Neural Symbolic LSTM Regression from Truncated Taylor Series

Tommaso Vaccari, Tommaso Felice Banfi
Politecnico di Milano
Dept. of Electronics, Information and Bioengineering
Milan, Italy
{tommaso.vaccari, tommasofelice.banfi}@mail.polimi.it

Abstract

We address the task of recovering closed-form expressions from truncated Taylor series. Using a small SymPy-generated grammar—polynomials together with $\sin$, $\cos$, and $\exp$ combined by $+, -, *, /$—we construct paired data of expressions and their order-4–6 expansions. A two-layer LSTM sequence-to-sequence model is trained to translate series tokens back to expressions. On a held-out test set of 500 pairs, greedy decoding achieves 91% exact matches and 94% token-level accuracy. We outline current limitations, including the restricted grammar, fixed expansion point, and noiseless synthetic data, and discuss possible extensions such as grammar-aware decoding and attention mechanisms.

1 Introduction

Symbolic regression aims to recover closed-form expressions from data, offering interpretability and compactness beyond purely numerical models. We consider a constrained variant: reconstructing a function from its truncated Taylor expansion.

This task exposes central difficulties for neural symbolic methods: deciding operator placement, nesting transcendental functions from limited local information, and handling non-identifiability, since distinct expressions may share the same expansion. Division further introduces algebraic instabilities near the expansion point.

As a first step, we present a minimal baseline: a sequence-to-sequence LSTM trained on synthetic data produced with `SymPy`.

2 Related Work

Neural sequence models such as seq2seq LSTMs have been widely applied to symbolic translation tasks, where their ability to capture sequential dependencies makes them a natural baseline. The FASEROH framework [1] demonstrated that synthetic datasets of symbolic objects can effectively support neural approaches to algebraic problems. Inspired by this idea, we generate paired expressions and Taylor expansions using a restricted grammar in `SymPy`.

3 Problem Formulation

Task. Given a truncated Taylor expansion around 0, predict its generating closed form. Let $\mathcal{T}_k[f](x) = \sum_{n=0}^{k} \frac{f^{(n)}(0)}{n!} x^n$ with $k \in \{4, 5, 6\}$. The goal is to learn a mapping $\mathcal{T}_k[f] \mapsto f$.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

Grammar. Expressions are composed from x, x^2, x^3, $\sin x$, $\cos x$, and $\exp x$ combined with $+, -, \times, /$ (1–3 terms). Each candidate is simplified/expanded in `SymPy`; identically zero forms and degenerate denominators are discarded, defining the function class $\mathcal{F}$.

Representation. A tokenizer τ maps expressions and series to sequences over a shared vocabulary V ($|V| = 161$), extended with `<SOS>`, `<EOS>`, `<PAD>`, and `<UNK>`. Each training pair is (x, y) with $x = \tau(\mathcal{T}_k[f])$ and $y = \tau(f)$.

Objective. A two-layer LSTM encoder–decoder models $p_\theta(y \mid x)$. Decoding is performed greedily by $\arg\max$ at each step.

Metrics. We report exact match, token accuracy, normalized Levenshtein similarity, and numerical fidelity on $[-1, 1]$ using MSE and MAE from `SymPy`/NumPy evaluation.

4 Dataset Construction

We adopt a lightweight grammar inspired by [1]. Base functions are $\{x, x^2, x^3, \sin x, \cos x, \exp x\}$ and operators $\{+, -, \times, /\}$. Random expressions combine 1–3 base terms, simplified in `SymPy`; identically zero forms and invalid divisions are resampled.

For each $f \in \mathcal{F}$, we compute its Taylor expansion at 0, truncated at $k \in \{4, 5, 6\}$ with the $O(x^{k+1})$ term removed. Functions and expansions are then tokenized: operators and parentheses are spaced, function names treated as atomic tokens, and exponentiation written as `^`. The shared vocabulary has 161 entries, including `<PAD>`, `<SOS>`, `<EOS>`, and `<UNK>`.

The final dataset comprises ∼137k *(expansion → function)* pairs. Expansions range from 5–45 tokens, functions from 5–19. Examples include $x^4/24 - x^2/2 + 1 \mapsto \cos x$ and $x^3/2 + x \mapsto x/\cos x$. All generation code is provided in the project notebook.

5 Method

5.1 Model Architecture

We employ a 2-layer LSTM encoder–decoder with embedding dimension 150 and hidden size 512. The encoder reads tokenized Taylor expansions and passes its final hidden state to the decoder, which autoregressively generates the target expression. Dropout is applied between layers to improve generalization. Training details and hyperparameters are reported in Appendix A. A complete pipeline example is given in Appendix C.

5.2 Loss Function

Training minimizes a weighted sum of Cross-Entropy (CE) and Weighted Mean Squared Error (WMSE). CE, $\mathcal{L}_{\text{CE}}$, enforces token-level accuracy. WMSE, $\mathcal{L}_{\text{WMSE}}$, compares predicted and target functions as continuous objects, emphasizing accuracy near the expansion center $x = k$:

$$\mathcal{L}_{\text{WMSE}} = \frac{1}{Z} \int_{k-\epsilon}^{k+\epsilon} w(x) \left(f_{\text{pred}}(x) - f_{\text{target}}(x)\right)^2 dx, \quad w(x) = \frac{1}{(x-k)^2 + \delta^2}. \tag{1}$$

Here $\delta > 0$ ensures stability, and Z normalizes the weighted integral. In practice, the integral is approximated on a grid of evaluation points $\mathcal{X}$, with token predictions kept differentiable via softmax.

The total loss is

$$\mathcal{L}_{\text{total}} = \alpha\, \mathcal{L}_{\text{CE}} + \beta\, \mathcal{L}_{\text{WMSE}},$$

where α, β balance syntactic and functional accuracy.

5.3 Decoding Strategy

At inference, sequences are generated autoregressively with greedy decoding:

$$\hat{y}_t = \arg\max_{y \in \mathcal{V}} p(y \mid \hat{y}_{<t}, x),$$

terminating at <EOS>.

As an extension, beam search [2, 3, 4] can be used, retaining the B best partial hypotheses at each step and improving the chance of recovering globally optimal expressions at higher computational cost.

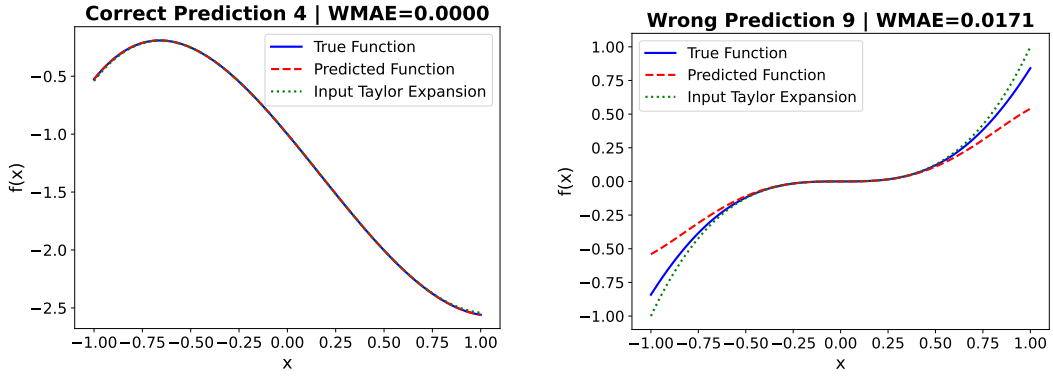
6 Evaluation Protocol and Results

We evaluate our models on the held-out 20% of the dataset (500 pairs) using three metrics. Exact Match (EM) measures the fraction of sequences that exactly reproduce the ground-truth expression, Token Accuracy (TA) quantifies the fraction of correctly predicted tokens, and Weighted Mean Squared Error (WMSE) evaluates numeric fidelity with higher weight near the Taylor expansion center, as defined in Equation 1. Predicted token sequences are converted back to symbolic expressions and evaluated over $\mathcal{X}$ to compute WMSE.

Table 1 reports test results comparing our two implementations: one trained with only cross-entropy loss ($\mathcal{L}_{\text{CE}}$) and another combining cross-entropy with WMSE ($\mathcal{L}_{\text{CE + WMSE}}$), while Figures 1a and 1b illustrate a correct and an incorrect prediction, respectively.

Table 1: Test set performance of LSTM models. EM: Exact Match, TA: Token Accuracy, WMSE: Weighted Mean Squared Error.

Model	EM (%)	TA (%)	WMSE (10^{-3})
LSTM ($\mathcal{L}_{\text{CE}}$)	88.0	88.0	1
LSTM ($\mathcal{L}_{\text{CE + WMSE}}$)	88.0	89.0	1



(a) Correct prediction.

(b) Incorrect prediction.

Figure 1: Comparison of predicted symbolic expressions for a truncated Taylor expansion. The WMSE loss emphasizes accuracy near the expansion center $x = 0$, as it is more relevant for local approximation.

7 Conclusion

We presented a proof-of-concept study for recovering compact closed forms from truncated Taylor expansions, a task with potential applications in physics, engineering, and symbolic computation.

Our approach employs a neural symbolic regression model based on a 2-layer LSTM, trained with both cross-entropy and a task-specific loss function. Experiments on synthetic datasets demonstrate strong exact-match and token-level accuracy, validating the feasibility of this framework.

For future work, we plan to investigate longer expansions, functions with centers other than $x = 0$, and the integration of attention mechanisms (see Appendix B), which we tested but found less beneficial for short sequences. Additionally, adopting beam search could further improve sequence decoding and overall prediction accuracy.

References

[1] Harrison B. Prosper and Sergei V. Gleyzer. Faseroh: Fast accurate symbolic empirical representation of histograms. *ML4Science GSoC Project Report*, 2022.

[2] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In Thang Luong, Alexandra Birch, Graham Neubig, and Andrew Finch, editors, *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, Vancouver, August 2017. Association for Computational Linguistics.

[3] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas, November 2016. Association for Computational Linguistics.

[4] Gábor Szűcs and Dorottya Huszti. Seq2seq deep learning method for summary generation by lstm with two-way encoder and beam search decoder. In *2019 IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 221–226, 2019.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[7] Xianyun Wen and Weibang Li. Time series prediction based on lstm-attention-lstm model. *IEEE Access*, PP:1–1, 01 2023.

[8] Xianyun Wen and Weibang Li. Time series prediction based on lstm-attention-lstm model. *IEEE access*, 11:48322–48331, 2023.

[9] Xuan Zhang, Xun Liang, Aakas Zhiyuli, Shusen Zhang, Rui Xu, and Bo Wu. At-lstm: An attention-based lstm model for financial time series prediction. In *IOP Conference Series: Materials Science and Engineering*, volume 569, page 052037. IOP Publishing, 2019.

[10] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 936–945. PMLR, 18–24 Jul 2021.

A Training Procedure and Hyperparameters

All models were trained on the synthetic dataset described in Section 4 using a 2-layer LSTM encoder-decoder with hidden dimension 512 and embedding size 150. The vocabulary size is 161 tokens.

We employed a combination of standard cross-entropy loss and a custom weighted loss (Section 5.2), with coefficients $\alpha = @TODO$ and $\beta = @TODO$, to balance token-level accuracy and functional fidelity. Optimization was performed with AdamW (learning rate $1\mathrm{e}{-3}$, weight decay $1\mathrm{e}{-5}$), and gradient norms were clipped at 10.0 to ensure stability.

Models were trained for 10 epochs with a batch size of 64. Teacher forcing was used during decoding, and sequences were padded to the batch maximum length. A learning rate scheduler reduced the learning rate by a factor of 0.1 if the training loss did not improve for 2 consecutive epochs.

B Attention Mechanism

To better capture long-range dependencies in token sequences, we explored integrating an attention mechanism into our LSTM decoder. Specifically, we implemented a *Bahdanau-style additive*

attention [5, 6], also commonly referred to as LSTM attention. Let $s_t \in \mathbb{R}^{d_h}$ be the decoder hidden state at timestep t and $h_i \in \mathbb{R}^{d_h}$ be the encoder hidden states for $i = 1, \ldots, L$. Attention scores and weights are computed as:

$$e_{t,i} = v_a^\top \tanh(W_a s_t + U_a h_i), \quad \alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{L} \exp(e_{t,j})}, \tag{2}$$

and the context vector c_t is obtained by a weighted sum of encoder states:

$$c_t = \sum_{i=1}^{L} \alpha_{t,i} h_i. \tag{3}$$

The decoder then generates the next token conditioned on both its current hidden state and the context:

$$p(y_t \mid y_{<t}, x) = \text{Softmax}(V[s_t; c_t] + b). \tag{4}$$

In our symbolic regression task, we integrated the Bahdanau-style attention mechanism as detailed previously. For the baseline dataset, we did not observe any notable improvements in performance. This is likely due to the generally short sequences, where local dependencies dominate and the decoder can already capture relevant information without additional context.

However, attention provides a clear advantage for longer token sequences, where the decoder needs to align distant input tokens with their corresponding outputs, as shown by several works in time series forecasting [7, 8, 9].

Looking forward, we plan to explore transformer-based architectures, which have already demonstrated strong performance on symbolic regression tasks [10].

C Pipeline Example

In the following example, we report a step-by-step illustration of the full pipeline from input truncated Taylor expansion to predicted symbolic expression. Figure 1a shows the plot comparing the true function, predicted function, and input Taylor expansion of this example.

Pipeline Example

Step 1: Input truncated Taylor expansion (tokenized)

$$f_{\text{Input}}(x) = -\frac{x^4}{24} + x^3 - \frac{x^2}{2} - 2x - 1$$

Step 2: Tokenization

$$\begin{aligned}\text{tokens} = [&\texttt{"-"}, \texttt{"x"}, \texttt{"**"}, \texttt{"4"}, \texttt{"/"}, \texttt{"24"},\\ &\texttt{"+"}, \texttt{"x"}, \texttt{"**"}, \texttt{"3"},\\ &\texttt{"-"}, \texttt{"x"}, \texttt{"**"}, \texttt{"2"}, \texttt{"/"}, \texttt{"2"},\\ &\texttt{"-"}, \texttt{"2"}, \texttt{"*"}, \texttt{"x"},\\ &\texttt{"-"}, \texttt{"1"}]\end{aligned}$$

Step 3: LSTM Encoder/Decoder

$$\text{encoded sequence} \xrightarrow{\text{LSTM}} \text{decoder states} \xrightarrow{\text{Greedy Decoding}} \text{predicted tokens}$$

Step 4: Token-level decoding

$$\begin{aligned}\text{predicted tokens} = [&\texttt{"x"}, \texttt{"**"}, \texttt{"3"}, \texttt{"-"}, \texttt{"exp"}, \texttt{"("}, \texttt{"x"}, \texttt{")"},\\ &\texttt{"-"}, \texttt{"sin"}, \texttt{"("}, \texttt{"x"}, \texttt{")"}]\end{aligned}$$

Step 5: Untokenization

$$f_{\text{pred}}(x) = x^3 - \exp(x) - \sin(x)$$