AN END-TO-END GRADING NEURAL NETWORK FOR MIDDLE-SCHOOL MATH PROBLEMS

BY

MENG LIN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Advisers:

Assistant Professor Nan Jiang
Professor Bruce Hajek

ABSTRACT

Mathematics homework grading is a common real-world task where a human grader checks a student's solution to a math problem against the answer key and gives a score. This thesis proposes a deep-learning-powered grader that takes the place of the human grader. The task is formulated as a classification problem. Given an answer key and a student's solution, the classifier needs to predict two metrics: (1) a four-class classification result that measures the completeness of the student's detailed steps and (2) a binary classification result that identifies whether the conclusion of the student's solution is accurate. A new model, Step Comparison Transformer (SCT), is introduced, and its performance is validated on a set of grading data provided by a commercial provider of artificial intelligence products for education.

To my parents, for their love and support. To the ECE department for providing me the template. To Learnable.ai for data and funding support.

ACKNOWLEDGMENTS

I thank my adviser, Prof. Bruce Hajek, for his kindness, generosity, and support. And I thank Professor Nan Jiang for the patient guidance, encouragement and advice he has provided throughout the time I have worked on the thesis. I sincerely appreciate the data, funding, and other help provided by Learnable.ai, an AI-EDU company that supported this work.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

There are three forms of problems in K-12 mathematical assignment: multiple-choice, fill-in-the-blank, step-by-step reasoning/calculation. The grading process of the first two is easy to automate thanks to their simplicity, formality and universality. However, the grading of step-by-step reasoning/calculation remains an open research problem. First, the information contained in the steps is rich and involves complex logic. Second, there are multiple solutions to the same problem which are equally valid but not necessarily captured by the answer key. Third, the solutions are written in both natural language and formal language, making the design of a good representation challenging. Fourth, much of the information is not fully observed. Fifth, a good grader must consider not only the correctness of the final conclusion but also the integrity of the entire reasoning process.

Given these complexities and challenges, researchers usually first limit the scope of the task. Kadupitiya et al. [1] and Lan et al. [2] focus on algebra and simple numerical problems. In addition to a constraint on the subject of the problems, their systems also depend on a detailed, elaborate grading rubric that specifies how to grade and when to mark scores down. This requirement makes real-world application of such systems unrealistic. Seo et al. [3], [4] focus on geometry problems. These works developed a system based on feature engineering and nonlinear optimization to solve SAT math questions that involve simple geometry. However, SAT problems are all multiple choice. Although it is possible to extend their existing work to the domain of step-by-step reasoning, no work has achieved any promising progress.

In this thesis it is assumed that a comprehensive human-level assessment of a student's problem comprises two metrics. One measures the integrity and rigidity of the steps. Since the degree of completeness is a scale, this

measurement gives a multi-class rank. The other measures the correctness of the final conclusion. Since the conclusion can be either right or wrong and there is no ambiguity, this measurement gives a binary pass/fail. Thus, the grading task can be formulated into two classification subtasks. Based on preliminary observations of the data and advice from pedagogical experts, we assign four classes to the first subtask: A, B, C, D, with descending rankings. The second subtask is a simple binary pass/fail.

For each round of grading, the machine grader is given a pair of answer key and student solution and should output a scale and a binary. There are other research fields whose major concerns are outputting a correctness or similarity score given a set of sentences. One well-studied problem is the answer selection task [5]. Different from doing pair-wise comparisons, the answer selection task deals with a question and a set of candidate sentences. It is concerned with identifying the candidate sentences that contain the correct answers to the target question. Traditional methods for tackling the answer selection task rely on feature engineering, extra sources and linguistic tools [6] [7]. In recent years, with increasing interest in applying deep learning techniques to natural language processing tasks, researchers have also proposed several deep learning methods [8], [9]. These deep learning models outperform traditional techniques. In addition, with sufficient training data, those methods do not need any feature-engineering effort or hand-coded resources.

Traditional machine learning and feature engineering methods have shown their limitations in handling this complex task of both grading and answer search. Meanwhile, recent progress on deep learning models in the field of natural language processing has demonstrated their huge potential in enabling transformative applications in many industries, including education. Inspired by such observations and advancements, we propose an end-to-end supervised deep learning method that aims to achieve human-level assessment and grading of K-12 step-by-step reasoning type math homework. The method resembles Compare-Aggregate Architecture [5] for the answer selection task. In short, the model first implicitly evaluates the importance of each step in the answer key. It also generates a one-to-one matching matrix between all the steps in the answer key and all the steps in the student's

solution. The importance vector and similarity matrix are then integrated to yield the assessment result.

This thesis will report experiments conducted on a data set labeled by a commercial provider of artificial intelligence products for education. The data set contains primary school and junior high school level Chinese mathematical problems. The subjects include simple algebra, arithmetic, geometry, and applications. Although experiments are conducted only in this scope, the approach is not domain specific and thus can be generalized.

Because the size of the data set is fairly small. The model overfits the training data severely. Several attempts were made, including regularization and auxiliary tasks, to mitigate the overfitting problem. This thesis provides detailed illustrations of these attempts and perspectives on their effectiveness.

CHAPTER 2

LITERATURE REVIEW

2.1 Deep learning based answer selection methods

There are several research fields related to this work. We briefly review comparatively new and popular works in these fields. Lai et al. [5] classify deep learning models for answer selection tasks into three classes:

1. **Siamese Architecture**. In the Siamese architecture, a set of input sentences (both candidate solutions and answer) passes through an identical sentence encoder to build the vector representations. Then the model compares the representations to determine the relevance score. There is no interaction between the sentences during the encoding process. An example of this architecture is reported in [10].

2. **Attentive Architecture**. In the attentive architecture, an attention mechanism is used to allow the information from an input sentence to influence the computation of the other sentence's representation. An example of this architecture is provided in [11].

3. **Compare-Aggregate Architecture**. In Compare-Aggregate Architecture, vector representations of some smaller units such as word representations are matched, then the matching results are aggregated to produce the final relevance score. An example of this architecture is provided in [12].

Most of those works are based on recurrent neural networks [13].

2.2 Mathematical word problem grading systems

Lan et al.[2] claim that theirs is the first mathematical language processing (MLP) system capable of evaluating open response mathematical questions and awarding partial credits. The system is developed for large scale MOOCS (massive open online courses), and the assessment results can include both ranking and pass/fail metrics. It can also point out steps where errors potentially occur. However, their work requires human graders to grade some solutions in a cluster of solutions first, and the cluster needs to be big enough to support satisfactory performance.

Badger [14] focuses on mathematics in general examinations. But the system collects solutions from the students through a fixed user interface in order to control the students' actions. Furthermore, the collected solution does not include natural language texts.

Kadupitiya et al.'s [1] approach combines the text similarity model and logical language based comparator for mathematical language. Like many other works, it only grades simple algebra and numerical problems. There will be a painfully large amount of human work to expand the method to a bigger scope.

CHAPTER 3

ALGORITHM DESIGN

This chapter introduces SCT (Step Comparison Transformer) and its detailed implementation. The model architecture is covered first, followed by the universal language encoder transformer [15] and the pre-training task adapted from Bert [16].

3.1 Model architecture

Figure 3.1 shows the entire structure of the model. Its inputs are answer keys and student solutions. Both are represented as matrices. Each row on an answer corresponds to a step. The length of a step varies. In order to form a matrix, we have to pad all steps to the length of the longest step on the answer key. Thus, the input matrix has shape $S \times l$, where S is the number of steps and l is the length of the longest step. Note that S is also not a fixed number. The answer key and student's solution for one question do not necessarily have to have the same number of steps. Most of the time,

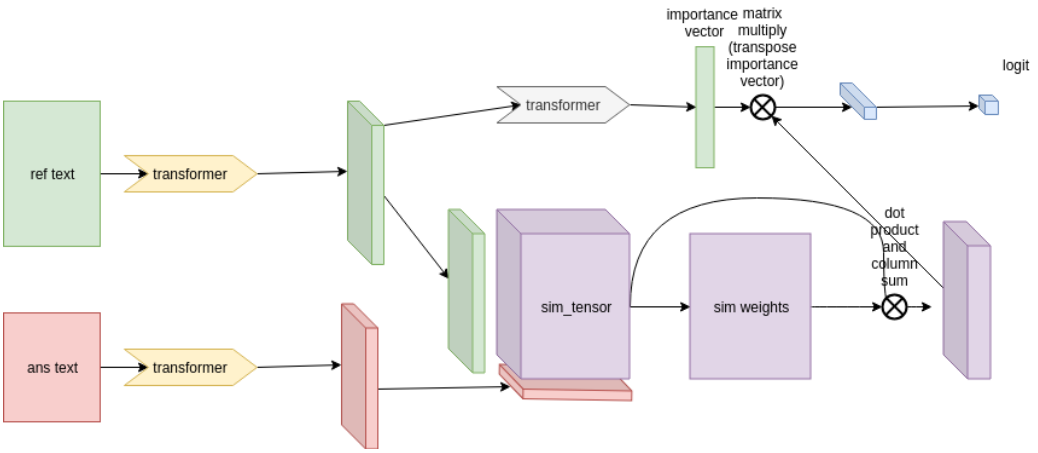


Figure 3.1: Whole model structure

the answer key has more steps because the editors wants to make sure the answer is elaborated.

If we ignore all the details of the model architecture, the tasks can be seen as typical k-class classification tasks. The architecture outputs a logit vector whose length is equal to the number of labels for the task. For the ranking task, there are four labels that indicate how close the student's solution is to the answer key. For the pass/fail task, there are two labels that indicate whether the right conclusion for the question exists. And the cost that the model tries to minimize is the cross entropy loss between sparse representations of ground-truth labels and logit vectors from the model.

Based on the model classes defined by [5], our model is a Compare-Aggregate Architecture because when comparing answer key and reply, the model matches small units to get matching representations. Then the model aggregates the matching representation to give a final relevance representation. For our model, the small units are steps.

$$\text{main loss} = \text{cross entropy}(\text{SCT}(\text{Answer, Reply}; \theta), \text{spare label logits}) \quad (3.1)$$

The model consists of three modules: the sentence encoding module which encodes each step on an answer to a vector representation; the importance evaluation module which evaluates the importance of each step on the answer key and presents it as a vector; and the similarity module which gives a similarity representation between all the steps on the answer key and all steps on student's solution. SCT then conducts weighted sum similarity representations for steps on answer keys with weights from the step importance vector to get a similarity vector between two answers. The vector then goes through a linear transformation with learned parameters to be a logit whose length is the number of labels.

3.1.1 Sentence encoder

The sentence encoder encodes each step on the answer key and the student's solution to a unit fixed depth representation. As shown in figure 3.1, it is based on a novel neural network language modeling approach called transformer [15]. Both inputs are encoded by identical transformers that share weights. After this step, the representation of the answers becomes a matrix that has shape $S \times d_l$ where d_l is the depth of encode representation. For simplicity, we name the matrix representation for answer key ASR, which stands for answer key step representation, and the matrix representation for student reply RSR, which stands for student's reply step representation. ASR has shape $l_k \times d_l$ where l_k is how many steps are there for the answer key. RSR has shape $l_k \times d_l$ where l_k is how many steps there are for the reply.

3.1.2 Importance vector

The encoded representations for the answer key are used to estimate the importance of each step. This process is achieved by feeding RSR through a one-layer transformer followed by a fully connected layer consisting of a linear transformation with learned weights and a sigmoid activation function. The intuition for this step is that when humans are grading a solution by a student, they usually only pay attention to a few steps that are crucial to the problem. If all the pivotal steps are correct, some typos or minor mistakes can be forgiven. The step importance estimator aims to implicitly learn to find those important steps from the answer key representation.

The model handles two tasks: the ranking task and the pass/fail task. The two share the same sentence encoding and similarity representation modules, but there are different importance vectors for each task. The intuition is that the sentence encoder and similarity between steps are independent of the type of assessment metric being used. Yet the importance vector is the defining difference between the two metrics. For the pass/fail task, only the conclusion step matters, whereas for the ranking tasks, all steps are important but on a smaller scale.

3.1.3 Similarity representation

SCT calculates a vector similarity representation between all the steps on ASR and RSR, and arranges them on a tensor with shape $l_k \times l_r \times (2 \times d_l)$. For the i^{th} step on the answer key and the j^{th} step on the student's solution, the similarity measure between them is on $ST_{i,j}$ where ST is an abbreviation for similarity tensor. The similarity measure is defined as

$$ST_{i,j} = \text{COS}(ASR_i, RSR_j) \circ \text{l1}(ASR_i, RSR_j) \tag{3.2}$$

where COS is the element-wise dot-product of normalized vectors by calculating the cosine similarity without summing over the hidden dimension, l1 difference is the element-wise absolute difference and $\circ$ is the concatenation operation.

We choose to concatenate the outputs of normalized dot-product operation and l1 diff operation to form ST because cosine similarity is an angular similarity and l1 difference is a line distance sensitive to the magnitude of the vectors. Intuitively, we can assume that the linear model can better evaluate the similarity with more information. Tai et al. [17] proved that the use of both distance measures is empirically better than the use of one distance measure.

The goal of this module is to have a similarity representation between each step from the answer key and a student's solution. For any step from the answer key, We define its similarity to a student's solution as its maximum similarity among all the steps from that student's solution. Thus, we have to select a similarity vector from several similarity vectors. To achieve this, we need (1) a more straightforward similarity representation that explicitly shows how similar two steps are and (2) an unambiguous way to 'select' a similarity representation from ST. To satisfy the first need, SCT forms a similarity matrix (SM) which has shape $l_k \times l_r$. An entry $SM_{i,j}$ on the matrix is a scalar whose magnitude reflects the proximity of ASR_i and RSR_j. $SM_{i,j}$ is calculated as weighted sum of elements on vector $ST_{i,j}$. The weight vector is learned from samples. To satisfy the second need, SCT applies softmax

with low temperature to each row on SM to build a relatively sparse weight map. For a vector $\mathbf{V}$ with length n:

$$\text{softmax}(\mathbf{V}) = \{\frac{\mathbf{e}^{\frac{\mathbf{t_i}}{\mathbf{T}}}}{\sum \mathbf{e}^{\frac{\mathbf{t_i}}{\mathbf{T}}}} : \ \forall \mathbf{i} \in (\mathbf{1} \cdots \mathbf{n})\} \tag{3.3}$$

where T is a positive real number called temperature that controls the smoothness of the softmax function. An element-wise weighted sum over all rows on similarity tensor that uses the matrix after softmax as a weight map is performed to approximately 'select' one similarity vector in a differentiable way. If the temperature for softmax function is very close to zero, for a certain row i, the weighted sum will select one vector $ST_{i,j}$ whose j index is:

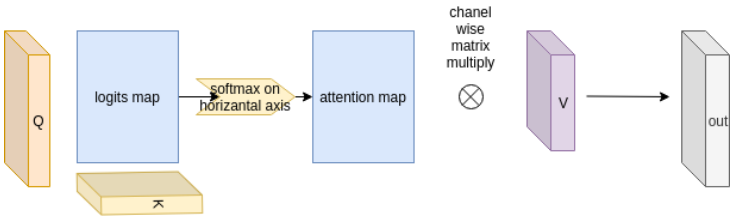
$$j = \text{argmax}_j SM_{i,j} \tag{3.4}$$

3.2 Transformer as sequence encoder

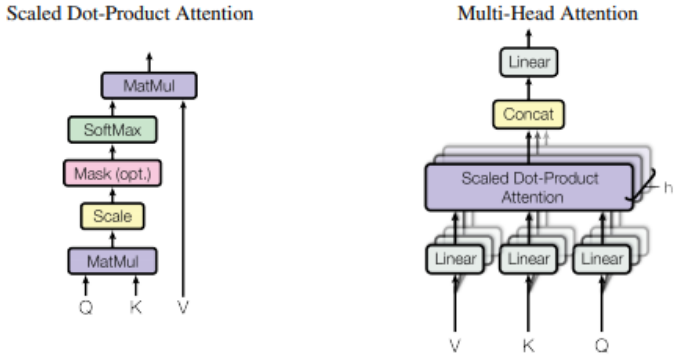
For decades, recurrent neural networks such as LSTM [13] and GRU [18] have been the state-of-the-art sequence modeling approaches. Transformer is a novel neural network based sequence modeling approach that solely relies on an attention mechanism to capture global dependencies. This model achieved state-of-the-art performance on language modeling and machine translation tasks. Furthermore, it is more time-efficient on the training phase because unlike recurrent networks, transformer's time efficiency does not depend on the length of the sequence. The following paragraph briefly introduces transformer and how to use it only as a sentence encoder.

Firstly, the transformer paper formulates the attention mechanism as a function that maps a query and a set of key-value pairs to an output,[1] where the query, keys, values, and output are all vectors. The output is a weighted sum of the values, where the weight matrix is computed with a compatibility function that takes a query and its corresponding key. The attention mechanism introduced on the paper is based on Dot-Product Attention.

[1] In practice, usually, there is one set of key-value pairs for one key.



(a) Dot-Product Attention



(b) Left figure is Scaled Dot-Product Attention and right figure is Multi-Head Attention. On the subfigure for Multi-Head Attention, the depth corresponds to words on the sequence.

Figure 3.2: Attention mechanism on transformer

$$\text{Dot-Product Attention}(Q, K, V) = softmax(QK^T)V \tag{3.5}$$

Because this attention uses dot-product, if the dimensionalities of Q K are large, the dot products will grow larger in magnitude and the softmax function will have smaller magnitude gradients. To solve the problem, with a simple normalization, we have:

$$\text{Scaled Dot-Product Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_l}})V \tag{3.6}$$

where d_l is the depth of Q, K, V. The transformer paper further revises Scaled Dot-Product Attention to Multi-Head Attention. For a Multi-Head Attention with k heads, the matrices Q, K, V, with shapes $d_q \times d_l$, $d_k \times d_l$, $d_v \times d_l$, are each linearly transformed k times into k matrices $[\hat{Q}_1, \cdots, \hat{Q}_k]$, $[\hat{K}_1, \cdots, \hat{K}_k]$, $[\hat{V}_1, \cdots, \hat{V}_k]$. Let $\hat{Q}_i$, $\hat{K}_i$, $\hat{V}_i$, $1 \leq i \leq k$ be elements on the set

of matrices on i^{th} positions. $\hat{Q}_i$, $\hat{K}_i$, $\hat{V}_i$ each have shapes $d_q \times d_l/k$, $d_k \times d_l/k$, $d_v \times d_l/k$. For all i, we apply Scaled Dot-Product Attention to $\hat{Q}_i$, $\hat{K}_i$, $\hat{V}_i$ to get $\hat{\text{out}}_i$. Then all $\hat{\text{out}}_i$ are concatenated back to get the result.

Transformer uses Multi-Head Attention in multiple ways. In this work, transformer is only used as a sequence encoder so we will only talk about self-attention. Within the current context, self-attention means a Multi-Head Attention whose inputs Q, K, V are from the same sequence embedding, but the sequence embedding goes through three independent linear transformations with different kernel weights to get Q, K and V. Figure 3.2 shows structures of Attention Mechanisms introduced in [15].

Recurrent neural networks such as LSTM and GRU automatically capture the relative and absolute position of each token, but transformers need extra source to tell the position of each token. Thus, transformer has "positional encodings" as the input embeddings at the beginning of the encoding process. The positional encoding has the same dimensionality as the word embeddings and each dimension corresponds to a sin or cos function with different frequencies.

$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i}/dmodel) \\ PE(pos, 2i+1) &= cos(pos/10000^{2i}/dmodel) \end{aligned} \tag{3.7}$$

where pos is the position and i is the dimension. It is recognized to be a good positional embedding because PE_{pos+k} can be represented as a linear transformation of PE_{pos}. Transformer also empirically proved its availability.

Intuitively, the position of a step in an answer is largely relevant to its similarity representation. Thus, before calculating the similarity tensor, we added positional encoding to ASR and RSR. Furthermore, since answer keys are usually more elaborate than student solutions, the absolute position for two steps with same meaning may not be close. We slightly modified the original positional embedding so that the pos in PE becomes the relative position of the step within the whole answer.

$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i}/dmodel/sequencelen \times maxlen) \\ PE(pos, 2i+1) &= cos(pos/10000^{2i}/dmodel/sequencelen \times maxlen) \end{aligned} \tag{3.8}$$

Transformer preserves the length of the sequence it encodes, but our goal is to get a vector representation for each step. Cer et al. [19] sum the hidden representation of the encoded sequence to get the vector representation. Collobert and Weston [20] select the maximum value over each dimension of the hidden units to get the vector representation. To get a sentence representation, we adopted the method of Bert, introduced by Devlin et al. [16], which is a representation learning method based on transformer. The model introduces two pretrain methods: the masked LM task and the next sentence prediction task. For the next sentence prediction task, the model has to generate a sentence representation. Bert's method is to append an E_{CLS} token to the beginning of a sequence. The hidden representation corresponds to the E_{CLS} token after the encoding process is used as the sentence embedding representation.

3.3 Regularization and auxiliary tasks

A big problem of the model is that there is no supervision for step similarity. The model only tries to minimize the main cost. Since the model is heavily parameterized and there are only a few data samples, it is easy for the model to overfit sample data without learning anything useful. And $SM_{i,j}$ may not be a similarity measure if there is no restriction on the linear transformation from $ST_{i,j}$ to $SM_{i,j}$.

To solve the problem, SCT minimizes a l1 regularization loss to the total loss of the model to restrict the magnitude of parameters. Data augmentation is also applied to help the sentence similarity module learn a real similarity metric. We also adopt the masked LM task mentioned above. Masked LM task first masks some percentage of the input tokens at random and then predicts only those masked tokens. To achieve this, a $[mask]$ token is added to the lexicon. Because the $[mask]$ token will never appear on the fine-tuning

stage, not all the tokens are substituted to $[mask]$ token. Instead, 80% of the time, tokens are substituted to $[mask]$, 10% of the time, tokens are substituted to randomly picked tokens that are not themselves, and 10% of the time, tokens are not changed.

In Bert, this masked LM task is only used to train word embedding representations. In my work, the masked LM task is used not only as a pretrain approach to initialize the word embeddings and the transformer encoder, but also as an auxiliary task which the model tries to learn to accomplish while learning to minimize the loss for the main task.

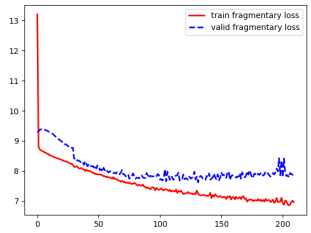
CHAPTER 4

EXPERIMENT

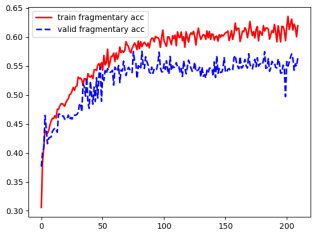
The experiment is performed on a dataset of real student assignment papers offered by Learnable.ai, an AI-EDU company. We split the sample data to training data and validation data. 80% of the data is used for training and 20% of the data is used for validation. The data contains 1185 samples in total. Each sample consists of an answer key, a student solution, the pass/fail label and the ranking label. The pass/fail label simply indicates if the correct answer exists. A detail worth mentioning is that for problems with multiple sub-questions, if the conclusion for the last sub-question is correct, no matter how the student answers the first few sub-questions, the solution would always receive a pass, indicating that the conclusion exists. The ranking grading label for an answer pair is one of the four levels."A" means that the student's answer is exactly the same as the answer key and the conclusion is correct. "B" means that the student's answer is similar to the answer key but there exists minor errors or missing steps. "C" means that the solution is very different from the answer key. "D" means that the solution is something random, not relevant to the problem at all. The data set includes question body for each sample but we did not use it. We implemented a crude rule-based line segmentation tool to split the answers into lists of steps.

4.1 Result

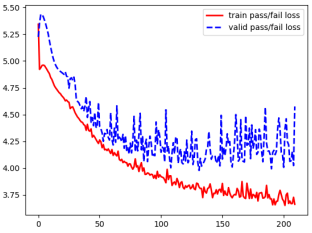
After tuning parameters, with heavy regularization, we get a balance on variation and bias. Figure 4.1 show that after training for certain epochs, the model continues to fit training data, but the model's performance on validation data did not drop nor improve. This phenomenon shows that early stopping may not be a necessity for this model. Figure 4.2 shows the con-



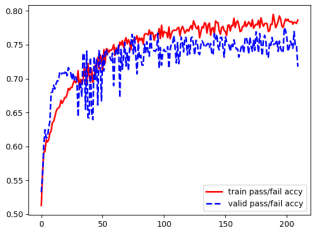
(a) fragmentary task cross entropy loss



(b) fragmentary task classification accuracy

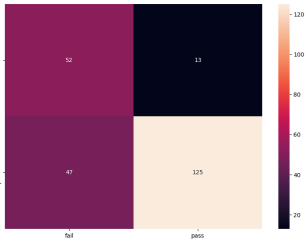


(c) pass/fail task cross entropy loss

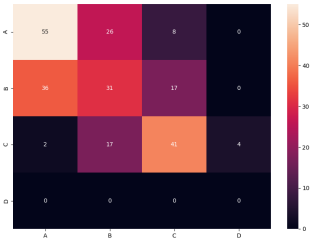


(d) pass/fail task accuracy

Figure 4.1: Training curves for both grading tasks



(a) confusion matrix for pass/fail grading task



(b) confusion matrix for fragmentary grading task

Figure 4.2: Confusion matrix for the two grading tasks, vertical axis corresponds to predicted labels, horizontal axis corresponds to ground truth labels

fusion matrices for the two tasks. Confusion matrices for the ranking task show that the model captures the relevance of the four labels: all the samples whose true labels are D are misclassified as C, the closest label to D. The model performs underwhelmingly, but that is likely due to its low sample efficiency, and sufficient samples are as yet unavailable

CHAPTER 5

CONCLUSION

This work is an attempt to build an end-to-end grading neural network for step-by-step reasoning math problems given an answer key and a student solution. The model adapts a Compare-Aggregate Architecture framework. It compares steps of the two inputs and calculates a vector that reflects the importance of each step from the answer key. Then the matching representations and importance vectors are used to get the final relevance representation. The performance of the model underwhelms but experiments show that the model has a low sample efficiency. Attempts on a larger data set can give a better evaluation of the model. Future works will include the following

1. **Better sentence representation.** While transformer is a state-of-the-art language modeling approach, steps for a solution to a mathematical step-by-step question require a very special language that is highly organized. We implemented a learned representation instead of using a human-coded parser in order to make the model flexible. With sufficient data, it can learn to grade simple problems in any domain (not only math). A possible research problem is to propose a sentence representation learning method specifically for formal language.

2. **Sparse importance vector.** Based on the intuition for the sparse importance vector, the vector should be sparse because for both tasks only few steps are very important. Ng [21] states that this goal is same as limiting the mean activation of each hidden neuron on the hidden representation over the input distribution to a small number. We did not use this method because it only works for fixed-length representation but the size of importance vector varies. Solutions to this problem is worth researching.

REFERENCES

[1] J. C. S. Kadupitiya, S. Ranathunga, and G. Dias, "Assessment and error identification of answers to mathematical word problems," in *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*. IEEE, 2017, pp. 55–59.

[2] A. S. Lan, D. Vats, A. E. Waters, and R. G. Baraniuk, "Mathematical language processing: Automatic grading and feedback for open response mathematical questions," in *Proceedings of the Second (2015) ACM Conference on Learning at Scale*. ACM, 2015, pp. 167–176.

[3] M. Seo, H. Hajishirzi, A. Farhadi, O. Etzioni, and C. Malcolm, "Solving geometry problems: Combining text and diagram interpretation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1466–1476.

[4] M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni, "Diagram understanding in geometry questions," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[5] T. M. Lai, T. Bui, and S. Li, "A review on deep learning techniques applied to answer selection," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 2132–2144.

[6] M. Heilman and N. A. Smith, "Tree edit models for recognizing textual entailments, paraphrases, and answers to questions," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1011–1019.

[7] M. Wang and C. D. Manning, "Probabilistic tree-edit models with structured latent variables for textual entailment and question answering," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1164–1172.

[8] M. Tan, C. dos Santos, B. Xiang, and B. Zhou, "LSTM-based deep learning models for non-factoid answer selection," *arXiv preprint arXiv:1511.04108*, 2015.

[9] Z. Wang, W. Hamza, and R. Florian, "Bilateral multi-perspective matching for natural language sentences," *arXiv preprint arXiv:1702.03814*, 2017.

[10] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, 1994, pp. 737–744.

[11] C. dos Santos, M. Tan, B. Xiang, and B. Zhou, "Attentive pooling networks," *arXiv preprint arXiv:1602.03609*, 2016.

[12] Y. Gong, H. Luo, and J. Zhang, "Natural language inference over interaction space," *arXiv preprint arXiv:1709.04348*, 2017.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] M. Badger, "Problem-solving in undergraduate mathematics and computer aided assessment," Ph.D. dissertation, University of Birmingham, 2013.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[17] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[19] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar et al., "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[20] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[21] A. Ng, "Sparse autoencoder," in *CS294 lecture notes*. Chonbuk National University, 2011.