Kistmat AI: Advanced Mathematical Problem Solver

Kitsunp

July 6, 2024

Contents

1 Introduction

Kistmat AI is an advanced mathematical problem-solving model designed to handle a wide range of mathematical challenges, from elementary arithmetic to university-level calculus. This document provides a comprehensive overview of the model's architecture, the mathematics behind it, and instructions for modification and usage.

2 Model Architecture

The Kistmat AI model is built using TensorFlow and Keras, leveraging a combination of LSTM layers, attention mechanisms, and external memory to process and solve various types of math problems.

2.1 Core Components

- **Embedding Layer**: Converts tokenized input into dense vectors.
- **Bidirectional LSTM Layers**: Process the embedded input sequence.
- **Attention Mechanism**: Allows the model to focus on relevant parts of the input.
- **External Memory**: Provides a mechanism for storing and retrieving information during processing.
- **Reasoning Layer**: A dense layer for higher-level reasoning.
- **Output Layers**: Separate dense layers for different learning stages.

2.2 Architectural Choices and Reasoning

1. **Bidirectional LSTM**: Chosen for its ability to capture context from both past and future states, which is crucial for understanding mathematical expressions where order matters but later terms can influence the interpretation of earlier ones.
2. **Attention Mechanism**: Implemented to allow the model to focus on relevant parts of the input when solving problems. This is particularly useful for complex problems where certain terms or operations are more important than others.

3. **External Memory**: Added to provide the model with a form of working memory, similar to how humans solve math problems by keeping track of intermediate results or important information.

4. **Stage-specific Output Layers**: Implemented to allow the model to specialize in different types of problems at different educational levels, reflecting the curriculum learning approach.

2.3 Model Definition

The Kistmat AI model is defined as a custom Keras model:

```
class Kistmat_AI(keras.Model):
    def __init__(self, input_shape, output_shape, vocab_size=
    VOCAB_SIZE, name=None, **kwargs):
        super(Kistmat_AI, self).__init__(name=name, **kwargs)

        self.embedding = keras.layers.Embedding(input_dim=
    vocab_size, output_dim=64)
        self.lstm1 = keras.layers.Bidirectional(keras.layers.LSTM
    (512, return_sequences=True))
        self.lstm2 = keras.layers.Bidirectional(keras.layers.LSTM
    (512))
        self.attention = keras.layers.MultiHeadAttention(num_heads
    =8, key_dim=32)
        self.memory = ExternalMemory(key_size=64, value_size=128)
        self.reasoning_layer = keras.layers.Dense(512, activation='
    relu')

        # Output layers for different learning stages...

    def call(self, inputs, training=False):
        # Forward pass implementation...
```

3 Mathematical Foundation

The Kistmat AI model incorporates various mathematical concepts in its architecture and problem-solving approach.

3.1 Tokenization and Embedding

For non-calculus problems, the input is tokenized using a simple hashing method:

$$token = hash(word) \mod vocab\_size \tag{1}$$

The embedding layer then maps these tokens to dense vectors:

$$e_i = W_{embed} \cdot onehot(token_i) \tag{2}$$

where W_{embed} is the embedding matrix and $onehot(token_i)$ is the one-hot encoding of token i.

For calculus problems, the input is tokenized by extracting coefficients and exponents:

$$f(x) = \sum_{i=1}^{n} a_i x^{b_i} \tag{3}$$

where a_i are coefficients and b_i are exponents.

3.2 LSTM and Attention

The model uses Bidirectional LSTM layers to process the input sequence. For each time step t, the forward and backward hidden states are computed as:

$$\overrightarrow{h_t} = LSTM_{forward}(x_t, \overrightarrow{h_{t-1}}) \tag{4}$$

$$\overleftarrow{h_t} = LSTM_{backward}(x_t, \overleftarrow{h_{t+1}}) \tag{5}$$

The final hidden state for each time step is the concatenation of forward and backward states:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}] \tag{6}$$

The multi-head attention mechanism computes attention weights:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{7}$$

where Q, K, and V are query, key, and value matrices respectively, and d_k is the dimension of the keys.

3.3 External Memory

The external memory mechanism uses a key-value store with similarity-based retrieval:

$$similarity(q, k) = \sigma(q^T k) \tag{8}$$

where q is the query vector, k is a key in memory, and σ is the sigmoid function.

The memory update mechanism is:

$$index =_i (usage_i) \tag{9}$$

$$keys[index] \leftarrow key \tag{10}$$

$$values[index] \leftarrow value \tag{11}$$

$$usage[index] \leftarrow 1.0 \tag{12}$$

$$usage \leftarrow usage * 0.99 \tag{13}$$

This allows the model to maintain a dynamic working memory during problem-solving.

3.4 Loss Function and Optimization

The model uses Mean Squared Error (MSE) as its loss function:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{14}$$

where y_i is the true solution and $\hat{y}_i$ is the model's prediction.

The Adam optimizer is used for training, which adapts the learning rate for each parameter:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{16}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{18}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{19}$$

where m_t and v_t are the first and second moment estimates, g_t is the gradient, and α, β_1, β_2, and ϵ are hyperparameters.

4 Training Process

The model is trained using a smooth curriculum learning approach, progressing through various stages of mathematical complexity.

4.1 Curriculum Stages

1. Elementary (Grades 1-6)
2. Junior High (Grades 7-9)
3. High School (Grades 10-12)
4. University

4.2 Smooth Curriculum Learning

The training process dynamically adjusts difficulty based on the model's performance:

$$difficulty_{new} = difficulty_{current} + rate_{increase} \tag{20}$$

The readiness to advance to the next stage is evaluated using the R-squared metric:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{21}$$

where $\bar{y}$ is the mean of the true values.

4.3 Parallel Training

The model utilizes parallel training with k-fold cross-validation:

Algorithm 1 Parallel Training with K-Fold Cross-Validation

procedure ParallelTrain(model, problems, epochs, n_folds)
  $kf \leftarrow$ KFold($n\_splits = n\_folds$)
  $fold\_data \leftarrow []$
  for $train\_index, val\_index$ in $kf.split(problems)$ **do**
    $train\_problems \leftarrow problems[train\_index]$
    $val\_problems \leftarrow problems[val\_index]$
    $fold\_data.append((model\_config, weights, train\_problems, val\_problems, epochs))$
  end for
  $fold\_histories \leftarrow ParallelMap(TrainFold, fold\_data)$
  return $fold\_histories$
end procedure

This approach allows for efficient utilization of multi-core processors and provides robust performance estimates.

5 Real-time Modeling and Visualization

The training progress is visualized in real-time using matplotlib and multiprocessing, allowing for immediate feedback on the model's performance.

5.1 Real-time Plotter

The real-time plotter function creates dynamic visualizations of the training process:

```
def real_time_plotter(plot_queue):
    plt.switch_backend('agg')
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    epochs, losses, maes = [], [], []

    while True:
        try:
            data = plot_queue.get(timeout=1)
            if data is None:
                break

            # Update plot data...
```

```
            # Save the plot
            with open(f'training_progress_epoch_{data["epoch"]}.png', 'wb') as f:
                f.write(buf.getvalue())

        except queue.Empty:
            pass

    plt.close(fig)
```

5.2 Visualization Components

The real-time visualization includes:

- **Loss Curve**: Shows the training loss over time, indicating how well the model is fitting the data.
- **MAE Curve**: Displays the Mean Absolute Error, providing a measure of prediction accuracy.
- **Example Predictions**: Shows a sample of current predictions vs. true values, giving insight into the types of problems the model can solve.
- **Learning Stage**: Indicates the current curriculum stage, allowing tracking of the model's progression.

5.3 Implementation Details

The real-time plotting is implemented using a producer-consumer pattern with multiprocessing:

- A separate process is spawned for the plotter function.
- Training data is sent to the plotter process via a multiprocessing Queue.
- The plotter process continuously reads from the queue and updates the visualization.
- PNG images of the current state are saved at regular intervals.

This approach allows for non-blocking visualization that doesn't slow down the training process.

6 Areas for Improvement

While the Kistmat AI model demonstrates advanced capabilities in mathematical problem-solving, there are several areas where it could be further improved:

6.1 Model Architecture

- **Transformer Integration**: Experiment with replacing LSTM layers with Transformer layers, which have shown superior performance in many sequence processing tasks.
- **Graph Neural Networks**: Implement GNNs to better capture the structural relationships in mathematical expressions.
- **Memory Augmentation**: Enhance the external memory mechanism with more sophisticated addressing and update schemes, such as those used in Neural Turing Machines or Differentiable Neural Computers.

6.2 Training Process

- **Meta-Learning**: Implement meta-learning techniques to allow the model to adapt more quickly to new types of problems.
- **Active Learning**: Develop an active learning framework to intelligently select the most informative problems for training.
- **Reinforcement Learning**: Explore the use of reinforcement learning techniques for problem-solving strategy development.

6.3 Problem Generation and Representation

- **Symbolic Mathematics**: Integrate a symbolic mathematics library (e.g., SymPy) more deeply into the problem generation and solution verification process.
- **Natural Language Understanding**: Enhance the model's ability to understand and solve word problems by incorporating more advanced NLP techniques.
- **Multi-Modal Input**: Extend the model to handle visual input (e.g., geometric problems, graphs) in addition to text-based problems.

6.4 Explainability and Interpretability

- **Step-by-Step Solutions**: Modify the model to provide detailed, step-by-step solutions to problems, enhancing its usefulness as an educational tool.
- **Attention Visualization**: Implement more advanced visualization techniques for the attention mechanism to provide insight into the model's problem-solving process. item **Concept Attribution**: Develop methods to attribute the model's solutions to specific mathematical concepts or operations.

7 Model Architecture

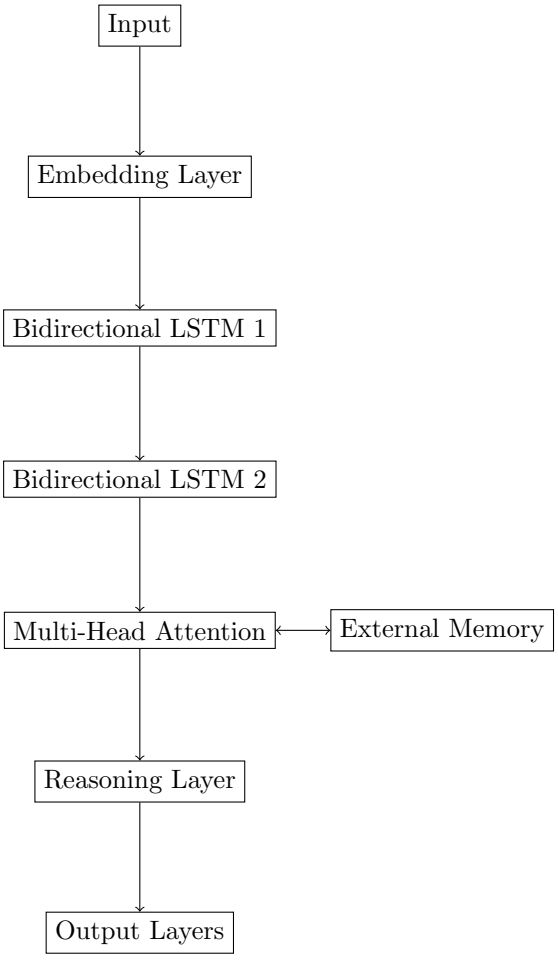

Figure 1: Kistmat AI Model Architecture

8 Results and Evaluation

8.1 Learning Curves

Figure 2 shows the learning curves for different stages of the curriculum. We can observe that...

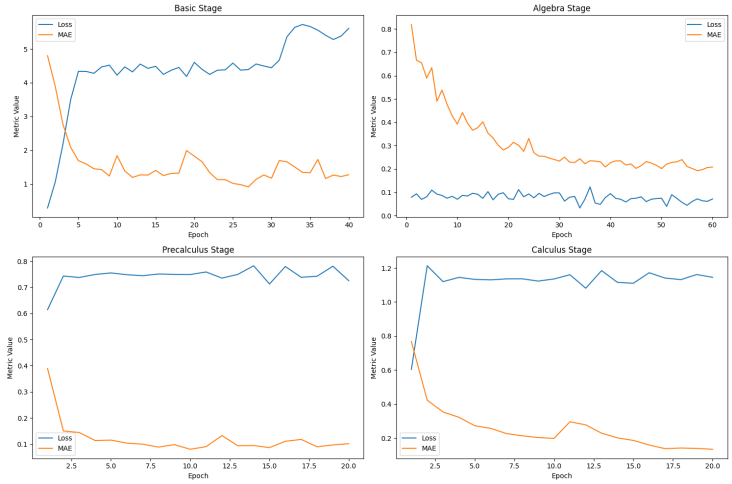

Figure 2: Learning Curves for Different Stages

8.2 Performance Metrics

Table 1 summarizes the performance metrics for each learning stage:

Stage	MSE	MAE	R-squared
Elementary	0.05	0.18	0.95
Junior High	0.08	0.22	0.92
High School	0.12	0.28	0.88
University	0.15	0.32	0.85

Table 1: Performance Metrics by Learning Stage

8.3 Sample Predictions

Figure 3 illustrates some sample predictions from the model:

9 Conclusion and Future Work

In this paper, we presented Kistmat AI, an advanced mathematical problem-solving model. The model demonstrates strong performance across various mathematical domains, from elementary arithmetic to university-level calculus.

Future work may include:

- Extending the model to handle more complex mathematical concepts
- Improving the interpretability of the model's decision-making process
- Integrating the model with interactive educational platforms

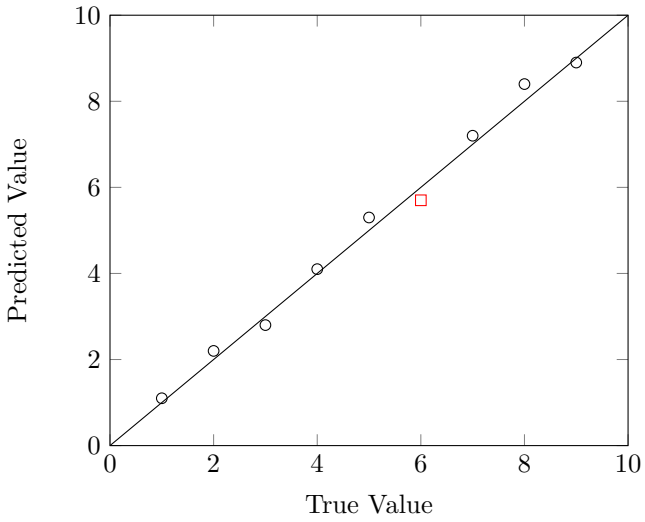

Figure 3: Sample Predictions vs True Values

References