Deep Learning-Informed Cryptocurrency Trading

1. Introduction

The cryptocurrency market is known for its extreme volatility and seemingly irrational price movements. Traditional trading models and rule-based bots often fall short when sudden swings and non-linear relationships emerge. Motivated by the challenge of predicting Bitcoin's future price movements with measurable accuracy, we set out to explore whether deep learning could help make sense of this chaotic time-series data.

Our goal was to predict Bitcoin's price within a ±5% margin of error - a target ambitious enough to be meaningful but grounded enough to avoid overfitting or hindsight bias. We framed this as a **regression problem**, since the task involves predicting a continuous numerical value rather than a classification label.

1.1 Related Work

While there has been a lot of work on cryptocurrency prediction using machine learning and deep reinforcement learning, few projects have focused solely on deep learning for regression without an RL agent component. We were inspired by projects such as "Automated Cryptocurrency Trading using Deep Reinforcement Learning" (Nick Kaparinos' GitHub repository), although we ultimately pursued a different methodology based on supervised deep learning rather than reinforcement learning.

Github inspiration:
https://github.com/NickKaparinos/Automated-Cryptocurrency-trading-using-Deep-RL

Our implementation is fully original and focused on regression models.

2. Methodology

We designed a modular system allowing for easy training and evaluation across multiple deep learning architectures, including:

- Multilayer Perceptron (MLP)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM)
- Transformer

Each model accepts a sliding window of historical Bitcoin prices as input and outputs a prediction for a future price. By maintaining a consistent input-output interface, we enabled direct comparison between model architectures.

The data preprocessing pipeline included normalizing price values to stabilize training and creating fixed-size sequences for input to the models. Missing dates in the historical datasets were filled, and time windows were structured to ensure chronological consistency.

During training, we used Mean Squared Error (MSE) as the loss function and tracked Mean Absolute Error (MAE) as a primary metric for evaluation. The Adam optimizer was generally used, with learning rates tuned per model. We also tracked the accuracy with which the model directed the price direction, since derivatives of a commodity are paramount to investment decisions; investing in an asset with negative derivative will result in loss.

Hyperparameter tuning involved adjusting the number of layers, number of units per layer, learning rates, and the window sizes. Each model was evaluated based on its validation mean absolute error (MAE) to ensure fair model comparisons.

We also focused on setting up a clean experiment management system. Each training run automatically logged key metrics, model configurations, and loss curves, enabling structured comparisons. A single main.py file served as the driver script, dynamically handling different models based on command-line arguments.

2.2 Data

We utilized historical cryptocurrency datasets from the following sources:

- CoinMetrics Community Data
- Kaggle Bitcoin Dataset
- Kaggle Ethereum Dataset
- Kaggle Solana Dataset

Although we focused primarily on Bitcoin for the initial experiments, the framework is general enough to extend to other coins. Significant preprocessing was required to:

- Fill missing dates
- Normalize prices
- Create consistent time window inputs and outputs

The preprocessing required that we combine several datasets into one large testing dataset. Specifically, bitcoin data from Kaggle from 2017 to 2019. Some data points were missing, so in order to make the data ‘nice’, we had to add a couple. We simply filled in the null data points by copying the previous data point to the new timestamp. This was required for the preprocessed data to work with the models. Adding data points may seem like data manipulation, but adding a few data points within a few million will have a negligible impact.

Preprocessing also retrieved the relevant data labels and data, with the labels being open, high, low, close, Volume BTC, and Volume USD. Batches of size 360, equivalent to 6 hours, were used for our model.

2.3 Metrics

The primary metric for evaluation was **Mean Absolute Error (MAE)** relative to the actual Bitcoin price. MAE was chosen because it provides an easily interpretable average error in the same units as the target variable, helping us understand typical deviations from actual market values. We also tracked **Mean Squared Error (MSE)** during training to capture the variance and penalize larger errors more heavily.

3. Challenges

The biggest challenge was engineering a clean, flexible codebase where all models could share the same interface. Deep learning models vary widely in input expectations (especially Transformers vs CNNs), so writing compatible data pipelines and main driver scripts took significant effort.

Another major challenge we faced was the complex preprocessing of time-series data. Cryptocurrency markets operate 24/7 without traditional market holidays, leading to gaps or anomalies in some datasets. Handling missing timestamps, aligning datasets, and generating uniform input windows proved to be a surprisingly involved and meticulous task that required careful engineering.

An additional challenge we encountered was the fact that our project relied on original research rather than reimplementing another groups' project. Constructing metrics for success and determining which models outperformed others necessitated lots of discussion and sometimes resulted in a sort of paralysis between all of the possible ways to implement the project. Of course, we managed to unblock ourselves and construct a successful project but we learned that generating original ideas was difficult.

In addition, handling metric logging across different architectures was nontrivial. CNNs, RNNs, and Transformers each have unique architectural requirements, and creating a shared framework without excessive special casing was a significant hurdle. We had to ensure that, regardless of model type, the system could automatically and consistently track metrics like MAE and MSE, making fair comparisons possible.

Finally, tuning hyperparameters for fair comparisons across models also proved to be a significant difficulty. Training deep learning models on financial time series data is highly sensitive to even minor changes in hyperparameters. Finding appropriate learning rates, batch sizes, number of epochs, and sequence lengths for each model required a great deal of experimentation and careful validation to avoid drawing misleading conclusions.

4. Results

Ranking	Model	Final Loss	Mean Absolute Error (USD)	Model Direction Accuracy
1	RNN	0.00042714	1621.413	54.0%
2	CNN	0.00044865	1805.642	53.8%
3	LSTM	0.00045700	1600.595	53.4%
4	MLP	0.00064806	1587.748	53.1%
5	TRANSFORMER	0.00087198	1732.9	53.1%

4.1 Results Analysis

Result rankings are based on final loss and the model's direction accuracy.

- Lowest loss: RNN
- Lowest mean absolute error (USD): MLP
- Best directional accuracy: RNN at 54%

All models performed similarly in direction prediction, around 53-54% accuracy, suggesting that capturing exact price movement direction remains difficult. Transformer and MLP models may need more training or architectural tuning for better results, since their losses are significantly higher than the RNN, CNN and LSTM.

Initially, we based our results on simply Mean Absolute Error. However, we found that this was not actually a very good metric because it does not actually reveal how good the model is and rather increases the chance that the model 'got lucky' and happened to land within our ±5% margin of error. Instead, we added the direction accuracy metric, which shows whether, at the time of prediction, whether the model predicted the price was rising or falling and comparing that to the true data. Based on our results, the metric demonstrates that our models could theoretically be used to make money since more often than not the prediction direction is correct.

4.2 Individual Model Analysis

MLP (Multilayer Perceptron)

Final Loss: 0.00064806

Final MAE (USD): 1587.748

The MLP showed strong initial convergence, rapidly reducing loss in early epochs. However, it plateaued earlier compared to sequence models, highlighting its limitations in modelling temporal

dependencies in crypto price data. While effective for capturing basic patterns and correlations, the MLP lacks a mechanism to remember or emphasise sequential information.

CNN (Convolutional Neural Network)

Final Loss: 0.00044865

Final MAE (USD): 1805.642

The CNN exhibited very stable and consistent convergence across epochs. Despite originally being designed for image and spatial data, the CNN effectively captured local time-based patterns in the crypto dataset, likely acting as a sliding window detector for short-term trends.

RNN (Recurrent Neural Network)

Final Loss: 0.00042714

Final MAE (USD): 1621.413

The RNN achieved the best overall performance, demonstrating strong capabilities in modelling sequential price data. Its design naturally suits the time-series prediction task, capturing dependencies between consecutive time steps. However, standard RNNs can still struggle with longer-term dependencies due to vanishing gradients.

LSTM (Long Short-Term Memory)

Final Loss: 0.00045700

Final MAE (USD): 1600.595

The LSTM model performed very competitively, achieving the lowest MAE among all models. LSTM's architecture is designed to remember long-term dependencies, making it highly suitable for volatile and sequential crypto price data. Its slightly higher loss compared to RNN could be due to minor overfitting or suboptimal hyperparameters.

TRANSFORMER

Final Loss: 0.00087198

Final MAE (USD): 1732.900

The Transformer showed the highest final loss but carries significant potential. Transformers excel when provided with large datasets and longer training regimes. In this experiment, it might not have fully converged due to insufficient epochs or data. Transformers offer advantages in capturing complex global patterns across longer time windows without recurrent bottlenecks.

5. Reflection and Discussion

Overall, deep learning shows real promise for cryptocurrency price prediction, especially when the focus is on probabilistic forecasting rather than exact numbers. Building a unified codebase early paid off, as it allowed us to easily test different models. All models demonstrated promise and performed convincingly.

Overall, MLP had the lowest mean squared error with a value of $1587.748, while RNN had highest model direction accuracy, predicting the correct derivative at a rate of 54%. The range of mean squared errors was between $1587.748-$1805.642, while the range of model direction accuracy was 53.1%-54%.

It surprised us how competitive the simple MLP was compared to more complex models like LSTMs and Transformers. This highlights that, for relatively structured time series like Bitcoin prices, simpler models can sometimes be more effective and efficient.

Additionally, working through the entire model pipeline - from preprocessing to evaluation - gave us a greater appreciation for the importance of engineering discipline in machine learning projects. Small bugs or inconsistencies early in data preparation could have easily led to misleading results later.

Through this project, we also saw firsthand that no "perfect" model exists. Even deep learning models have limitations, particularly when faced with highly stochastic and externally influenced markets like cryptocurrency.

Future work could include:

- Incorporating sentiment analysis (news, tweets) alongside historical prices
- Expanding to multi-asset trading strategies
- Fine-tuning Transformer-based models for even longer time dependencies
- Testing different feature sets (e.g., volume, market cap) beyond price

Deep learning's ability to model nonlinear relationships makes it a natural fit for messy financial data, and our experiments reinforce that view. However, model interpretability and risk management will continue to be important considerations in real-world deployment.

6. Code Repository

GitHub Repo: https://github.com/ilanbrauns/DL_crypto_trading/tree/main

7. References

- Kaparinos, Nick. "Automated Cryptocurrency Trading using Deep Reinforcement Learning." GitHub, 2021. https://github.com/NickKaparinos/Automated-Cryptocurrency-trading-using-Deep-RL
- CoinMetrics Community Network Data. https://coinmetrics.io/community-network-data/
- Kaggle BTC, ETH, and SOL Datasets.