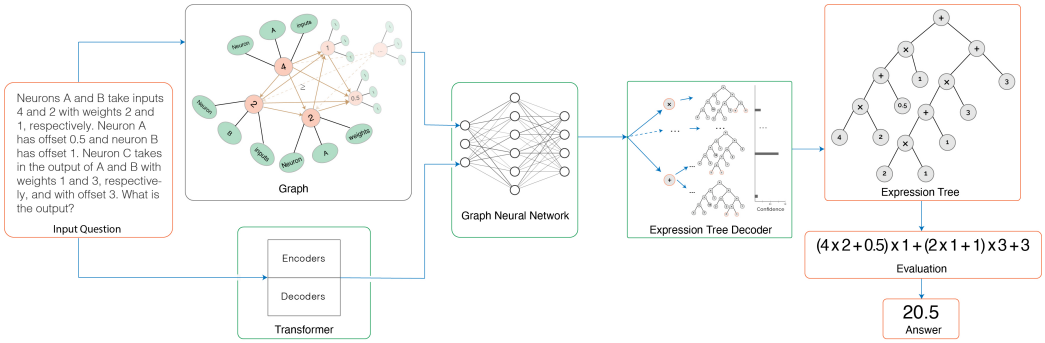

Figure 1: Architecture for solving Machine Learning questions. We create the question text embedding by a Transformer and also a graph representing the relationship between words and values and among values. These are passed through a graph neural network to create a vector in latent space which is then decoded to generate expression trees. The process of inputting the same question into the model is repeated a fixed number of times, resulting in multiple outcomes due to dropout in the GNN. The expression tree with the highest confidence is parsed to get the expression and the final value.

2.2. Architecture

Figure 1 illustrates our architecture. The model takes a question as input returns a numerical answer. Each input question is passed into a Transformer model to encode the question and also a graph which parses the question into its words and numerical components. The Transformer creates an embedding for the input question. The graph collects the words and numerical values of the question and formulates relationships between words and values and between values in the question. The embedding of the question from the Transformer and the graph are passed together into a GNN to create an embedding, which is subsequently decoded by a Tree Decoder to generate an expression tree. The model evaluates the generated expression tree and returns the computed value as output.

Graph Representation A Machine Learning question consists of words and quantities, and our method generates an expression tree which upon evaluation provides an answer. In order to capture the relationship between the problem text and the output tree, we represent the question as a graph where the nodes represent values and words, and the edges represent relationships between nodes as shown in Figure 1. Our model finds the maximum-likelihood estimate tree given the constructed graph and the possible node values.

Expression Tree Decoder A key feature of our architecture is the generation of an expression tree for each question which is then evaluated to produce an answer. The decoder generates the tree in a depth-first fashion using the current node and the latent encoding of the tree to generate either an operator or number for that node and to create children if applicable.

2.3. Additional Components

Our primary architecture is aimed at answering open-response numerical questions. Here we describe the alterations made to handle multiple choice questions and problem hint generation.

2.3.1. Solution Ranking

To answer multiple-choice questions, we sample our model multiple times to create a solution distribution, rank each solution in order of the frequency of appearance, and select the most frequent solution as the answer. Our architecture reflects this algorithm by performing a beam-search to return a list of answers from most to least confident. Although our model is able to produce results which are not dependent on answer choices, we find it important that our model take advantage of this extra information. This naturally lends itself to finding the most probable expression tree, conditioned on the fact that it must evaluate to one of the answers.

We also seek to create a system which generates incorrect answer questions close to the correct answer, similar to human-created multiple choice exams. We define an answer as "close" to the correct answer if a series of small perturbations of the correct expression tree yields another expression tree that evaluates to that answer. We consider two types of perturbations: rotating the tree, and adjusting the leaf nodes. Rotations are similar to those of balanced binary search trees, and are performed for trees with at least 5 nodes. We also consider adjusting leaf nodes by multiplying or adding a constant value. Both strategies are used to create sufficient incorrect answers to present at the model.

2.3.2. Hint Generation

In addition to solving questions, we also generate hints for questions answers. Our generated hints provide a range of outputs, from classifying questions by topic, to partial solutions, to full solutions, as well as provide partial or full examples with different values for the variables. Both partial solutions and example solutions with other values help students fill-in missing knowledge gaps and learn by example. By enabling the model to generate hints, we are able to leverage the understanding of the model to aid the understanding of learners stumped by Machine Learning questions. This provides a more responsive and scalable system than relying on instructor assistance alone.

The generated hints in our model are categorized into three types:

- Expression hints: include expressions or sub-expressions with variables from the question. Each hint is an expression which is generated using a partial solution of the question.
- Value hints: include expressions or sub-expressions with numerical values from the question. These are types of hints that directly provide information from the values used in the question.
- Example hints: include expressions or sub-expressions with values different from those found in the problem. These are generated using different values from the ones used in the question, demonstrating steps of the solution by examples.

Figure 2 depicts self-contained hints and the progression of providing hints to the learner until a full solution is be achieved.

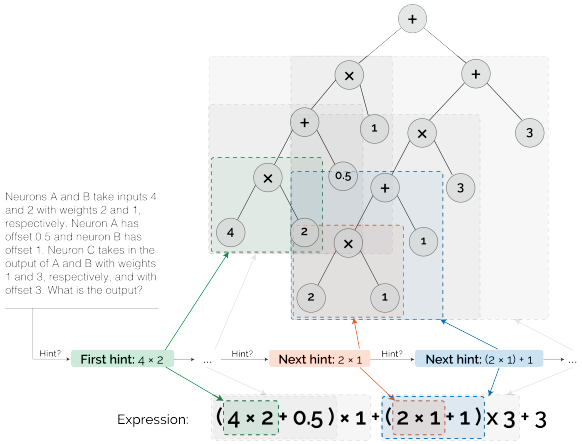

Figure 2: Generating successive hints for a machine learning problem. The expression tree is traversed in a binary search tree order starting from the left-most non-leaf node to give the sub-expression rooted at each non-leaf node as hint.

2.4. Model Evaluation

The evaluation of our model is two-fold; we determine its accuracy in reformulating the expected mathematical expression to solve the problem, as well as its accuracy in achieving the correct computed result. We refer to these performance metrics as:

- Expression accuracy: number of answers with the correct expression.
- Value accuracy: number of answers with the correct numerical value.

We take random disjoint training and test sets with an 0.8/0.2 split. We determine the accuracy of our model by calculating the percentage of questions which the model gets correct. To determine the accuracy per topic, we collect a mapping between each topic and the templates for that topic. We recover the topic of a question which has been correctly responded to by removing numerical values and representations in the question and matching it to one of the original templates.

Class Details and Grading MIT's 6.036 Introduction to Machine Learning [2], has around 500 students. Graded assignments for the course include:

- Exercises: short quizzes submitted before lectures.
- Quizzes: timed questions that do not contain coding questions.
- Homeworks: mostly coding problems.

2. MIT 6.036 website: introml.odl.mit.edu.

Quiz and homework questions are typically more difficult than exercise questions. We use questions from all the above, excluding questions which consist of input images that are required for the solution and excluding coding questions. Overall, around 58% of the questions are multiple choice, and 42% are open questions. Exercise questions do not consist of required input images or coding problems.

For exercises, MIT students are given an unlimited number of attempts at a problem. If they ever respond with the correct answer they receive 1 point for the problem, otherwise they received a 0. To prevent our model from simply brute-forcing answers our grading scheme is as follows:

- For open-response questions (ORQ): we give our model one chance to provide an answer, and if the answer is correct, we get 1 point, otherwise no points.
- For multiple-choice questions (MCQ): with n choices and the correct answer chosen at attempt t, we give our model $\frac{n-t}{n-1}$ of a point.

2.5. Implementation Details

Our code is in Python, using the libraries Pytorch[3], Deep Graph Library (DGL)[4], HuggingFace's Transformers[5], and comparisons using the OpenAI API[6]. DGL is primarily used for its graph convolutional network representation, while the Transformers library provides the necessary tokenizers and pre-trained models. All experiments are run on a standard cloud instance optimized for compute. Training time is around a day, augmentation time is less than a minute, whereas test time is milliseconds per question. We make our code available in the SM. Our implementation takes roughly 3 hours to train, including data augmentation, model training, and model evaluation. Data augmentation takes less than 30 seconds to perform, creating 14,000 questions from 140 paraphrased questions. Model training takes about 6.5 minutes per epoch, and evaluating the model on the 2,800 test examples takes about 1.5 minutes.

3. Results

We find that our model achieves grade A performance on its evaluation, when evaluated on both open-response questions and multiple-choice questions. A summary of the results are shown in Table 1 and Figure 3. This is performed at super-human speed, answering each question in milliseconds, whereas humans typically take minutes to solve questions. Table 2 shows example questions and answers generated by our model for each topic. The SM provides 60 additional example questions and answers generated by our model, five for each of the course topics. Figure 4 shows an example expression tree automatically generated by our model given a question. The SM provides 25 expression trees automatically generated by our model from questions in our dataset.

Table 3 shows a multiple-choice question, its associated answer choices, and the output produced by our model. For this example, the model is sampled 100 times and selects the

3. https://github.com/pytorch/pytorch
4. https://github.com/dmlc/dgl
5. https://github.com/huggingface/transformers
6. https://beta.openai.com

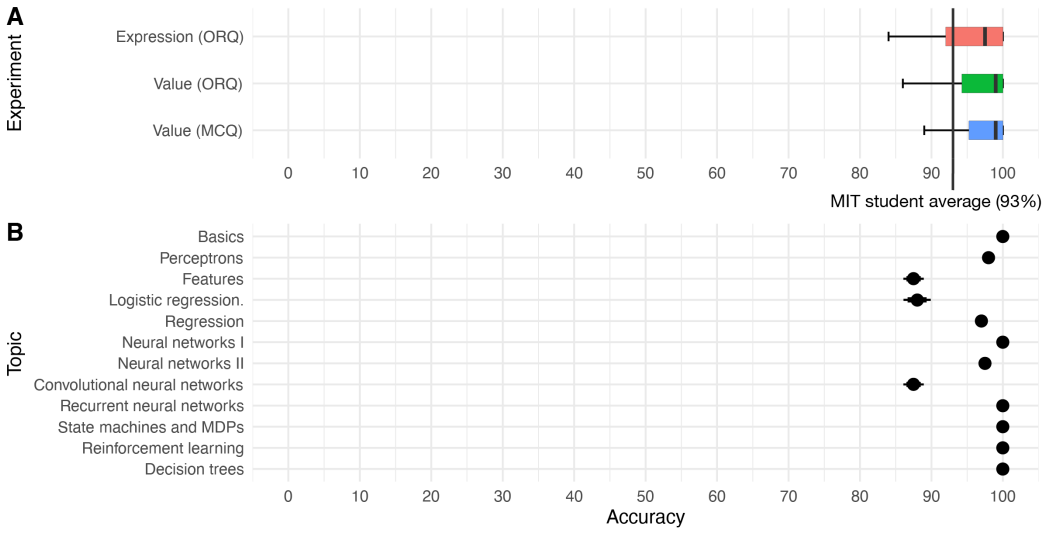

Figure 3: Accuracy distributions across **(A)** evaluation methods and **(B)** topics. The line overlay indicates the average student score.

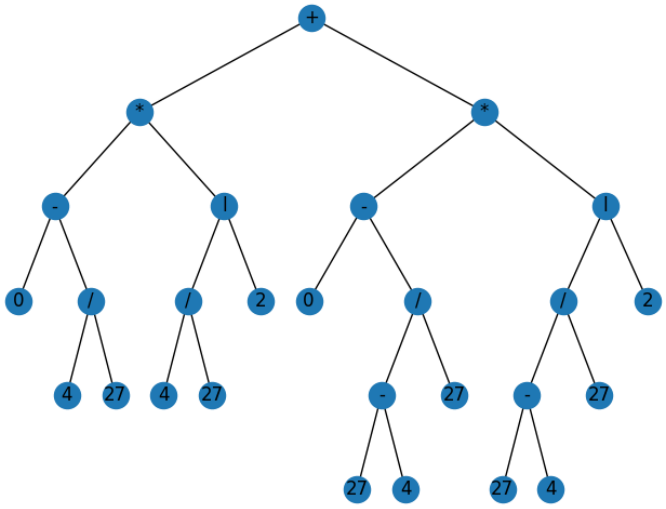

Figure 4: An automatically generated expression tree for the question "What is the entropy of the left side of a region containing 27 points where the plane has 45 points in total and 4 points on the left are positive?" Note "l" denotes the log operator.

highest probable answer from its 4 choices, where the correct answer is 1.5. Our model finds the correct answer in the first attempt. Table 4 shows an example question and the resulting types of hints and three different types of hints generated by our model. This extension of the model to procedurally generate hints based on the solution equation allows it to provide insightful hints for learners struggling with the problem.

Topic	Question	Answer
Basics	Compute the magnitude of $[10, 10, 1]$.	14.18
Perceptrons	If the decision boundary of a classifier is θ, where $\theta = (4, 1)$, how does it classify point p, where $p = (2, -4)$?	4
Features	A point 1 has label 1. Compute the margin of a classifier on this point. Let the θ of the classifier be -1 and the θ_0 of the classifier be -1.	2
Logistic Regression	If we have $x = (0, -1)$, $\theta = (1, -2)$, and $\theta_0 = -3$, then what is the result of $\theta x + \theta_0$?	-1
Regression	If we let $\theta = 1$ and $\lambda = 0.5$, what is the mean squared error of the given points $[(2, 0), (1, 1)]$?	2.25
Neural Networks I	Neurons A and B take inputs 4 and 2 with weights 2 and 1, respectively. Neuron A has offset 0.5 and neuron B has offset 1. Neuron C takes in the output of A and B with weights 1 and 3, respectively, and with offset 3. What is the output?	20.5
Neural Networks II	Compute the ReLU output of neuron C which takes the output of neuron A with weight 1 and neuron B with weight 2 and offset 1. Neuron B has input 0 and offset 1. Neuron A has input -1 and weight 2 with offset 0.5.	1.5
Convolutional Neural Networks	What is the minimum number of padding needed to maintain the same output size if the input image is 50 by 50 and the filter is 17 by 17?	8
State Machines and Markov Decision Processes	Consider the input $x_t = [7, 14, 13, 10]$ to a state machine with equations $s_t = f(s_{t-1}, x_t)$ and $y_t = g(s_t)$. Compute y_4 if our initial conditions are $s_0 = 2$, $f(s_{t-1}, x_t) = \max(s_{t-1}, x_t)$, and $g(s_t) = 4s_t$.	56
Reinforcement Learning	Let $q = 2$. After Q learning, what is q if $a = 0.3$ and $t = 8$?	3.8
Recurrent Neural Networks	Let $s_0 = 1.5$, $w = 1.5$, and $x = [1, 0, 2]$. Compute s_3 if $s_t = w * s_{t-1} + x_t$.	9.31
Decision Trees	Consider a 1D classification line on a 2D plane. There is a total of 46 points, 30 of which are on the right and the rest on the left of the boundary. 5 points on the left are classified positive. What is the entropy of the left region?	0.90

Table 2: A sample question from each of the machine learning course topics and the answer generated by our model.

Topic	Question	Multiple Choice Options	Output
Regression	Let 1 be the optimal θ by mean squared error. Given the data points $[(0, 0), (1, -1), (2, y)]$ and $\lambda = 1$, compute the value of y.	$[-0.24, -2.24, -0.64, -53.9]$	$[-0.24]$

Table 3: Multiple-choice question and answers. An example of the questions given to the model, the answers generated by the adversarial generator, and the output answers made by the model in order of the appearance. Note that the final value in output is the correct answer of the question, and the answering stops once the models achieves the correct answer. The correct answer is selected in the first attempt.

Question	A neural network has input x_1 with weight w_1 that goes into neuron A. Neuron A also has input OA that has weight w_{OA}. Neuron C inputs the output of neuron A with weight w_{AC}. Neuron C also has input OC that has weight w_{OC}. Neurons output the sum of the products of each input with their respective weight. at has weight w_{OA}. Neuron C inputs the output of neuron A with weight w_{AC}. Neuron C also has input OC that has weigh w_{OC}. Neurons output the sum of the products of each input with their respective weight. What is the output of neuron C if $x_1 = 2$, $w_1 = 1$, $OA = 0.5$, $w_{OA} = 2$, $w_{AC} = 1$, $OC = 1$, and $w_{OC} = 3$?
Answer	$(OA * w_{OA}) * w_{AC} + OC * w_{OC} = (0.5 * 2) * 1 + 1 * 3 = 4$
Expression Hints	1. $(OA * w_{OA})$ 2. $(OA * w_{OA}) * w_{AC}$ 3. $OC * w_{OC}$
Value Hints	1. $0.5 * 2 = 1$ 2. $(0.5 * 2) * 1 = 1$ 3. $1 * 3 = 3$
Example Hints	For an example where $OA = 8, w_{OA} = 9, w_{AC} = 3, OC = 2, w_{OC} = 4$: 1. $8 * 9 = 72$ 2. $(8 * 9) * 3 = 216$ 3. $2 * 4 = 8$

Table 4: Example questions and hints generated by our model.

3.1. Baseline and Ablation Study

To baseline the performance of our model, we use 120 open-response questions, with 10 questions per topic. We use a small number of questions since unlike our model which returns the numerical answer to the question, the responses of GPT-3 vary in length and content, thus requiring manual parsing of the responses given by GPT-3. We also perform an ablation study without the graphs and GNN such that the result of the Transformer feeds directly into the tree decoder. We compare the results of these two models with that of GPT-3 tuned on machine learning in Table 5. GPT-3 completely fails, achieving an overall value accuracy average of 7%, while our original model achieves an overall value accuracy of 90% and the reduced model achieves an overall value accuracy of 87%. We present a detailed breakdown of the results comparing these three models by topic in Table 5. To achieve these results, we train the models twice and take its averaged performance to mitigate randomness. These results highlight that although GPT-3 is able to generate text about concepts in Machine Learning, it completely fails in solving Machine Learning problems. This is expected, since GPT-3 has no understanding of simple topics in Linear Algebra and Calculus, such as the magnitude of a vector.

3.2. Data Augmentation Performance

We evaluate the performance of our data augmentation by varying the magnitude of the number of augmentations performed. For each topic, we downsample the number of produced augmentations, and we present the results in Table 6. All topics generally decrease in performance as the number of augmentations decrease. However, we find topics which have many more questions to decay more gracefully than topics which contain fewer canonical questions. Certain topics may have spikes of higher performance at lower augmentations, which may occur due to the smaller solution space as the number of augmentations is de-

Week	Topic	Our Model	Our Model without GNN	GPT-3
1	Basics	1.00	0.50	0.00
2	Perceptrons	1.00	0.95	0.20
3	Features	0.65	0.50	0.00
4	Logistic regression	0.70	0.50	0.20
5	Regression	1.00	0.75	0.10
6	Neural networks I	1.00	0.95	0.00
7	Neural networks II	1.00	0.90	0.00
8	Convolutional neural networks	0.90	0.95	0.00
9	Recurrent neural networks	1.00	1.00	0.20
10	State machines and MDPs	1.00	1.00	0.10
11	Reinforcement learning	1.00	1.00	0.00
12	Decision trees	1.00	0.90	0.00
	Overall average over topics	0.94	0.83	0.07

Table 5: A comparison of the ORQ performance of our model vs. our model without GNN vs. GPT-3.

		Augmentations per Question					
Week	Topic	100	50	25	13	6	3
1	Basics	1.00	0.42	0.11	0.01	0.07	0.00
2	Perceptrons	0.98	0.71	0.35	0.22	0.22	0.07
3	Features	0.86	0.61	0.36	0.11	0.07	0.00
4	Logistic regression	0.86	0.76	0.12	0.02	0.00	0.02
5	Regression	0.97	0.64	0.30	0.05	0.00	0.00
6	Neural networks I	1.00	0.86	0.34	0.01	0.05	0.06
7	Neural networks II	0.97	0.91	0.13	0.02	0.01	0.00
8	Convolutional neural networks	0.86	0.91	0.54	0.12	0.06	0.04
9	Recurrent neural networks	1.00	0.61	0.23	0.00	0.05	0.00
10	State machines and MDPs	1.00	0.95	0.26	0.12	0.13	0.00
11	Reinforcement learning	1.00	1.00	0.64	0.00	0.02	0.00
12	Decision trees	1.00	0.80	0.60	0.00	0.00	0.00
	Overall average over topics	0.96	0.77	0.33	0.06	0.06	0.02

Table 6: Performance of the model as the number of augmentations per question varies across all 12 topics. To reduce randomness and variability, these values are averaged across the performance from retraining the model four times.

crease.d To account for randomness and variability within the model, the model is retrained four times, and we display the averaged results for each topic.

3.3. Recursion

We also demonstrate the capability of our system to solve questions which are recurrent in nature. We convert values to variables and allow for relationships across such variables. This is shown in the example in the Table, whose paraphrased template lays the foundation to allow for a variable depth recursive expression tree to be learned by the model. In the example, this is done by creating a relationship between the length of the input x and the expected amount of recursive times needed to yield the solution, $s_{|\{x\}|}$. This process is done by our data augmentation on recurrent neural network questions, allowing for the model to learn the recursive nature of such problems, and as a result, data augmentation generates the data by which the model is trained to solve recursive problems. This is reflected in a score of 100% on the recurrent neural networks topic in Table 1.

3.4. Limitations

Despite demonstrating grade A performance on questions from MIT's Introduction to Machine Learning course, our model has several limitations. We provide examples of questions with incorrect answers in the SM. A key component missing in our current model is the ability to comprehend questions involving pictorial components. In addition, our model is unable to write code for machine learning-related coding questions, which are present in MIT's Introduction to Machine Learning course. We are currently expanding upon the capabilities of our current system to address these limitations.

4. Conclusion

We present the first machine learning system capable of solving University undergraduate level Machine Learning questions with high accuracy, evaluated on both open-response questions and multiple-choice questions and demonstrating an overall grade A performance, and superseding MIT students. This work expands upon the current field of training machine learning models to learn courses, and we demonstrate the ability to create a model capable of solving machine learning problems by leveraging data augmentation techniques, such as templating and paraphrasing, a Transformer model, and a solution ranking procedure. In addition, we find our model capable of solving questions which are recurrent in nature.

We also explore hint generation, opening the door to creating a teacher model which procedurally generates hints for learners. By leveraging the data augmentation component of our system with the hint generator, we are able to give students hints which support their understanding of the problem. This may automate labor intensive parts of running a course with minimal supervision. In addition, hint generation and providing methods of solving a problem demonstrates a much higher level of explainability compared to solely giving a numeric answer, which helps advance the area of explainable AI. An extension of this work would be to expand upon the current size of training data, scaling up the data augmentation process to multiple courses. We are currently working on solving discrete math questions, specifically from MIT's Mathematics for Computer Science course, 6.042. In the future we would like to extend the set of questions handled to include coding questions and as well as question with visual components, using multi-skilled and multi-modal models.

References

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Artur d'Avila Garcez, Marco Gori, Luis C. Lamb, Luciano Serafini, Michael Spranger, and Son N. Tran. Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning. *arXiv e-prints*, art. arXiv:1905.06088, May 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference*

of the North American Chapter of the Association for Computational Linguistics, pages 4171–4186, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Bugeun Kim, Kyung Seo Ki, Donggeon Lee, and Gahgene Gweon. Point to the expression: Solving algebraic word problems using the expression-pointer transformer model. In *Conference on Empirical Methods in Natural Language Processing*, pages 3768–3779, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *Annual Meeting of the Association for Computational Linguistics*, pages 271–281, 2014.

Luis C. Lamb, Artur Garcez, Marco Gori, Marcelo Prates, Pedro Avelar, and Moshe Vardi. Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective. *arXiv e-prints*, art. arXiv:2003.00330, February 2020.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Conference on Empirical Methods in Natural Language Processing*, pages 2841–2852, 2020.

Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, pages 1–12, 01 2013.

Jinghui Qin, Lihui Lin, Xiaodan Liang, Rumin Zhang, and Liang Lin. Semantically-aligned universal tree-structured solver for math word problems. 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning

with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140): 1–67, 2020.

Kelly Rivers and Kenneth R. Koedinger. Automatic generation of programming feedback: A data-driven approach, 2013.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using GPU model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

John Stamper, Tiffany Barnes, Lorrie Lehmann, and Marvin Croy. The Hint Factory: Automatic generation of contextualized help for existing computer aided instruction, 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2018.

Lieven Verschaffel, Stanislaw Schukajlow, Jon Star, and Wim Van Dooren. Word problems in mathematics education: A survey. *ZDM*, pages 1–16, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing Huang. A knowledge-aware sequence-to-tree network for math word problem solving. In *Conference on Empirical Methods in Natural Language Processing*, pages 7137–7146, 2020.

Zhipeng Xie and Shichao Sun. A goal-driven tree-structured neural model for math word problems. In *International Joint Conference on Artificial Intelligence*, pages 5299–5305, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. Graph-to-tree learning for solving math word problems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937, 2020.