Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project
Final Report

Representation Learning for Financial Time-Series Forecasting

by

Antony Krymski

Email: agk123@ic.ac.uk
GitHub username: agk-123
Repository: https://github.com/ese-msc-2023/irp-agk123

Supervisors:

Prof. Paul Bilokon
Dr. Tom Davison

August 2024

Acknowledgements

I would like to thank Dr. Paul Bilokon and Dr. Tom Davison for their continued support, guidance and feedback throughout this project. Their expertise and mentorship have been invaluable.

Contents

1 Introduction **3**
1.1 Literature Overview . . . 3

2 Methods **4**
2.1 Parameters . . . 4
2.2 Data Source and Preprocessing . . . 4
2.2.1 Data Source . . . 5
2.2.2 Data Preprocessing . . . 5
2.2.3 Train-Test Split . . . 6
2.3 Baseline Models . . . 6
2.3.1 Persistence Model . . . 6
2.3.2 Zero Model . . . 6
2.3.3 Mean Model . . . 6
2.3.4 Linear Regression Model . . . 7
2.4 LSTM Model . . . 7
2.4.1 Overview . . . 7
2.4.2 Architecture . . . 8
2.5 Contrastive Predictive Coding (CPC) . . . 9
2.5.1 Overview of Contrastive Predictive Coding . . . 9
2.5.2 Data Generation for CPC . . . 10
2.5.3 CPC Model Architecture . . . 11
2.5.4 Training . . . 13
2.5.5 Evaluation of CPC Embeddings . . . 13
2.6 Sharpe Optimisation with Linear Regression . . . 14
2.6.1 Sharpe Ratio Definition . . . 14
2.6.2 Training & Evaluation . . . 14

3 Results **15**
3.1 Performance Comparison . . . 15
3.2 LSTM Model Performance . . . 15
3.3 Linear Regression Model Performance . . . 16
3.4 Mean and Zero Models: Evidence of Mean-Reverting Behaviour . . . 16
3.5 Sharpe Optimisation Results . . . 16

4 Analysis **17**

5 Conclusions and Limitations **20**
5.1 Challenges in Implementation . . . 20
5.2 Strengths of the Proposed Solution . . . 20
5.3 Limitations of the Study . . . 20
5.4 Future Work and Recommendations . . . 21
5.5 Conclusion . . . 21

Abstract

The accurate forecasting of financial time series remains a significant challenge due to the stochastic nature of the underlying data. To improve prediction accuracy, feature engineering has become a vital aspect of forecasting financial assets. However, engineering features manually often requires domain expertise. We propose to utilise an automated feature generation architecture, Contrastive Predictive Coding (CPC), to generate embeddings as input to improve the performance of downstream financial time series forecasting models. To benchmark the effectiveness of our approach, we evaluate forecasting models on predicting the next day's log return on various foreign exchange markets with and without embeddings. Finally, we assess our CPC architecture by employing the same trained encoder on different currency pairs and calculating the Sharpe ratio of our strategies.

1 Introduction

Analysing financial time series is an integral component of stock trading, risk management and econometrics. Auto-regressive integrated moving averages and linear regression are traditionally used and serve as the foundation for many modern forecasting techniques [1]. However, these methods often falter when confronted with the highly volatile and stochastic nature of financial datasets, where the assumptions of stationarity and linearity do not hold [1].

Advancements in machine learning have allowed for more complex temporal patterns to be captured for both analysis and forecasting of time series [2]. Deep Learning (DL) and neural networks enable the extraction of complex patterns and representations which have the potential to enhance predictive accuracy. Notably, Long Short-Term Memory (LSTM) networks have been shown to be able to forecast time series accurately to a degree [3].

However, LSTMs still struggle when attempting to forecast non-deterministic time series. This is partially due to the large amount of data required for the LSTM to learn more complex patterns. Feature engineering is typically utilised to supplement models with further data but this is a time-consuming step that requires extensive domain knowledge and can introduce bias.

We propose to implement Contrastive Predictive Coding (CPC) architecture as presented by Aaron van den Oord et al. to extract features from financial time series [4]. To assess the performance of this approach, linear regression and LSTM models are trained with and without the CPC-generated embeddings and compared based on accuracy metrics. Several baselines models such as the persistence model and naive mean model are implemented as a baseline. By conducting this comparative analysis, we aim to evaluate the relative performance and potential advantages of CPC in high-quality embeddings based unpredictable and highly variable stochastic financial time series.

1.1 Literature Overview

"Representation Learning with Contrastive Predictive Coding", introduces CPC as an innovative approach for unsupervised learning [4]. CPC leverages a self-supervised learning framework that predicts future observations in a latent space while contrasting correct futures against negative samples. This approach enables the extraction of meaningful patterns from large datasets without the need for extensive labelled data. CPC's strength lies in its ability to learn compact and informative representations, making it particularly useful for applications where labelled data is scarce. However, CPC was experimented on a deterministic environment and has yet to be applied successfully to financial time series that exhibit stochastic behaviour.

The paper ”Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks” by Guokun Lai et al. proposes a novel deep learning framework, LSTNet, for multivariate time series forecasting [5]. LSTNet attempts to capture both short-term local dependencies and long-term patterns in time series data utilising convolutional neural networks and recurrent neural networks in combination. In addition, a recurrent-skip mechanism along with an autoregressive component is implemented for the handling of long-term dependencies and scaling issues. Despite the model outperforming benchmarks on certain non-stochastic datasets, it failed to statistically outperform baselines on the one financial dataset it was tested on.

”Contrastive Predictive Coding for Anomaly Detection in Multi-Variate Time Series Data” introduces a novel approach called Time-series Representational Learning through CPC (TRL-CPC) for detecting anomalies in multivariate time series data [6]. The proposed method presents an effective way to learn representations using the CPC architecture and a non-linear transformation function. The datasets utilised in this study are not similar to financial time series data, which are notoriously more difficult to predict due to having higher levels of randomness and volatility. Furthermore, the dataset used includes more than one feature, which makes evaluating the CPC architecture's impact on the results more difficult.

The lack of evaluating automated features generated by the CPC architecture in a stochastic environment is what forms the basis of our research.

2 Methods

Here we outline our methodology used to investigate the effectiveness of CPC embeddings in improving financial time series forecasting. Our experiment framework consists of data sourcing and preprocessing, baseline model development, neural network implementation, and the CPC architecture.

2.1 Parameters

For our experiment the following variables were kept constant:

- **windows = 10:** The number of sliding windows used to divide the time series data.
- **timesteps = 25:** The length of each window.
- **features = 1:** The number of features in the input data (just the closing price).
- **code size = 32:** The dimensionality of the encoded representation, also known as the size of the latent space.
- **batch size = 512:** The number of samples processed by the model in one forward/backward pass during training.

2.2 Data Source and Preprocessing

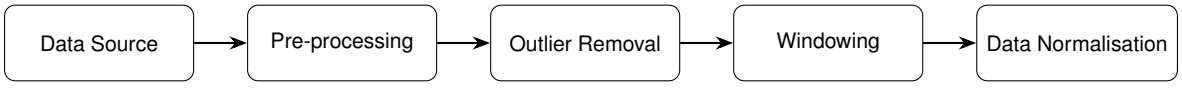


Figure 1: Data Preprocessing Pipeline

2.2.1 Data Source

The dataset employed in this study and for the training of the CPC architecture comprises daily closing prices of the US Dollar versus the Japanese Yen exchange rate (USDJPY). In addition, our encoder trained on USDJPY is then domain-adapted on two other datasets: the US Dollar versus the Singaporean Dollar (USDSGD) and the Euro versus the Great British Pound (EURGBP). All data was downloaded in CSV format, obtained from Yahoo Finance [7], and subsequently processed to prepare it for analysis. Data from November 1996 till August 2024 was utilised.

2.2.2 Data Preprocessing

Log Returns Calculation: The raw closing prices were transformed into log returns to stabilise the variance and achieve stationarity. This is a prerequisite for many statistical and machine-learning models due to their assumption of the input data distribution being consistent over time [8]. The log return, r_t, at time t was calculated as follows:

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right) \tag{1}$$

where P_t denotes the closing price at time t.

Data normalisation: The data was then normalised using min-max scaling. This normalisation is beneficial when training neural networks as it improves convergence during training (due to gradient-based optimisation algorithms) and ensures compatibility with activation functions, to name a few [8]. The normalised value, x_t, was computed as:

$$x_t = 2\left(\frac{x_t - \min(x)}{\max(x) - \min(x)}\right) - 1 \tag{2}$$

where $\min(x)$ and $\max(x)$ represent the minimum and maximum values of the time series, respectively.

Sliding Window Approach: The sliding window technique was utilised to convert the normalised time series data into overlapping sub-sequences, called ”windows”. Each window has 250 timesteps (nearly one year of trading days) and is slid across the data at a fixed length of 1.

The pseudocode for the sliding window approach is provided below:

```
FUNCTION create_windows(data, timesteps, stride = 1):
    SET windows TO empty list
    FOR i FROM 0 TO (LENGTH of data - timesteps) STEP stride:
        SET window TO data SLICE from
            index i TO (i + timesteps)
        APPEND window TO windows
    END FOR

    RETURN windows
END FUNCTION
```

Where:

- `data`: represents the normalised time series data.
- `timesteps`: is the fixed length of each sliding window (250 in our case).
- `stride`: defines the step size for moving the window across the data.
- `window`: one singular window of fixed length.
- `windows`: list of overlapping windows.

2.2.3 Train-Test Split

We split the dataset with a $80:20$ ratio with the following distribution of indices:

Set	Index Range	Proportion of Data
Training Set	$0, 1, 2, \ldots, i-1$	80%
Testing Set	$i, i+1, i+2, \ldots, \texttt{len}(X)-1$	20%

Table 1: Time-series split of the dataset into training and testing sets.

2.3 Baseline Models

To establish a benchmark for model performance, several baseline models were implemented. This allows us to assess the quality of CPC embeddings and whether they have extracted relevant features.

2.3.1 Persistence Model

The persistence model is a naive, recursive-like forecasting method that assumes the future value of the time series is the most recent observed value. This can be expressed as:

$$\hat{y}_{t+1} = y_t \tag{3}$$

2.3.2 Zero Model

The zero model predicts that all future values log returns will be zero. This serves as a test for mean reverting behaviour of the asset in question where returns periodically return to zero. We define it as:

$$\hat{y}_t = 0 \tag{4}$$

2.3.3 Mean Model

The mean model predicts that all future values will be equal to the mean of the observed training data. The predicted value, $\hat{y}$, is given by:

$$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{5}$$

where n represents the number of observations in the training set.

2.3.4 Linear Regression Model

A linear regression model was used with lagged time series data as an input and non lagged time series data as the target variable. We attempt to determine as a baseline if there is a linear relationship between the price at time $t-1$ and the price at time t. The regression equation is represented as:

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j X_j \tag{6}$$

where β_0 is the intercept, β_j are the coefficients for each lagged feature X_j, and p is the number of lagged features.

2.4 LSTM Model

2.4.1 Overview

To explore whether a simple model with our embeddings as input can outperform neural networks, a basic LSTM model was implemented. The LSTM model, first introduced by Sepp Hochreiter et al. [9] is a type of recurrent neural network (RNN) designed to be particularly suitable for sequential data. LSTMs are different to standard RNNs as they are capable of capturing both short-term and long-term patterns, helping to mitigate the vanishing gradient problem [9].

An LSTM cell consists of a memory cell, an input gate, a forget gate and an output gate, which together control the flow of information.

The operations of an LSTM cell can be summarised by the following equations:

- **Forget Gate:** decides which information from the previous cell state c_{t-1} should be discarded. It is defined as:

  $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{7}$$

  where f_t is the forget gate vector, σ is the sigmoid activation function, W_f are the weights, h_{t-1} is the previous hidden state, x_t is the current input, and b_f is the bias.

- **Input Gate:** decides which new input will be stored in the cell state. This involves two steps:

  $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{8}$$

  $$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{9}$$

  where i_t is the input gate vector, $\tilde{c}_t$ is the candidate cell state vector, and W_i, W_c are the weights with corresponding biases b_i, b_c.

- **Cell State Update:** c_t is updated by combining the previous cell state and the new candidate cell state:

  $$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{10}$$

  where $\cdot$ denotes element-wise multiplication.

- **Output Gate:** The output gate determines the next hidden state h_t based on the updated cell state c_t:

  $$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{11}$$

  $$h_t = o_t \cdot \tanh(c_t) \tag{12}$$

  where o_t is the output gate vector, and W_o is the weight matrix with bias b_o.

Through gate regulation, LSTMs maintain and update a memory cell state over time. This allows for the network to capture dependencies that span across many time steps in sequential data.

2.4.2 Architecture

The follow architecture for the LSTM was utilised:

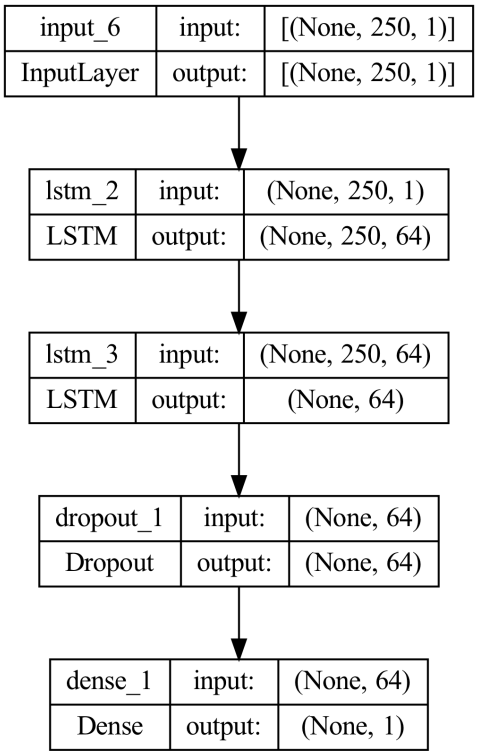


Figure 2: LSTM Architecture

Training Procedure: The LSTM model was trained using the Mean Squared Error (MSE) loss function with the Adam optimiser used to update the model weights.

The Adam optimiser was chosen due to the following reasons:

- Stochastic Gradient Descent and Momentum optimisers although with their own benefits, have a global learning rate that is fixed for all parameters throughout training. This can lead to inefficient updates when different parameters require different learning rates. Adam, on the other hand, computes individual adaptive learning rates for each parameter. For our use-case where limited time is available for parameter tuning the Adam optimiser was the clear choice.
- Unlike the other optimisers, Adam features an adaptive mechanism which can tolerate our input of noisy and stochastic data.
- Finally, Adam performs well in cases of sparse gradients, which can occur in LSTM networks when some weights receive very few updates. This can happen if the LSTM network gets to a local minima, or in our case, finds features that seem to perform well but could be improved minimally.

Due to the relatively large model size compared to the number of data points, training was done for a total of **40** epochs. This is to prevent possible overfitting on the train set.

2.5 Contrastive Predictive Coding (CPC)

CPC is an unsupervised representation learning approach designed to "learn the representations that encode the underlying shared information between different parts of the (high-dimensional) signal" [4]. Unlike traditional supervised learning methods, CPC generates embeddings that capture the temporal structure and dependencies inherent in time series data.

2.5.1 Overview of Contrastive Predictive Coding

The CPC's architecture's objective is to maximise the mutual information between a context vector and future observations in a latent space. The model consists of an encoder network that transforms raw inputs into a lower-dimensional representation and a context network that aggregates these representations over time to predict future sequences [4].

The CPC architecture implemented in this study is tailored for financial time series data and utilises windows as data points. The flow of data through the CPC model can be described as follows:

- **Encoder Network:** denoted as f_θ, maps each window of raw input data, $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,T})$, where i denotes the window index and T is the window length, into a lower-dimensional latent representation z_i. This process is represented by the equation:

$$z_i = f_\theta(X_i) \tag{13}$$

  where $z_i \in R^d$ is the latent representation of window i, and f_θ is a function parameterised by θ (in our case a series of one-dimensional convolutional layers followed by a dense layer).

- **Context Network:** denoted as g_ϕ, aggregates the sequence of window embeddings $\{z_1, z_2, \ldots, z_n\}$ to produce a single context vector c defined as:

$$c = g_\phi(z_1, z_2, \ldots, z_n) \tag{14}$$

  where $c \in R^h$ and g_ϕ is a function parameterised by ϕ, implemented using a Gated Recurrent Unit (GRU). The GRU only returns the final hidden state to output a single context vector that has aggregated information from all window embeddings.

- **Contrastive Objective:** is to maximise the similarity between the context vector c and the true future latent representation z_{n+k}, while minimising the similarity with the negative sample. We do this by minimising the binary cross-entropy loss function where the predicted probability $\hat{y}$ for the positive sample being the correct future latent representation is computed as:

$$\hat{y} = \sigma\left(\text{mean}(c \cdot z_{n+k})\right) \tag{15}$$

  where σ denotes the sigmoid function. The binary cross-entropy loss, L, is then computed as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right) \tag{16}$$

  where y_i represents the binary label (1 for positive samples, 0 for negative samples), and $\hat{y}_i$ represents the predicted probability obtained from the sigmoid activation.

2.5.2 Data Generation for CPC

The generation of negative samples is arguably the most important aspect of the CPC architecture. If negative samples are too easy to differentiate between positive samples, the model will fail to create relevant and predictive embeddings. In contrast, if negative samples are too similar to the positive samples, the model will fail to learn and high-quality embeddings will not be generated. Therefore, a custom data generator was developed to create batches of data specifically for contrastive learning. This generator produces sequences of time windows, with each batch containing both positive and negative pairs, along with corresponding labels indicating whether a given pair is a true or false match.

Positive Sample Generation: For each index or window in the time series data, the positive sample represents the next consecutive window of the time series.

The following pseudo-code illustrates the process for generating positive samples:

```
FOR each position i IN data LENGTH - (timesteps * (n_windows + 1)):
    SET context_windows TO empty list
    FOR each w IN range(n_windows):
        SET window TO data SLICE from
            (i+(w*timesteps)) TO (i + (w + 1) * timesteps)
        APPEND window TO context_windows
    SET y_positive TO data SLICE from
        (i+(n_windows*timesteps)) TO (i+(n_windows+1)*timesteps)
END FOR
```

Negative Sample Generation: Conventional CPC methodologies typically involve selecting random samples from different sequences or distant parts of the same sequence. In our approach, negative samples are generated using a normal distribution of random noise based on the parameters of the input window. This is based on the Black-Scholes framework which models financial time series as a stochastic process [10]. Generating negative samples in this way allows for the model to learn to differentiate from random noise, therefore creating high-quality embeddings.

To generate a negative sample, a random normal distribution is created:

$$y_{\text{negative}} \sim \mathcal{N}(\mu_{y_{\text{negative}}}, \sigma_{y_{\text{negative}}}) \tag{17}$$

where $\mu_{y_{\text{negative}}}$ and $\sigma_{y_{\text{negative}}}$ are the mean and standard deviation of the input window.

The pseudo-code for generating negative samples is as follows:

```
FOR each position i IN data LENGTH - (timesteps * (n_windows + 1)):
    SET context_windows TO empty list
    FOR each w IN range(n_windows):
        SET window TO data SLICE from
            (i + w * timesteps) TO (i + (w + 1) * timesteps)
        APPEND window TO context_windows
    SET y_negative_base TO data SLICE from
        (i + (n_windows - 1) * timesteps) TO (i + n_windows * timesteps)
    GENERATE y_negative USING Gaussian noise WITH
        mean(y_negative_base) AND std(y_negative_base)
END FOR
```

This negative sampling strategy introduces noise that mimics random characteristics of financial time series, forcing the CPC model to learn patterns that are more predictive than randomness.

In Figure 3 one can see an example of an input window and its corresponding positive/actual sample and negative sample.

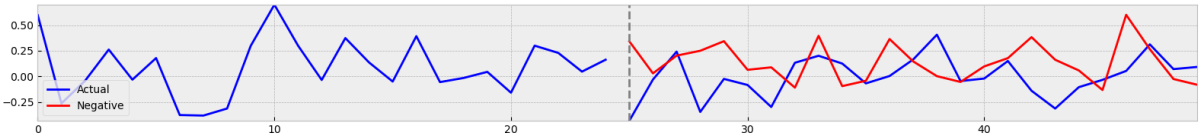
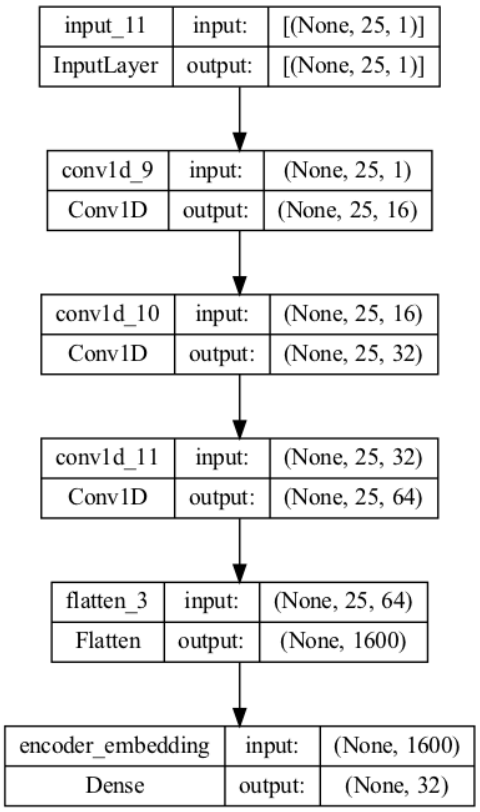


Figure 3: CPC Positive and Negative Sample Exampels

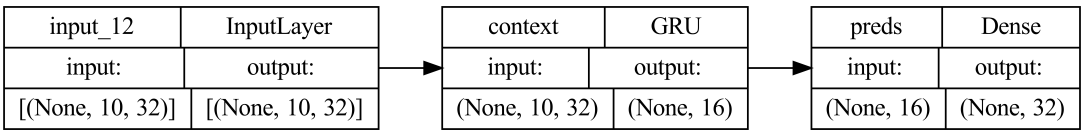
2.5.3 CPC Model Architecture

The CPC model architecture is depicted in Figure 4. The architecture consists of an encoder network, which transforms input windows into embeddings, and a context network, which aggregates these embeddings over time to predict future observations.



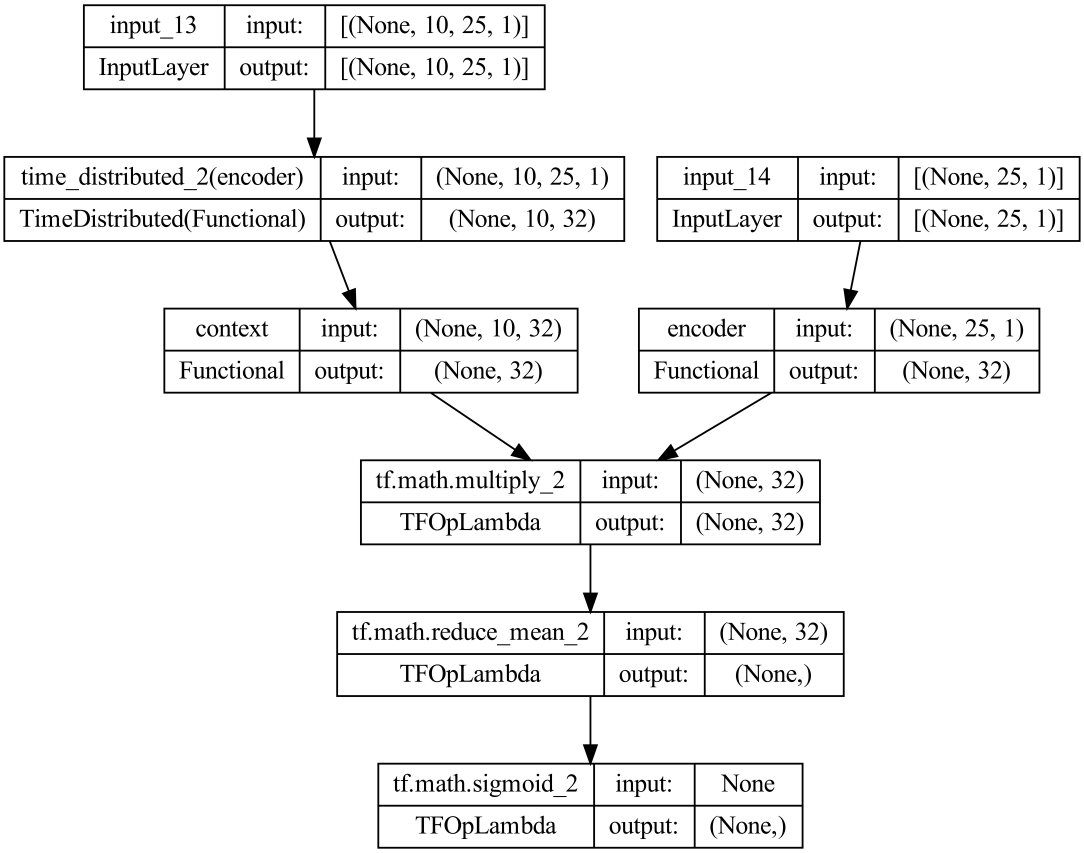
(a) Encoder Architecture

Encoder: The encoder model utilises three one-dimensional convolution layers with different sizes to extract relevant features from the input. This is then flattened to generate a single embedding for that window.



(b) Context Architecture

Context: The context model utilises one GRU layer to extract temporal features from the sequence of embeddings generated by the encoder from the windowed input.



(c) Overall CPC Architecture

Overall CPC Model: The overall CPC model integrates the encoder and context models to predict future latent representations based on past input windows.

Figure 4: Overall CPC Model Architecture

2.5.4 Training

During training, for every epoch the binary accuracy is displayed for both train and test set to monitor potential overfitting. The CPC architecture was ran for **100** epochs, after which overfitting to the train set was very apparent.

2.5.5 Evaluation of CPC Embeddings

After training, the encoder is frozen and used to create embeddings for both the training and testing datasets. These embeddings are then evaluated using the architectures mentioned earlier.

To analyse the embeddings from a visual standpoint, t-SNE, a dimensionality reduction technique was utilised to reduce the number of components to two. The K-means algorithm was then used to colour the embeddings in t-SNE vector space. Using the elbow method, it was determined that the ideal number of clusters was 4. Figure 5 shows the K-Means clustering.

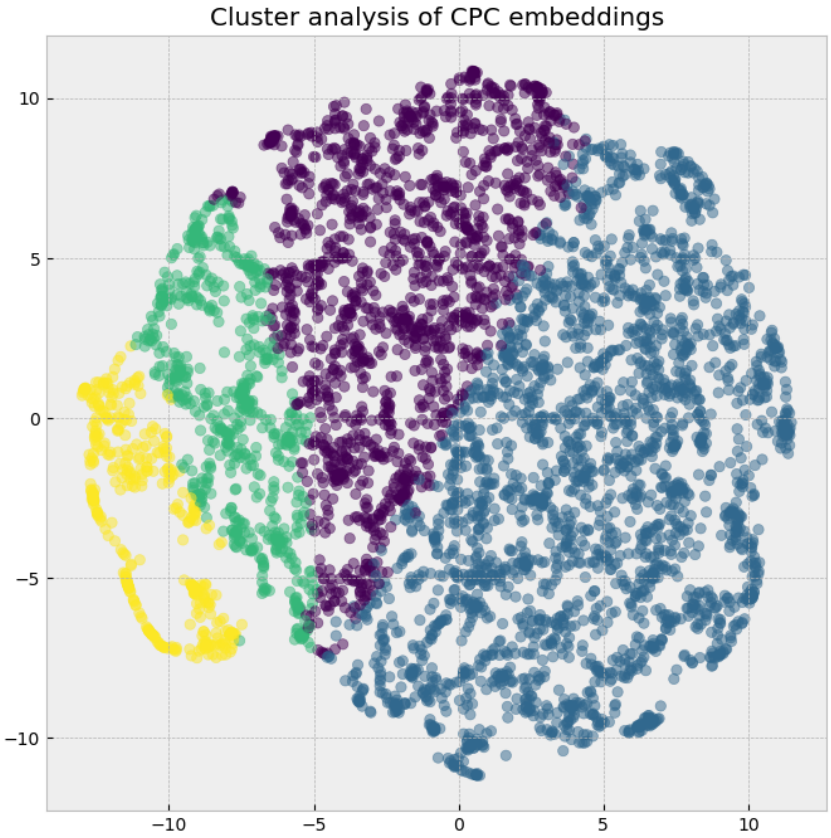


Figure 5: Clustering of embeddings in two-dimensions

Visually, one can clearly see distinct clustering and segragation of the different clusters. This suggests that the architecture is in fact learning and producing high-quality embeddings that effectively identify different groups in the input data.

2.6 Sharpe Optimisation with Linear Regression

To utilise the embeddings in a formal and financial context we have used our embeddings in combination with Linear Regression and evaluated its performance using the Sharpe Ratio.

2.6.1 Sharpe Ratio Definition

William F. Sharpe defined the Sharpe ratio as: ”the mean excess return per unit of standard deviation of excess return” [11]. Mathematically, it is defined as:

$$\text{Sharpe Ratio} = \frac{E[R - R_f]}{\sigma_R} \tag{18}$$

where R is the portfolio return, R_f is the risk-free rate (often assumed to be zero for simplicity in some contexts), and σ_R is the standard deviation of the portfolio returns. To analyse the Sharpe ratio, we multiply by $\sqrt{252}$, assuming 252 trading days in a year.

2.6.2 Training & Evaluation

We will evaluate our model on the test set of the USDJPY dataset and compare the Sharpe ratio to a simple buy-and-hold strategy Sharpe ratio.

As a further test to show the quality of the embeddings generated, the encoder trained on USDJPY will then be used to generate embeddings on two other datasets: USDSGD and the EURGBP, to show our CPC architecture is capable of generating both generalist and specific features.

3 Results

In this section, we present the results of our experiments to prove the benefit of the embeddings the CPC architecture is capable of generating. The models assessed include baseline models (Persistence, Zero, Mean), a Linear Regression model (LR), an LSTM network, and an LR model utilising CPC (CPC-LR). The performance of each model was evaluated using two key metrics: RMSE and MAE. A further percentage difference in each model's performance relative to the CPC-LR model was calculated.

3.1 Performance Comparison

The percentage difference is calculated using the formula:

$$\text{Percentage Difference} = \left(\frac{\text{Error of Model} - \text{Error of CPC-LR}}{\text{Error of CPC-LR}} \right) \times 100 \tag{19}$$

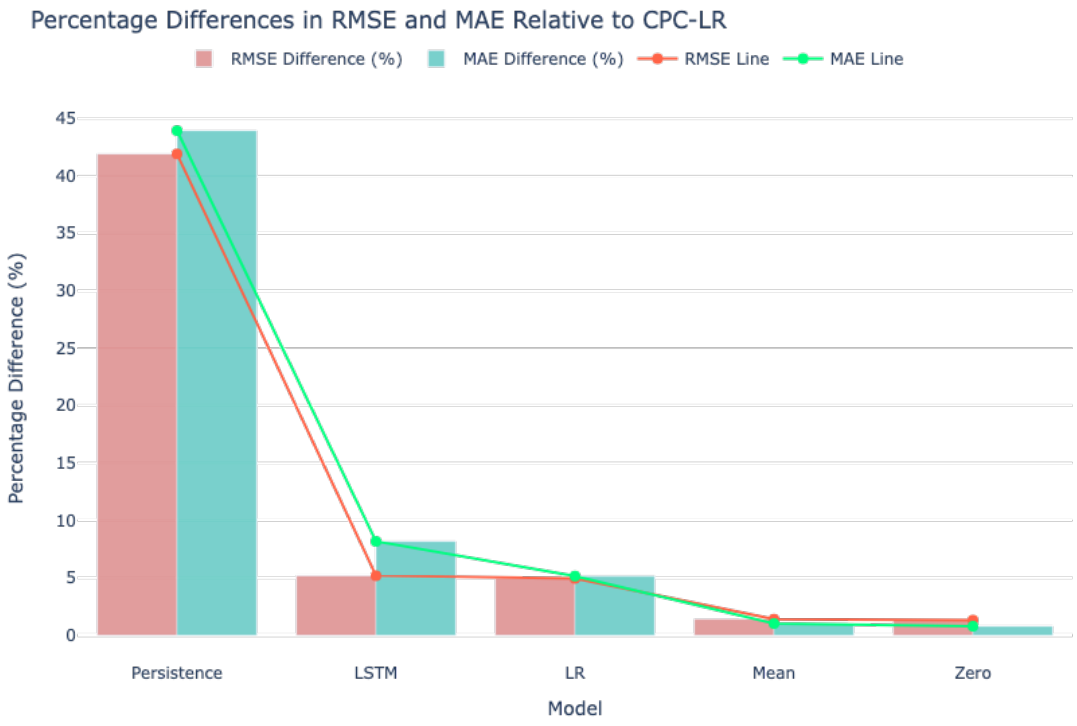


Figure 6: RMSE and MAE relative to CPC-LR

3.2 LSTM Model Performance

Despite the consensus that LSTMs are designed to avoid the gradient vanishing problem and should be able to capture temporal dependencies [9] , the results suggest that the LSTM model

struggles to learn meaningful features from the USDJPY time series data. This can be attributed to the following factors:

- Financial time series are inherently noisy and are highly stochastic. This is due to a variety of factors (geopolitical events, macroeconomic indicators, etc.) that together contribute to an almost random behaviour. LSTM models require a substantial amount of data or features to learn effectively and therefore struggled with our task.
- LSTM architectures have more parameters than their non-neural network counterparts. If the number of parameters is significantly larger than the number of data points, overfitting has a high probability of occuring. This concern is exacerbated further when an LSTM is trained on noisy data.

3.3 Linear Regression Model Performance

The LR model, despite its simplicity and significantly fewer parameters compared to the LSTM model, outperforms the LSTM in forecasting the USDJPY. This surprising result can be explained by several reasons:

- Linear Regression, being a simpler model with fewer parameters, is less prone to overfitting compared to an LSTM. It does not attempt to capture complex non-linear patterns in the data, which can be beneficial due to the inherent noisy nature of the our input.
- The LR model provides stable parameter estimates even with relatively small datasets. This can be beneficial as the relationship between input and output might not be highly complex or might vary frequently due to external market conditions.
- The financial time series of exchange rates often exhibit linear trends or mean-reverting behaviour over short periods. The LR model, which inherently assumes a linear relationship between input variables and the output, can effectively capture these linear or near-linear trends without the need for complex model architectures.

3.4 Mean and Zero Models: Evidence of Mean-Reverting Behaviour

Both the Mean and Zero models perform close to each other and have the lowest error rate among the other models. This outcome strongly suggests that the USDJPY rate exhibits a mean-reverting behaviour, a common characteristic in many financial time series.

- The Mean model, which predicts future values as the mean of the historical data, works well in scenarios where the time series is mean-reverting. The near-zero error indicates that the exchange rate often returns to its average return over time, making the Mean model a reliable predictor.
- Similarly, the Zero model, which assumes that all future log returns will be zero (implying no change in price), performs well, further supporting the hypothesis of mean reversion. A zero prediction effectively suggests that the current price level is expected to persist, which aligns with a market that lacks strong directional trends and frequently reverts to a mean.

3.5 Sharpe Optimisation Results

The table below presents the Sharpe ratios for different currency pairs using two different strategies: the LR model using CPC embeddings generated on the USDJPY dataset and the buy-and-hold strategy of the currency the embeddings are being evaluated on.

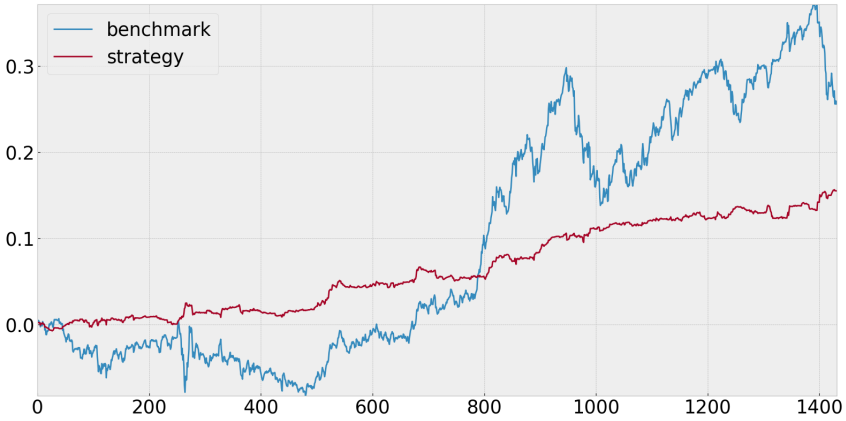
Table 2: Sharpe Ratios for Different Currency Pairs

Currency Pair	Strategy Sharpe Ratio	Buy and Hold Sharpe Ratio
USDJPY	1.312	0.5298
USDSGD	0.9802	-0.3512
EURGBP	0.7405	-0.1216

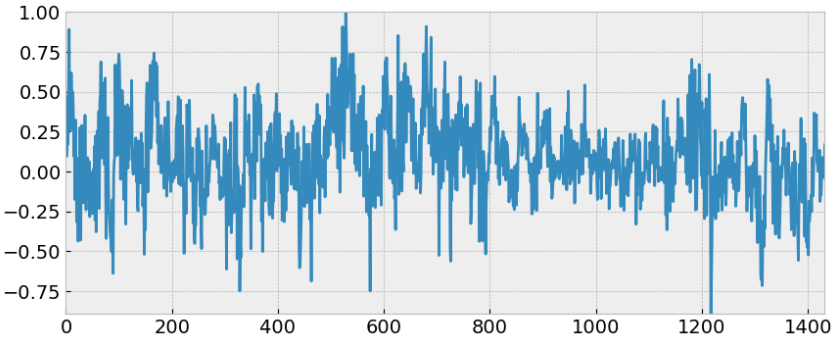
4 Analysis

The strategy Sharpe ratios for all currency pairs evaluated are substantially higher than their corresponding buy-and-hold Sharpe ratios. This indicates that the CPC-based strategy is able to capture underlying patterns and signals in the data that are not apparent to traditional methods. The CPC model's ability to generalise and be fine-tuned on other currency pairs underscores its ability to learn general market sentiment.

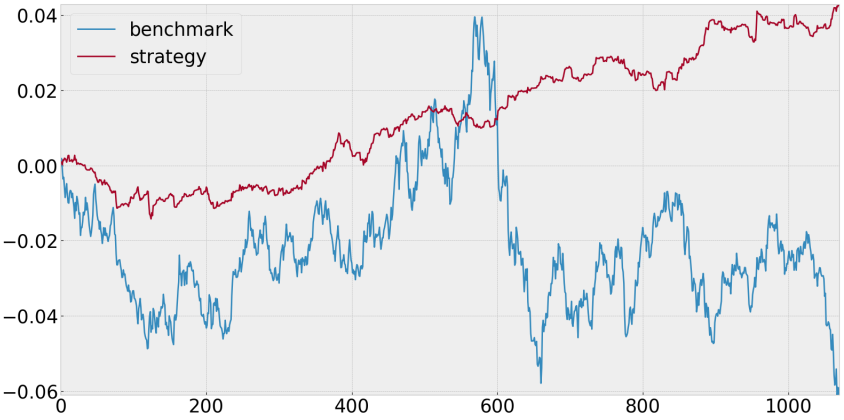
In the plots below, allocations were scaled to be between -1 and 1. This does not affect the calculation of the Sharpe ratio and is done just for plotting purposes.



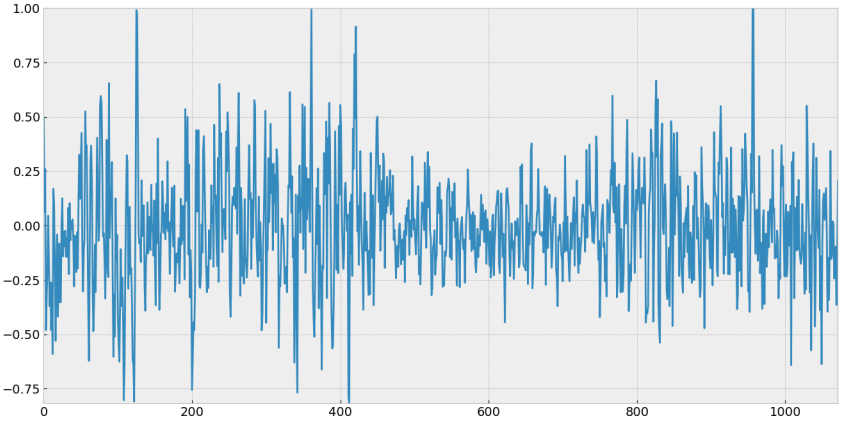
(a) USDJPY Strategy vs. Benchmark Buy-and-Hold



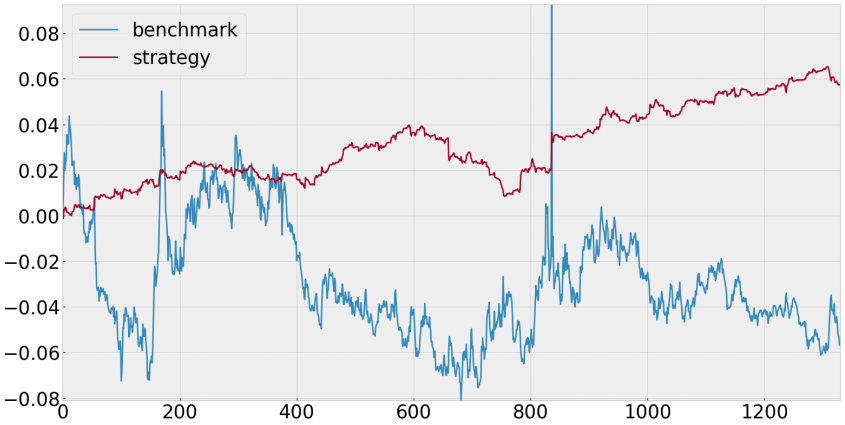
(b) USDJPY Allocations



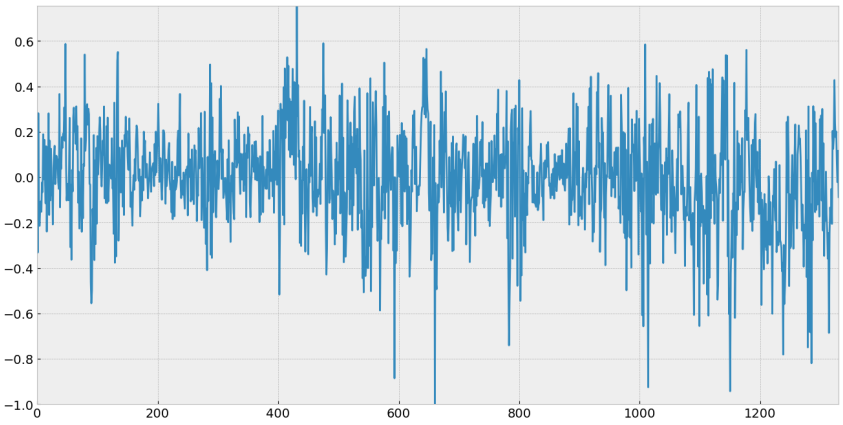
(c) USDSGD Strategy vs. Benchmark Buy-and-Hold



(d) USDSGD Allocations



(e) EURGBP Strategy vs. Benchmark Buy-and-Hold



(f) EURGBP Allocations

Figure 7: Performance and Allocations for USDJPY, USDSGD, and EURGBP

From Figure 7, the following is observed:

Observations from Return Plots: Upon examining the return plots, it is evident that the strategies exhibit an upward, almost linear trend on average. Additionally, there are sporadic jumps upwards in our strategies returns that coincide with significant movements in the original stock, as indicated by the returns of the benchmark buy-and-hold strategy. This pattern suggests that the features extracted by the model provide relevant information on which trades to execute and, more importantly, the optimal timing for these trades. The ability of the strategy to capture upward trends while also responding swiftly to substantial market movements indicates that the model effectively identifies profitable trading opportunities based on the underlying features.

Analysis of Allocations: The allocations plotted over time can be characterised by frequent fluctuations both upwards and downwards. This behaviour is consistent with the expected strategy of a market maker, whose role is to provide liquidity by constantly buying and selling, thereby ensuring a smooth market operation. The high-frequency nature of these allocations reflects an attempt to capture small price discrepancies, making quick trades that capitalise on these minor variations. Such behaviour is advantageous in forex markets, where tight spreads and high liquidity present numerous opportunities for rapid trades.

Furthermore, the allocations appear to be, on average, equally distributed between positive and

negative positions indicating that the model is finding opportunities to go both long and short. This is a positive outcome as this type of neutral portfolio behaviour tends to perform well in the mean-reverting behaviour of foreign exchange markets. Since prices tend to return to a central value over time taking both long and short positions allows the model to profit from both upward and downward movements.

5 Conclusions and Limitations

Throughout the implementation of this project, there were several limitations and turning points that are vital for one to understand before conducting further research.

5.1 Challenges in Implementation

One of the key challenges was this endeavour being relatively novel; adapting from existing literature to a CPC architecture that learns representations from a stochastic windowed time series proved to be difficult. Treating each window as a distinct time step required careful consideration to ensure that the context vector correctly aggregated information across these windows/inputs.

The creation of negative samples that were appropriately challenging for the CPC model is a section that requires special attention. The objective was to design negative samples that provided sufficient difficulty to the learning process without causing the model to fail completely. This comes with several issues, namely, how does one know if a sample is too easy for the model to differentiate from or if the model has found very high-quality embeddings? Is the best negative sample one that is random and based on the Black-Scholes framework? This should be investigated further.

5.2 Strengths of the Proposed Solution

The results of this study illustrate that CPC can generate high-quality features or embeddings. The architecture has the potential to save significant amounts of time and effort compared to traditional feature engineering methods and does not require one to be a domain expert. With appropriate tuning, there is no fundamental reason why CPC cannot identify the same features that would be manually engineered.

Moreover, by fine-tuning on different currency pairs, we have shown that the embeddings generated by the CPC model are versatile and can be employed for a variety of downstream tasks. It is likely that CPC would perform even better in non-stochastic time series environments, due to the lack randomness and noise. The generation of negative samples may need to be adjusted if one were to use this architecture on other types of windowed time series.

5.3 Limitations of the Study

This project had a three-month time constraint which proved to be a limitation in the implementation of this project. Given more time, more complex neural networks would be explored and all models would be hyperparameter tuned to ensure best possible comparison.

The quantity of data available is an inherent problem in finance. For example, the USDJPY non-cross-currency has just above 7000 daily closing prices available through Yahoo Finance. If one were to use neural networks to find more intricate and complex relationships between the features/embeddings generated by the CPC model, the amount of data would be severely

limiting and would make overfitting on the train set a significant concern. As a result, it is challenging to assess the true capacity of deeper architectures in this context.

Finally, the current analysis does not account for transaction costs or the practicalities of executing trades at the closing price of each day, which is often unrealistic in real-world financial markets due to slippage and market impact. These factors could affect the strategy's performance and should be considered in any practical application of the model.

5.4 Future Work and Recommendations

Several paths for future research and improvement are evident. A further exploration of different neural network architectures and hyperparameter tuning could provide a better understanding of the CPC model's capabilities. Utilising other datasets that may have more historical data could also reduce the chance of overfitting. Finally, adding more data augmentation techniques, particularly for generating negative samples, could further improve the CPC model's ability to learn meaningful features from the input.

5.5 Conclusion

Overall, a CPC architecture was built that outputs relevant, high-quality embeddings for downstream financial forecasting time-series tasks. Negative samples in the architecture were generated non-traditionally using the Black-Scholes framework as a basis. These embeddings were used to successfully forecast several different foreign exchange currency pairs and beat all traditional benchmarks. Significant alpha was found demonstrated by a Sharpe ratio beating the traditional buy-and-hold benchmark and ultimately helped in achieving our objective of producing an edge against the market.

In conclusion, the architecture developed can be used to generate features automatically without the need for manual feature engineering and domain expertise. This could impact further research into stochastic time series forecasting and improve the current state of the art.

References

[1] Ping-Feng Pai and Chih-Sheng Lin. A hybrid arima and support vector machines model in stock price forecasting, 2005.

[2] Angelo Casolaro, Vincenzo Capone, Gennaro Iannuzzo, and Francesco Camastra. Deep learning for time series forecasting: Advances and open problems. *Information*, 14(11), 2023.

[3] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401, 2018.

[4] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[5] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks, 2018.

[6] Theivendiram Pranavan, Terence Sim, Arulmurugan Ambikapathi, and Savitha Ramasamy. Contrastive predictive coding for anomaly detection in multi-variate time series data, 2022.

[7] Yahoo Finance. Yahoo finance stock lookup, 2024. Accessed: 2024-08-29.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[10] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[11] William F. Sharpe. The sharpe ratio, [].