Deep Hedging: Enhancing Options Hedging Strategies with Deep Learning Models

Ajeetkumar Rai
Department of Applied Mathematics and Statistics
Stony Brook University
ajeetkumar.rai@stonybrook.edu

Abstract

Human expertise plays a vital role in options trading, particularly in delta hedging. While doing delta hedging, expert traders are required who can trade successfully. The Black-Scholes Method has always been a benchmark model for traders. However, due to some limitations, this method is not sufficient. Therefore, in this decade our researchers started using deep learning techniques which proved useful in reducing human effort and what we call Deep Hedging today.

1 Introduction

In quantitative finance, one of the famous Black-Scholes models useful for traders in their estimations of options prices. But there are some flaws in this model, like the model works only in a perfect market and we know that the market is generally not perfect, which is just an assumption. Implied volatility, which tells us about the movement of the market, leaves a ripple effect in the underlying asset, and the movement of the underlying asset is passed on to the options price. Implied volatility has been ignored in this model. For these reasons, Black-Scholes models cannot be used fully, but yes, they can be used for approximation and as a benchmark. In recent years we started using deep hedging, in which we can predict the delta using current stage delta and market data.

2 Literature Review

The concept of deep hedging was first introduced by Buehler, Hans and Gonon, Lukas and Teichmann, Josef and Wood, Ben, on February 8, 2018. In their research, they show how Greek hedging, and statistical hedging work, and their drawbacks. Later they introduce deep hedging and their implementation. In this paper, researchers use Reinforcement learning and LSTM for training the model. They proposed the idea that by using market data and the current

state of the delta it is possible to predict the next stage delta and iterating this and rebalancing minimizing hedging error is possible. Later other researchers worked on other deep learning models such as feed-forward neural networks, variants of sequential models, deep learning-based clustering models, and so on.

3 Pre-requisites

Derivative: A financial derivative is a security whose value is tied to the value of another security, called the underlying.

Derivative Securities: A financial contract is a derivative security, or a contingent claim if its value at expiration is exactly determined by the market price of one or more cash instruments called the underlying.

Forward and Futures Contracts: A forward or futures contract is a security that obligates the holder to buy a certain underlying asset at a certain price at a certain time in the future.

Options: An option is a security that gives its owner, the holder, the right (not the obligation) to buy or sell an underlying asset at a certain price on a certain date or dates.

European Options: A European-style option on a security is the right (not obligation) to buy in a call option or to sell in a put option the security at strike price K at expiry T.

Call & Put:

1. Consider a case in which a trader is long, has bought a call with strike $K = 100$, then $F(T) = \max[S(T) - K, 0]$.

2. Consider a case in which a trader is long or has bought a put with strike $K = 100$, then $F(T) = \max[K - S(T), 0]$.

3. Consider a case in which a trader is short, has sold or written, a call with strike $K = 100$, then $F(T) = -\max[S(T) - K, 0]$.

4. Consider a case in which a trader is long or has sold or written, a put with strike $K = 100$, then $F(T) = -\max[K - S(T), 0]$.

Notations: T = expiration date, the expiry, of the derivative; t = time index (typically at some time $t < T$); $S(t)$ = the price of the underlying at time t; $F(S(t), t)$ = the derivative price at time t given $S(t)$; $F(t) = F(S(t), t)$, when the context is clear.

4 Black-Scholes Model

The Black-Scholes method is used for estimating the theoretical price of an option. However, due to some assumptions, the Black-Scholes method is not applicable in an imperfect market. Black-Scholes expects some market information such as,

- C: Call option price
- S: Current stock price
- K: Strike price of the option
- r: Risk-free interest rate (a number between 0 and 1)
- σ: Volatility of the stock return (between 0 and 1)
- t: Time to option maturity (in years)
- N: Normal cumulative distribution function

Using these values, one can calculate the theoretical price of an option.

The Black-Scholes formula for calculating the theoretical price of an option is given by:

$$C(S,t) = N(d_1)S - N(d_2)Ke^{-rt}$$

Where:

$$d_1 = \frac{1}{\sigma\sqrt{t}}\left(\ln\left(\frac{S}{K}\right) + t\left(r + \frac{\sigma^2}{2}\right)\right)$$

$$d_2 = \frac{1}{\sigma\sqrt{t}}\left(\ln\left(\frac{S}{K}\right) + t\left(r - \frac{\sigma^2}{2}\right)\right)$$

And $N(x)$ is the cumulative distribution function of the standard normal distribution, given by:

$$N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}z^2}dz$$

5 Delta

Although in literature five different Greeks introduced such as Delta, Theta, Gamma, Vega, and Rho. Our interest is with delta which is used to measure the rate of change of the option price concerning changes in the underlying asset price. We called delta-neutral when delta remains zero for any t where t belongs to $[0, T]$. That means we are trying to minimize the differences between underlying assets and their options price. From equation (1) we can obtain delta as follows.

$$\Delta_t = (t, S_t) = N(d_1)$$

6 Problem Statement

Our objective is to find a delta-neutral state for given t, t belongs to $[0, T]$, such that the relative difference of prices between underlying assets and options should be minimal. In order to find the delta-neutral state we are utilizing deep learning models considering market data and current delta state t is available such that we want to predict the next delta at time $t + 1$.

6.1 Methodology

We implemented a feed-forward neural network, a feed-forward network using SVR, and a sequential model recurrent neural network. Metrics such as Mean, Standard Deviation, Skewness, Kurtosis, Percentile 5, Percentile 95, VaR (95%), and CVaR (95%) have been used for evaluation.

6.1.1 Algorithm

- Define the number of paths
- Initialize initial stock price (S), current delta, and stochastic integral.
- For each t, t belongs to $[0, T]$, iterate this
  - Generate noise (G) with respect to stock price.
  - Compute dt (t/T) and using volatility calculate the delta of the stock price.
  - Pass the stock price and initial delta in the neural network and get the output.
  - Using the output, initial stochastic integral, delta of stock price and S update stochastic integral.
  - Update stock price (S + delta of stock price)
- Calculate the payoff using the final stock price and target price
- Compute hedging error (theoretical Black-Scholes price + stochastic integral - Payoff).
- Calculate the Loss function MSE of hedging error.

7 Models

Although there are many models in deep learning, we can choose the model by understanding the data. If we have data that has both features and targets, then a feed-forward neural network (FNN) is most appropriate. Such type of model works well for tabular data. If we have image data then the Convolution neural network (CNN) works very well. If we have time-series data then Recurrent neural networks (RNN) are appropriate. Apart from these, there are many variants that can be used as per the data.

7.1 Feed-Forward Neural Network (FNN)

:

In a feed-forward neural network (FNN), we initialize some weights and linear combinations of the feature matrix (X) and initialized weights along with the Bais term pass to the activation function. Activation functions like tanh,

linear, ReLu, sigmoid, etc. can be used according to the data. As per requirement, after passing through the first activation function, more of what we call a hidden layer can be added. Finally, there is the output layer which reveals the predictions of the model. This process is known as forward propagation. After that model compares predicted values with the actual values. The loss function is used for comparison and computing these values. Once the loss value is calculated, based on the values model updated their weights in all layers and this process is known as back-propagation. For back-propagation, we use optimizers such as ADAM, gradient descent, etc. This forward propagation and back-propagation continue until the defined parameters are found. For example, if a total number of iterations (n) is defined then the model will update their weights n times. This process is known as hyperparameter tuning, and a few of them are the number of hidden layers, nodes, activation functions, output, optimizers, batch size, etc. These parameters can affect the model performance significantly.

1. **Input Layer**: $\mathbf{x}$ represents the input vector.
2. **Hidden Layers**: $\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$ where $\mathbf{h}^{(l)}$ is the output of layer l, σ is the activation function, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector of layer l.
3. **Output Layer**: $\mathbf{y} = \sigma(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)})$ where $\mathbf{y}$ is the output vector, L is the number of layers, and $\mathbf{W}^{(L)}$ and $\mathbf{b}^{(L)}$ are the weight matrix and bias vector of the output layer.

The key equations here are the weighted sum $\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ and the activation function σ, which introduces non-linearity into the network. Through training, the weights and biases are adjusted to minimize the difference between the predicted output $\mathbf{y}$ and the true output.

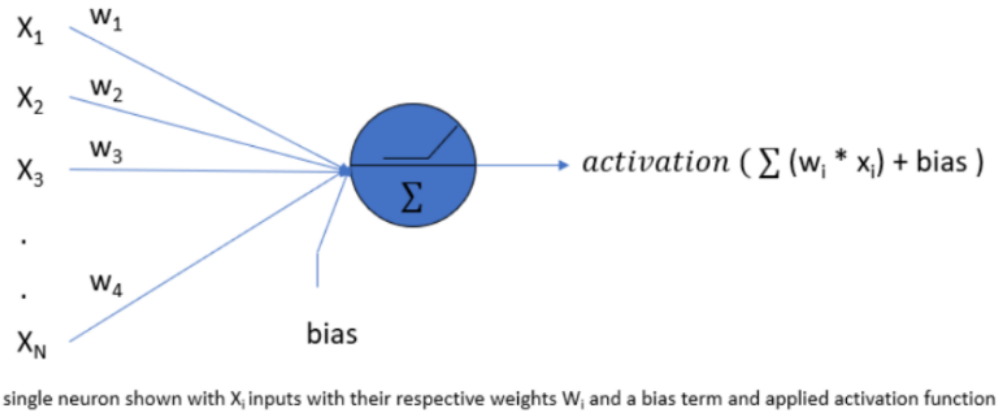

Figure 1: Neural Network

7.2 SVR

We are solving the regression problem and we saw that feed-forward neural networks can be implemented for regression problems, but apart from this, there are other models that can prove to be useful. Not only neural networks but kernel-based models, tree-based models, and probabilistic models can be used to solve regression problems. We will look at kernel-based models here. Since kernel-based models can be used for both regression and classification. When we solve classification it's Support Vector Machine (SVM) and in the case of regression, it is Support Vector Regression (SVR). There are two variants of SVR for discrete cases namely ε-SVR and ν-SVR.

7.3 Regression

Let Y be the regressant (target variable) and $X = (X_1, \ldots, X_d)$ be the regressors (factors).

$$Zf = Y - f(X) - C = residual$$

$$\overline{Zf} = Y - f(X) = residualw/ointercept$$

Generalized Regression

$$\min_{f \in F, C} E(Zf) = \min_{f \in F} \min_{C} E(\overline{Zf} - C) \quad \{errorprojection\}$$

$$= \min_{f \in F} D(\overline{Zf}) \quad and \quad C \in S(\overline{Zf}) = \arg\min_{C} E(\overline{Zf} - C)$$

7.4 ϵ-SVR: Discrete Case

Given: training data $X_\ell = (x_i, y_i)_{i=1}$, where $x_i \in R^n$, $y_i \in R$, $y = (y_1, \ldots, y_\ell)$.

Find: hyperplane $f_{w,b}(x) = w^T x + b$, $(w, b) \in R^{n+1}$ that optimally fits the data.

Let $z = z(w, b)$ be a random variable taking with equal probabilities $p = \frac{1}{\ell}$ the components $(y_i - f_{w,b}(x_i))_{i=1}^{\ell}$.

Given $C > 0$, $\nu \in (0, 1]$.

Denote $L(\xi) = \max\{0, |\xi| - \varepsilon\} = [|\xi| - \varepsilon]^+ =$ "-insensitive loss function".

The ϵ-SVR (Vapnik, 1995) objective is:

$$\min_{w,b} E[L(z(w, b))] + \frac{1}{2C}\|w\|_2^2$$

Let $\nu = 1 - \alpha$. Consider:

$$\min_{\varepsilon}(1 - \alpha)\left(\varepsilon + \frac{1}{1 - \alpha} E[|z(w, b)| - \varepsilon]^+\right)$$

$= (1-\alpha)\overline{q}_\alpha(|z(w, b)|) = \langle\langle z(w, b))\rangle\rangle_\alpha = CVaRnorm[Bertsimasetal., 2014; Mafusalov\&Uryasev, 2016]$

7.5 ν-SVR: Discrete Case

The ν-SVR [Schölkopf et al., 2000] objective is:

$$\min_{w,b,\varepsilon} \nu \left(\varepsilon + \frac{1}{\nu} E[L(z(w,b))] \right) + \frac{1}{2C} \|w\|_2^2$$

The -SVR [Malandii & Uryasev, 2022] objective is:

$$\min_{w,b} \langle \langle z(w,b)) \rangle \rangle_\alpha + \frac{1}{2C} \|w\|_2^2$$

The -SVR [Takeda et al., 2010] objective is:

$$\min_{w,b} CVaR_\alpha \left(\frac{y - C}{f(w,b)} \|w\| \right)$$

8 Results

The below PnL plots show results obtained from the feed-forward neural network (FFNN), ϵ-SVR, and ν-SVR. In the left-hand side plot, the distribution of hedging error is shown, and in the right-hand side plot, we can see the quantiles as well. These models are trained with 64 nodes in 6 hidden layers with activation functions LeakyReLU and tanh, and ADAM optimizers were used. In a Feed-Forward Neural Network (FNN), the loss function MSE was used, whereas in ϵ-SVR and ν-SVR, separate loss functions were used with additional parameters (0.1) and (0.4). This plot can be improved with a larger number of iterations, paths, and other parameters with the cost of computation time.

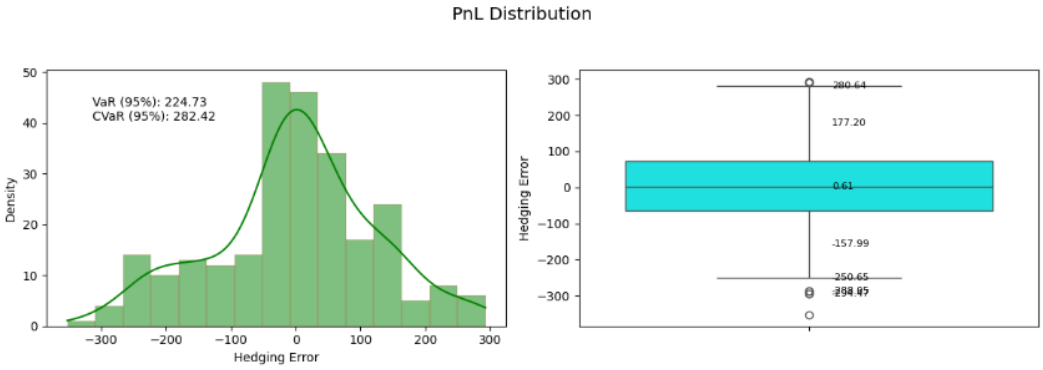

Figure 2: PnL for Feed-Forward Neural Network(FNN)

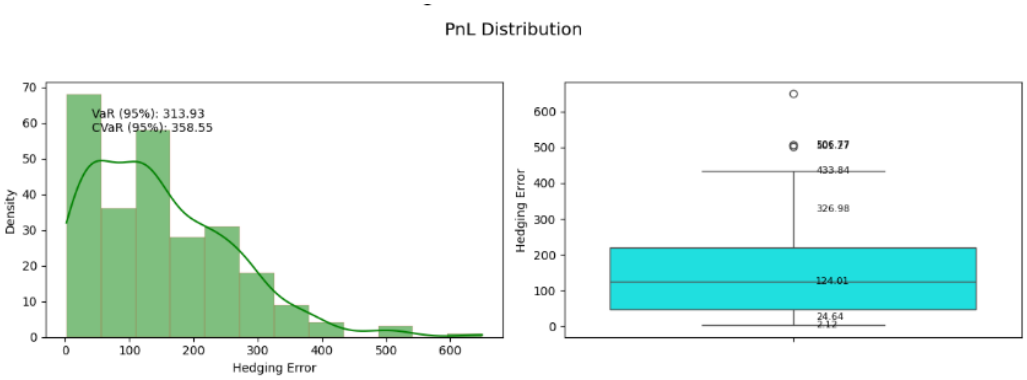

Figure 3: PnL for ϵ-SVR

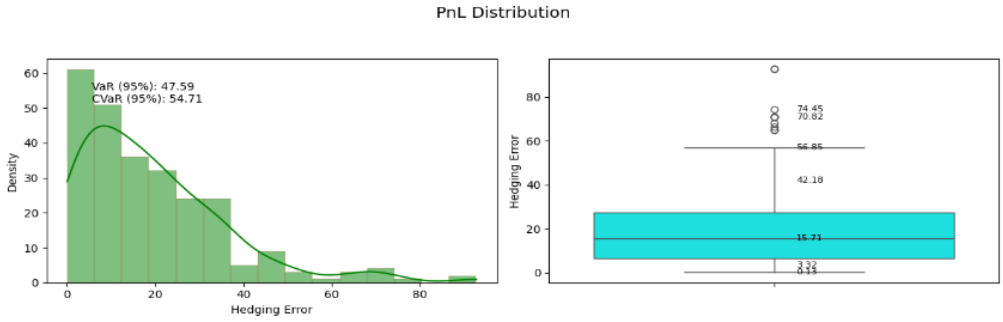

Figure 4: PnL for ν-SVR

9 Metrics and Performance

In the table below we can see how our model is performing. We have chosen mean, kurtosis, skewness, and percentiles as metrics but we are more interested in VaR and CVaR which look very good for -SVR

Table 1: Metrics for Neural Network (FNN), ϵ-SVR, ν-SVR

Metrics	Neural Network (FNN)	ϵ-SVR	ϵ-SVR
Mean	-2.339	138.315	19.539
Standard Deviation	138.051	106.767	17.051
Skewness	0.049	1.182	1.426
Kurtosis	0.099	1.952	2.255
Percentile(5%)	-228.037	10.619	1.189
Percentile(95%)	227.848	330.813	54.555
VaR(95%)	224.734	313.932	47.587
CVaR(95%)	282.420	358.545	54.712

The plot given below shows the predictions at different times t where t belongs to $[0, T]$. We have rebalanced delta $t = 100$ times, and it can be observed that the Black-Scholes model looks very stable due to its assumptions, while our model is learning after each time t.

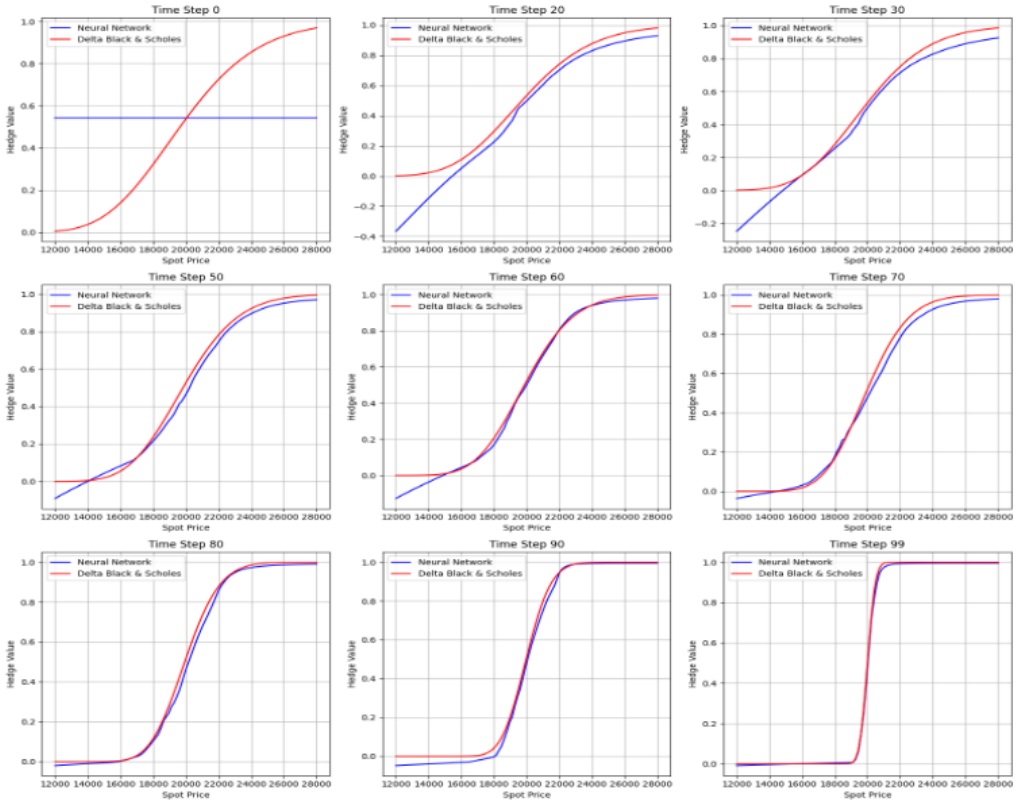

Figure 5: Delta at time t for FNN

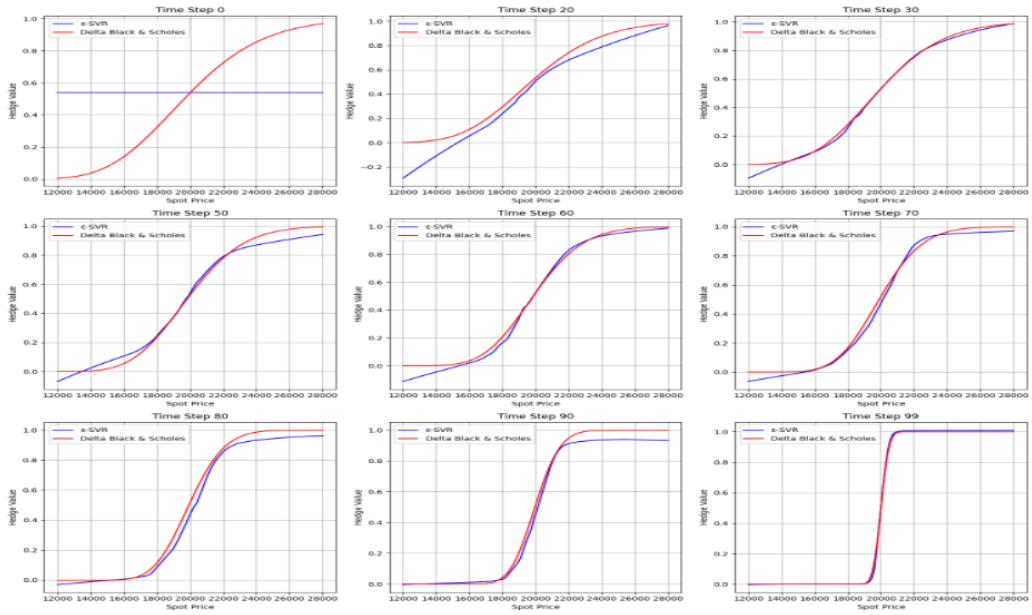

Figure 6: Delta at time t for ϵ-SVR

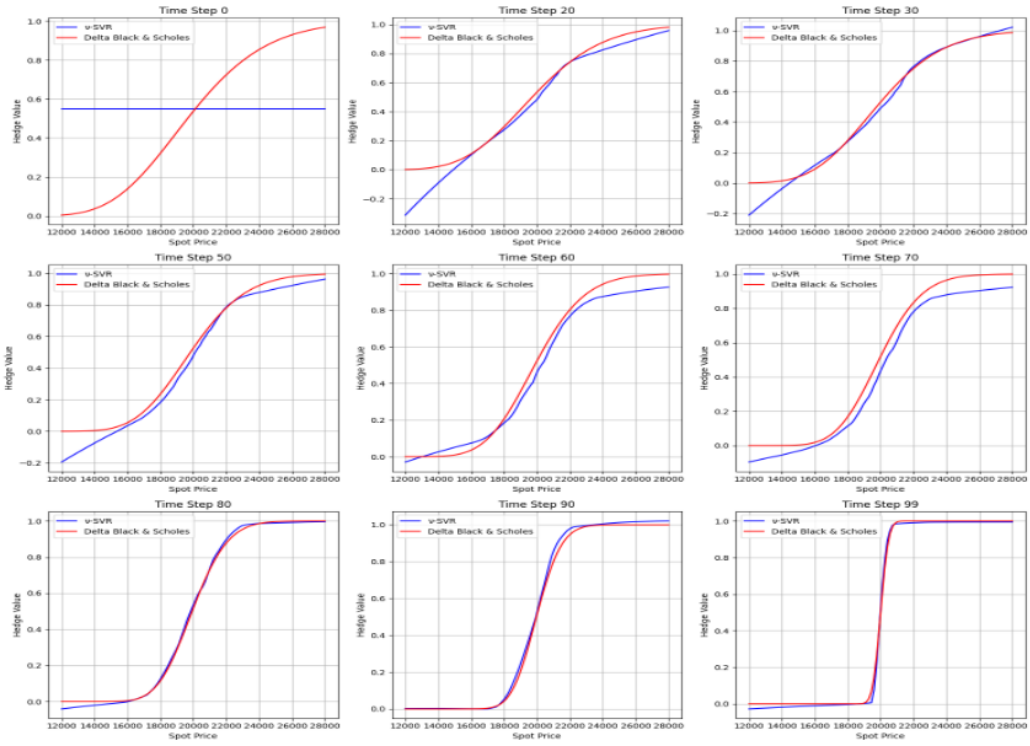

Figure 7: Delta at time t for ν-SVR

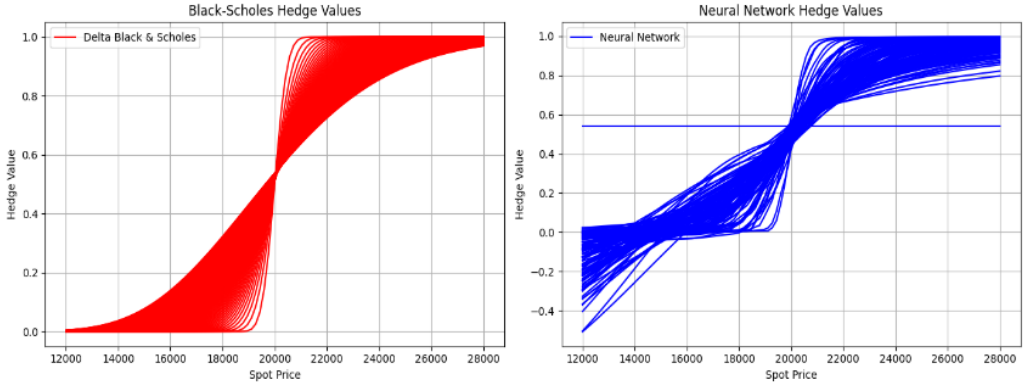

Figure 8: Overall Delta for FNN

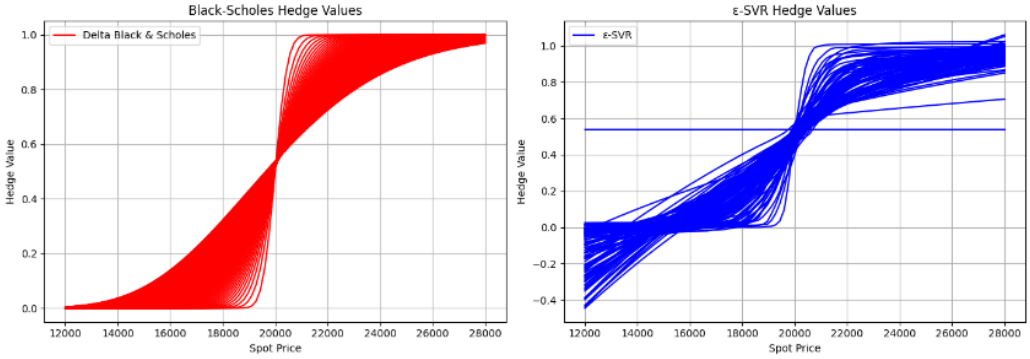

Figure 9: Overall Delta for ϵ-SVR

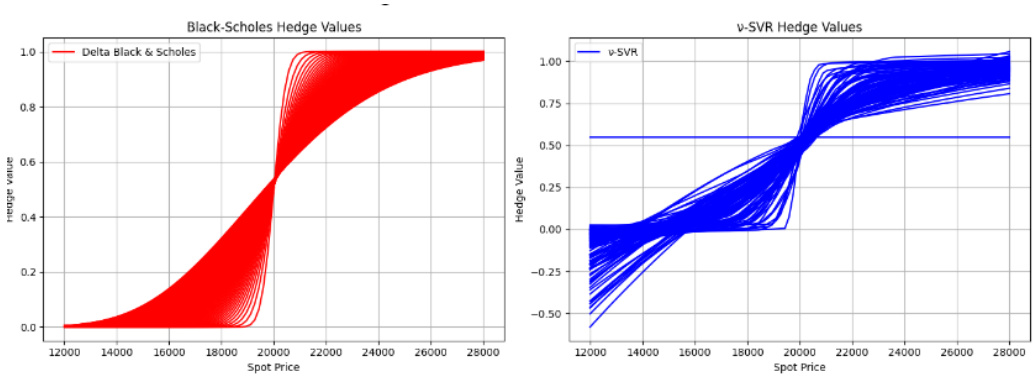

Figure 10: Overall Delta for ν-SVR

10 Conclusion

In conclusion, our investigation indicates that ν-SVR outperforms other models in delta simulation. We opted not to consider VaR and CVaR metrics for SVR due to the positively skewed distribution resulting from the absolute value of residuals in the loss function. Our results demonstrate that given adequate time for model execution, deep learning models can be highly effective, enabling the attainment of delta-neutral states. However, further research in this nascent field is warranted, as it holds promise for even greater advancements in hedging strategies.

References

1. Buehler, Hans and Gonon, Lukas and Teichmann, Josef and Wood, Ben, Deep Hedging (February 8, 2018).

2. https://naomiehalioua.medium.com/deep-hedging-how-to-understand\
-one-of-the-most-difficult-ai-defi-concept-in-less-than-6mn\
-481c8222c1ee

3. https://stackoverflow.com/questions/54414392/convert-sklearn\
-svm-svc-classifier-to-keras-implementation

4. https://www.tensorflow.org/tutorials/customization/custom_training\
_walkthrough

5. https://machinelearningmastery.com/time-series-prediction-\lstm-recurrent-neural-netwo

6. https://github.com/keras-team/keras/issues/2588