Exam Taking with a Mixture of Experts Model

Gan Du
gdu32@gatech.edu

Muaz Maqbool
mmaqbool3@gatech.edu

Kurt Niemi
kniemi3@gatech.edu

Demetrius Rowland
drowland9@gatech.edu

Abstract

General purpose language models have achieved tremendous success in conversation, exam-taking, answering inquiries, etc. However, they face a number of challenges, including the inability to correct for certain failures on a given task without retraining the full model and the inability to identify where knowledge is located in the model. In this paper, we propose a mixture-of-experts (MOE) model, which aims to tackle some of these challenges. In particular, our MOE is modular, in that each expert, if deemed problematic, can be replaced by a more competent one. It is also more interpretable– the knowledge possessed by the network can be ascribed to the knowledge of the appropriate expert. We test performance of our MOE model on three tasks– word scrambling, reading comprehension, and polynomial minimization, and observe an 80% success rate on an exam of 50 multiple choice problems, compared to a 60% success rate for GPT-3.5. In addition, we highlight key drivers for the MOE's success and identify promising future directions.

1. Introduction/Background/Motivation

In this paper, we aim to build a mixture-of-experts model that outperforms a benchmark language model on a constructed exam. Specifically, we require that each expert use substantially fewer parameters than the benchmark model and that the expert is specialized to the domain it is required to answer. The exam that we have constructed consists of fifty multiple choice questions with topics "word scrambling", in which the goal is to unscramble a word correctly (e.g. "stak" should be correctly unscrambled to the word "task"), "reading comprehension", in which the goal is to answer questions about a body of text to test the exam-taker's understanding, and "polynomial minimization", in which the goal is to return the input x_{min} which minimizes a polynomial of the form $ax^2 + bx + c$. All of these tasks were extracted from a previous paper [4], with the exception of polynomial minimization, which we have added here. In addition to the expert models we have constructed or finetuned, we have introduced a router model that maps the text of the input problem to a corresponding label, i.e. one of "word scrambling", "reading comprehension", and "polynomial minimization", in order for the problem to be solved by the appropriate expert. We have configured the router to be learnable as well, and details of the training are included in the methodology section.

Comparison with Current Practice

Current approaches to solving exam problems with language models involve pretraining the language model on large bodies of text, typically scraped from the internet as in the CommonCrawl data set used to train GPT-3, and asking the model to take the exam. Examples of this approach include "Language models are few shot learners" [4] and "Emergent analogical reasoning in large language models" [5].

This approach has several drawbacks, many of which stem from its lack of modularity. The first is the inability to isolate and correct for failure modes. If, for example, the modelers notice that a pretrained generalist model like GPT-3 fails to do well on arithmetic problems, one of the only currently available solutions is to fine-tune on arithmetic datasets, which may be fairly expensive given GPT-3's size. A second drawback is the inability to isolate information which ought to be kept private. For example, if a generalist model were trained on a person's private information like home address, etc., there is no guarantee that this could not be used inadvertently or maliciously for another downstream task. A final drawback is its lack of interpretability, particularly due to the inability to locate the model's knowledge in the network or interpret the network's activation values.

Proposed Advantages

With our mixture-of-experts model, we hope to correct for some of the deficiencies mentioned above. First, for correction of failure modes, it is straightforward and much less expensive to swap out or finetune a smaller component of the model responsible for a given class of problems. For example, if we notice our ensemble model is doing poorly on arithmetic problems, we simply freeze all other parameters in the model and adjust only those parameters corresponding to the "arithmetic" expert. Second, for isolation of private information, an "expert" could be constructed, designated as the only model allowed to answer questions about a person's private information, and used only when necessary. This would prevent leakage of information about, say, a person's address to other tasks. Last, for interpretability, constructing an ensemble of specialized models helps to alleviate this issue, since the relevant expert will be the one responsible for information about a given problem, e.g. the word scrambling model will have information about word-scrambling problems, and smaller models are, in general, easier to interpret.

Data Description

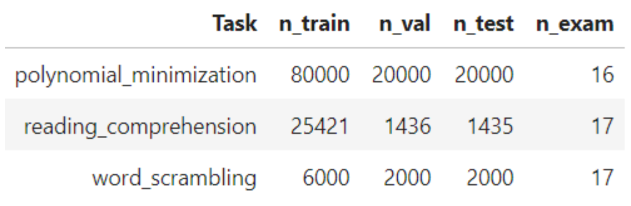
We have used three distinct datasets to train our three expert models.

For **polynomial minimization**, we construct a synthetic input dataset X consisting of the coefficients a, b, c of a polynomial $ax^2 + bx + c$ and an output dataset Y consisting of the argminima of these polynomials, i.e. -b/(2*a). These coefficients are sampled i.i.d. from a uniform distribution on [-5, 5], and the absolute value is taken of "a" to ensure a global minimum.

For **reading comprehension**, RACE-M, a subset of ReAding Comprehension dataset from Examinations (RACE) dataset, was downloaded from https://www.cs.cmu.edu/ glai1/data/race/. The RACE-M dataset consists of 7,139 passages and 28,293 questions from Chinese middle school English exams. 3-6 questions are designed for each passage to evaluate reading skills of understanding and reasoning. Each question is associated with 4 candidate answers and one of which is correct.

For **word scrambling**, the following dataset(s) on HuggingFace (https://hugginface.co) were created: kurtn718/scrambled_words_multiple_choice and kurtn718/scrambled_words . The first dataset is multiple choice questions, and the second one is for predicting the unscrambled word. A scrambled word is defined identically to Section 3.9.2 - A1 task in the "Language Models Are Few-Shot Learners" paper.

The following table indicates our train/validation/testing split, together with the number of exam questions, for each task's datasets.



Task	n_train	n_val	n_test	n_exam
polynomial_minimization	80000	20000	20000	16
reading_comprehension	25421	1436	1435	17
word_scrambling	6000	2000	2000	17

DSPY

For comparison with the LLM finetuning, DSPy (https://github.com/stanfordnlp/dspy) is used. It automatically created high quality prompts using the declared input and output descripton. In the compile mode, it bootstraps the high-quality examples (demonstations) from the training samples which is used to improve the runtime output. Major advantage of using DSPy is that it improves the results on custom data without the need to finetune the LLM which saves both the time and compute cost. In this paper, we compare the results of DSPy with the finetuned LLM using all the three datasets.

2. Approach

We now provide descriptions of the methodologies for our three experts, along with a description of the benchmark DSPY model.

Polynomial Minimization

For the polynomial minimization task, we employ an RNN, whose architecture has been modified to better suit known algorithms for solving the task. In particular, we know that gradient descent, given sufficiently many iterations and sufficiently small learning rate, will converge to the correct solution on a quadratic polynomial since they are convex. Therefore, we will aim to modify the RNN architecture so that each timestep of the RNN's forward pass mimics one iteration of gradient descent.

This inspires the following architecture for our RNN, the full derivation of which can be found in the appendix here,

$$h_{t+1} = \sigma_h(W_h h_t + x_t^T W_{ih} h_t + W_i x_t + b_h)$$
$$o_{t+1} = \sigma_o(h_{t+1} + b_o)$$

with a hidden dimension of 1, an input dimension of 3, an output dimension of 1, σ_h, σ_o fixed as the identity activations, W_h, W_{ih}, W_i being learnable weight matrices of dimensions 1x1, 3x1, and 1x3 respectively, b_h, b_o as learnable biases of dimensions 1x1 and 1x1, respectively.

The key insight is that we have added a cross term to the RNN equations between the input and hidden state, in order to match the idealized implementation of gradient descent using an RNN. We then find good weights as usual by gradient descent on training examples, with X being the polynomial coefficients $[a, b, c]$ and Y being the correct argminima of the corresponding polynomials.

Word Scrambling

In creating the expert model for answering multiple choice questions of scrambled words, the Mistral 7B model was fine-tuned using 8 bit quantization using the Low-rank approximation (LORA algorithm). A model was also fine-tuned to perform unscrambling without multiple choice answers.

Given that few shot prompting from the ”Language Models are Few-Shot Learners” was able to unscramble words via prompting techniques, that demonstrates that LLM’s are capable of unscrambling words. Because of this it was expected that fine-tuning would further improve overall accuracy, as well as give us the output that we are expecting (either a multiple choice answer, or the unscrambled word).

One thing new explored in our experiments is answering multiple choice questions in word unscrambling (i.e. does including the correct unscrambled word as a multiple answer choice - make it much easier for a LLM to get the correct answer - due to spelling errors in the training data).

Reading Comprehension

In creating the expert model for answering the reading comprehension questions, a TinyBERT encoder with one additional classification layer was fine-tuned by the AdamW algorithm.

The highest accuracy on the RACE dataset was achieved by Jiang *et al.* [2] with a model constructed upon ALBERT-xxlarge model. In light of this, we choose the TinyBERT as the encoder for our model. TinyBERT is a BERT-like model with 7.5 million parameters, which is significantly less than 110 million parameters of the BERT Base model.

As described by Jiang *et al.* [2], comparing to the traditional multi-choice setup, reconstructing the multi-choice to single-choice problem helped improve the performance of the LLM on reading comprehension tasks. To implement this approach, we use the concatenation of the passage, questions, and answers as the inputs of the Encoder model overview. The ground truth is $y \in \{0, 1\}$ for each of the inputs.

The score for each answer is calculated with the sigmoid activation on the output from the classification layer. Correspondingly, the cross entropy loss function is re-formulated as in [2]:

$$loss = -\sum_i (ylog(score_i) + (1 - y)log(1 - score_i))$$

where $score_i$ is the score for the i-th answer of the question, $i \in \{1, 2, 3, 4\}$. The answer chosen by the model is selected by the answer with the highest score for the question. The accuracy is then evaluated by the division between the number of questions that are correctly answered by the model and the total number of questions.

DSPY

Using DSPy, we develop a simple Chain-of-Thought (CoT) module using declarative DSPy syntax which tells the CoT module of the expected input and output. We then compiles this CoT module using the DSPy compiler called teleprompter. The compiler bootstraps examples from the training data and generates the prompts without manual prompt engineering or fine-tuning the LLM. For each dataset, we test two versions. For the uncompiled version, we test it using just the defined modules without the bootstrapped examples. For the compiled version, we train it first to bootstrap the examples.

Router

For the router, we need a model which maps input text describing the problem to the corresponding problem type, in order for it to be answered by the appropriate expert. To this end, we have leveraged the ”all-MiniLM-L12-v2” sentence embeddings. This model maps sentences or paragraphs to vectors of dimension 348 which, ideally, capture the semantic content of the input text. It was pretrained on a task which requires selection of the target sentence, from a randomly assorted set of candidate sentences, that correctly pairs with a given input sentence. More details can be found here [3].

To accomplish our routing task, we start by embedding each question in the compiled exam to its 348-dimensional vector under our sentence-embedding model. We then define a learnable linear map W_r from the embedding space to the ”expert” space, which consists of three dimensions, and for which each dimension corresponds to the appropriate expert, i.e. one of word scrambling, reading comprehension, and polynomial minimization. For training the router, we set the input training data X to be a collection of 800 problems, each having equal probability of belonging to one of the three classes. The output training data Y is the correct label for the problem. If we let se denote the sentence embeddings, the router can be described with

$$router(x) = softmax(W_r\, se(x)) \tag{1}$$

To get a good router, we freeze the sentence embedding model and update the weights W_r as usual with gradient descent, with the training objective defined to be the cross entropy loss between the model outputs and the true labels Y.

Problems Encountered

Polynomial Minimization

For polynomial minimization, RNNs are known to suffer from the vanishing gradients problem. We tried to solve this by engineering a loss function that averages the distance between the model output and the true polynomial argminimum across timesteps– this agrees with the intuition that at each timestep, the model's output should get closer to the polynomial's global minimum. However, perhaps due to the simplicity or algorithmic nature of the problem, we observed that this did not provide a significant improvement, so we have omitted this change to the loss function.

Word Scrambling

On the non-multiple choice word scrambling problem, initially it was thought that there was in issue in the fine-tuning process as the accuracy was 1% (2 out 200 in testing dataset correct) when the accuracy in the benchmark was around 10%.

It turns out that the dataset scrambled all the letters in the word - as opposed to scrambling all letters except the first and last letter, so it was an unfair comparison as that appears to be a harder problem for LLMs.

When generating a new dataset that had the first and last letters not scrambled, the results are closer to the benchmark and when comparing the same base model against a fine-tuned one, the results match our expected results.

Reading Comprehension

The state-of-the-art result for the reading comprehension task on RACE dataset is fine-tuned on the ALBERT-xxlarge model with transfer learning from multiple QA (question answering) datasets [2]. Since our model is constructed based on a small-size BERT model, TinyBERT, and trained on a relatively small dataset without transferred knowledge, we anticipate a compromised performance compared to the state of the art.

Besides, the last binary classification layer of our model gives output for each individual answer, without distinguish the question or passage it is based on. We applied a customized collate function to batch the data points by questions instead of the individual answers.

DSPY

For language model, we used the Mistral-7B model as the LLM. For evaluating of Maths dataset, we used an error margin of 0.1 to consider the answer correct. For the comprehenion and scramble dataset, we used exact answer match to evaluate the model. Using the DSPy signature (CoT module), we generate the answers using LLM and postprocess it before evaluating it. The Mistral-7B model prepends some of the context from the question which makes the automated evaluation difficult. Hence, we postprocess it to extract the final answer from the response. Initial prompt for the math dataset is: Given the A,B,C coefficients, calculate the final value Y using the formula Y= -B/(2*A) of it. Initial prompt for the scramble dataset: Answer multiple choice questions from the given options A,B,C,D Initial prompt for the comprehension dataset: Given the context, answer the question by choosing from given options.

Router

For the router, we initially tried to use a general-purpose language model like llama-7b, for which we would then replace the final layer with a linear one that maps to the appropriate expert. However, training this model proved to be too slow and expensive, and we suspected that additional internal updates to the model's weights would be needed for it to work correctly. As a result, we decided to use sentence-embeddings instead, and we found the training to be less complicated and more successful.

3. Experiments and Results

Polynomial Minimization:

For training the polynomial minimization RNN, we use 80,000 examples of the inputs X and outputs Y, synthesized as described in the "Data Description" section. The training process involved copying the input training data (X) T times, where T denotes an adjustable hyperparameter representing the sequence length, splitting into batches, running a forward pass through the RNN, evaluating the MSE loss between the true argminima Y and the model outputs $\hat{Y}$, and updating the weights with a backward pass.

Each hyperparameter is tuned by evaluating the model's performance on a validation set of 20,000 examples. For each hyperparameter choice ϕ taken from the cross product of the sets Learning Rates = {.1, .05, .01}, Batch Sizes = {100}, Num Epochs = {10}, Sequence Lengths = {10, 20}, Betas = {(0.9, 0.999), (0.99, 0.99)} (for the Adam optimizer), we have identified the best hyperparameters, which minimize loss on the validation set, as learning rate = .01, batch size = 100, number of epochs = 10, sequence length = 20, betas = (0.9, 0.999). The learning curves for the best hyperparameters are included in the appendix here, and results for all hyperparameters are included in the supplementary material.

As indicated by the learning curves, most of the model's learning seems to happen in the first epoch. We notice some overfitting, as the training loss is below the validation loss, and the model's loss does not seem to decrease substantially over the epochs. However, this is not deemed to be concerning, as the validation loss is within an acceptable amount, and does not increase with epoch count. Overall, the trained model is determined to be successful, as we will demonstrate shortly by looking at its performance on a test set of 16 exam problems. Furthermore, since we have a ground truth, we can check whether the RNN parameters match those suggested by the implementation of the gradient descent algorithm using an RNN.

The model parameters found by training are

$$W_h = \begin{bmatrix}1.0588\end{bmatrix} \quad W_i = \begin{bmatrix}0.0102\\-0.1821\\-0.0016\end{bmatrix}^T \quad W_{ih} = \begin{bmatrix}-0.4163\\0.0057\\0.0021\end{bmatrix}$$

Note that these parameters agree with equations (2) and (3) describing the RNN implementation of gradient descent. Here, all components are near zero, except W_h which is close to 1 as in the equation (2), the first component of W_{ih}, which is near -2 * lr for a learning rate of .2, and the second component of W_i, which is near -lr (with the same learning rate of .2). These match the ideal implementation almost exactly, so we can be confident that our RNN is implementing an approximate version of gradient descent, and we use it solve the exam problems later on.

Word Scrambling

For the multiple choice word and non-multiple choice word scrambling problem, success was measured by accuracy. On a testing set 200 questions were randomly sampled the results are shown below.

The base model is the Mistal 7b model. Results are not given for the multiple choice context / base model as it is unable to consistently output a multiple choice answer. When comparing the fine tuned model to the benchmark in the "Language Models are Few-Shot Learners" for the non multiple choice we see an improvement of accuracy (17.5% for fine tuned vs around 11%).

Problem	Model	Metric	Percentage
MC	Base	NA	NA
MC	Finetuned	199 / 200	99.5%
Non MC	Base	6 / 200	3.0%
Non MC	Finetuned	35 / 200	17.5%

Table 1. Multiple Choice Word Scrambling

Overall the experiments for seeing if a small (compared to a ChatGPT 3.5/4) model can be improved with finetuning is a success and failure. It is successful in the sense that accuracy is improved - but such a low accuracy from a model is not really that useful. The authors were hoping for about the same accuracy in the 175B model (around 65%)

Reading Comprehension

The reading comprehension task uses accuracy as the metric. The parameters are tuned by evaluating the metric of the model on a validation set of 1,436 questions. Each hyperparameter is chosen from the the sets: Learning Rates = {1e-5, 3e-5, 5e-5}, Batch Sizes (of questions) = {4, 8, 16}, Num Epochs = {15, 20}. The best hyperparameters were identified by the maximized accuracy on the validation set, of 37%, as batch size = 8, learning rate = 5e-5 for 10 epochs following with 1e-5 for another 10 epochs.

The learning curve of two set of hyperparameters are included in the appendix here. The learning curve of the best model is not appended but shows a similar trend as the appended curves. Evidence of overfitting is identified, as the training accuracy passes the validation accuracy and keeps increasing along with the epochs. Potential cause could be the limited size of the training set. The validation accuracy generally increases but not convergent by the end of the epoch, indidating that the model can be potentially further finetuned.

The benchmark in the "Language Models are Few-Shot Learners" for RACE-m data shows a 42% accuracy. Considering the limited size of the training dataset and the finetuned model, the performance is acceptable yet improvable.

DSPY

As we can see in the table, using DSPy on the comprehension datasetimproves the results by 30 percent without any finetuning or prompt hacking. However, it sturggles to answer correctly on the anaylical datasets i.e. scramble and maths dataset. For the scramble dataset, we argue that choosing the correction option is difficult. For future directions, we can ask the model to rearrange the letters instead of selection, which will enforce the model to

Dataset	Compile	Metric	Percentage
Comprehension	False	452 / 1436	31.5%
Comprehension	True	959 / 1436	66.8%
Scramble	False	799 / 2000	40.0%
Scramble	True	508 / 2000	25.4%
Maths Test	False	3 / 16	18.8%
Maths Test	True	3 / 16	18.8%

Table 2. Dataset Metrics

reason correctly. For the maths dataset, we argue that the maths dataset needs domains specific training, therefore it underperforms. If we summarize this experiment, we can see that the DSPy system improved the knowledge retrieval rather then improving the analytical skills of the LLMs. Hence we know both the strength and weakness of the DSPy system.

Router

For training the router, we have compiled a list of 800 multiple-choice (four options) exam problems, where each problem is sampled randomly from the training data for the word scrambling, reading comprehension, and polynomial minimization tasks and has an equal probability of belonging to each. Each question follows the format "Answer the following question with A, B, C, or D. {question} {choices} {body}".

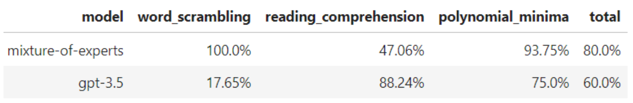
Each question is then mapped to its embedding using the "all-MiniLM-L12-v2" sentence embedding model. The training process involves splitting the training data (the inputs are the exam problems, the outputs are the class labels corresponding to the problem type– 0, 1, 2 for word scrambling, reading comprehension, and polynomial minimization respectively) into batches, and for each batch running a forward pass on the model via equation (1), obtaining the cross entropy loss between the model outputs and the true class labels, and finally backpropagating the loss gradients to the weights and updating them with the Adam optimizer.

For hyperparameter tuning, we have chosen learning rate and betas (for the Adam optimizer) from the sets $\{0.1, 0.05, 0.01\}, \{(0.9, 0.999), (0.99, 0.99)\}$ respectively. The criterion for choosing the hyperparameters was the model performance on a validation set of 200 exam problems. The best hyperparameters were found to be learning rate = .1, betas = (.99, .99). The learning curves using the best hyperparameters are recorded in the appendix here, and results for all hyperparameters are included in the supplementary material.

Overall, the router training is quite successful, and the router achieves 100% accuracy on the holdout set of 50 exam problems. We therefore encounter no issue in deploying this in our mixture-of-experts model when taking the exam.

Exam Results

Finally, we combine the models for word scrambling, reading comprehension, polynomial minimization, and the router in order to construct the mixture-of-experts model. We construct an exam of fifty problems, with 17 for word scrambling, 17 for reading comprehension, and 16 for polynomial minimization. We then map each problem to its appropriate expert using the router and have the expert return the answer choice. Additionally, we compare against a benchmark model, namely GPT-3.5, where we have obtained answers to each problem by querying the OpenAI API and receiving a response. All exam questions, together with answers from both the model and the benchmark, are included in the supplementary material. The below table compares the accuracy of our mixture-of-experts model and the benchmark model on this exam:



model	word_scrambling	reading_comprehension	polynomial_minima	total
mixture-of-experts	100.0%	47.06%	93.75%	80.0%
gpt-3.5	17.65%	88.24%	75.0%	60.0%

As indicated by the table, the mixture-of-experts model is quite successful, beating GPT-3.5 by a resounding 20 percentage points. Our word scrambling model does particularly well, with 100% accuracy, compared to the benchmark accuracy of 17%. This makes sense, as there is no reason to believe that GPT-3.5, which performs tokenization at a word-level, is capable of the character-level manipulation required to unscramble a word.

Future Directions

In this paper, we have investigated strategies for solving a set of problems with an ensemble of experts. A direction we have not explored is to build subrouters, i.e. models which can route a given input problem to an even more specialized expert. An example of this would be for a high-level router to map a mathematics problem to the "mathematics" class and for a low-level router to map this problem to one of several math categories, e.g. algebra, calculus, etc. This would modularize our framework even further, improving the benefits of failure isolation and interpretability. Finally, some of our work involved a high degree of hand-crafting, particularly for the polynomial minimization task. In the future, we would like to see work that automates the construction of experts, perhaps with an approach not unlike that described by Duvenaud [1].

4. Appendix

RNN for polynomial minimization – Derivation

Note that the action of RNN's can be described by the following equations, for hidden state h_t, input x_t, and output o_t, and activation functions σ_h, σ_o:

$$\begin{aligned} h_{t+1} &= \sigma_h(W_i x_t + W_h h_t + b_h) \\ o_{t+1} &= \sigma_o(W_o h_{t+1} + b_o) \end{aligned}$$

Note also that gradient descent, with learning rate lr, for a quadratic polynomial in a variable h, $ah^2 + bh + c$ can be described by

$$\begin{aligned} h_{t+1} &= h_t - lr * \frac{d}{dh}(ah^2 + b * h + c)|_{h=h_t} \\ &= h_t - lr * (2ah_t + b) \end{aligned}$$

Observe that this has remarkable similarities with the hidden state update from the RNN description, so we will continue the gradient descent derivation as follows

$$\begin{aligned} h_{t+1} &= h_t - lr * (2ah_t + b) \\ &= \begin{bmatrix} 1 \end{bmatrix} h_t + \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} -2 * lr \\ 0 \\ 0 \end{bmatrix} h_t + \begin{bmatrix} 0 & -lr & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\ &= \sigma_h(W_h h_t + x_t^T W_{ih} h_t + W_i x_t) \\ o_{t+1} &= \sigma_o(h_{t+1}) \end{aligned}$$

with

$$W_h = \begin{bmatrix} 1 \end{bmatrix}, W_{ih} = \begin{bmatrix} -2 * lr \\ 0 \\ 0 \end{bmatrix}, W_i = \begin{bmatrix} 0 & -lr & 0 \end{bmatrix} \quad (2)$$

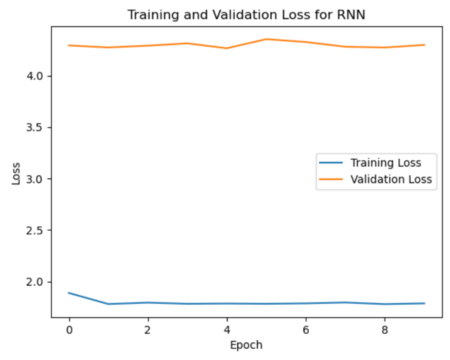
$$x_t = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \sigma_h(z) = z, \quad \sigma_o(z) = z \quad (3)$$

This is nearly identical to the equations describing the actions of a vanilla RNN. However, there is an additional cross term between x_t, h_t. We will add this to our model, suggesting the following architecture for our RNN

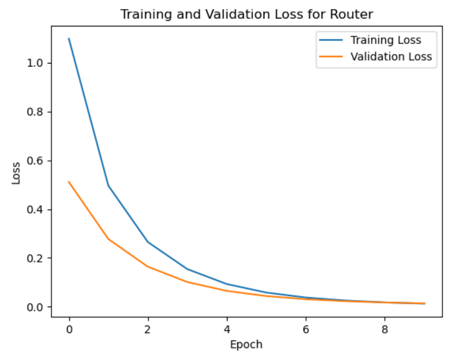
$$\begin{aligned} h_{t+1} &= \sigma_h(W_h h_t + x_t^T W_{ih} h_t + W_i x_t + b_h) \\ o_{t+1} &= \sigma_o(h_{t+1} + b_o) \end{aligned}$$

with a hidden dimension of 1, an input dimension of 3, an output dimension of 1, σ_h, σ_o fixed as the identity activations, W_h, W_{ih}, W_i being learnable weight matrices of dimensions 1x1, 3x1, and 1x3 respectively, b_h, b_o as learnable biases of dimensions 1x1 and 1x1, respectively.

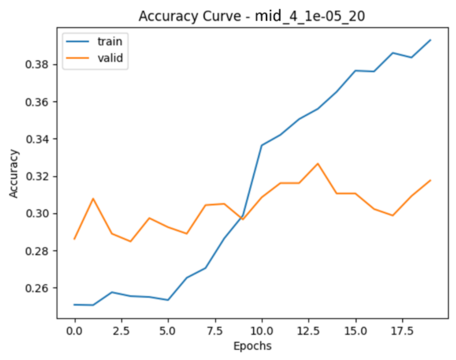
Polynomial Minimization– Training Results



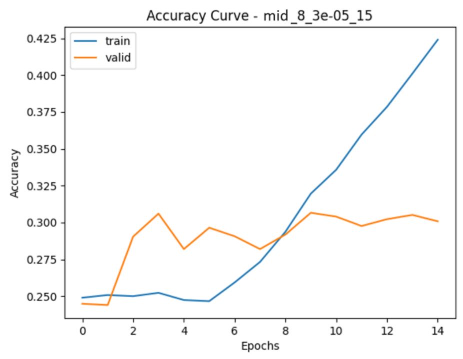
Router– Training Results

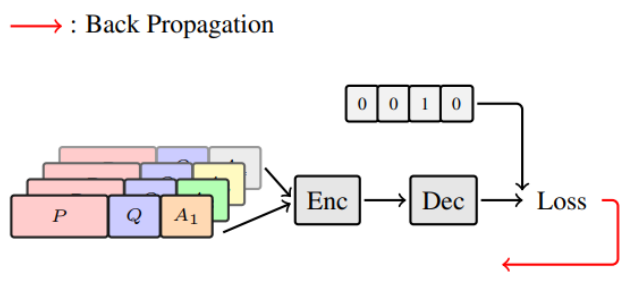


Reading Comprehension– Training Results 1

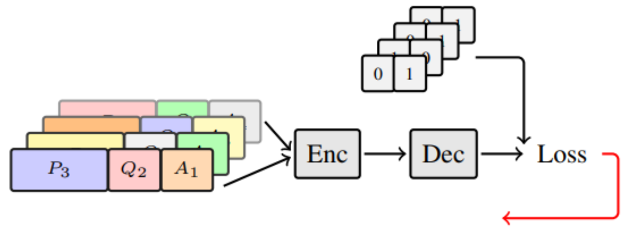


Reading Comprehension– Training Results 2





(a) Multi-choice Model



(b) Single-choice Model

An overview of Multi-choice Model and Single-choice Model

References

[1] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, 11 2014. 6

[2] Yufan Jiang, Shuangzhi Wu, Jing Gong, Yahui Cheng, Peng Meng, Weiliang Lin, Zhibo Chen, and Mu li. Improving machine reading comprehension with single-choice decision and transfer learning, 2020. 3, 4

[3] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. 3

[4] Nick Ryder Melanie Subbiah Jared Kaplan Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray Benjamin Chess Jack Clark Christopher Berner Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei Tom B. Brown, Benjamin Mann. Language models are few-shot learners. *https://arxiv.org/abs/2005.14165*, 2020. 1

[5] Holyoak K.J. Lu H. Webb, T. Emergent analogical reasoning in large language models. *Nat Hum Behav 7, 1526–1541*, 2023. 1

5. Work Division

Student Name	Contributed Aspects	Details
Gan Du	Model training / Evaluation / Paper writing	Performed everything related to Reading Comprehension: Processing Data / Training Models / Evaluating Results / Writing Paper
Muaz Maqbool	Model training / Evaluation / Paper writing	Evaluated DSPy on all three datasets. Compared results with the finetuned LLM, uncompiled DSPy CoT, compiled DSPy CoT
Kurt Niemi	Model training / Evaluation / Paper writing	Performed all aspects of everything related to Word scrambling (Dataset generation / Training Models / Evaluating results / Writing paper)
Demetrius Rowland	Model training / Evaluation / Paper writing	Created and trained the RNN architecture for solving the polynomial minimization; Created and trained the router for the appropriate expert selection; Wrote code that queries the benchmark GPT-3.5 model for the exam answers