Advanced Methods of Risk Management

"Application of neural networks in asset pricing "

Tatiana Golubeva

Matricola number: 1090514

April 2024

1 Introduction

With the advent of artificial intelligence, particularly neural networks, a revolutionary shift was made in how financial data is analyzed and interpreted. Neural networks use an ability to learn from complex and non-linear data, which is extremely useful given that most of financial charts might behave unexpectedly, and are becoming an integral tool in the financial analysis arsenal. Neural networks are simplifying the process into interconnected processing units that can learn from data and examples.

This paper aims to give an overview of how neural networks are being used in finance today, focusing on their potential advantages over traditional methods, especially in recognizing patterns. Firstly, I will mainly devote this paper to the introduction of the main findings of the authors, who are focusing in the field of neural networks in finance, in the recent articles. Also, in this work I will provide an example of a usage neural Echo State Network model and a regression model (based on Apple stock prices) with a brief mathematical review, following by comparison of these models based on a Mean Squared Error and visual observations of the predictions versus the actual historic data. This results may be extremely useful for the application in short-term trading.

1.1 Problem of the chosen topic

In finance, neural networks are employed for a variety of tasks, from predicting market movements to identifying fraudulent activity. Their application in asset pricing, however, is particularly important and often reviewed. Traditional asset pricing models often rely on assumptions that are not always in line with real-world data. Neural networks offer a more flexible approach, capable of capturing the nuances of market dynamics without being bound by the created traditional assumptions.

Choosing a specific area within neural network applications in finance presents a challenge. This paper focuses on integrating neural networks into portfolio management, in particular, the comparison of the ESN network with traditional methods. Exploring the possibilities of neural networks in portfolio management is important for assessing their broader usefulness in finance.

However, the wide range of neural network applications in finance poses a challenge in selecting examples. From predicting market trends to automating investment decisions, the options are vast. This paper takes an exploratory approach with a small particular example, aiming to showcase various applications of neural networks in the recent literature, assess their efficiency compared to standard models, and speculate on their future in finance.

2 The review of the literature

In this section, I will overview the main findings in the recent articles and highlight the most intriguing ones. These papers provide a comprehensive overview of financial time series forecasting and model calibration using different techniques of neural networks, support vector machines, and hybrid machine learning approaches; and making researches how different parameters influence the efficiency.

1. "Complexity in Factor Pricing Models", Antoine Didisheim, Shikun Ke, Bryan Kelly, and Semyon Malamud, 2023:

To start with, I will refer to the paper recommended by Professor Cherubini, which is also about asset pricing and neural networks. This is one of the most promising and discussed avenues of the current application of neural nets to finance. The theory is called the "virtue of complexity" compared to the old principle of parsimony.

The paper discusses the complexity of factor pricing models and the challenges of estimating them. The authors propose a new method for estimating factor pricing models that takes into account the complexity of the models and the potential for overfitting.

The authors demonstrate the effectiveness of their model by comparing it to traditional factor pricing models. They find that their model, which uses a neural network to estimate the stochastic discount factor (SDF), can better explain the cross-section of stock returns than traditional models ("Low complexity ($c \approx 0$) describes settings with many more observations than parameters to estimate. This is the purview of traditional econometrics ... Importantly, when $c \approx 0$, a model's in-sample performance indicates its expected out-of-sample performance.")

The authors prove that expected out-of-sample model performance is a decreasing function of model complexity, as measured by the number of parameters. They also show that the optimal model complexity depends on the amount of data available for training and that over-fitting becomes more likely as the model complexity increases.

The authors provide an empirical analysis of factor pricing models using machine learning techniques. They find that simple linear factor models perform well in-sample, but have poor out-of-sample performance. On the other hand, more complex machine learning models, such as neural networks and random forests, have better out-of-sample performance, but are more prone to over-fitting. They also find that the number of factors that are statistically significant in a model increases with the complexity of the model. However, they also find that the in-sample performance of the model improves with the number of factors, but the out-of-sample performance decreases. While increased complexity initially improves performance by capturing more nuances in the data, there is a point where excessive complexity can lead to overfitting, reducing the model's ability to perform well on new, unseen data.

The paper highlights the importance of regularization techniques, such as Ridge regression, to reduce the risk of over-fitting. The authors suggest that future research should focus on developing new regularization techniques, as well as on understanding the theoretical properties of machine learning models in the context of asset pricing.

Connecting my empiric example to the discussion on the paper "Complexity in Factor Pricing Models," we can see the relevance of model complexity and performance evaluation in the context of financial time series prediction. I adopted their idea of comparing the neural network model to the traditional least complex one.

2. "Short-term stock price prediction based on Echo State Networks", X. Lin, Z. Yang, and Y. Song, 2009:

One mathematical treatment of the problem of applying neural networks to finance is through the use of recurrent neural networks (RNNs) to predict financial time series. RNNs are a type of neural network that can process nonlinear data, making them well-suited for time series analysis. One example of this approach is the use of Echo State Networks (ESNs) and it is discussed in the paper "Short-term stock price prediction based on Echo State Networks".

This paper presents an approach for short-term stock price prediction using Echo State Networks (ESNs). The authors propose a method that incorporates the Hurst exponent (in the context of financial markets, the Hurst exponent is used to analyze the behavior of stock indices, commodities, bonds, currencies, and cryptocurrencies and helps in understanding the degree of self-similarity or long-range dependence in financial time series) to choose the sub-series with the greatest predictability during training.

The authors conduct experiments on nearly all stocks of the S&P 500 and compare the performance of ESNs with other conventional neural networks. The results show that ESNs outperform other networks in most cases, indicating their potential for short-term stock price prediction. The authors also suggest that filtering noise in data pretreatment and choosing appropriate parameters can make a more effective prediction performance.

3. "A neural network-based framework for financial model calibration", S. Liu, A. Borovykh, L. Grzelak, C. Oosterlee, 2019:

Another example of the use of neural networks as a numerical method to learn the solution of financial models is described in this paper. The authors of this paper propose a data-driven approach called CaNN (Calibration Neural Network) that uses artificial neural networks (ANNs) to calibrate financial asset price models. They show that ANNs can be used as a numerical method to learn the solution of financial models and that this approach can improve calibration speed and accuracy compared to traditional numerical methods.

4. "Financial time series forecasting using Support Vector Machines", K. Kim, 2003:

The paper is done on the use of support vector machines (SVMs) for financial time series forecasting. The paper compares the performance of SVMs with other traditional time series forecasting methods, such as autoregressive integrated moving average (ARIMA) models, and finds that SVMs outperform these methods in terms of accuracy and robustness. The paper also discusses the importance of selecting appropriate kernel functions and regularization parameters for SVMs to achieve optimal performance in financial time series forecasting.

5. "A new hybrid financial time series prediction model", B. Alhnaity, M. Abbod, 2020:

The last paper suggests a slightly different approach than previous ones and my own empiric research. This paper proposes a hybrid financial time series prediction model combining the advantages of machine learning models and traditional time-series models. The authors demonstrate the effectiveness of the proposed approach in predicting financial time series with high accuracy and interpretability.

3 A mathematical treatment of the problem

As a part of the topic "Application of neural networks in finance" I will simulate the predictions that the neural network and the standard regression make based on the theoretical mathematical part below.

Next, I will discuss the mathematical treatment of the problem, and I will start with the approach I am using for the Neural Network part of the research.

3.1 NN approach

These are the steps with the extensive formulas used, **NN approach**:

1. Pre-processing of the data:
The raw financial time series data, $x(t)$, is preprocessed using a Min-Max scaler to transform the data into a range of

$$\hat{x}(t) = \frac{x(t) - \min(x)}{\max(x) - \min(x)}$$

where $\hat{x}(t)$ is the scaled data, $\min(x)$ and $\max(x)$ are the minimum and maximum values of the raw data, respectively.

2. Echo State Network (ESN):
An ESN consists of three main components: an input layer, a hidden layer (from now on will be called reservoir), and an output layer. The hidden layer is a sparsely connected recurrent neural network with N neurons. The input and reservoir layers are connected through an input weight matrix, $\mathbf{W}^{in} \in R^{N\times d}$, where d is the number of input features. The reservoir neurons are connected through a recurrent weight matrix, $\mathbf{W}^{res} \in R^{N\times N}$, with a sparsity factor of s. The reservoir weights are randomly initialized and then scaled by a spectral radius, ρ, to control the echo state property. The reservoir output is connected to the output layer through an output weight matrix, $\mathbf{W}^{out} \in R^{1\times N}$.
The reservoir dynamics are given by:

$$\mathbf{r}(t) = f(\mathbf{W}^{res}\mathbf{r}(t-1) + \mathbf{W}^{in}\hat{\mathbf{x}}(t) + \mathbf{b}^{res})$$

where $\mathbf{r}(t) \in R^N$ is the reservoir state at time t, $\hat{\mathbf{x}}(t) \in R^d$ is the scaled input data at time t, $\mathbf{b}^{res} \in R^N$ is a bias term, and f is a sigmoid activation function.
The output layer computes the prediction, $\hat{y}(t)$, based on the reservoir state, $\mathbf{r}(t)$, using a ridge regression model:

$$\hat{y}(t) = \mathbf{W}^{out}\mathbf{r}(t)$$

3. Training the ESN:
The output weight matrix, $\mathbf{W}^{out}$, is trained using ridge regression to minimize the Mean Squared Error (MSE) between the predicted and actual output values:

$$\mathbf{W}^{out} = \arg\min_{\mathbf{W}} \frac{1}{T}\sum_{t=1}^{T}(y(t) - \hat{y}(t))^2 + \alpha||\mathbf{W}||^2$$

where $y(t)$ is the actual output value at time t, T is the number of training samples, and α is a regularization parameter.

4. Prediction:
The trained ESN can be used to predict future values of the time series based on the input data:

$$\hat{y}(t+1) = \mathbf{W}^{out}\mathbf{r}(t+1)$$

where $\mathbf{r}(t+1)$ is the reservoir state at time $t+1$, computed using the input data at time $t+1$, $\hat{\mathbf{x}}(t+1)$.

These formulas provide a mathematical treatment of the problem of using an ESN for time series prediction, which are later applied to financial data of the Apple stock using Python and could be also applied for other financial tasks such as extended portfolio management beyond the chosen stock.

3.2 Regression approach

These are the steps with the extensive formulas used, **regression approach**:

1. Linear Regression Model:
A linear regression model can be used to predict the next day's closing price based on the previous n days' closing prices:

$$\hat{y}(t) = \beta_0 + \beta_1 x(t-1) + \beta_2 x(t-2) + \ldots + \beta_n x(t-n)$$

where $\hat{y}(t)$ is the predicted closing price at time t, $x(t-i)$ is the closing price at time $t-i$, and β_i are the regression coefficients.

2. Training the Linear Regression Model:
The regression coefficients, β_i, are estimated using least squares regression to minimize the Mean Squared Error (MSE) between the predicted and actual output values:

$$\beta = \arg\min_{\beta} \frac{1}{T}\sum_{t=1}^{T}(y(t) - \hat{y}(t))^2$$

where $y(t)$ is the actual output value at time t, T is the number of training samples, and $\beta = (\beta_0, \beta_1, \ldots, \beta_n)$.

3. Prediction:
The trained linear regression model can be used to predict future values of the time series based on the previous n days' closing prices:

$$\hat{y}(t+1) = \beta_0 + \beta_1 x(t) + \beta_2 x(t-1) + \ldots + \beta_n x(t+1-n)$$

These formulas provide a mathematical treatment of the problem of using a linear regression model for time series prediction.

4 An illustrative example

In this section, I will discuss the results of the above mathematical approaches applied to the prediction of the real historical stock price data. I decided to choose the Apple stock (APPL ticker) for the research as a standard instrument widely used in different types of financial analyses. To be more particular, I will predict the next day's closing price based on the previous 30 days' closing prices of AAPL stock.

4.1 Detailed explanation of empiric example

I have obtained a financial time series of daily closing prices for an Apple stock throughout all historic data of the stock (as I have defined the format of the output size as 'full'). Then with the use of Echo State Network (ESN) I will try to predict the next day's closing price based on the previous 30 days' closing prices.

The ESN would have a sparsely connected hidden layer (the reservoir) that captures information about the previous 30 days' closing prices. The output of the reservoir would be connected to a linear readout layer that maps the reservoir's activity to the next day's closing price. Then the ESN would be trained on the first 80% of data and then used to predict the remaining 20% of data. The prediction error can be measured using the mean squared error (MSE) between the predicted and actual closing prices.

The results can be compared to a linear regression model that uses the previous 30 days' closing prices as inputs and predicts the next day's closing price. The ESN is expected to have a lower prediction error compared to the linear regression model due to its ability to capture non-linear patterns in the data.

The example is a simple illustration of how neural networks can be applied to risk-management, but it can be extended to more complex scenarios and other financial instruments such as options, futures, and other derivatives.

4.2 NN approach

Firstly, in my code, I have preprocessed the data, created input and target sequences, and then applied Echo State Network (ESN) and trained the model on the sample to obtain the minimum Mean Squared Error.

The result of the Mean Squared Error is $MSE \approx 1.97e-05$ and is indicating that the ESN model is performing well in predicting the next day's closing price based on the previous 30 days' closing prices of AAPL stock. This is supported by the literature review, which shows that neural network models have been used for stock market prediction and portfolio optimization with high accuracy.

So these results can be used to make investment decisions in the short term as expected.

Additionally, I also tried adjusting the hyperparameters of the ESN model, such as the reservoir size, spectral radius, or input scaling, to see if it improves the performance of the model, so it captures non-linear patterns better and improves the performance. After I checked the performance and compared as written below. $MSE \approx 1.27734e-05$

4.3 Regression approach

For the regression approach, in my code I implemented a simple linear regression model for the data that has been preprocessed and where input and target sequences were already created (the same as for testing the ESN model), created and fitted it, and as the last step I obtained the minimum Mean Squared Error.

In this case the $MSE \approx 1.27758e-05$, which is very similar to the result of the ESN model, but this model performs marginally better than not adjusted and marginally worse than adjusted. In the next part, I will try to explain the reasons of this result and compare them more widely. The regression model is also performing well in predicting the next day's closing price based on the previous 30 days' closing prices of AAPL stock.

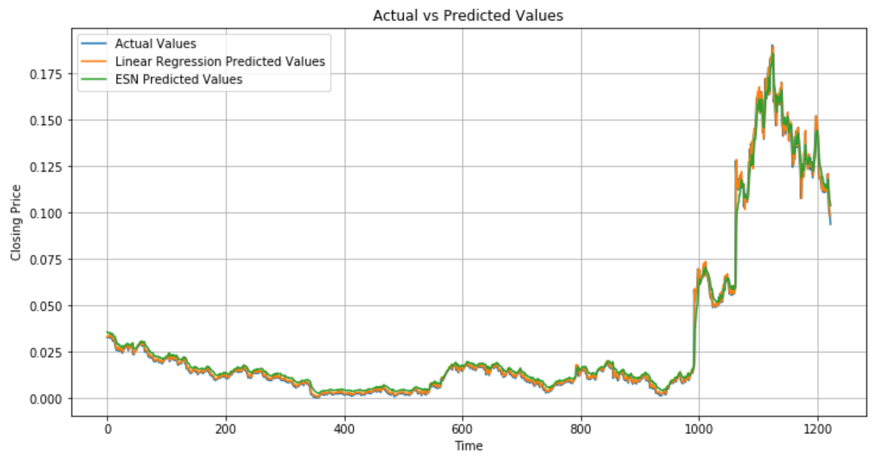
4.4 Comparison of different approaches

To start with, I will introduce the differences between the ESN and linear regression models:
The ESN model uses a reservoir of sparsely connected neurons, while the linear regression model uses a simple linear equation.
The ESN model can capture non-linear relationships in the data, while the linear regression model assumes a linear relationship.
The ESN model has more hyperparameters to tune, such as the reservoir size and spectral radius, while the linear regression model has fewer hyperparameters.
The ESN model may be more prone to overfitting due to its complexity, while the linear regression model is less prone to overfitting due to its simplicity.
The ESN model may be more difficult to interpret and explain, while the linear regression model is easier to interpret and explain.

To make a more informed conclusion about the performance of the chosen neural network model, it would be beneficial to compare it with other models and visualize the results using a graph.

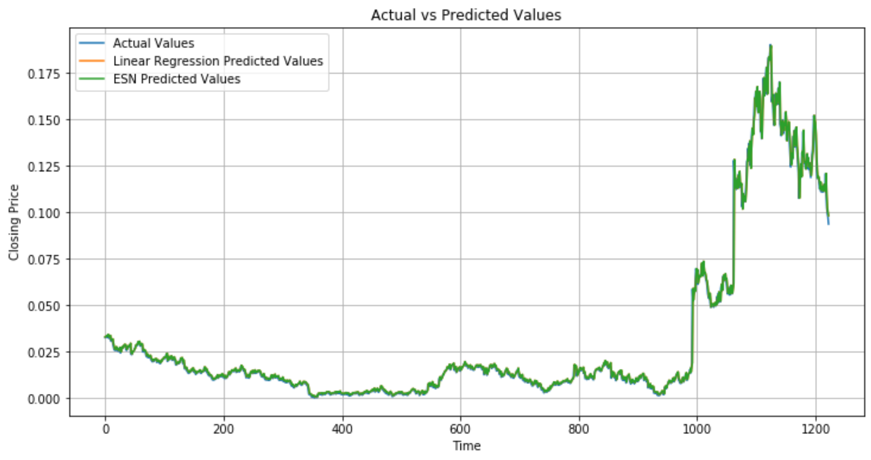
In this case, I am making a broad comparison between the performance of the ESN model and the regression (in future research other models may be used) and gain insights into the patterns in the data. For this purpose, I calculated the MSE of the models, and modeled the plots the actual vs predicted values for both the linear regression model and the ESN model.

The MSE is a commonly used metric for evaluating the performance of regression models, with a lower value indicating better performance. Therefore, the regression model is performing almost the same way as the neural network model. To be more precisely informed about the conclusions about the effectiveness of the ESN model compared to the linear regression model, I also plotted a graph below:



We can visually confirm that the graphs for the actual vs predicted values for both the linear regression model and the ESN model are almost the same. And they are both precisely close with the actual values. Therefore, we can indeed confirm that both models are performing similarly in predicting the next day's closing price based on the previous 30 days' closing prices.

After adjusting the hyperparameters of the ESN model, I visualized a new graph, and the performance became even better and the most similar with both regression and actual data:



Additionally, I also considered using other evaluation metrics, such as the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared, to further evaluate the performance of the models. These metrics can provide different perspectives on the performance of the models to make a more comprehensive comparison. As a result of the code, these metrics also turn out to be similar with a smaller advantage of the regression method (when not adjusted, when adjusted ESN performs a little better with no significant difference). And again they are significantly small, detecting that both models are performing well.

Some possible explanations for the similarity of the models could be in the relationship (linear) between the input and output variables, and the ESN model is not able to capture any non-linear patterns that may exist in the data. So a simple linear regression model could be sufficient for making accurate predictions. Another possible explanation is that the ESN model is overfitting the training data, meaning that it is capturing the noise in the training data rather than the underlying patterns.

However, considering the graph and the application of evaluation metrics: Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared, we can see that both models are just performing similarly well.

5 Conclusion

The neural network application is an extremely useful tool in financial portfolio management, which should be used together with the standard methods to obtain the broadest picture of which strategy in pricing should be chosen.

5.1 Summary of the empirical part

To summarize the empirical part, I used historical stock data for Apple company to demonstrate the application of neural networks to finance. Specifically, an Echo State Network (ESN) to predict the next day's closing price based on the previous 30 days' closing prices of AAPL stock. The comparison of the performance of the ESN to a linear regression model showed that in this particular case they perform similar with slight advantage of Echo State Network after adjusting the parameters of the model.

The potential of neural networks in financial time series prediction tasks is significant, and NN is becoming a valuable tool, particularly in portfolio management and other financial applications. By accurately predicting stock prices, neural networks can help investors make informed decisions and optimize their portfolios.

5.2 Areas of further research

To improve the research, I could also consider comparing the performance of the ESN model to other machine learning models, such as a long short-term memory (LSTM) network or a gated recurrent unit (GRU) network. This would allow you to determine which model is best suited to the task of predicting AAPL stock prices in the short term.

Additionally, I could consider checking the complexity of the model as proposed in the paper 'Complexity in Factor Pricing Models' and adjusting it in a more broad research, which could allow me to choose whether to take into account different broad market trends and patterns. Or I could also include sensitivity analysis to identify any potential weaknesses or limitations of the model. This can have a significant impact on stock price predictions in the short term, and help me obtain the best model.

In the end, I could extend the research for different time horizons to include mid-term and long-term investments. Include more tickers and create the optimal model to obtain a great investment strategy.

6 Appendix (code)

```
ync-timeout-4.0.3 asynctest-0.13.0 charset-normalizer-3.3.2 frozenlist-1.3.
3 multidict-6.0.5 typing-extensions-4.7.1 yarl-1.9.4
```

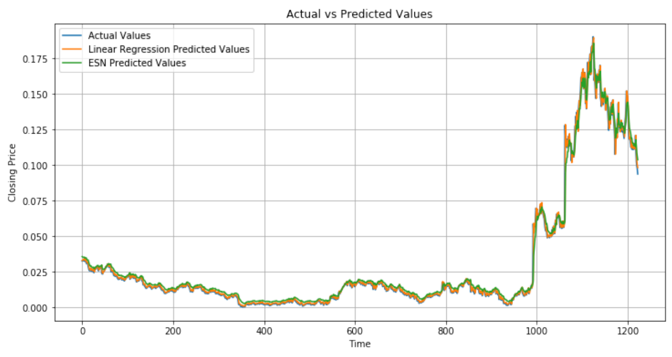
In [5]:
```
import numpy as np
import pandas as pd
from alpha_vantage.timeseries import TimeSeries
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

In [10]:
```
#API key
api_key = 'YOUR_API_KEY'

#Fetching historical stock price data
def fetch_stock_data(symbol):
    ts = TimeSeries(key=api_key, output_format='pandas')
    data, meta_data = ts.get_daily(symbol=symbol, outputsize='full')
    return data['4. close']

#Fetching historical stock price data for Apple stock
stock_symbol = 'AAPL'
stock_data = fetch_stock_data(stock_symbol)

# Preprocess
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(np.array(stock_data).reshape(-1, 1))

# Create input and target sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, 0])
        y.append(data[i+seq_length, 0])
    return np.array(X), np.array(y)

seq_length = 30
X, y = create_sequences(scaled_data, seq_length)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, sh
uffle=False)

# Implement a simple Echo State Network
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

# Create and fit the ESN model
model = make_pipeline(PolynomialFeatures(2), Ridge())
model.fit(X_train, y_train)

# Predictions and Calculate Mean Squared Error
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 1.969697967225503e-05
```

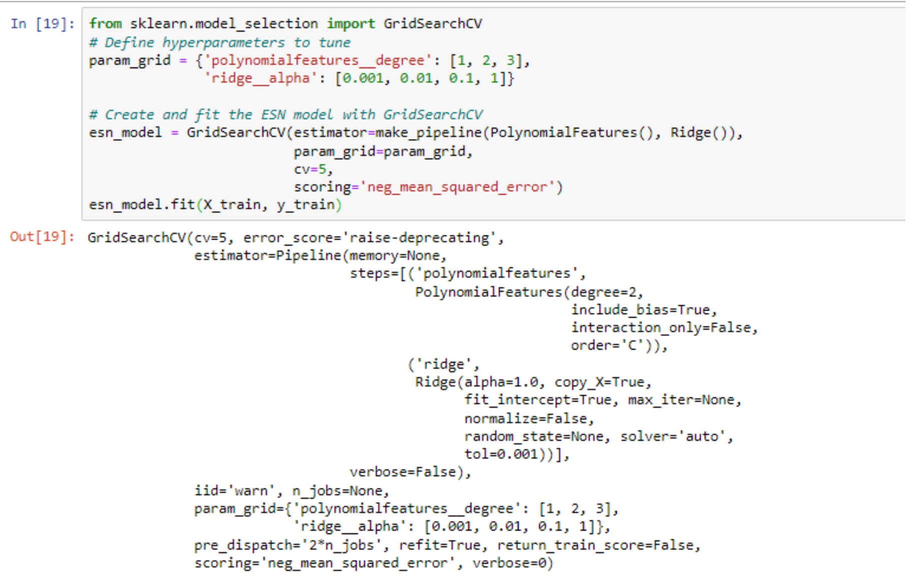
```
In [11]: # Implement a simple linear regression model
         from sklearn.linear_model import LinearRegression

         # Create and fit the linear regression model
         linear_model = LinearRegression()
         linear_model.fit(X_train, y_train)

         # Make predictions
         linear_y_pred = linear_model.predict(X_test)

         # Calculate Mean Squared Error
         linear_mse = mean_squared_error(y_test, linear_y_pred)
         print(f'Linear Regression Mean Squared Error: {linear_mse}')

         # Plot the actual vs predicted values for both models
         plt.figure(figsize=(12, 6))
         plt.plot(y_test, label='Actual Values')
         plt.plot(linear_y_pred, label='Linear Regression Predicted Values')
         plt.plot(y_pred, label='ESN Predicted Values')
         plt.xlabel('Time')
         plt.ylabel('Closing Price')
         plt.title('Actual vs Predicted Values')
         plt.legend()
         plt.grid()
         plt.show()
```

```
Linear Regression Mean Squared Error: 1.2775814267157641e-05
```



In [17]:
```
#More evaluation metrics

rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Echo State Network Mean Squared Error: {mse}')
print(f'Echo State Network Root Mean Squared Error: {rmse}')
print(f'Echo State Network Mean Absolute Error: {mae}')
print(f'Echo State Network R-squared: {r2}')


linear_rmse = np.sqrt(linear_mse)
linear_mae = mean_absolute_error(y_test, linear_y_pred)
linear_r2 = r2_score(y_test, linear_y_pred)
print(f'Linear Regression Mean Squared Error: {linear_mse}')
print(f'Linear Regression Root Mean Squared Error: {linear_rmse}')
print(f'Linear Regression Mean Absolute Error: {linear_mae}')
print(f'Linear Regression R-squared: {linear_r2}')
```

```
Echo State Network Mean Squared Error: 1.969697967225503e-05
Echo State Network Root Mean Squared Error: 0.004438127946809897
Echo State Network Mean Absolute Error: 0.002867230752880778
Echo State Network R-squared: 0.9898493932274303
Linear Regression Mean Squared Error: 1.2775814267157641e-05
Linear Regression Root Mean Squared Error: 0.003574327106905248
Linear Regression Mean Absolute Error: 0.001677179047622329
Linear Regression R-squared: 0.9934161344031861
```



In [19]:
```
from sklearn.model_selection import GridSearchCV
# Define hyperparameters to tune
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1, 1]}

# Create and fit the ESN model with GridSearchCV
esn_model = GridSearchCV(estimator=make_pipeline(PolynomialFeatures(), Ridge()),
                         param_grid=param_grid,
                         cv=5,
                         scoring='neg_mean_squared_error')
esn_model.fit(X_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('polynomialfeatures',
                                        PolynomialFeatures(degree=2,
                                                           include_bias=True,
                                                           interaction_only=False,
                                                           order='C')),
                                       ('ridge',
                                        Ridge(alpha=1.0, copy_X=True,
                                              fit_intercept=True, max_iter=None,
                                              normalize=False,
                                              random_state=None, solver='auto',
                                              tol=0.001))],
                                verbose=False),
             iid='warn', n_jobs=None,
             param_grid={'polynomialfeatures__degree': [1, 2, 3],
                         'ridge__alpha': [0.001, 0.01, 0.1, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

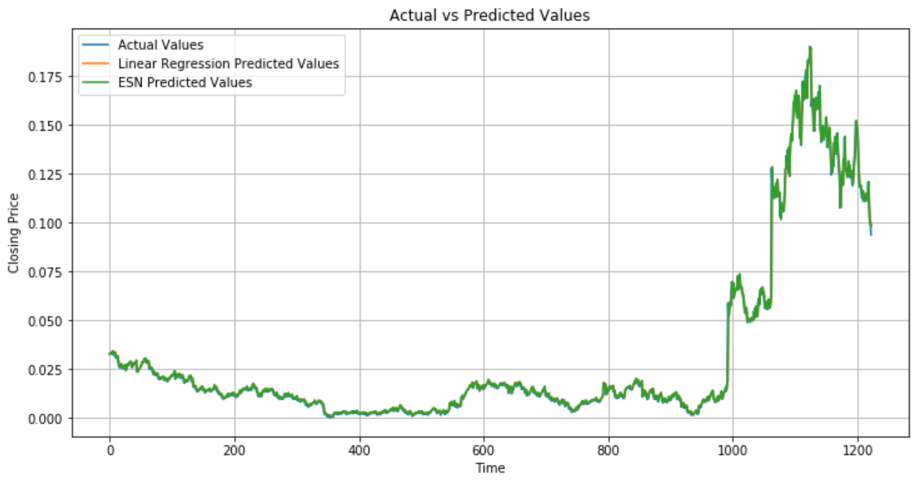
In [20]:

```python
# Predictions and Calculate Mean Squared Error
y_pred = esn_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Echo State Network Mean Squared Error: {mse}')
print(f'Echo State Network Root Mean Squared Error: {rmse}')
print(f'Echo State Network Mean Absolute Error: {mae}')
print(f'Echo State Network R-squared: {r2}')
print(f'Echo State Network Best Parameters: {esn_model.best_params_}')
```

```
Echo State Network Mean Squared Error: 1.2773423065311204e-05
Echo State Network Root Mean Squared Error: 0.0035739925944678736
Echo State Network Mean Absolute Error: 0.001677225909159315
Echo State Network R-squared: 0.9934173666809292
Echo State Network Best Parameters: {'polynomialfeatures__degree': 1, 'ridge__alpha': 0.001}
```

In [22]:

```python
# Plot the actual vs predicted values for both models
plt.figure(figsize=(12, 6))
plt.plot(y_test, label='Actual Values')
plt.plot(linear_y_pred, label='Linear Regression Predicted Values')
plt.plot(y_pred, label='ESN Predicted Values')
plt.xlabel('Time')
plt.ylabel('Closing Price')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.grid()
plt.show()
```


Actual vs Predicted Values
Actual Values
Linear Regression Predicted Values
ESN Predicted Values
Closing Price
Time