DIPLOMARBEIT

Deep Mean-Variance Hedging using LSTM RNNs

ausgeführt am

Institut für
Stochastik und Wirtschaftsmathematik
TU Wien

unter der Anleitung von

Univ.Prof. Dipl.-Math. Dr.rer.nat. Thorsten Rheinländer

durch

Philipp Ladislaus Wilhelm Knoll, BSc
Matrikelnummer: 01617936

Wien, am 27. Jänner 2023

Autor	Betreuer

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Suche nach Hedging-Strategien für Derivate ist ein zentrales Problem der Finanzmathematik und sowohl für institutionelle als auch für private Investoren von großem Interesse. Herkömmliche Hedging-Verfahren erfordern in der Regel die Spezifikation und Kalibrierung eines Asset-Preis-Modells, bevor eine Hedging-Strategie berechnet werden kann. Als Alternative zu diesen Ansätzen haben Buehler et al. das Deep Hedging Framework [3] basierend auf neuronalen Feed-Forward-Netzwerken eingeführt. Die vorliegende Arbeit adaptiert dieses Framework in zwei Bereichen. Erstens ist das Netzwerk, das zur Bestimmung der Hedging-Strategie verwendet wird, nicht mehr eine Folge verbundener neuronaler Feed-Forward-Netzwerke. Stattdessen wird eine echte rekurrente Topologie verwendet, die aus hierarchisch angeordneten long short-term memory (LSTM) Zellen besteht. Zweitens wird das Hedging-Optimierungsproblem als ein Mean-Variance-Hedging-Problem formuliert, bei dem das Ziel darin besteht, den erwarteten quadratischen Hedging-Fehler unter einem äquivalenten Martingal-Maß zu minimieren. Dieser Ansatz kann auch Marktfriktionen in Form von Transaktionskosten und Handelsrestriktionen in das Optimierungsproblem einbeziehen, d.h. Phänomene der realen Welt, die in traditionellen Hedging-Ansätzen normalerweise nicht berücksichtigt werden. Eine Implementierung des LSTM-basierten Ansatzes wird in numerischen Experimenten unter vier verschiedenen Rahmenbedingungen und zwei verschiedenen Klassen von Asset-Preis-Modellen illustriert und bewertet, wobei das neuronale Netzwerk basierend auf dem Loss bei Out-of-Sample-Testdaten ähnliche oder bessere Performance erzielt als die Benchmark-Strategien.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Finding hedging strategies for derivatives is a central problem of mathematical finance and of great interest for both institutional and retail investors. Traditional hedging techniques usually require specification and calibration of an asset price model prior to calculating a hedging strategy. As an alternative to these approaches Buehler et al. introduced the Deep Hedging framework [3] based on feed-forward neural networks. This thesis adapts this framework in two areas. First, the network used to determine the hedging strategy is no longer a sequence of connected feed-forward neural networks. Instead, a truly recurrent topology consisting of hierarchically organized long short-term memory (LSTM) cells is used. Secondly, the hedging optimization problem is formulated as a mean-variance hedging problem, where the aim is to minimize the expected squared hedging error under an equivalent martingale measure. This approach can also incorporate market frictions in the form of transaction costs and trading constraints into the optimization problem, i.e., real-world phenomena which are usually not considered in traditional hedging approaches. An implementation of the LSTM-based approach is illustrated and evaluated in numerical experiments under four different settings and two different classes of asset price models, where the neural network performs similarly or better than the benchmark strategies, based on the loss on out-of-sample test data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 27. Jänner 2023

Philipp Ladislaus Wilhelm
Knoll, BSc

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

1. Introduction

Option pricing and finding optimal hedging strategies is a central problem in financial mathematics. In general, this problem can be stated as

$$\inf_{c,\vartheta} \rho(H - c - (\vartheta \cdot S)_T)$$

for a given claim H, tradable asset S and risk measure ρ. The value $c \in \mathbb{R}$ denotes the amount of initial cash injection and $\vartheta \in L(S)$ a self-financing strategy. The space $L(S)$ consists of all integrands for which the stochastic integral w.r.t S is well defined.
Solving this optimization problem is usually not straightforward and computationally intensive. Additionally, the solutions depend on the risk measure ρ as well as on the underlying asset price model, necessitating previous model choice and calibration of model parameters. As an alternative to theses techniques, Buehler et al. introduced the"Deep Hedging" framework in [3]. This approach uses deep learning methodology in the form of feed-forward neural networks to numerically solve the hedging optimization problem for convex risk measures ρ. The neural network uses a "semi-recurrent" topology with separate feed-forward networks for each time step and the output for each time step being part of the input vector for the next time step. The framework also allows for market frictions in the form of transaction costs and trading constraints, real-world features that are often not accounted for in traditional hedging approaches.

This thesis adapts this framework in two fundamental ways:
First, the model topology in this thesis is "fully recurrent", using gated recurrent neural networks in the form of hierarchically organized long short-term memory (LSTM). This choice of topology is intuitive, as recurrent neural networks are specifically designed and specialized to process sequence data such as time series.
Second, instead of the hedging problem being formulated for convex risk measures, the optimization problem is stated as a mean-variance hedging problem which uses a quadratic criterion. Only mean variance hedging under the risk-neutral measure is considered.

The "LSTM Hedging" approach is illustrated and analysed in numerical experiments for two different classes of asset price models: an exponential Lévy model (Normal Inverse Gaussian Model [2]) and a stochastic volatility model (4/2 stochastic volatility model [9]).

Regarding the structure of the thesis, Chapter 2 serves as an introduction into deep learning and recurrent neural networks. Chapter 3 first treats the mathematical prerequisites (3.1) to later formulate the mean-variance hedging optimization problem (3.2) and its explicit solution in the case of exponential Lévy models (3.2.1). Next, Chapter 4 combines the definitions and results from the previous chapters, discusses the discrete-time mathematical setting including trading restrictions and transaction costs (4.1) and illustrates how LSTM RNNs can be used to find close-to-optimal mean-variance hedging strategies (4.2). Finally, this approach is examined and evaluated within the numerical experiments in Chapter 5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2. Deep Learning and RNNs

This chapter introduces the models and algorithms used in later chapters, and discusses the mechanisms used for training these models. First, feedforward neural networks are introduced, followed by an explanation of the training process using gradient descent, back-propagation and the Adam optimization algorithm. In the next step, recurrent neural networks and their properties are discussed, before this chapter is concluded by introducing the LSTM-Cell as a modern RNN-Architecture.

2.1. Feedforward Neural Networks

Feedforward neural networks, also referred to as multilayer perceptrons, are among the most simple and straightforward deep learning models. They are used to approximate some function f^* which is usually not explicitly given. Depending on the problem at hand, this could be a classifier mapping inputs to a category or class, or, in the case of a regression problem, a function with real-valued outputs. This specific type of neural network is called "feedforward", because the values are propagated forward from input to output without any kind of feedback or recurrence.
A feedforward neural network can be represented as a composition of multiple functions $F_l : \mathbb{R}^n \to \mathbb{R}^m$, which are commonly referred to as **layers** The outputs of each layer, with exception of the last one (the so-called output layer), are passed as inputs to the next layers. All layers except the input and output layers are referred to as **hidden layers**. The computations done in each layer consist of an affine transformation of the input values, followed by the application of a non-linear activation function. The purpose of this activation function is to introduce non-linearity into the model. This is done ,because only performing affine transformations between layers would ultimately lead to the model output being an affine transformation of the model inputs, which would limit the neural network's ability to approximate arbitrary functions f^*. [8]
A possible layout for a feedforward neural network with 3 input nodes, two hidden layers with 4 nodes each, and a 3-node output layer is shown in Figure 2.1. All this information can be combined in the following mathematical definition of a neural network [3]:

Definition 1. *Let $L, N_0, N_1, ..., N_L \in \mathbb{N}$, $\sigma : \mathbb{R} \to \mathbb{R}$ and for any $l = 1, ..., L$ let $W_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ denote an affine function. A function $F : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as*

$$F(x) = W_L \circ F_{L-1} \circ \cdots \circ F_1 \text{ with } F_l = \sigma \circ W_l \text{ for } l = 1, \ldots, L-1$$

is called a feedforward neural network. The activation function σ is applied component-wise. L denotes the number of layers, $N_1, ..., N_{L-1}$ denote the dimensions of the hidden layers and N_0, N_1 of the input and output layers, respectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

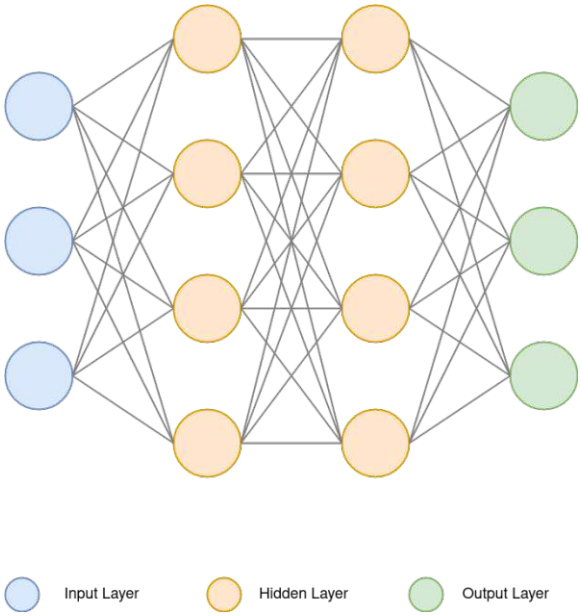

Figure 2.1.: Example architecture of a feedforward neural network

The affine functions W_l are given as $W_l(x) = A^l x + b^l$ for some $A^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^l \in \mathbb{R}^{N_l}$. The number A^l_{ij} is interpreted as the weight of the edge connecting the node i of layer $l-1$ to node j of layer l.
The set of all neural networks mapping from $\mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ with activation function σ is denoted by $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$
Feedforward neural networks are appropriate tools for function approximation, due to the introduction of non-linearity via applications of the activation function and the multi-layer architecture. The following theorem ([13],[3]) shows the exact approximation properties that these networks offer.

Theorem 1. *(Universal Approximation) Suppose σ is bounded and non-constant. The following statements hold:*

- *For any finite measure μ on $(\mathbb{R}^{d_0}, \mathcal{B}(\mathbb{R}^{d_0}))$ and $1 \leq p < \infty$, the set $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$ is dense in $L^p(\mathbb{R}^{d_0}, \mu)$*

- *If in addition $\sigma \in C(\mathbb{R})$, then $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$ is dense in $C(\mathbb{R}^{d_0})$ for the topology of uniform convergence on compact sets.*

This theorem's results only concern the set $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$. However, as any output component within a neural network in $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ can be thought of as a neural network from $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$, the results generalize to neural networks with higher-dimensional output.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.1.1. Training feedforward neural networks

Training a neural network refers to the process of repeatedly updating the network parameters (weights and biases) to achieve a "better" approximation of the target function f^*. The goal is to find model parameters that locally minimize the cost function associated with the problem at hand. The cost function is chosen based on the specific problem formulation and task. Popular choices include *mean-squared-error* for regression and *cross entropy* for classification tasks.
While, from a mathematical standpoint, aiming to find a global minimum of the cost function might seem reasonable, in a machine learning setting, global minima often lead to *over-fitting*, meaning a model performs well on training data, but comparatively poorly on unseen testing data. This, in combination with the fact that for non-convex cost functions the predominantly used algorithms have neither a guarantee of convergence towards a global minimum nor a way to determine if an achieved minimum is local or global, means that the goal of finding a local minimum is not only more realistic but also more desirable for prediction.
In the case of neural networks, cost function minimization is usually achieved through some version of *gradient descent*, while the updates of the model parameters are calculated using *back-propagation*.

Gradient descent

Gradient descent is a numerical method to find a (local) minimum of a real valued function $f : \mathbb{R}^n \to \mathbb{R}$. The gradient at a point $x_0 \in \mathbb{R}^n$ indicates the direction of maximum rate of increase. Consequently, if $\nabla f(x_0)$ is the direction of maximum rate of increase, the negative gradient $-\nabla f(x_0)$ points in the direction of maximum decrease of f at x_0. Assuming a starting point $x^{(0)}$, consider the point $x^{(0)} - \alpha \nabla f(x^{(0)})$. Then, Taylor's theorem implies

$$f(x^{(0)} - \alpha \nabla f(x^{(0)})) = f(x^{(0)}) - \alpha ||\nabla f(x^{(0)})||^2 + \mathcal{O}(\alpha)$$

Therefore, if α is sufficiently small and $\nabla f(x^{(0)}) \neq 0$ this leads to

$$f(x^{(0)} - \alpha \nabla f(x^{(0)})) \leq f(x^{(0)})$$

meaning that the point $x^{(0)} - \alpha \nabla f(x^{(0)})$ is an improvement over $x^{(0)}$ when trying to find a (local) minimum of f.
This leads to the following minimization algorithm [4]:

1. Choose a starting point $x^{(0)} \in \mathbb{R}^n$
2. Calculate the step size α_k for $(k+1)$-th iteration
3. Given the result $x^{(k)}$ of the k-th iteration, calculate $x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x^{(k)})$
4. Repeat steps 2 and 3 until some kind of stopping criterion is reached

The scalar value α_k is referred to as *step size* in this context, while it is usually called *learning rate* when talking about neural network training. This value is allowed to be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

non-constant, as many improved versions of this basic gradient descent algorithm adapt the step size for each step to avoid numerical problems and "bad" local minima.

In machine and deep learning applications, the cost function often decomposes into a sum over the training samples of a per-sample loss function, for example:

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}_{data}} L(x, y, \theta) = \frac{1}{N}\sum_{i=1}^{N} L(x^{(i)}, y^{(i)}, \theta)$$

with the per-sample loss function L, input data x, target data y , number of samples N and θ denoting the vector of trainable model parameters. The gradient to be calculated is the gradient w.r.t θ:

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{N}\sum_{i=1}^{N} L(x^{(i)}, y^{(i)}, \theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

In this case, one gradient descent step would require looping over the entire dataset, which can be quite large with millions of samples in many cases. Instead, one can approximate the gradient by sampling a so-called *minibatch* $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ of size $m << N$ from the training data, calculating the gradient of the (normed) sum of losses for the minibatch

$$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

and using this gradient in the gradient descent update step. This way, the parameters are updated more often, but the updates only use a approximation of the true gradient. This algorithm is usually referred to as (minibatch) **stochastic gradient descent** (SGD) [8].

Back-propagation

In feedforward nerual networks, the inputs are propagated from the input layer to the first hidden layer, then to the the following hidden layers (if they exist), and eventually to the output layer. During training, this propagation continues from the output layer to the cost function $J(\theta)$. This process is called **forward-propagation**, as the results of each calculation are propagated forward through the network.
The **back-propagation** algorithm then propagates the cost information backwards through the network to calculate the gradients used to update the network parameters. Using the algorithm one can calculate the partial derivatives of the cost function J w.r.t. to each model parameter and adjust the parameters accordingly using some variation of gradient descent.
In order to understand this algorithm, one has to think of a neural network as a computational graph. In these graphs, each node represents a variable which can be either a scalar, vector, matrix or even a tensor. Operations are functions of one or more variables returning a single variable. If a variable results from applying an operation to one or more variables, a directed edge is drawn from these variables to the result of the operation.
The back-propagation algorithm is based on repeated application of the chain rule [8]:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Theorem 2. *Let $x \in \mathbb{R}^m, y \in \mathbb{R}^n$ and $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ be sufficiently differentiable functions. If $y = g(x)$ and $z = f(y)$ then*

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Rewritten in vector notation this reads

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

where $\frac{\partial y}{\partial x}$ denotes the Jacobi matrix of g.

Applying this theorem allows one to find algebraic expressions for the gradients. However, directly using this theorem to compute gradients may be infeasible for larger network architectures. Often, there are reoccurring expressions within the gradient calculation that can be stored instead of recomputed to speed up this process at the cost of higher memory usage.
To illustrate this, consider a computational graph describing the computation of a scalar $u^{(n)}$. The goal is to calculate the gradients with respect to the input nodes $u^{(i)}$, $i \in 1, ...n_i$. Assume that each output can be calculated one after the other, starting from $u^{(n_i+1)}$ up to $u^{(n)}$. The gradients can then be calculated using the chain rule

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}} \tag{2.1}$$

with $Pa(u^{(i)})$ denoting the set of parent nodes of $u^{(i)}$.
When calculating the gradients backwards from $\frac{\partial u^{(n)}}{\partial u^{(n)}} = 1$, each gradient should be saved and used in place of $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ in Equation 2.1 to avoid recalculation of reoccurring expressions and allow for a computational cost proportional to the number of edges in the graph.
Assuming a feedforward neural network as defined in Definition 1 with loss function L, the back-propagation steps for this network can now be derived. Let a^l denote the activation of the l-th layer, meaning the result of $A^l x + b^l$ without application of the activation function. and let h^l denote the post-activation layer outputs, meaning $h^l = \sigma(a^l)$. The example above then leads to the following algorithm:

- Assign $\nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$ to the variable g
- For $l = L, L-1, ..., 1$:
  - Convert gradient on the layer output to pre-activation:
    Assign $\nabla_{a^l} J = g \odot \sigma'(a^l)$ to variable g
  - Calculate gradients w.r.t weights and biases:
    $\nabla_{b^l} J = g$
    $\nabla_{A^l} J = g {h^{l-1}}^T$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

– Propagate gradients backwards to the next lower layer's outputs:
Assign $\nabla_{h^{l-1}} J = A^{l^T} g$ to the variable g

The gradients $\nabla_{b^l} J$ and $\nabla_{A^l} J$ can then be used immediately for the parameter update step, or used with the chosen gradient-based optimization method. The symbol $\odot$ once again denotes the component-wise product, which in this case stems from σ being a scalar function being applied component-wise. [8]

Adam Optimizer

Adam (Adaptive Moment Estimation) is an optimization algorithm first introduced in [15]. As it is one of the most used optimization algorithms in deep learning today, it will be the algorithm of choice for the models used in later chapters. Adam is a variation of stochastic gradient descent, using not a single learning rate, but a separate learning rate for each network parameter. The algorithm also uses exponential moving averages of the gradient (m_t) as well as the squared gradient (v_t) as estimates of the gradients first and second moments. This can be represented using the following equations:

$$\begin{aligned} g_t &= \nabla_\theta f(\theta) \\ m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_1) \cdot g_t \odot g_t \end{aligned}$$

where $\beta_1, \beta_2 \in [0, 1)$ denote the exponential decay rates, $f(\theta)$ the function to be optimized, $\odot$ the component-wise multiplication and t the current iteration step. The moving average vectors are initialized as zero vectors, meaning $m_0 = v_0 = \mathbf{0}$. This leads to these estimators being biased towards zero in early iterations, which is why the bias corrected estimators

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned}$$

are used instead. As $\beta_1, \beta_2 \in [0, 1)$, this bias correction diminishes with increasing number of iterations.
Given a step size $\alpha \in \mathbb{R}^+$ the model parameters are updated using the following update rule:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where $\epsilon > 0$ is a hyper parameter added in the denominator to prevent division by zero.

The parameters recommended by the authors of [15] are:

- $\alpha = 0.001$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\epsilon = 10^{-8}$

which are used by most implementations (note that TensorFlow uses $\epsilon = 10^{-7}$ per default [7]).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2. Recurrent Neural Networks

Recurrent neural networks (RNNs) are a family of neural networks that are specialized to process sequences. For the purposes of this thesis, *sequence* refers to a collection $x^{(1)}, \ldots, x^{(\tau)}$, where each $x^{(i)}$, $i \in 1, \ldots, \tau$ is a n-dimensional vector.
While using feedforward networks for sequences is of course possible, one quickly runs into problems for long-length input sequences as the high number of network parameters make this architecture infeasible. RNNs circumvent these problems by using *parameter-sharing*, meaning that some parts of the model use the same parameters across time steps. This not only reduces the number of parameters that need to be trained, but also enables RNNs to process variable length sequences, while feedforward neural networks are not able to generalize to input sizes not seen during training. Parameter-sharing is also useful for many typical RNN applications, as when processing sequences, one often wants the network to treat input information similarly, independent of the time index it occurs at. For example, in natural language processing, when trying to extract certain information from a sentence, one would want a network to recognize this information no matter the location within the sentence.
To further define recurrent neural networks, it is necessary to understand the concept of **unfolding a computational graph**. While feed-forward neural networks can simply be viewed as a function from input to output space, RNNs are more similar to dynamical systems. Consider a dynamical system driven by some external signal $x^{(t)}$

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \tag{2.2}$$

where $s^{(t)}$ is referred to as the state of the system, which contains information of the past signal values of x. For a finite number of time steps, the system's graph can be unfolded by repeatedly applying 2.2. For example

$$s^{(3)} = f(s^{(2)}, x^{(3)}; \theta) = f(f(s^{(1)}, x^{(2)}; \theta), x^{(3)}; \theta) \tag{2.3}$$

leads to an expression without recurrence that can be represented using acyclic graph.
Many recurrent neural networks use equation 2.2 to compute its *hidden units*, which are usually subsequently used to calculate the networks outputs. These hidden units, commonly denoted by $h^{(t)}$, function as a record of relevant information from past input values. As the hidden units have a fixed dimension, but the input sequence up to t can be arbitrarily long, they are inherently lossy.
The unfolded recurrence at time step t can be represented by a function $g^{(t)}$ that takes the sequence up to time t as input, but the unfolded structure allows this function to be factorized into repeated applications of f

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, \ldots, x^{(1)}) = f(h^{(t-1)}, x^{(t)}; \theta)$$

This way, the model is specified as transition from state to state and has a fixed input size in each step, not a sequence of arbitrary length. Additionally, the transition function f and the parameters θ are the same for each time step. Therefore, only the model f has to be learned and can easily be generalized to arbitrary sequence length, without having to learn a model $g^{(t)}$ for every sequence length.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

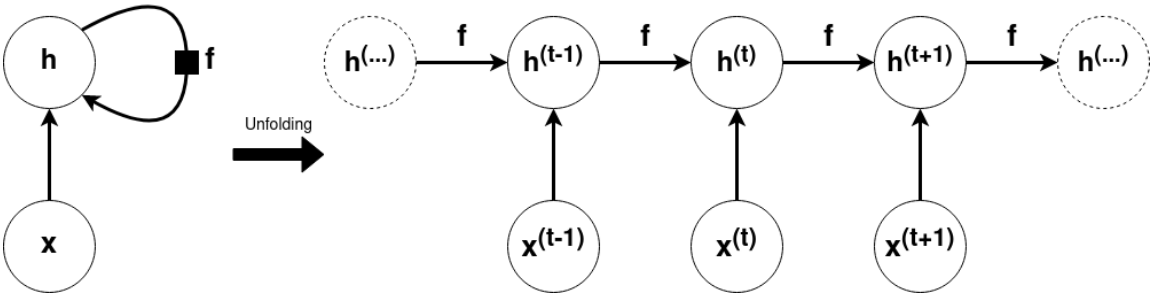

Figure 2.2.: Folded and unfolded computational graph

Figure 2.2 shows a simple, output-less RNN as a circuit diagram and then as an unfolded graph, in which every node is now associated with a time step. The black square in the left diagram denotes the application of f with a one time step delay. [8]

2.2.1. Forward propagation equations

Using graph unfolding and parameter sharing, one can define many different RNN architectures. A basic one will serve as an illustration on how to derive the forward propagation equations of RNNs, meaning the computations necessary to obtain a models output(s) given the inputs. Figure 2.3 shows the structure of this recurrent neural network, which is similar to the structure depicted in Figure 2.2. However, this network introduces the output nodes o, the loss function L, and the true labels or targets y.
The hidden units $h^{(t)}$ for a given time step t are computed as the sum of the matrix-vector product of W and the previous time step's hidden units, the matrix vector matrix-vector product of U and the input at the current time and a bias vector b, followed by the application of an activation function σ. The value at the output node $o^{(t)}$ for a given time step t is computed as the matrix vector product of V and the current time step's hidden units plus a bias vector c. After application of another activation function, the so-called *output function* σ^{out}, the values are passed to the loss function along with the target value $y^{(t)}$.
For $t \in 1, \ldots, \tau$, this can be summed up using the following equations:

$$
\begin{aligned}
a^{(t)} &= Wh^{(t-1)} + Ux^{(t)} + b \\
h^{(t)} &= \sigma(a^{(t)}) \\
o^{(t)} &= Vh^{(t)} + c \\
\hat{y}^{(t)} &= \sigma^{out}(o^{(t)})
\end{aligned}
$$

This means that the trainable model parameters are the matrices U, V, W and the vectors b, c. The loss function in this architecture is given as the sum of the per time step losses

$$L := \sum_t L^{(t)}$$

[8]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

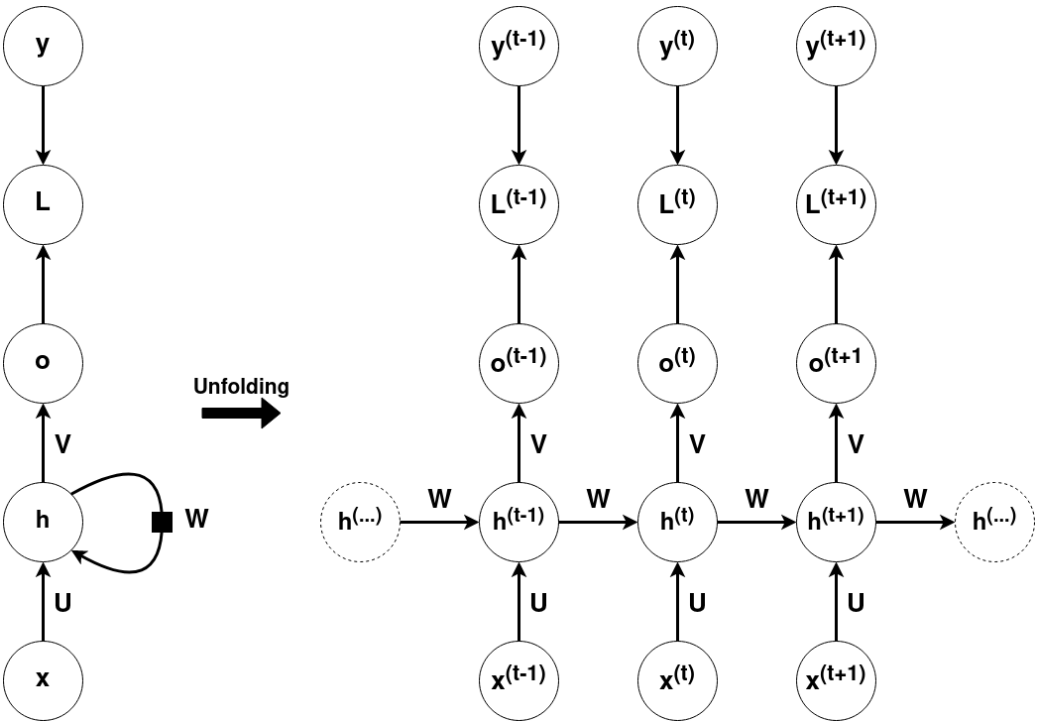

Figure 2.3.: Basic RNN Architecture

2.2.2. Training recurrent neural networks

Training recurrent neural networks follows the same principles as training feedforward neural networks. The minimization of the cost function is achieved by applying the back-propagation algorithm to the unfolded recurrent neural network.
However, due to the concept of parameter sharing used in RNNs, one has to resort to a small modification, because computing the gradient of the cost function with regard to a parameter in the typical sense would encompass the contributions of this parameter to the gradient over all possible edges in the computational graph, which is not compatible with the algorithm introduced in Section 2.1.1. Therefore, the parameters in each time step are treated as non-shared for the back-propagation step by introducing dummy variables that simple are copies of the original parameters. For example, instead of using the matrix W, the gradients are calculated as if every time step t used a matrix $W^{(t)}$, that contains the same values as W. The parameter gradient is then given as the sum of the parameter gradients for the dummy variables.

Using the same recurrent neural network as in Section 2.2.1, one can calculate the gradients w.r.t. the parameters recursively starting from the nodes preceding the total loss

$$\frac{\partial L}{\partial L^{(t)}} = 1$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The next step is to calculate the gradient w.r.t. the outputs $o^{(t)}$

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

Going to the next nodes, the hidden units $h^{(t)}$, one has to start iterating backwards from time step τ due to the connections between the hidden unit nodes. For time step τ the result is

$$\nabla h^{(\tau)} L = V^T \nabla_{o^{(\tau)}} L$$

Iterating backwards in time from $t = \tau - 1$ to $t = 1$ yields

$$\begin{aligned}\nabla_{h^{(t)}} L &= \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^T (\nabla h^{(t+1)} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T (\nabla_{o^{(t)}} L) \\ &= W^T (\nabla h^{(t+1)} L) J_\sigma^{h^{(t+1)}} + V^T (\nabla_{o^{(t)}} L)\end{aligned}$$

where $J_\sigma^{h^{(t+1)}}$ denotes the Jacobian matrix of the activation function σ associated with the hidden units at time step $t + 1$.
Now the parameter gradients can be computed using the dummy variables and gradients aggregation as mentioned above:

$$\begin{aligned}\nabla_c L &= \sum_t \left(\frac{\partial o^{(t)}}{\partial c}\right)^T (\nabla_{o^{(t)}} L) = \sum_t {}^T (\nabla_{o^{(t)}} L) \\ \nabla_b L &= \sum_t \left(\frac{\partial h^{(t)}}{\partial b^{(t)}}\right)^T (\nabla_{h^{(t)}} L) = \sum_t J_\sigma^{h^{(t)}} (\nabla_{h^{(t)}} L) \\ \nabla_V L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_V o_i^{(t)} = \sum_t (\nabla_{o^{(t)}} L) h^{(t)T} \\ \nabla_W L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{W^{(t)}} h_i^{(t)} \\ &= \sum_t J_\sigma^{h^{(t)}} (\nabla_{h^{(t)}} L) h^{(t-1)T} \\ \nabla_U L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{U^{(t)}} h_i^{(t)} \\ &= \sum_t J_\sigma^{h^{(t)}} (\nabla_{h^{(t)}} L) x^{(t)T}\end{aligned}$$

[8]

2.2.3. Deep Recurrent Networks

The recurrent neural network structure used in the previous section can be decomposed into three blocks of parameters and transformations:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

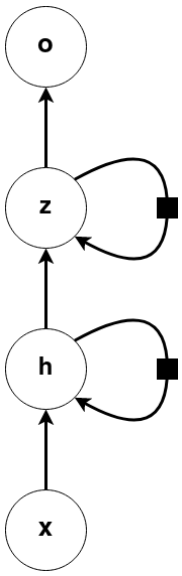

Figure 2.4.: Deep RNN architecture with two layers of hidden units

1. from input to hidden state
2. from previous hidden state to the next hidden state
3. from hidden state to output

Each of these corresponds to an affine transformation followed by an application of an activation function. This is often called a shallow transformation as it can be performed by a single layer neural network. For many applications, it may be useful to consider adding depth to any of these three blocks, as experimental evidence suggests that this may be required to approximate the target functions.
The models used in later chapters will use multiple groups of hidden units organized hierarchically, as depicted in Figure 2.4. Intuitively, the lower layers of hidden units serve as a kind of pre-processing layers, to allow the following layers to better capture the relevant information.
Moreover, it should also be mentioned that deeper architectures are more computationally intensive to train, as not only do more parameters have to be trained, but also the paths between variables become longer, leading to more intensive back-propagation steps. [8]

2.2.4. LSTM

Recurrent neural network suffer a problem called *vanishing and exploding gradients*. This stems from the fundamental design of these networks, as their structure and parameter sharing leads to repeated applications of the same functions.
To better illustrate this problem, consider the recurrence relation

$$h^{(t)} = W^T h^{(t-1)}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

which can be interpreted as a very basic recurrent neural network with linear activation function and no inputs. This recurrence can be simplified to

$$h^{(t)} = (W^t)^T h^{(0)}$$

If the matrix W admits an eigenvalue decomposition $W = Q\Lambda Q^T$ with orthogonal Q, this can again be rewritten as

$$h^{(t)} = Q\Lambda^t Q^T h^{(0)}$$

From this representation, it is clear that the eigenvalues with magnitude less than 1 decay to zero, while eigenvalues with magnitude greater than 1 explode. This leads to the components of $h^{(0)}$, which are not aligned with the eigenvector corresponding to the eigenvector with biggest magnitude, to be discarded eventually.[8]

To address this, many approaches have been introduced. One of them are the so called **gated RNNs**, their most prominent member being **long short-term memory (LSTM)**. The LSTM architecture introduces three so-called gates, i.e., the input, output and forget gates, that control the flow of information within the network and consist of trainable parameters. The LSTM RNN is defined by the following equations:

$$\tilde{c}^{(t)} = \tanh(Uh^{(t-1)} + Wx^{(t)} + b) \tag{2.4}$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \tag{2.5}$$

$$h^{(t)} = o^{(t)} \odot g(c^{(t)}) \tag{2.6}$$

where the function g is usually the hyperbolic tangent. $c^{(t)}$ is referred to as the memory-cell at t, $\tilde{c}^{(t)}$ as the candidate activation at t. Equation 2.4 shows, that the candidate activation is computed from the hidden unit-vector from the previous time step $h^{(t-1)}$ and the input vector $x^{(t)}$. To get the memory-cell in Equation 2.5, this is multiplied component-wise with the so called **input gate** $i^{(t)}$ and added to the component-wise product of the previous memory-cell and the **forget gate** $f^{(t)}$. The hidden units, which function as the LSTM cell outputs are then computed in Equation 2.6 as the component-wise product of the **output gate** $o^{(t)}$ and the memory-cell after application of the activation function g.
These relationships are shown graphically in Figure 2.5. The three gates are simple RNNs defined by the following equations:

$$i^{(t)} = \sigma(U_i h^{(t-1)} + W_i x^{(t)} + b_i)$$
$$f^{(t)} = \sigma(U_f h^{(t-1)} + W_f x^{(t)} + b_f)$$
$$o^{(t)} = \sigma(U_o h^{(t-1)} + W_o x^{(t)} + b_o)$$

where σ denotes the logistic activation function $\sigma(z) = \frac{1}{1-e^{-z}}$ which limits the gates to values between 0 and 1.
As one can easily see, the number of parameters used in LSTM networks is significantly higher than in simpler architectures, leading to higher memory and training time requirements. To be precise, the parameter number of a LSTM cell is $4((n+m)n+n)$ with n

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

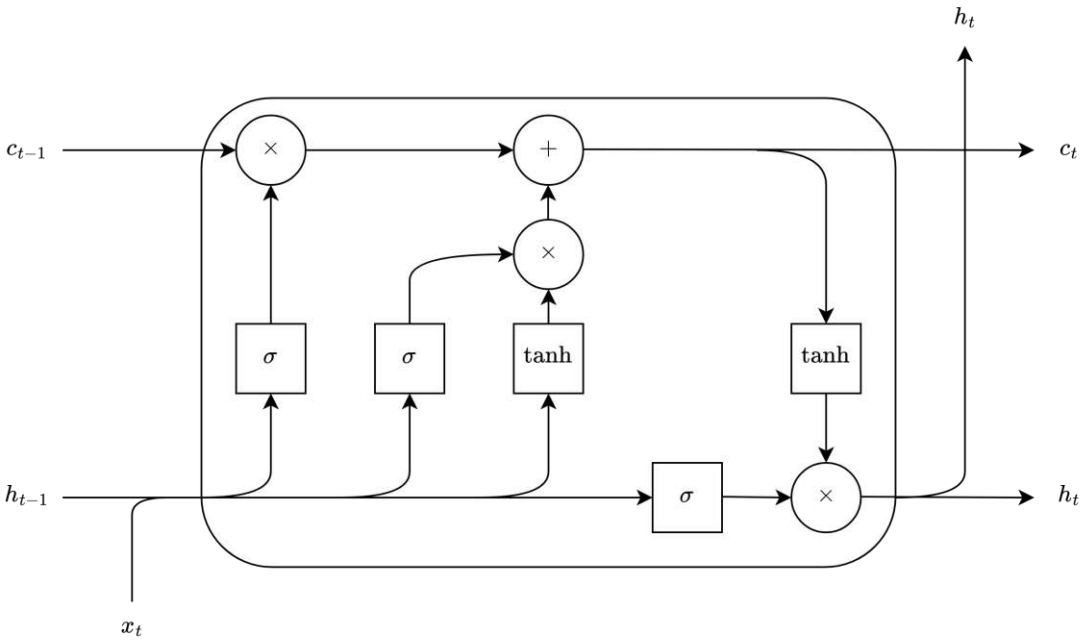

Figure 2.5.: Structure of an LSTM cell. Squares denote feed forward layers with the labelled activation function, circles denote element-wise operations, and joining and splitting arrows denote concatenation and copying respectively.

denoting the dimension of the hidden unit vector and m denoting the dimension off the inputs, compared to $2n^2+mn+2n$ for the simpler architecture described in 2.2.1. However, LSTM RNNs have been shown to have impressive results for several applications and are one of the most used RNN architectures.[20]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3. Mean-Variance Hedging

3.1. Prerequisites

Before introducing the mean-variance hedging approach, some prerequisite definitions and results have to be formulated. These are split into stochastic calculus prerequisites and a short introduction into hedging and mathematical finance.

3.1.1. Stochastic Calculus

Mathematical Setting

First, to provide a setting, a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is assumed. *Completeness* in this case refers to the property that all subsets of $\mathcal{F}$-null-sets are contained in $\mathcal{F}$. Additionally, a filtration $\mathbb{F} := (\mathcal{F}_t)_{t\in[0,T]}$ is assumed, with $T \in (0, \infty)$ denoting the time horizon. The following properties of the filtration are assumed:

- F_0 contains only sets of probability 0 or 1
- $\mathbb{F}$ is complete: Each $\mathcal{F}_t$ contains all subsets of $\mathcal{F}$-null-sets
- $\mathbb{F}$ is right-continuous: $\mathcal{F}_t = \bigcap_{s>t} \mathcal{F}_s \ \forall t \in [0, T]$

In the context of mathematical finance, the probability measure $\mathbb{P}$ is usually referred to as the **statistical measure** to differentiate it from the martingale measures introduced in 3.1.2.

Local Martingales and Semi-martingales

Some classes of stochastic processes, namely martingales, local martingales and semi-martingales, are of great importance to stochastic calculus and mathematical finance. For this reason, they are defined and discussed in this section.

Definition 2. *A stochastic process $M = (M)_{t\in[0,T]}$ is called a **martingale** if it fulfils the following conditions:*

- *M is $\mathbb{F}$-adapted: M_t is $\mathcal{F}_t$-measurable for all $t \in [0, T]$*
- *M is integrable: $\mathbb{E}[|M_t|] < \infty$ for all $t \in [0, T]$*
- *M has the martingale-property: $\mathbb{E}[M_t|\mathcal{F}_s] = M_s$ for all $s < t$*

*If the equality in the last condition is replaced by a $\leq$ or $\geq$ the process is called **super-martingale** or **sub-martingale** respectively.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

While martingales are an immensely important concept, the processes belonging to this category are not suitable to model some patterns and behaviours occurring in the real world. For this reason, another class of processes is often used and examined in mathematical finance, the so-called local martingales.

Definition 3. *An adapted cadlag process $M = (M)_{t\in[0,T]}$ is called a **local martingale** if there exists a sequence of stopping times $(\tau_n)_{n\in\mathbb{N}}$, $\tau_n \nearrow T$ such that the stopped process M^{τ_n} is a martingale for all n. The sequence $(\tau_n)_{n\in\mathbb{N}}$ is referred to as **localizing sequence**.*

*Additionally, a property of a stochastic process is said to hold **locally** if there exists a sequence $(\tau_n)_{n\in\mathbb{N}}$, $\tau_n \nearrow T$ such that the property holds for the stopped process M^{τ_n} for all n.*

To generalize the martingale concept even further, semi-martingales are introduced. Defining this class of processes requires one to first define predictable stochastic processes and processes of finite variation.

Definition 4. *The σ-algebra on $\mathbb{R}\times\Omega$ that is generated by the adapted and left-continuous processes is called the **predictable σ-algebra**. If a process is measurable w.r.t. the predictable σ-algebra, it is called **predictable**.*

Definition 5. *A stochastic process is said to be of **finite variation** if and only if it can be written as a difference of two increasing stochastic processes.*

Definition 6. *A stochastic process X that can be written as*

$$X = X_0 + A + M$$

*with A being a process of finite variation and M being a local martingale, both starting in 0, is called **semi-martingale**. If the process A is predictable, X is called a **special semi-martingale** and the, in this case, unique decomposition $X = X_0 + A + M$ is referred to as the **canonical decomposition** of X.*

Quadratic Variation and Co-variation

Definition 7. *Let X be a semi-martingale and $(\Pi_n)_{n\in\mathbb{N}}$ a sequence of random partitions. A random partition in this case means a finite sequence of finite stopping times $0 = T_0^n \leq T_1^n \leq \cdots \leq T_{k_n}^n = T$. It is assumed that the grid size of $(\Pi_n)_{n\in\mathbb{N}}$ converges a.s. to zero. The process $[X]$ defined by*

$$[X] := \lim_{n\to\infty} \sum_{i=0}^{k_n-1} \left(X^{T_{i+1}^n} - X^{T_i^n}\right)^2 \text{ in ucp}$$

*is called the **quadratic variation process** of X. This process is increasing and adapted.*

*The **quadratic co-variation process** $[X,Y]$ of two semi-martingales X and Y is defined via polarization as*

$$[X,Y] := \frac{1}{2}\left([X+Y] - [X] - [Y]\right).$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Definition 8. *A martingale M is called a* ***square-integrable*** *martingale (or L^2-martingale) if $\mathbb{E}[[M]_T] < \infty$.*
A semi-martingale X is called square-integrable if it is special with canonical decomposition $X = X_0 + M + A$, with M being a square-integrable martingale and A having square-integrable variation.

Definition 9. *If X and Y are two locally square-integrable semi-martingales, then there exists a unique predictable process $\langle X, Y \rangle$ such that $[X, Y] - \langle X, Y \rangle$ is a local martingale. This process is referred to as* ***conditional quadratic variation process*** *or simply* ***angle bracket process****.*

Stochastic Integration

The gains from trading in a financial asset are modelled as stochastic integrals, with the integrand representing the trading strategy and the integrator representing the price process of the financial asset.
While closer discussion of the interpretation of these integrals in mathematical finance is found in Section 3.1.2, the mathematical concept and properties of stochastic integrals are treated in this section.

As rigorously defining stochastic integrals is a quite lengthy and technical process that extends far beyond the scope of this thesis, only the main properties and results of stochastic integrals are stated in this section, while referring to [5] for a full definition.

Stochastic integrals are defined for semi-martingale integrators X and predictable integrands ϑ and usually denoted as $\int \vartheta \, dX$ or $(\vartheta \cdot X)$. The space of integrands for which the stochastic integral w.r.t. X is well defined is denoted by $L(X)$.
Due to stochastic integrals only being defined for predictable integrands, occasionally the left-continuous integrand ϑ_- is used instead to guarantee predictability.
To better understand the structure of the processes resulting from stochastic integration, the following result shows a way to approximate these processes.

Theorem 3. *Let X be a semi-martingale, ϑ an adapted cadlag process and (Π_n) a sequence of random partitions with grid size converging to zero. Then*

$$\int \vartheta_- \, dX = \lim_{n \to \infty} \sum_{i=1}^{k_n - 1} \vartheta_{T_i^n} \left(X^{T_{i+1}^n} - X^{T_i^n} \right) \text{ in ucp}$$

[18]

Theorem 4. *Let X, Y be semi-martingales and $\vartheta \in L(X)$, $\eta \in L(Y)$. Then*

$$\left[\int \vartheta \, dX, \int \eta \, dY \right] = \int \vartheta \eta \, d[X, Y]$$

In particular

$$\left[\int \vartheta \, dX \right] = \int \vartheta^2 \, d[X]$$

[18]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Theorem 5. ***(Integration by parts)*** *Let X, Y be semi-martingales. Then XY is also a semi-martingale that can be represented in the following way*

$$XY = X_0Y_0 + \int X_- \, dY + \int Y_- \, dX + [X, Y]$$

[5]

Theorem 6. ***(Itô's formula)*** *Let X be a semi-martingale and $f : [0, \infty) \times \mathbb{R} \to \mathbb{R}$ a $\mathcal{C}^{1,2}$-function. Then $f(t, X_t)$ is also a semi-martingale and*

$$\begin{aligned} f(t, X_t) =& f(0, X_0) + \int_0^t f_t(s, X_{s-}) \, ds \\ &+ \int_0^t f_x(s, X_{s-}) \, dX_s \\ &+ \frac{1}{2} \int_0^t f_{xx}(s, X_{s-}) \, d[X^c]_s \\ &+ \sum_{0<s\leq t} (f(s, X_s) - f(s, X_{s-}) - f_x(s, X_{s-})\Delta X_s) \end{aligned}$$

with f_t denoting the derivative of f w.r.t. the first variable, f_x ans f_{xx} denoting the first and second derivative w.r.t. the second variable, respectively, and $[X^c]_t = [X]_t - \sum_{0<s\leq t}(\Delta X_s)^2$. [5]

Theorem 7. ***(Itô isometry)*** *Let M be a local martingale and let ϑ be a predictable process with*

$$\mathbb{E}\left[\int_0^T \vartheta_t^2 \, d[M]_t\right] < \infty$$

Then $\int \vartheta \, dM$ is a square-integrable martingale and

$$\mathbb{E}\left[\left(\int_0^T \vartheta_t \, dM_t\right)^2\right] = \mathbb{E}\left[\int_0^T \vartheta_t^2 \, d[M]_t\right]$$

[18]

Kunita-Watanabe Decomposition

The Kunita-Watanabe decomposition [16] allows for representation of square-integrable martingales as a sum of a stochastic integral w.r.t. a given martingale M and a process strongly orthogonal to M. This can also be interpreted as orthogonal projection onto the space of stochastic integrals w.r.t. M. This decomposition relies on results for Hilbert spaces, which is why the Hilbert space of square-integrable martingales has to be defined first.

Definition 10. *The square-integrable martingales M, for which $\mathbb{E}[[M_T]] < \infty$, equipped with the scalar product*

$$(M, N) := \mathbb{E}[M_T N_T]$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

form a Hilbert space $\mathcal{M}^2$. The space $\mathcal{M}^2$ is complete, as each element M can be isometrically identified with its terminal value M_T in the complete Hilbert space $L^2(\mathbb{P})$.
Two elements $M, N \in \mathcal{M}^2$ are called ***orthogonal*** *if $(M, N) = 0$, and* ***strongly orthogonal*** *if MN is a martingale.*
The Hilbert space of square integrable martingales starting in 0 is denoted by $\mathcal{M}_0^2$, while $\mathcal{M}_{loc}^2$ denotes the processes which are locally in $\mathcal{M}^2$. [18]

The following definition introduces the space $L^2(M)$ of "square-integrable" stochastic integrands w.r.t. M as well as the space $\mathcal{S}(M)$ of stochastic integrals w.r.t. M obtained from these integrands. The latter space is the Hilbert sub-space that the square-integrable martingale is projected onto for the Kunita-Watanabe decomposition.

Definition 11. *For $M \in \mathcal{M}_{loc}^2$ the space $L^2(M)$ is defined as the space of all predictable processes with*

$$\mathbb{E}\left[\int_0^T \vartheta_t^2 \, d[M_t]\right]^{\frac{1}{2}} < \infty$$

The space of stochastic integrals $\mathcal{S}(M) := \left\{\int \vartheta \, dM \mid \vartheta \in L^2(M)\right\}$ is a closed sub-space of $\mathcal{M}^2$ and stable under stopping, leading to the name ***stable sub-space*** *generated by M. [18]*

Now all definitions and results needed for stating the Kunita-Watanabe decomposition have been introduced.

Theorem 8. ***Kunita-Watanabe decomposition*** *Let $M \in \mathcal{M}_{loc}^2$ and $N \in \mathcal{M}^2$. Then there exists a unique decomposition*

$$N_t = N_0 + \int_0^t \vartheta_s \, dM_s + L_t$$

with $\vartheta \in L^2(M)$, $L \in \mathcal{M}_0^2$ and L strongly orthogonal to all elements of $\mathcal{S}(M)$. Additionally, $\langle M, L \rangle = 0$. [18] [16]

This decomposition is later used directly to define mean-variance hedging strategies.

3.1.2. Mathematical Finance

This section focusses on introducing necessary mathematical finance concepts and defining terms used throughout the following sections and chapters.

Strategies

It is assumed that an investor has the possibility to invest in two different assets. First, a risky asset whose price process S is usually modelled as a semi-martingale. This can, for example, be interpreted as a stock traded on an exchange.
The other asset R represents a risk-free savings account. It is generally modelled as a continuous, strictly positive process of finite variation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The number of units of the risky asset held at any time is modelled by the predictable process $\vartheta_t \in L(S)$, while the adapted and R- integrable process ψ models the amount of holdings in the savings account. Together, the pair (ϑ, ψ) is called a **strategy**.
The value process of this strategy is given as

$$V_t = \vartheta S_t + \psi R_t$$

A strategy is called **self-financing** if its value process can be represented in the following way:

$$V_t = V_0 + \int_0^t \vartheta_u \, dS_u + \int_0^t \psi_u \, dR_u$$

This, in simple terms, means that the portfolio value at each time only stems from trading in the available assets, without providing or withdrawing capital besides the initial capital V_0.

The setting can be simplified by choosing R as numeraire, which means dividing every financial quantity by R. This yields a trivial savings account $R \equiv 1$ and the price process S/R for the risky asset. It follows from integration by parts, that (ϑ, ψ) is self-financing for (S, R) if and only if (ϑ, ψ) is self-financing for $(S/R, 1)$. As $dR = 0$, for a strategy to be self financing, only ϑ and V_0 have to be specified, as ψ and V_t are then uniquely determined by the self-financing constraint. Therefore, from here on out, only the discounted price process S/R of the risky asset (denoted by S for notational convenience) and the trivial savings account are used.
In this new setting, strategies and value processes are defined in the following way:

Definition 12. *A predictable, S-integrable process ϑ is called a **strategy**.*
*The **value process** $V = V(c, \vartheta)$ for a given initial capital c and strategy ϑ is defined as*

$$V_t = c + \int_0^t \vartheta_u \, dS_u$$

Depending on the setting and problem formulation, it may be necessary to restrict oneself to a subset of possible strategies. This subset is then called the **admissible strategies**. Due to the problem-dependent nature of admissible strategies, they are defined when needed in the following sections and chapters.

Arbitrage

In simple terms, an arbitrage opportunity allows an investor to achieve a riskless non-negative return by investing in a risky asset. This is related to the existence of so-called **equivalent martingale measures** by the **first fundamental theorem of asset pricing**.

Definition 13. *A strategy is called an **arbitrage opportunity** if its value process fulfills:*

1. $V_0 \leq 0$
2. $V_T \geq 0$ $\mathbb{P}$-*a.s.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3. $\mathbb{P}(V_T > 0) > 0$

Definition 14. *A probability measure $\mathbb{Q}$ equivalent to $\mathbb{P}$ is called an **equivalent martingale measure** (ELMM), if S is a $\mathbb{Q}$-local-martingale. The set of all equivalent martingale measures is denoted by $\mathcal{M}^e$.*
*If $\mathbb{Q}$ is only absolutely continuous w.r.t. $\mathbb{P}$, it is simply referred to as a **martingale measure** and the set of all such $\mathbb{Q}$ is denoted by $\mathcal{M}$.*

This allows for the definition of admissible strategies, at least for within this section.

Definition 15. *Assuming $\mathcal{M}^e \neq \emptyset$, a strategy ϑ is **admissible** if its value process V is a $\mathbb{Q}$-super-martingale for all $\mathbb{Q} \in \mathcal{M}^e$.*

The aforementioned first fundamental theorem of asset pricing states that the existence of "no free Lunch with vanishing risk" is equivalent to the existence of an equivalent martingale measure [6]. As "no free Lunch with vanishing risk" is a too general concept for the scope of this thesis, only the implication connecting absence of arbitrage and existence of equivalent martingale measures is stated.

Theorem 9. ***First fundamental theorem of asset pricing (FTAP)*** *If there exists an equivalent martingale measure for S, then there are no arbitrage opportunities with admissible strategies. [18] [6]*

Hedging and Complete Markets

For this section the following assumptions are made:

- $\mathcal{M}^e \neq \emptyset$
- A strategy ϑ is deemed **admissible** if the stochastic integral $\int \vartheta \, dS$ is a $\mathbb{Q}$-martingale for all $\mathbb{Q} \in \mathcal{M}^e$

In order to define complete and incomplete markets, the idea of a **claim** has to be introduced first.

Definition 16. *A **claim** H is an $\mathcal{F}_T$ measurable random variable.*

Claims are commonly used to model derivatives written on the risky asset S. In the case of an European option this would mean that $H = h(S_T)$ for some function h. For example, a claim representing a European call option with strike K is given by $H = (S_T - K)^+$.

Hedging in the sense of mathematical finance refers to approximating a claim by the value process V_T of an admissible strategy ϑ and initial capital c. In which sense the payoff is approximated depends on the problem formulation. This thesis focusses on mean-variance hedging which uses a quadratic criterion and is introduced in 3.2. In some settings however, all claims can be represented perfectly via trading, not just approximated. These settings are called **complete markets**.

Definition 17. *A claim H is called **attainable**, if there exists an attainable strategy ϑ and initial capital c such that*

$$H = c + \int_0^T \vartheta_t \, dS_t \tag{3.1}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

With the notion of attainable claims introduced, complete markets can be defined.

Definition 18. *The quintuplet $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}, S)$ is called a **market**. If all bounded claims H within a market are attainable, the market is called **complete**. Markets that are not complete are called **incomplete**.*

Similar to the first FTAP, the **second fundamental theorem of asset pricing** relates the completeness of a market to the uniqueness of equivalent martingale measures.

Theorem 10. ***Second fundamental theorem of asset pricing (FTAP)*** *An arbitrage free market is complete if and only if there exists exactly one equivalent martingale measure. [18] [6]*

Consider a bounded claim H in a complete market. Due to completeness, H can be represented as

$$H = c + \int_0^T \vartheta_u \, dS_u \tag{3.2}$$

Now fix a martingale measure $\mathbb{Q}$ and consider the closed $\mathbb{Q}$-martingale M generated by H

$$M_t := \mathbb{E}_{\mathbb{Q}} [H \mid \mathcal{F}_t]$$

This martingale is square-integrable due to H being bounded and the contractivity of the conditional expectation. If the market is complete, M should be representable as a stochastic martingale w.r.t. S. According to Theorem 8, M can at least be decomposed into a stochastic integral w.r.t. S and a strongly orthogonal process L. These two components reflect the attainable and unattainable part of the claim H. This means that, for a market to be complete, all $N \in \mathcal{M}^2$ need to be in the stable subspace $\mathcal{S}(S)$ generated by S. This property of S is called the **predictable representation property**.

Definition 19. *A process $M \in \mathcal{M}^2_{loc}$ has the **predictable representation property (PRP)** if $\mathcal{S}(M) = \mathcal{M}^2$*

In this case, the process L in the Kunita-Watanabe decomposition vanishes for all possible M, and the market is complete.

Considering again a bounded claim H, now in a complete market, the constant c in 3.1 equates to the initial capital required to drive the self-financing strategy ϑ. As his initial capital allows an investor to completely replicate the claim H, it can be thought of as the fair price of H.
As the market is complete, the second FTAP implies the existence of a unique martingale measure $\mathbb{Q}$. Taking expectations w.r.t. $\mathbb{Q}$ in 3.1 yields

$$\mathbb{E}_{\mathbb{Q}} [H] = c$$

This shows that in complete markets one can calculate the fair price of a claim by taking expectation w.r.t. the martingale measure. The strategy ϑ is referred to as the **hedging strategy** of H.

In incomplete markets, a claim is in general not attainable, meaning that it cannot be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

replicated by a hedging strategy. In this case, the goal is to find an admissible strategy that best approximates the claim. Not only is hedging not as straightforward in incomplete markets, pricing a claim is more challenging as well, as ,according to the second FTAP, one has to deal with an infinite number of martingale measure ($\mathcal{M}^e$ contains at least two elements and is a convex set).
One hedging approach for these settings is **mean-variance hedging** which is introduced in the following section.

3.2. Mean-Variance Hedging

In incomplete markets not every claim is attainable. Trading in the market's risky asset to hedge the claim still exposes the investor to residual risk. The idea behind mean-variance hedging is to minimize this risk according to a quadratic criterion.
This thesis focusses on the "martingale case", where the minimization problem is formulated under a martingale measure. While mean-variance hedging can also be formulated under the statistical measure, the theory is substantially more complex, extending beyond the scope of this thesis.

For this section, the following assumptions are made:

1. S is a locally square-integrable $\mathbb{P}$-martingale
2. A strategy ϑ is deemed admissible if $\vartheta \in L^2(S)$
3. H is a square-integrable, $\mathcal{F}_T$-measurable random variable

The goal of minimizing the quadratic risk can be formalized in the following way:
Minimize

$$\mathbb{E}\left[\left(H - c - \int_0^T \vartheta_t \, dS_t\right)^2\right] \tag{3.3}$$

over all admissible strategies $\vartheta \in L^2(S)$ and initial capitals c.
This is already reminiscent of the Kunita-Watanabe decomposition, but in order to apply it in this case, the random variable H needs to be replaced with its closed martingale

$$V_t = \mathbb{E}\left[H \mid \mathcal{F}_t\right] \tag{3.4}$$

If one now identifies the square-integrable martingale $V \in \mathcal{M}^2$ with its terminal value $V_T = H$, it is clear that 3.3 can be solved by orthogonally projecting V onto the subspace spanned by the constants and the stable subspace $\mathcal{S}(S)$. This is achieved by taking the Kunita-Watanabe decomposition (Theorem 8) of V w.r.t. S:

Theorem 11. ***Optimal mean-variance strategy.*** *For a square-integrable claim H, consider the Kunita-Watanabe decomposition of the process V defined in 3.4*

$$V = \mathbb{E}\left[H\right] + \int \vartheta^H \, dS + L^H \tag{3.5}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

with $\vartheta \in L^2(S)$ and $L \in M_0^2$ strongly orthogonal to all elements of $\mathcal{S}(S)$.
The optimal initial capital and strategy in the minimization problem 3.3 are then given by $c^* := \mathbb{E}[H]$ and $\vartheta^* := \vartheta^H$.
The optimal strategy is unique in the sense that for two optimal strategies, their stochastic integrals w.r.t. S are indistinguishable.

The process $c^* + \int \vartheta^*$ can be interpreted as the attainable part of H which can be replicated, while L represents the unattainable and unhedgeable part of H. Therefore, the variance of L_T is of interest to quantify the remaining risk:

$$R(\vartheta) := \mathbb{E}\left[L_T^2\right]$$

[18] While finding the Kunita-Watanabe decomposition for claims written on a general process S can be difficult, there are approaches for the class of exponential Lévy processes, one of which will be discussed in the following section.

3.2.1. Laplace Method for Exponential Lévy Processes

If the price of the risky asset is modelled as an **exponential Lévy process** $S_t = \exp(X_t)$ for a Lévy process X and the claim is given as $H := f(S_T)$ where f fulfils certain conditions discussed below, the optimal strategy and initial capital in the mean-variance hedging sense can be calculated explicitly as complex integrals.
The following section introduces Lévy processes as well as the cumulant generating function. Next, the bilateral Laplace transform is introduced, followed by a discussion of the necessary conditions for f. Finally, the theorem containing the explicit integral representations of strategy, value process and hedging error is stated.

Lévy Processes

Definition 20. *A process X on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ is called a **Lévy process**, if it has the following properties:*

1. *X is adapted w.r.t. $\mathbb{F}$*
2. *$X_0 = 0$ a.s.*
3. *The paths of X are a.s. right-continuous with left limits (càdlàg)*
4. *X has stationary increments: $X_t - X_s \sim X_{t-s}$ for all $s < t$*
5. *The increments of X are independent of the past: $X_t - X_s$ is independent of $\mathcal{F}_s$ for all $s < t$*

[14]

The distribution of the process X is determined by the law of X_1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Definition 21. *For a Lévy process X define*

$$D := \left\{ z \in \mathbb{C} \;\middle|\; \mathbb{E}\left[e^{\Re(z)X_1} < \infty\right] \right\}$$

*For $t > 0$ the **cumulant function** κ of X is defined as:*

$$\kappa : D \to \mathbb{C}$$
$$e^{\kappa(z)t} = \mathbb{E}\left[e^{zX_t}\right]$$

In many cases, the cumulant function κ has a simpler structure than the distribution of X. The cumulant function exists at least for $z \in \mathbb{C}$ with $\Re(z) = 0$. In this case, $\kappa(z) = \kappa(iu)$ corresponds to the characteristic exponent

$$\psi(u) := \log \mathbb{E}\left[e^{iuX_1}\right]$$

which uniquely determines the distribution of X_1 and therefore of the entire process X.

For this section, the following two assumptions are made:

- S is a exponential Lévy process: $S_t = S_0 \exp(X_t)$ for a Lévy process X and constant $S_0 > 0$
- S is a square integrable $\mathbb{P}$-martingale

Bilateral Laplace Transform

Definition 22. *Let $f : \mathbb{R} \to \mathbb{C}$ be a Borel-measurable function. Its **bilateral Laplace transform** $\tilde{f}$ is given as*

$$\tilde{f}(z) := \int_{-\infty}^{\infty} f(x) e^{-zx}\, dx$$

for any $z \in \mathbb{C}$ such that the integral exists.

The following lemma shows that the bilateral Laplace transform always exists on a vertical strip in the complex plane. This strip can be empty, a single vertical line, or even a open or closed half-plane or all of $\mathbb{C}$.

Lemma 1. *Suppose that $\tilde{f}(a)$ and $\tilde{f}(b)$ exist for $a, b \in \mathbb{R}$ with $a \leq b$. Then $\tilde{f}(z)$ exists for all z with $a \leq \Re(z) \leq b$.*

Proof: The statement follows from the following estimate for $a \leq \Re(z) \leq b$:

$$|f(x)e^{-zx}| = |f(x)|e^{-\Re(z)x} \leq |f(x)e^{-ax}| + |f(x)e^{-bx}|$$

[14]

□

The bilateral Laplace transform determines a function uniquely and can be inverted to obtain the original function from its transform. There are multiple ways of achieving this inversion, on being the so called **Bromwich inversion integral**:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Theorem 12. ***Bromwich inversion integral.*** *If the function $v \mapsto f(R+iv)$ is integrable, then*

$$f(x) = \frac{1}{2\pi i} \int_{R-i\infty}^{R+i\infty} \tilde{f}(z) e^{zx} \, dz$$

[19]

Integral Representation of Payoffs

The Laplace method requires more conditions regarding the claim to be hedged than the general Definition 16. For the remainder of this section, H represents a European contingent claim written on S. In mathematical terms, H denotes a square-integrable, $\mathcal{F}_T$-measurable random variable given by $H = f(S_T)$ for some function $f : (0, \infty) \to \mathbb{R}$. It is also assumed that the payoff function f admits an integral representation of the following form

$$f(s) = \int s^z \, \Pi(dz) \tag{3.6}$$

for a complex measure Π on a vertical strip $\{z \in \mathbb{C} \mid R_1 \leq \Re(z) \leq R_2\}$ where $\mathbb{E}\left[e^{2R_1 X_1}\right] < \infty$ and $\mathbb{E}\left[e^{2R_2 X_1}\right] < \infty$.
While this does of course limit the classes of payoff functions this method can be applied to, many claims, e.g. calls, puts, power calls and puts, self-quanto calls and puts and digital options, admit this integral representation [14]. For these payoff functions, the measure Π is derived using the Bromwich inversion integral.

Example: Integral Representation of European Call As an example, as well as for use in Chapter 5, the derivation of the integral representation for a European call is shown here.

Theorem 13. *For a European call $H := (S_T - K)^+$, the integral representation in the form of 3.6 is*

$$(s - K)^+ = \int_{R-i\infty}^{R+\infty} s^z \, \Pi(dz)$$

with $R > 1$ and

$$\Pi(dz) := \frac{1}{2\pi i} \frac{K^{1-z}}{z(z-1)} \, dz$$

Proof: As $s > 0$, one can define $s = S_0 e^x$ and subsequently $g(x) := (S_0 e^x - K)^+$. Now calculate the bilateral Laplace transform of g:

$$\begin{aligned}
\tilde{g}(z) &= \int_{-\infty}^{\infty} (S_0 e^x - K)^+ e^{-zx} \, dx \\
&= \int_{-\infty}^{\infty} \mathbb{1}_{\{S_0 e^x - K \geq 0\}} (S_0 e^x - K) \, dx \\
&= \int_{\ln(\frac{K}{S_0})}^{\infty} (S_0 e^x - K) \, dx
\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

For $\Re(z) > 1$ this integral can be calculated as:

$$\begin{aligned}\int_{\ln(\frac{K}{S_0})}^{\infty} (S_0 e^x - K)\, dx &= \frac{S_0 e^{(1-z)x}}{1-z} + \frac{K e^{-zx}}{z}\Bigg|_{x=\ln(\frac{K}{S_0})}^{\infty} \\ &= \frac{S_0^z k^{1-z}}{z-1} - \frac{S_0^z k^{1-z}}{z} \\ &= \frac{S_0^z K^{1-z}}{z(z-1)}\end{aligned}$$

Applying the Bromwich inversion integral with $R > 1$ to $\tilde{g}(z)$ then yields

$$\begin{aligned}g(x) &= \frac{1}{2\pi i}\int_{R-i\infty}^{R+i\infty} \tilde{g}(z) e^{zx}\, dz \\ &= \frac{1}{2\pi i}\int_{R-i\infty}^{R+i\infty} \frac{S_0^z K^{1-z}}{z(z-1)} e^{zx}\, dz \\ &= \frac{1}{2\pi i}\int_{R-i\infty}^{R+i\infty} s^z \frac{K^{1-z}}{z(z-1)}\, dz\end{aligned}$$

which is the claimed integral representation.

□

For more concise notation, define the following quantities:

$$\begin{aligned}\alpha(y,z) &:= \kappa(y) + \kappa(z) \\ \beta(y,z) &:= \kappa(y+z) - \kappa(y) - \kappa(z) - \frac{(\kappa(y+1)-\kappa(y))(\kappa(z+1)-\kappa(z))}{(\kappa(y+1)-\kappa(y))(2)} \\ \gamma(z) &:= \frac{\kappa(z+1)-\kappa(z)}{\kappa(2)}\end{aligned}$$

Optimal Strategy

Using the notation introduced in this section, the following theorem shows the optimal solution to the mean-variance hedging problem. In addition to the optimal trading strategy, the value process of the claim can be calculated, which determines the optimal initial capital.

Theorem 14. *Define*

$$\begin{aligned}H_t &:= \int e^{\kappa(z)(T-t)} S_t^z\ \Pi(dz) \\ \xi_t &:= \int \gamma(z) e^{\kappa(z)(T-t)} S_{t-}^{z-1}\ \Pi(dz)\end{aligned}$$

Then the mean-variance value of a claim $H = f(S_T)$ with integral representation 3.6 at time t and current price $S_t = s$ is given by

$$V_t := H_t|_{S_t=s} = \int e^{\kappa(z)(T-t)} s^z\ \Pi(dz)$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and the optimal mean-variance hedging strategy ϑ^H *is given by*

$$\vartheta^H = \xi$$

The variance of the hedging error

$$R(\vartheta^H) := \mathbb{E}\left[\left(V_0 + \int_0^T \vartheta_t^H \, dS_t - f(S_T)\right)^2\right]$$

equals

$$R(\vartheta^H) = \int\int R(y,z) \; \Pi(dy)\Pi(dz)$$

with

$$R(y,z) := \int_0^T e^{\kappa(y+z)t + \alpha(y,z)(T-t)} \, dt$$

[18]

The proof of this theorem is omitted due to its length and dependence on many lemmas and estimates, extending beyond the scope of this thesis. For full details, refer to [14]. It should be noted that the theorem proved in [14] does not assume S to be a square-integrable martingale, leading to a more general result.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4. Using Deep Learning for Mean-Variance Hedging

After all necessary Deep Learning and Mathematical Finance concepts and definitions were introduced in the previous chapters, they are combined in this chapter to leverage the abilities of (recurrent) neural networks to approximate mean-variance hedging strategies. The optimization problem 3.3 is in general difficult to solve and compute, especially for more sophisticated models like stochastic volatility models. Therefore, an approximation using neural networks can be preferable, as computation of the strategy using an already trained network is very efficient. Additionally, this approach allows for the inclusion of trading restrictions and trading costs, which are usually not considered in the context of mean-variance Hedging.

The first section defines the mathematical setting. It differs from the one defined in chapter 3 due to the outputs of neural networks representing trading strategies being finite-dimensional, which requires a market with discrete time scale.

In the second section, the Mean-Variance Hedging optimization problem 3.3 is reformulated according to the setting provided in 4.1. Afterwards, the RNN architecture used for the Deep Learning approach is introduced and the optimization problem is adapted to minimize over network parameters instead of trading strategies. Finally, the implementation of the architecture used for the numerical experiments is and explained.

4.1. Setting

Consider a fixed time horizon T and trading dates $0 = t_0 < \ldots < t_n = T$ and let $(\Omega, \mathcal{F}, \mathbb{P})$ be a finite probability with $\Omega = \{\omega_1, \ldots, \omega_N\}$ and $\mathbb{P}[\{\omega_k\}] > 0 \; \forall k \in 1, \ldots, N$.
The information available at each time step is modelled by the r-dimensional random variables $(I_k)_{k\in 1,\ldots,n}$. These random variables contain the price processes of all tradable assets as well as auxiliary information, e.g., interest rates, macro-economic indicators, or news data.
The filtration $\mathbb{F} := (\mathcal{F}_k)_{k\in 1,\ldots,n}$ is defined as the filtration generated by $(I_k)_{k\in 1,\ldots,n}$, meaning that each σ-algebra $\mathcal{F}_k$ contains all information available up to t_k. This definition implies that all asset price processes are adapted w.r.t. $\mathbb{F}$.
The market contains d tradable assets, the prices of which are represented by a d-dimensional stochastic process $(S_k)_{k\in 1,\ldots,n}$. These tradable assets do not exclusively consist of primary assets, but can also contain liquid derivatives and other securities.
The derivative portfolio to be hedged is represented by an $\mathcal{F}_T$-measurable random variable H, which corresponds to a *claim* in chapter 3 and is referred to as such throughout this chapter. The time horizon T is chosen to be equal to the maximum maturity of the port-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

folio represented by H.
To hedge the claim H, an investor can trade in S using an $\mathbb{R}^d$-valued, $\mathbb{F}$-adapted process $(\vartheta_k)_{k\in 1,\dots,n-1}$. The process $(\vartheta_k)_{k\in 1,\dots,n-1}$ is, analogous to chapter 3, referred to as *trading strategy*. The i-th component of $\vartheta_k = (\vartheta_k^1, \dots, \vartheta_k^d)$ represents the holdings in the i-th asset from time t_k to t_{k+1}. For notational convenience, define $\vartheta_{-1} = \vartheta_n := 0$.
The set of all unconstrained trading strategies is denoted by $\mathcal{G}^u$.

For simplification, it is assumed that all payments are accrued with a risk free overnight rate. This means that an interest rate of zero can be assumed, as can that all payments occur at time T.
Additionally, instruments with true optionality (e.g., American options) are not considered, neither as part of the claim H, nor the tradable assets. [3]

4.1.1. Trading Constraints

In order to model more realistic hedging scenarios, trading constraints have to be taken into account. Due to liquidity, asset availability or trading restrictions, an unconstrained strategy may be impossible to be implemented by an investor. If, for example, one of the components of S describes an option that is only available for trading in the time interval $[T_1, T_2]$ with $0 < T_1 < T_2 \leq T$, the possible holdings in this option outside of $[T_1, T_2]$ are limited to $\{0\}$. An investor may also face trading restrictions, like simple "no-short"-restrictions, or more complicated ones like maximum values for the traded Vega.

Trading constraints are modelled by restricting ϑ_k to a set $\mathcal{G}_k$. This set is given as the image of a continuous $\mathcal{F}_k$ measurable function $G_k : \mathbb{R}^{d(k+1)} \to \mathbb{R}^d$, meaning $\mathcal{G}_k := G_k(\mathbb{R}^{d(k+1)})$.
Given an unconstrained strategy $\vartheta^u \in \mathcal{G}^u$, its constrained "projection" is given successively via

$$(G \circ \vartheta^u)_k := G_k\left((G \circ \vartheta^u)_0, \dots, (G \circ \vartheta^u)_{k-1}, \vartheta_k^u\right) \tag{4.1}$$

The set of constrained strategies obtained this way is denoted by $\mathcal{G} := (G \circ \mathcal{G}^u) \subset \mathcal{G}^u$.

To give an example, a "no-short"-constraint (all holdings must be non-negative) could be implemented in this setting by defining G_k in the following way:

$$G_k\left(\vartheta_0, \dots, \vartheta_k\right) := \left((\vartheta_k^1)^+, (\vartheta_k^2)^+, \dots, (\vartheta_k^d)^+\right)$$

with $(\cdot)^+$ denoting the positive part.

4.1.2. Transaction Costs

As for trading constraints, to model realistic hedging scenarios, transaction costs have to be considered.

Transaction costs in the context of this thesis will be modelled in the following way: Trading a position $n \in \mathbb{R}^d$ at time t_k will incur transaction costs of $c_k(n)$. The transaction costs are therefore defined by the functions $c_k : \mathbb{R}^d \to \mathbb{R}$. It is assumed that $c_k(0) = 0$ and that each c_k is $\mathcal{F}_k$-measurable, non-negative and upper-semi-continuous.
The traded position corresponds to the change of holdings from one point in time to the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

next and can therefore be calculated as $\vartheta_k - \vartheta_{k-1}$. This way, the total transaction costs of a strategy ϑ up to the maturity T are given as:

$$C_T(\vartheta) := \sum_{k=0}^{n} c_k(\vartheta_k - \vartheta_{k-1}) \tag{4.2}$$

Two "standard" choices for the functions $c_0, \dots c_n$ are the following:

- **Proportional transaction costs**: For fixed constants $c_k^i > 0$, $i \in \{1, \dots, d\}, k \in \{0, \dots, n\}$ define $c_k(n) := \sum_{i=1}^{d} c_k^i S_k^i |n^i|$
- **Fixed transaction costs**: For fixed constants $c_k^i > 0$, $i \in \{1, \dots, d\}, k \in \{0, \dots, n\}$ and $\epsilon > 0$ define $c_k(n) := \sum_{i=1}^{d} c_k^i 1_{|n^i| \geq \epsilon}$

As mentioned before, $\vartheta_{-1} = \vartheta_n = 0$, which implies liquidation of all positions at time T.

For the simulations in later chapters, if trading costs are modelled, proportional trading costs with time-independent constants ($c_0^i = \dots = c_n^i$) will be used.

4.1.3. Portfolio Value and Loss

The cumulated profit at time T from trading in S using strategy ϑ is given as

$$(\vartheta \cdot S)_T := \sum_{k=0}^{n} \vartheta_k \cdot (S_{k+1} - S_k)$$

Additionally, an initial capital of c is assumed, which can be interpreted at the price the claim H is sold for.

Combining the definitions from above, the portfolio value of an investor trying to hedge a claim H with strategy ϑ and initial capital c is given as

$$PL_T(H, c, \vartheta) := -H + c + (\vartheta \cdot S)_T - C_T(\vartheta)$$

To be congruent with the notation used in 3, the focus will not lie on the portfolio value $PL_T(H, c, \vartheta)$, but the loss of the hedging strategy (c, ϑ) for claim H at time T. This is simply defined as

$$L_T(H, c, \vartheta) := -PL_T(H, c, \vartheta) = H - c - (\vartheta \cdot S)_T + C_T(\vartheta)$$

which, if one ignores the trading cost term, is the discrete time analogue of the quantity $(H - c - \int_0^T \vartheta_t \, dS_t)$ from 3.3.

4.1.4. Mean-Variance Hedging

The mean-variance hedging problem formulated in 3.3 can now be adapted to the setting described in section 4.1.

The goal is the same as before:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In incomplete markets, claims may not be perfectly replicable by trading in the available assets, exposing an investor trying to hedge the claim to residual risk given by L_T. This risk is to be minimized under a quadratic criterion over all trading strategies and initial capitals.
This can be formalized similar to 3.3:

Minimize

$$\mathbb{E}\left[L_T(H,c,\vartheta)^2\right] = \mathbb{E}\left[\left(H - c - (\vartheta \cdot S)_T - C_T(\vartheta)\right)^2\right] \tag{4.3}$$

over all strategies $\vartheta \in \mathcal{G}$ and initial capitals c.

Up to now, the problem has only been adapted to the new mathematical setting, the minimization problem still has to be solved. In this setting, however, it is possible to introduce the RNN-approximation approach used to find a (close-to-)optimal pair of strategy and initial capital. For this, the model used for this approximation has to be introduced.

4.2. Stacked LSTM Model

4.2.1. Model Architecture

The architecture used for the approximation approach described in this thesis is a stacked LSTM recurrent neural network, similar to the deep neural networks described in 2.2.3.
In particular, this means that there are m LSTM-Cells, each with a number of hidden units of n_{h_i} for $i \in \{1,\dots,m\}$. In each time step t, the calculated hidden units $h_i^{(t)}$ of the i-th LSTM cell are not only passed "horizontally" to the i-th LSTM-Cell in the next time step $t+1$, but also "vertically" as regular inputs to the $(i+1)$-th LSTM-Cell.
The hidden units of the m-th LSTM-Cell are passed to a single-layer feed-forward neural network with an output dimension of d and linear activation function $\sigma(x) = x$. This feed-forward network combines the hidden units of the m-th and final LSTM-Cell into the model outputs $(\vartheta_0,\dots,\vartheta_{n-1})$ in each time step.
These outputs represent an unrestricted trading strategy $\vartheta \in \mathcal{G}^u$, but as the optimization problem 4.3 is stated as minimization over $\mathcal{G}$ and not $\mathcal{G}^u$, one has to apply the trading restrictions according to 4.1 to obtain $(G \circ \vartheta) \in \mathcal{G}$.
The model inputs are samples of the r-dimensional random variables $(I_k)_{k\in 1,\dots,n}$ introduced in section 4.1.

Assuming that the dimensionality of the hidden units is the same for every LSTM cell, the total number of parameters is $4h((m-1)(2h+1)+i+h+1)+d(h+1)$, with h and i denoting the dimensionality of the hidden units and inputs respectively.

Denote by $\mathcal{RNN}^{LSTM}_{M,d_i,d_o}$ the set of all stacked LSTM networks with input dimension d_i, output dimension d_o and at most M parameters. Additionally, $\mathcal{RNN}^{LSTM}_{M,d_i,d_o} = \{F^\theta : \theta \in \Theta_{M,d_i,d_o}\}$ with $\Theta_{M,d_i,d_o} \subset \mathbb{R}^q$ (for some q depending on M) denoting the parameter space for the elements of $\mathcal{RNN}^{LSTM}_{M,d_i,d_o}$.

Using this model, the goal is to find suitable parameters to minimize the loss function described in the following section.

The model architecture is depicted graphically in Figure 4.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

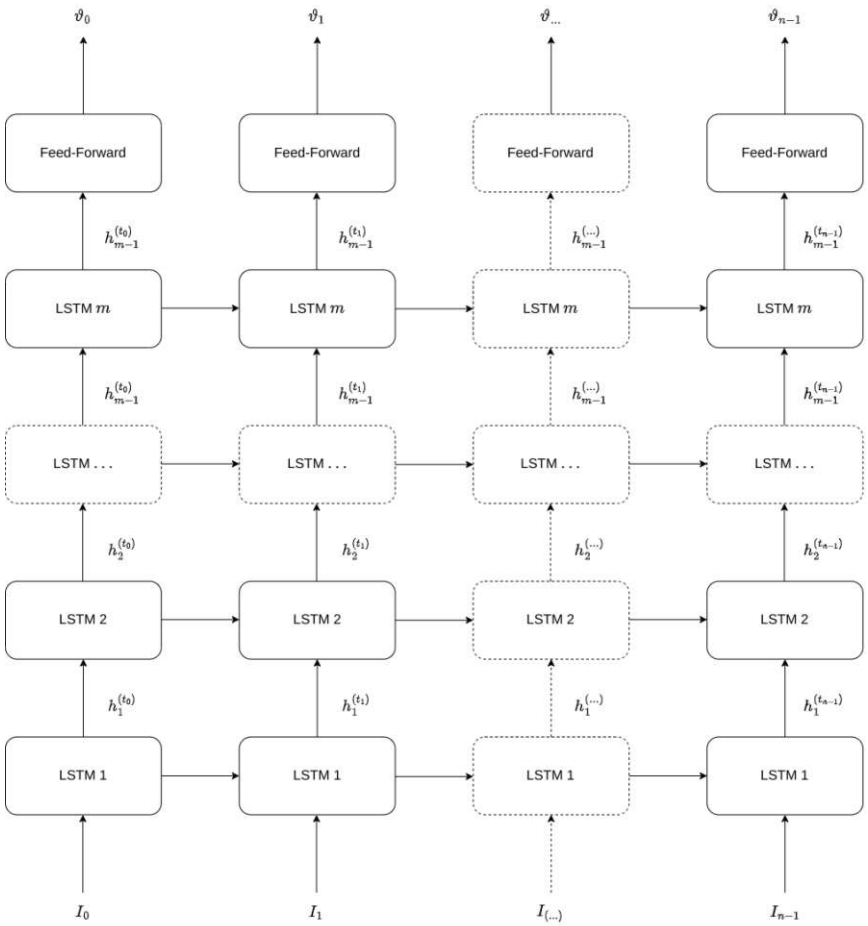

Figure 4.1.: Stacked LSTM Architecture: There are m LSTM-Cells arranged hierarchically, followed by a single layer feed-forward network combining the m-th LSTM-Cell's hidden units into the final output ϑ_i for each time step.

4.2.2. Cost Function

The objective of this model is to minimize a certain cost function. As the goal is to approximate the optimal strategy and initial capital for problem 4.3, the cost function is chosen to reflect this. One fact to be considered is that the model outputs an unrestricted strategy while the minimization problem 4.3 is formulated as a minimization over all restricted strategies. The minimization problem can, however, be stated equivalently over all unrestricted trading strategies in the following way:

Minimize

$$\mathbb{E}\left[L_T(H, c, G \circ \vartheta)^2\right] = \mathbb{E}\left[(H - c - (G \circ \vartheta \cdot S)_T - C_T(G \circ \vartheta))^2\right] \tag{4.4}$$

over all strategies $\vartheta \in \mathcal{G}^u$ and initial capitals c.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The problems 4.3 and 4.4 are equivalent, as $G \circ \vartheta = \vartheta$ for $\vartheta \in \mathcal{G}$ and $G \circ \vartheta \in \mathcal{G}$ for $\vartheta \in \mathcal{G}^u$. As not all $\vartheta \in \mathcal{G}^u$ can be represented by a stacked LSTM RNN with finite number of parameters, the problem is once again restated for the following set of strategies

$$\begin{aligned}\mathcal{G}^M &:= \left\{(\vartheta_k)_{k\in 1,\dots,n-1} \mid (\vartheta_k)_{k\in 1,\dots,n-1} = F(I_0,\dots,I_{n-1}), F \in \mathcal{RNN}^{LSTM}_{M,r,d}\right\} \\ &= \left\{(\vartheta^\theta_k)_{k\in 1,\dots,n-1} \mid (\vartheta^\theta_k)_{k\in 1,\dots,n-1} = F^\theta(I_0,\dots,I_{n-1}), \theta \in \Theta_{M,r,d}\right\}\end{aligned}$$

The set $\mathcal{G}^M$ consists of all trading strategies that can be represented by stacked LSTM RNNs with at most M parameters.
Substituting $\mathcal{G}$ with $\mathcal{G}^M$ and subsequently the parameter space $\Theta_{M,r,d}$ in 4.4 yields the final optimization problem:

Minimize

$$\mathbb{E}\left[L_T(H, c, G \circ \vartheta^\theta)^2\right] = \mathbb{E}\left[\left(H - c - (G \circ \vartheta^\theta \cdot S)_T - C_T(G \circ \vartheta^\theta)\right)^2\right] \tag{4.5}$$

over $\theta \in \Theta_{M,r,d}$ and initial capitals c.

As the initial capital c is not modelled as an output of the model, it can instead be interpreted as an additional parameter of the stacked LSTM RNN. Extending the parameter space $\Theta_{M,r,d} \subset \mathbb{R}^q$ by one dimension to $\Theta_{M,r,d,c} \subset \mathbb{R}^{q+1}$ and including the inital capital c^θ in the parameter vector allows 4.5 to be represented purely as an optimization problem over the parameter space $\Theta_{M,r,d,c}$.

To reiterate, the original problem 4.3 has first been restated as a minimization problem over all unrestricted strategies and now has been rewritten as a minimization problem over the parameters of a recurrent neural network. To now numerically find close-to optimal parameters, the cost function is defined as

$$\begin{aligned}J(\theta) :=& \mathbb{E}\left[\left(H - c^\theta - (G \circ \vartheta^\theta \cdot S)_T - C_T(G \circ \vartheta^\theta)\right)^2\right] \\ =& \sum_{k=1}^{N}\left(H(\omega_k) - c^\theta - (G \circ \vartheta^\theta \cdot S)_T(\omega_k) - C_T(G \circ \vartheta^\theta)(\omega_k)\right)^2 \cdot \mathbb{P}(\{\omega_k\})\end{aligned} \tag{4.6}$$

which can be numerically minimized using conventional deep learning techniques discussed in chapter 2.
It should be noted that this approach does not explicitly require the trading strategy to be admissible. However, as the results from Chapter 5 show, the model does not appear to find any arbitrage strategies.

4.2.3. Implementation

The model architecture described in section 4.2.1 was implemented in Python using the Deep Learning Library TensorFlow [7].
While TensorFlow offers a simple framework for building deep learning models in the form of keras.Sequential, this cannot be used in this case, as a per-sample cost function is expected. Due to the cost function 4.6 depending on the outputs at each time-step as well

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

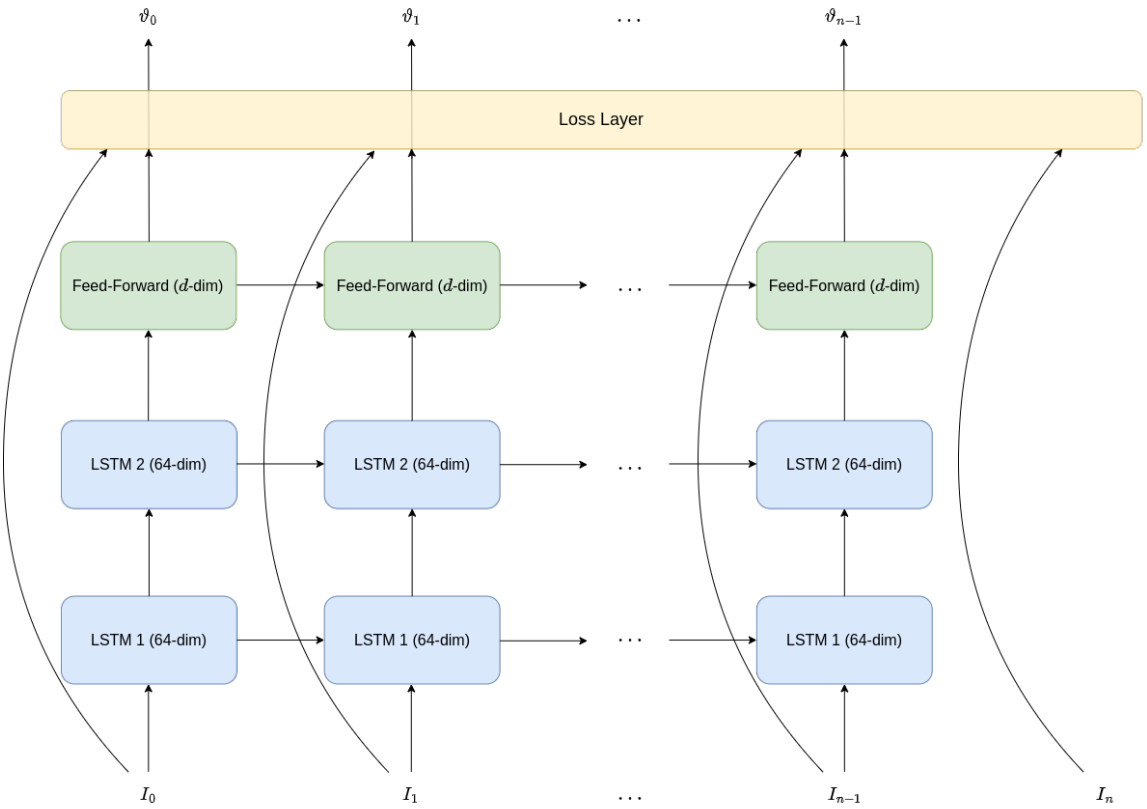

Figure 4.2.: Example architecture of the stacked LSTM model with two hierarchically organized LSTM-Cells ($m = 2$)

as the input vector (to calculate the stochastic integral and cost terms), a layer spanning over the entire unrolled RNN graph is defined to handle loss calculation. This layer also contains the variable c representing the initial capital.

An example structure is depicted graphically in Figure 4.2. Note that while the model only uses $(I_k)_{k\in 1,\dots,n-1}$ to calculate the output strategy $(\vartheta_k)_{k\in 1,\dots,n-1}$, the input I_n is still considered in the loss calculation because it is needed for computing the trading result over the last period in the stochastic integral term.
Technically, the model does output a value for ϑ_n, but as this value is not considered in the loss calculation and does not serve any real purpose, it is replaced with zero in the final output to signify total liquidation in the last time step. This means that the output has the structure $(\vartheta_0, \dots, \vartheta_{n-1}, 0)$.

The LSTM-Cells and Feed-Forward network do not consider possible trading constraints and, as such, output an unconstrained strategy. In the loss layer, before loss calculation, however, the function G is applied to the unconstrained strategy and the results of this are used as final outputs.
Similarly, trading costs are calculated in the loss layer using the function trading_costs and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

are included in the cost function. However they are not part of the final model output. The implementation uses the Keras Functional API to hierarchically organize the LSTM-Cells, Feed-Forward network and loss layer.

The loss layer is implemented via subclassing of the base class tensorflow.keras.layers.Layer. This layer spans over all time steps and contains the trainable variable c. The cost function 4.6 is implemented by the function mvh_loss, which is added to this layer. The loss layer is not only connected to the outputs of the Feed-Forward network but also the inputs, as the asset prices are needed for loss calculation. Finally, the loss layer outputs a constrained trading strategy, with the last component being zero, as discussed above. The combined stacked LSTM model is implemented as a subclass of tensorflow.keras.Model. At initialization, parameters for the input shape, number of LSTM layers, number of hidden units per layer, the portfolio of claims, indices indicating the tradable assets within the input vector, the parameter ϵ for transaction cost calculation and a function G representing the trading restrictions have to be specified.

The claim portfolio is expected to be given as an instance of the dataclass OptionPortfolio, which consists of a list of instances of the class Option and a list of indices indicating the underlying of each option in the input vector. The payoff of the claim portfolio is given as the sum of the outputs of the member functions tf_payoff of all claims in the claim portfolio. These functions expects a batch of paths of the input vectors as inputs and can output either a tensor of rank 1 or a scalar value.

The shape of the input tensor is expected to be (num_samples, num_time_steps, num_infos), with num_samples denoting the number of training samples (N), num_time_steps denoting the number of time steps ($n+1$), and num_infos denoting the number of components of the information vectors (d).

The code of the implementation can be found in Appendix A in Listing A.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5. Numerical Experiments

In this chapter, the approach introduced in chapter 4 is applied to two different asset price models: an exponential Lévy model and a stochastic volatility model. In both cases, the RNN strategy will be compared to a benchmark strategy in various ways to gauge effectiveness and applicability of this approach.

5.1. Exponential Lévy Model

First, the RNN approach is tested on an exponential Lévy model, namely the Normal Inverse Gaussian model. The hedging strategies calculated by the stacked LSTM model will be compared to a discretized version of the optimal mean-variance hedging strategy derived in Theorem 14.

5.1.1. Setting

The Normal Inverse Gaussian model (NIG model) is an asset price model in which the price of the risky asset $S_t = S_0 \exp(X_t)$ is modelled as the exponential of a Normal Inverse Gaussian process $(X_t)_{t\in[0,T]}$. To further define a NIG process, one has to first define the Normal Inverse Gaussian distribution:

Definition 23. *The **Normal Inverse Gaussian (NIG) distribution**, denoted by $NIG(\alpha, \beta, \delta, \mu)$ is a continuous probability distribution defined by the density function*

$$f_{NIG}(x) := \frac{\alpha\delta}{\pi} e^{\delta\sqrt{\alpha^2-\beta^2}+\beta(x-\mu)} \frac{K_1(\alpha\sqrt{\delta^2+(x-\mu)^2})}{\sqrt{\delta^2+(x-\mu)^2}}, \quad x \in \mathbb{R}$$

where K_1 denotes the modified Bessel function of second kind and index 1.
The possible parameter values are $\alpha > 0$, $\beta \in (-\alpha, \alpha - 1)$, $\delta > 0$ and $\mu \in \mathbb{R}$.
[1]

Using this probability distribution, the NIG process can be defined in the following way:

Definition 24. *A stochastic process $(X_t)_{t\in[0,T]}$ is called a **Normal Inverse Gaussian process** if its increments are NIG distributed. This means, that for $h \geq 0$ such that $0 \leq t \leq t + h \leq T$*

$$X_{t+h} - X_t \sim NIG(\alpha, \beta, \delta h, \mu h)$$

This process is a Lévy process with the cumulant function

$$\kappa_{NIG}(z) := \mu z + \delta\left(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+z)^2}\right)$$

[1]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Throughout section 3.2.1 it is assumed that the process $S_t = S_0 \exp(X_t)$ is a square integrable martingale. This is equivalent to the conditions $\kappa(1) = 0$ and $\kappa(2) < \infty$.
From the expressions

$$\kappa(1) = \mu + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta+1)^2}\right)$$

and

$$\kappa(2) = 2\mu + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta+2)^2}\right)$$

one can gather the conditions $\alpha^2 - \beta^2 \geq 0$, $\alpha^2 - (\beta+1)^2 \geq 0$ and $\alpha^2 - (\beta+2)^2 \geq 0$. This leads to the following conditions for the parameters α and β:

$$-\alpha \leq \beta \leq \alpha - 2, \quad \alpha \geq 1 \tag{5.1}$$

where the second condition guarantees that the interval for β in the first condition is not empty.

Next, define the function

$$\phi(\beta) := \mu + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta+1)^2}\right), \quad \beta \in [-\alpha, \alpha - 1]$$

which describes $\kappa(1)$ as a function of β. Plugging in the bounds of the domain of ϕ yields

$$\phi(-\alpha) = \mu - \delta\sqrt{2\alpha + 1}, \quad \phi(\alpha - 1) = \mu + \delta\sqrt{2\alpha + 1}$$

As the derivative

$$\phi'(\beta) = \delta\left(\frac{\beta+1}{\sqrt{\alpha^2 - (\beta+1)^2}} - \frac{\beta}{\sqrt{\alpha^2 - \beta^2}}\right)$$

approaches positive infinity towards the bounds of the domain and only has one root at $\beta = -\frac{1}{2}$, the function is increasing in the interval $[-\alpha, \alpha - 1]$.
Therefore, for $\kappa(1) = \phi(\beta) = 0$ to be true for some β, the interval $[\phi(-\alpha), \phi(\alpha - 1)]$ must contain zero, which leads to the condition

$$|\mu| \leq \delta\sqrt{2\alpha + 1} \tag{5.2}$$

[12]
Now we can solve the equation $\phi(\beta) = 0$ to find the parameters to ensure $S_t = S_0 \exp(X_t)$ is a square-integrable martingale.

Theorem 15. *The stochastic process $S_t = S_0 \exp(X_t)$, where X_t is a NIG-process, is a square-integrable martingale if and only if the conditions 5.1 and 5.2 are satisfied and*

$$\beta = -\frac{1}{2} - \operatorname{sgn}(\mu)\sqrt{\frac{\alpha^2\mu^2}{\mu^2 + \delta^2} - \frac{\mu^2}{4\delta^2}}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Proof: As described above, for $\kappa(1)$ to be zero and $\kappa(2) < \infty$, the conditions 5.1 and 5.2 need to be satisfied and the parameter β has to solve the equation

$$\phi(\beta) = \mu + \delta\left(\sqrt{\alpha^2 - \beta^2} - \sqrt{\alpha^2 - (\beta+1)^2}\right) = 0$$

Rearranging and squaring this equation yields

$$\beta^2 + \beta + \frac{\mu^2}{4\delta^2} - \frac{\alpha^2\mu^2}{\mu^2+\delta^2} + \frac{1}{4} = 0$$

The solutions of this equation are

$$\beta_{1,2} = -\frac{1}{2} \pm \sqrt{\frac{\alpha^2\mu^2}{\mu^2+\delta^2} - \frac{\mu^2}{4\delta^2}} \tag{5.3}$$

Plugging these solutions into the original equation shows that the correct solution depends on the sign of μ and is given by

$$\beta = -\frac{1}{2} - \operatorname{sgn}(\mu)\sqrt{\frac{\alpha^2\mu^2}{\mu^2+\delta^2} - \frac{\mu^2}{4\delta^2}} \tag{5.4}$$

[12]

□

5.1.2. Experiment

For the numerical illustration, a discretized version of the Normal Inverse Gaussian model will be used. For a fixed $n \in \mathbb{N}$ and time horizon $T \geq 0$, the asset price is simulated at the time points $0 = t_0 < \ldots < t_n = T$. This is achieved using the Euler scheme.
The parameters of the Normal Inverse Gaussian process are:

- $\alpha = 75.49$
- $\beta = 0.4984357$
- $\delta = 3.024$
- $\mu = -0.04$

These parameters, with the exception of β, are taken from [14], while β was calculated according to Theorem 15 to guarantee the simulated process being a martingale. These parameters satisfy all conditions in Theorem 15, as one can easily verify. The initial value of the price process will be $S_0 = 100$. For this numerical experiment, transaction costs are assumed to be zero.

The claim to be hedged is an at-the-money call option ($K = S_0 = 100$) expiring in one month ($T = \frac{21}{252}$). The hedging position can be adjusted daily, meaning that $n = 21$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Benchmark Strategy

To assess the performance of the RNN strategy, a benchmark strategy has to be introduced. For this experiment, the mean-variance hedging strategy from Theorem 14 will be used. To reiterate, this means trading in the risky asset using the strategy

$$\xi_t := \int \gamma(z) e^{\kappa(z)(T-t)} S_{t-}^{z-1} \, \Pi(dz) \tag{5.5}$$

and initial capital

$$c := \int e^{\kappa(z)T} S_0^z \, \Pi(dz) \tag{5.6}$$

This would constitute the optimal hedging strategy in the mean-variance sense, if continuous time trading was possible.
As the setting for this experiment is time discrete, the benchmark strategy will be strategy 5.5 evaluated at the discrete time points $0 = t_0 < \ldots < t_N = T$.

The claim to be hedged will be a European Call option with maturity T and strike $K = 100$. Using Theorem 13, the integrals in 5.5 and 5.6 can be rewritten as

$$\xi_k = \frac{1}{2\pi i} \int_{R-i\infty}^{R+i\infty} \gamma(z) e^{\kappa(z)(T-t_k)} S_k^{z-1} \frac{K^{1-z}}{z(z-1)} \, dz \tag{5.7}$$

$$c = \frac{1}{2\pi i} \int_{R-i\infty}^{R+i\infty} e^{\kappa(z)T} S_0^z \frac{K^{1-z}}{z(z-1)} \, dz \tag{5.8}$$

for $R > 1$. These integrals do not depend on the specific choice of R. For numerical evaluation this value is chosen to be $R = 1.1$.

The integral 5.8 must yield a real value, therefore one can equivalently calculate the integral

$$\int_{R-i\infty}^{R+i\infty} \Re\left(\frac{1}{2\pi i} e^{\kappa(z)T} S_0^z \frac{K^{1-z}}{z(z-1)} \right) dz$$

Using the substitution $z = R + iv$ yields

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} \Re\left(\frac{1}{2\pi i} e^{\kappa(R+iv)T} S_0^{R+iv} \frac{K^{1-R-iv}}{(R+iv)(R+iv-1)} \right) dv$$

which can now be calculated numerically using the Python function scipy.integrate.quad.

The same argumentation can be applied to 5.7, yielding

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} \Re\left(\gamma(R+iv) e^{\kappa(R+iv)(T-t_k)} S_{t_k}^{R+iv-1} \frac{K^{1-R-iv}}{(R+iv)(R+iv-1)} \right) dv$$

which can, once again, be calculated numerically using scipy.integrate.quad.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

RNN Strategy

The RNN strategy is computed using similar implementations to the one discussed in section 4.2.3. As there is no definite way to determine the optimal topology of the RNN, several models were trained for a combination of LSTM layers and hidden units. The tested number of layers were 1, 2, 4 and 8, while the tested number of hidden units per layer were 8, 16, 32 and 64. The network input vector at each time point contains only the asset price at that moment, so $I_k = S_{t_k}$.

The stacked LSTM Models are trained on a training set of 70,000 sample paths and evaluated on a validation set of 10,000 sample paths after each epoch. If the validation loss does not decrease for 15 epochs, training is stopped. The trained models are subsequently evaluated on a test set of 20,000 sample paths and compared to the benchmark strategy on this test set.

Results

The performance of each combination of layers and hidden units is evaluated via the loss of the trained model on the test set, with the minimum marked by a box. The results of this are shown in Table 5.1.

Number of hidden units	Number of layers			
	1	2	4	8
8	0.47474253	0.45257995	0.42978394	3.10601521
16	0.44929951	0.45258984	0.4277997	0.42693883
32	0.44305009	0.43092746	0.42477739	0.42760918
64	0.4335461	0.42361307	0.42558199	0.43312702

Table 5.1.: Losses of trained models on the test set (Normal Inverse Gaussian model)

Table 5.1 suggests that two LSTM layers of 64 hidden units each constitute the optimal choice for this problem. If training time and model simplicity are of concern, one might also choose two LSTM layers of 32 hidden units each, as this cuts the number of model parameters (and training time per epoch) roughly in half.
It should be noted that the "8-layer 8-hidden unit" model apparently got stuck in a local minimum and did not decrease the loss function further, leading to an extremely high loss on the test set, while the other "8-layer" models managed to achieve losses which are comparable to the other models.

The optimal model with two layers and 64 hidden units will now be analysed further.
Figure 5.1 compares the distribution of the hedge errors between the benchmark and RNN strategies. While the histograms seem quite similar, one can see that the benchmark strategy hedging errors are concentrated slightly closer to zero than the RNN strategy errors. This can be confirmed by comparing the losses on the test data set: The RNN strategy yields a loss of 0.42361307 while the benchmark strategy performs slightly better at 0.413948. Considering the fact that the benchmark strategy is a discretized version of the true optimal trading strategy, one can conclude that the RNN strategy performs reasonably

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

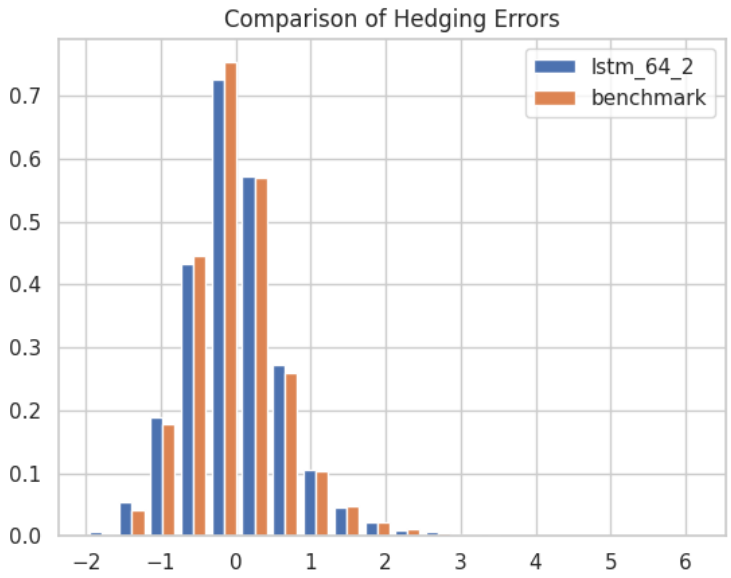

Figure 5.1.: Hedging error comparison between mean-variance hedging benchmark and 2-layer 64-unit stacked LSTM model (Normal Inverse Gaussian model).

well.

In regards to pricing, the two approaches again yield quite similar results, with the benchmark approach pricing the option at 2.29024604, while the RNN approach prices the option slightly higher at 2.2956936. This, combined with the 95% and 99% Value-at-Risk of the

	Benchmark	RNN
c	2.290246	2.295694
Loss	0.413948	0.423613
95% VaR	1.270878	1.283621
99% VaR	2.041777	1.989227

Table 5.2.: Comparison of price, loss, and 95%/99% VaR of absolute hedge errors between the benchmark and RNN strategies on the test data set (Normal Inverse Gaussian model).

absolute hedge error are shown in Table 5.2. A histogram of the the absolute differences of the benchmark and RNN hedge errors for each test data path is depicted in Figure 5.2. From this histogram, one can conclude that the strategies perform similarly well not only aggregated but also on a per-path level.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

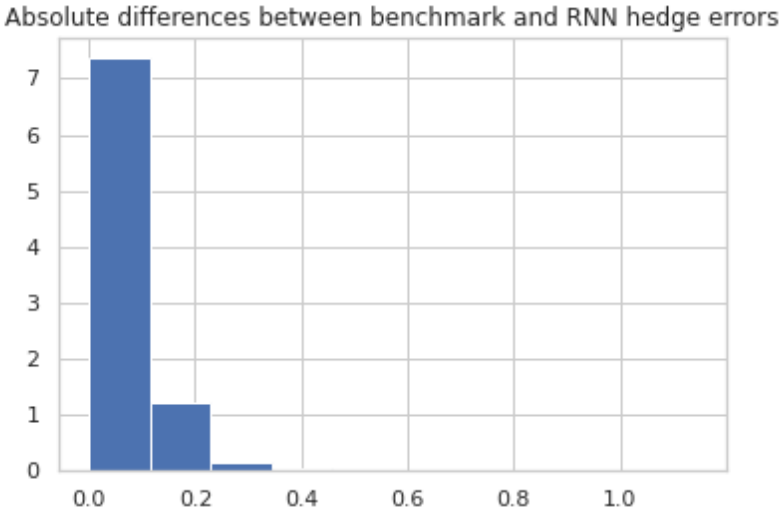

Figure 5.2.: Histogram of absolute hedge error differences for each path (Normal Inverse Gaussian model).

5.2. Exponential Lévy Model - Transaction Costs

The common approaches to solving the mean-variance hedging problem assume absence of transaction costs. As this does not reflect real world markets, the RNN model includes proportional transaction costs in it's loss function as discussed in section 4.1.2. While the proportional transaction costs were set to zero for the previous experiment, now a non-zero proportional cost factor is assumed.

5.2.1. Setting

The same setting as in section 5.1 is assumed, the only exception being the addition of non-zero proportional transaction costs.

5.2.2. Experiment

As before, a discretized version of a Normal Inverse Gaussian model with parameters:

- $\alpha = 75.49$
- $\beta = 0.4984357$
- $\delta = 3.024$
- $\mu = -0.04$

with initial value $S_0 = 100$ is assumed. The proportional trading cost factors are chosen to be constant over time with $c_0 = \ldots = c_n := 2^{-6}$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The benchmark strategy will be the same as in 5.1, a discretized version of the mean-variance hedging strategy from Theorem 14.

again, 16 models are trained for all possible combinations of 1, 2, 4 and 8 LSTM layers and 8, 16, 32 and 64 hidden units per layer. The network input vector at each time point contains only the asset price at that moment, so $I_k = S_{t_k}$.

The stacked LSTM Models are trained on a training set of 70,000 sample paths and evaluated on a validation set of 10,000 sample paths after each epoch. If the validation loss does not decrease for 15 epochs, training is stopped. The trained models are subsequently evaluated on a test set of 20,000 sample paths and compared to the benchmark strategy on this test set.

Results

The performance of each combination of layers and hidden units is evaluated via the loss of the trained model on the test set, with the minimum marked by a box. The results of this are shown in Table 5.3.

Number of hidden units	Number of layers			
	1	2	4	8
8	0.68132752	0.42960787	0.25011209	3.10431433
16	0.67694676	0.24679595	0.24333169	0.20446205
32	0.62059528	0.19986248	0.18782304	0.22110181
64	0.24768192	0.19711398	0.18369022	0.18988693

Table 5.3.: Losses of trained models on the test set (Normal Inverse Gaussian model)

According to table 5.3, 4 LSTM layers of 64 hidden units each constitute the optimal topology choice for this problem with a test loss of 0.18369022. This table also shows that when including transaction costs into the optimization problem, a larger number of layers and hidden units is required to reach acceptable minima. For example, the models with only one LSTM layer (with the exception of the "1-layer 64-unit" model) only achieve test losses between 0.6 and 0.7, while adding layers and hidden units per layer drastically decreases the test loss. However, this effect seems to be capped at four layers, as the 8 layer models (with the exception of the "8-layer 32-unit model") do not perform better then the 4 layer models with the same number of hidden units. Again, the "1 layer 8 unit" model fails to converge properly and only reaches a test loss of 3.10431433.

Comparing the RNN strategy to the benchmark strategy, one can easily see that the RNN strategy vastly outperforms the benchmark. As the benchmark does not factor in transaction costs, the hedge errors are shifted to the right, as transaction costs have a positive sign in the hedge error calculation. However, even when adjusting the price of the benchmark strategy to the one f the RNN strategy, which includes transaction costs, it still performs significantly worse than the LSTM model. This is shown graphically in Figure 5.3.
The difference in the hedge errors does not only stem from "bad" pricing, as even when considering only the variance of the hedge error, the RNN strategy hedge errors have a vari-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 5.3.: Hedging error comparison between mean-variance hedging benchmark and 4-layer 64-unit stacked LSTM model (Normal Inverse Gaussian model with transaction costs).

ance of 0.183257 while the (adjusted) benchmark hedge errors have a variance of 1.006040. This suggests that the the RNN model actually includes transaction cost considerations into the strategy on a per-path level and does not only charge a higher price to compensate for the average cost increase.

However, the price determined by the LSTM model is still significantly higher than in the "no-cost" setting at 6.504342, compared to 2.295694 (LSTM without transaction costs) and 2.290246 (benchmark). Losses, prices and 95% and 99% Value-at-Risk of the absolute hedge error are summarized in Table 5.4

	Benchmark	Benchmark with RNN price	RNN
c	2.290246	6.504342	6.504342
Loss	8.727918	3.066037	0.183690
95% VaR	4.702037	2.677056	0.806843
99% VaR	5.907022	3.044445	1.989227

Table 5.4.: Comparison of price, loss, and 95%/99% VaR of absolute hedge errors between the benchmark and RNN strategies on the test data set (Normal Inverse Gaussian model).

Quite surprisingly, when including transaction costs, the model achieves a significantly lower test loss than when not including them. Figure 5.4 shows a histogram of the hedge errors of the optimal model from 5.1 and the "4-layer 64-unit" model discussed in this section. As one can easily see, the model including transaction costs has hedge errors

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

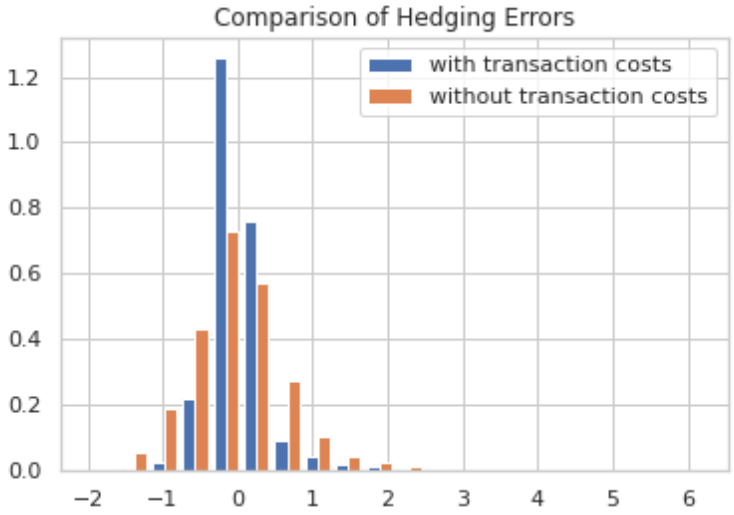

Figure 5.4.: Hedging error comparison between the RNN strategies with and without trading costs (Normal Inverse Gaussian model).

with lower variance than the "no cost" model. This is also evident from the test losses of 0.42361307 and 0.18369022 for the "cost" and "no cost" models respectively.

5.3. Stochastic Volatility Model

While exponential Lévy models are useful asset price models, particularly in this setting, as the optimal mean-variance hedging strategy can be computed relatively easily, other classes of asset price models offer features and properties that better reflect real-life markets. One of this classes are stochastic volatility models, in which the asset's volatility is not assumed to be constant or deterministic over time, but usually modelled as a separate stochastic process. The standard stochastic volatility model is the Heston model [11], which resembles the Black-Scholes-Merton Model, with the addition of the volatility not being constant, but instead modelled as a Cox-Ingersoll-Ross process, with the Brownian motions driving both the asset and volatility processes being, in general, correlated. The Heston model, although well known and widely used, still has shortcomings, for example that the volatility process is not bounded away from zero. For this reason, the 4/2 stochastic volatility model [9], in which the volatility is uniformly bounded away from zero, will be used in the following numerical experiments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.3.1. Setting

In the 4/2 stochastic volatility model, the risk-neutral dynamics of the asset price $(S_t)_{t\in[0,T]}$ are given by the SDE

$$\frac{dS_t}{S_t} = rdt + \left(a\sqrt{V_t} + \frac{b}{\sqrt{V_t}}\right) dZ_t \tag{5.9}$$

where the process $(V_t)_{t\in[0,T]}$ follows the dynamics

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t}\, dW_t \tag{5.10}$$

for parameters $r, \kappa, \theta, \sigma > 0$ and $a, b \in \mathbb{R}$. The processes Z and W are Brownian motions with instantaneous correlation $d\langle W, Z\rangle_t = \rho dt$ for $\rho \in [-1, 1]$. Additionally, if $b \neq 0$, assume that the Feller condition $2\kappa\theta \geq \sigma^2$ is satisfied.

The volatility term in 5.9 is a linear combination of the volatility term in the Heston model and the volatility term in the 3/2 stochastic volatility model [10].

As this thesis only treats mean-variance hedging for martingales, the discounted asset price process $\tilde{S}_t := S_t e^{-rt}$ is required to be martingale. According to [9], the process $\tilde{S}_t$ is a true martingale if the Feller conditions are satisfied both under the risk-neutral and historical measures, which leads to the following two conditions:

- $2\kappa\theta \geq \sigma^2$
- $2\kappa\theta + 2\rho\sigma b \geq \sigma^2$

By choosing parameters that satisfy these conditions, the discounted asset price process is guaranteed to be a true martingale.

5.3.2. Experiment

For the numerical illustration, a discretized version of the 4/2 stochastic volatility model will be used. For a fixed $n \in \mathbb{N}$ and time horizon $T \geq 0$, the asset price and volatility are simulated at the time points $0 = t_0 < \ldots < t_n = T$. This is achieved using the Euler scheme with correlated standard normal samples for the two processes.
The parameters are:

- $\kappa = 10$
- $\theta = 0.02$
- $\sigma = 0.2$
- $a = 0.5$
- $b = 0.0002$
- $r = 0$
- $\rho = -0.7$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

As one can easily verify, these parameters satisfy both conditions to guarantee the discounted asset price process to be a martingale.
The initial values for the price and volatility process are $S_0 = 100$ and $V_0 = 0.02$, respectively. For this numerical experiment, transaction costs are assumed to be zero.

The claim to be hedged is an at-the-money call option ($K = S_0 = 100$) expiring in one month ($T = \frac{21}{252}$). The hedging position can be adjusted daily, meaning that $n = 21$.

Benchmark Strategy

The benchmark strategy for this numerical experiment will be a **delta hedging** strategy. This means that the holdings of the underlying asset correspond to the sensitivity of the value process H_t of the claim H w.r.t. changes in the price of the underlying. This quantity will be denoted by $\Delta(t, s, v) := \frac{\partial H_t}{\partial S_t}\Big|_{S_t=s, V_t=v}$.
The benchmark strategy is therefore given by

$$\xi_t := \Delta(t, S_t, V_t)$$

To calculate this strategy, an expression for the value process of the claim H is necessary, which can then be evaluated twice to numerically approximate the partial derivative w.r.t. the price of the underlying.
As elaborated in [9], the price of a payoff $F(Y_T)$ with $Y_T = \log(S_T)$ can be determined using the following formula:

$$\begin{aligned} e^{-r(T-t)}\mathbb{E}\left[F(Y_T) \mid \mathcal{F}_t\right] &= \frac{e^{-r(T-t)}}{2\pi}\mathbb{E}\left[\int_{\mathcal{Z}} e^{-izY_T}\left(\int_{\mathbb{R}} e^{izy}F(y)dy\right)dz \;\middle|\; \mathcal{F}_t\right] \\ &= \frac{e^{-r(T-t)}}{2\pi}\int_{\mathcal{Z}} \Psi_{t,T}(-iz)\hat{F}(z)\; dz \end{aligned} \tag{5.11}$$

where $\hat{F}$ denotes the generalized Fourier transform of F, $\mathcal{Z}$ denotes the strip of regularity of this transform and $\Psi_{t,T}$ denotes the conditional generalized characteristic function of the log-price:

$$\Psi_{t,T}(u) := \mathbb{E}\left[e^{uY_T} \mid \mathcal{F}_t\right]$$

for $u \in D_{t,T} \subseteq \mathbb{C}$, which denotes the domain in which the function is well-defined.

As proven in [9], the conditional generalized characteristic function $\Psi_{0,t}$ is given by

$$\begin{aligned} \Psi_{0,t}(u) = &\exp\left\{uY_0 + \frac{\kappa^2\theta}{\sigma^2}t + u\left(r - ab - \frac{a\rho\kappa\theta}{\sigma} + \frac{b\rho\kappa}{\sigma}\right)t + u^2(1-\rho^2)abt\right\} \\ &\cdot \left(\frac{\sqrt{A_u}}{\sigma^2\sinh\left(\frac{\sqrt{A_u}}{2}t\right)}\right)^{m_u+1} V_0^{\frac{1}{2}+\frac{m_u}{2}-\frac{\kappa\theta}{\sigma^2}-\frac{ub\rho}{\sigma}} \left(K_u(t) - \frac{ua\rho}{\sigma}\right)^{-(\frac{1}{2}+\frac{m_u}{2}+\frac{\kappa\theta}{\sigma^2}+\frac{ub\rho}{\sigma})} \\ &\cdot \exp\left\{\frac{V_0}{2}\left(-\sqrt{A_u}\coth\left(\frac{\sqrt{A_u}t}{2}\right) + \kappa - ua\rho\sigma\right)\right\} \frac{\Gamma(\frac{1}{2}+\frac{m_u}{2}+\frac{\kappa\theta}{\sigma^2}+\frac{ub\rho}{\sigma})}{\Gamma(m_u+1)} \\ &\cdot {}_1F_1\left(\frac{1}{2}+\frac{m_u}{2}+\frac{\kappa\theta}{\sigma^2}+\frac{ub\rho}{\sigma}, m_u+1, \frac{A_uV_0}{\sigma^4\sinh^2(\frac{\sqrt{A_u}}{2}t)\left(K_u(t)-\frac{ua\rho}{\sigma}\right)}\right) \end{aligned}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

with

$$A_u := \kappa^2 - 2\sigma^2\left(u\left(\frac{a\rho\kappa}{\sigma} - \frac{1}{2}a^2\right) + \frac{1}{2}u^2(1-\rho^2)a^2\right)$$

$$m_u := \frac{2}{\sigma^2}\sqrt{\left(\kappa\theta - \frac{\sigma^2}{2}\right)^2 - 2\sigma^2\left(u\left(\frac{b\rho}{\sigma}\left(\frac{\sigma^2}{2} - \kappa\theta\right) - \frac{b^2}{2}\right) + \frac{u^2}{2}(1-\rho^2)b^2\right)}$$

$$K_u(t) := \frac{1}{\sigma^2}\left(\sqrt{A_u}\coth\left(\frac{\sqrt{A_u}}{2}t\right) + \kappa\right)$$

Here, ${}_1F_1$ denotes the hypergeometric confluent function:

$${}_1F_1(a;b;z) := \sum_{n=0}^{\infty}\frac{a^{(n)}z^n}{b^{(n)}n!}$$

where $a^{(0)} = 1$ and $a^{(n)} = a(a+1)(a+2)\dots(a+n-1)$.

From this expression for Ψ_0, t, the general formula for $\Psi_{t,T}$ can be deduced easily.

To calculate the delta hedging strategy, the pricing formula 5.11 can be interpreted as a function u of t, S_t, and V_t, which can in turn be numerically evaluated at the points $(t, S_t - h, V_t)$ and $(t, S_t + h, V_t)$ for small $h > 0$ to calculate the symmetric difference quotient

$$\Delta(t, S_t, V_t) \approx \hat{\Delta}(t, S_t, V_t) := \frac{u(t, S_t + h, V_t) - u(t, S_t - h, V_t)}{2h}$$

as an approximation of the claim's delta. This approximation will be used as the benchmark strategy

$$\xi_t = \hat{\Delta}(t, S_t, V_t)$$

For this experiment, a value of $h = 10^{-4}$ was used. Expressions for the generalized Fourier transform of the payoff function and the strip $\mathcal{Z}$ were taken from [17].

RNN Strategy

As in section 5.1, 16 models are trained for all possible combinations of 1, 2, 4 and 8 LSTM layers and 8, 16, 32 and 64 hidden units per layer. The RNN models are implemented using, again, an implementation similar to the one discussed in 4.2.3. The network input vector at each time point contains only the asset price at that moment, so $I_k = S_{t_k}$.

As before, the stacked LSTM Models are trained on a training set of 70,000 sample paths and evaluated on a validation set of 10,000 sample paths after each epoch. If the validation loss does not decrease for 15 epochs, training is stopped. The trained models are subsequently evaluated on a test set of 20,000 sample paths and compared to the benchmark strategy on this test set.

Results

The losses on the test set of the 16 trained models are shown in table 5.5, with the minimum marked by a box. According to the data shown in table 5.5, the optimal model topology

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Number of hidden units	Number of layers			
	1	2	4	8
8	0.04894227	0.03936565	0.02951627	0.39152917
16	0.03715733	0.0336134	0.03005624	0.02991065
32	0.03527585	0.03319656	0.02878462	0.39135444
64	0.0340624	0.03063314	0.02815739	0.39146277

Table 5.5.: Losses of trained models on the test set (4/2 stochastic volatility model)

for this specific problem consists of four LSTM layers with 64 units each. However, should model simplicity and per-epoch training time be of concern, the "4-layer 32-unit" and "4-layer 16-unit" models offer comparable performance at a significant lower number of parameters. In general, the "4-layer" models seem to perform best for this specific setting, achieving four of the five lowest losses.
Again, almost all of the models with eight LSTM layers seem to get stuck in "bad" local minimums and only reach very poor test dataset losses. The "8-layer 16-unit" model, however, converges properly and achieves the fourth lowest test dataset loss overall.

As the "4-layer 64-unit" model achieved the lowest test dataset loss, the following analysis will focus on this model. Figure 5.5 compares the distribution of the hedge errors between the benchmark and RNN strategies. This figure shows, that the distribution of hedge errors is quite similar between the benchmark and RNN strategies, with the RNN strategy hedge errors appearing to be of smaller variance than the benchmark hedge errors. This is confirmed by the fact, that the RNN strategy test dataset loss is 0.02815739, while the benchmark test dataset loss is slightly higher at 0.028832. Therefore, the RNN strategy outperforms the benchmark strategy w.r.t. the mean-variance hedging criterion.

The two approaches yield similar prices, with the benchmark approach pricing the call option at 0.827093, while the RNN prices it slightly higher at 0.820873. Losses, prices and 95% and 99% Value-at-Risk of the absolute hedge error are summarized in Table 5.6

	Benchmark	RNN
c	0.827093	0.820873
Loss	0.028832	0.028157
95% VaR	0.340705	0.339834
99% VaR	0.476324	0.493351

Table 5.6.: Comparison of price, loss, and 95%/99% VaR of absolute hedge errors between the benchmark and RNN strategies on the test data set (4/2 stochastic volatility model).

Again, the two approaches do not only perform similarly when aggregated, but also on a per-path level, which can be seen in Figure 5.6, which depicts a histogram of the the absolute differences of the benchmark and RNN hedge errors for each test data path.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

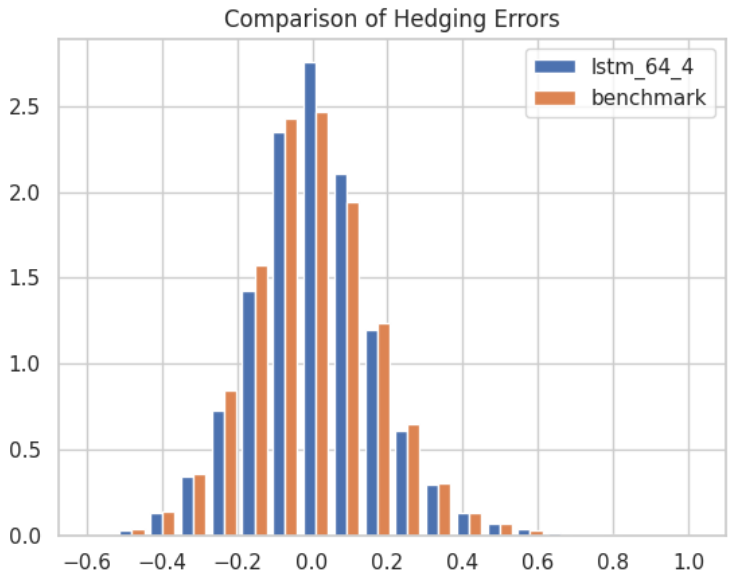

Figure 5.5.: Hedging error comparison between mean-variance hedging benchmark and 4-layer 64-unit stacked LSTM model (4/2 stochastic volatility model).

5.4. Stochastic Volatility Model - Trading in additional Option

If one can only trade in the risky asset S, the markets generated by stochastic volatility models like the Heston and 4/2 model are incomplete, as there are two sources of randomness present, but only one tradable asset. However, under some conditions, the market can be completed by allowing trading in a second option written on S.

This motivates the extension of the previous setting to also allow trading in another option with longer maturity.

5.4.1. Setting

The same setting as in section 5.3 is assumed. This means that the dynamics for the price and volatility processes are given by 5.9 and 5.10, respectively and that the conditions guaranteeing the discounted price process to be a martingale are satisfied.

5.4.2. Experiment

As before, a discretized version of the 4/2 stochastic volatility model with parameters

- $\kappa = 10$
- $\theta = 0.02$
- $\sigma = 0.2$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

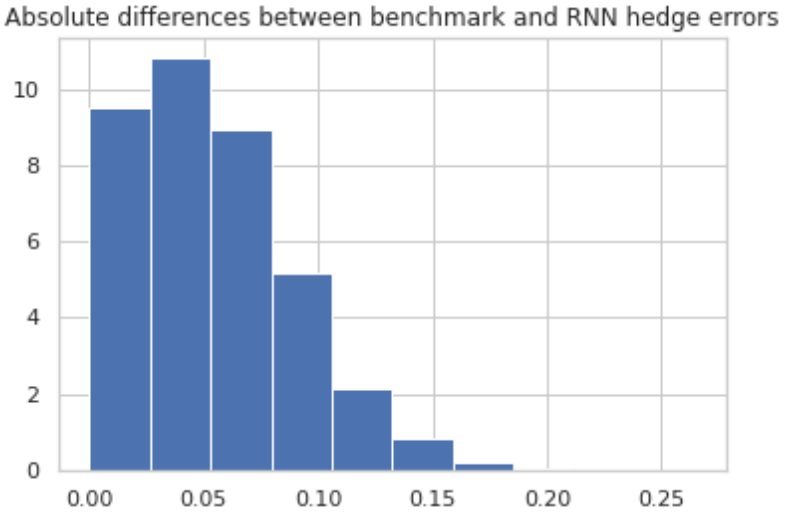

Figure 5.6.: Histogram of absolute hedge error differences for each path (4/2 stochastic volatility model).

- $a = 0.5$
- $b = 0.0002$
- $r = 0$
- $\rho = -0.7$

with initial values $S_0 = 100$ and $V_0 = 0.02$ will be used and transaction costs are assumed to be zero.
Again, the claim to be hedged is an at-the-money call option ($K = S_0 = 100$) expiring in one month ($T = \frac{21}{252}$), with daily adjustment in the hedging position ($n = 21$). Additionally, trading in an at-the-money call option with a maturity $T' = 2T$ of 2 months is possible. This claim will be denoted by F and its value process by F_t. The value process of this option was calculated by numerically evaluating the pricing formula 5.11 at each time step for each path.

Benchmark Strategy

The benchmark strategy for this numerical experiment will be a **delta-sigma hedging** strategy.
To better illustrate the concept of delta-sigma hedging, consider the hedging portfolio consisting of -1 units of claim H, Δ units of the asset S and Σ units of the second claim F. A delta-sigma strategy aims to eliminate both the delta (sensitivity of option price w.r.t. price change of underlying) and vega (sensitivity of option price w.r.t. volatility change of underlying) of the hedging portfolio.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The vega of the claim H will be denoted by $\mathcal{V}^H(t,s,v) := \frac{\partial H_t}{\partial V_t}\Big|_{S_t=s,V_t=v}$, and similarly, $\mathcal{V}^F$ will denote the vega of the claim F. In an analogous manner, Δ^H and Δ^F denote the deltas of the claims H and F, respectively.

Eliminating the portfolio vega is achieved by holding an amount Σ of the traded option equal to the ratio of the vega of the option to be hedged and the vega of the traded:

$$\Sigma := \frac{\mathcal{V}^H}{\mathcal{V}^F}$$

The holdings in the risky asset S are now chosen to offset both the delta of the holdings in H as well as in F:

$$\Delta := \Delta^H - \Sigma\Delta^H$$

This will constitute the benchmark strategy for the following experiment. The values of Δ^H and Δ^F can be approximated as discussed in 5.3.2, while $\mathcal{V}^H$ and $\mathcal{V}^F$ can similarly approximated by

$$\mathcal{V}^{H/F}(t,S_t,V_t) \approx \hat{\mathcal{V}}^{H/F}(t,S_t,V_t) := \frac{u^{H/F}(t,S_t,V_t+h) - u^{H/F}(t,S_t,V_t-h)}{2h}$$

The chosen values for h were $h = 10^{-4}$ for the delta approximation and $h = 10^{-6}$ for the vega approximation.

RNN Strategy

Once again, 16 models for are trained for all possible combinations of 1, 2, 4 and 8 LSTM layers and 8, 16, 32 and 64 hidden units per layer. The RNN models are implemented using, again, a implementation similar to the one discussed in 4.2.3. The network input vector at each time point now consists of both the underlying asset price as well as the value of the traded option at that moment, so $I_k = (S_{t_k}, F_{t_k})^T$.

As before, the stacked LSTM Models are trained on a training set of 70,000 sample paths and evaluated on a validation set of 10,000 sample paths after each epoch. If the validation loss does not decrease for 15 epochs, training is stopped. The trained models are subsequently evaluated on a test set of 20,000 sample paths and compared to the benchmark strategy on this test set.

Results

The losses on the test set of the 16 trained models are shown in table 5.5, with the minimum marked by a box. According to this table, the optimal network topology consists of 4 LSTM layers with 32 hidden units each. Alternatively, 2 layers of 16 hidden unit LSTM cells may also be a viable choice, as this model performs only slightly worse, while being significantly smaller in both number of layers and number of hidden units (and therefore parameters)

Once again, almost all “8 layer” models appear to get stuck in suboptimal local minima of the loss function and fail to reach test losses comparable to the other models. Interestingly, the “8 layer 64 units” model, which is the largest tested model by number of parameters,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Number of hidden units	Number of layers			
	1	2	4	8
8	0.00554427	0.00476589	0.0046732	0.02454056
16	0.00492431	0.00464781	0.00467911	0.02456822
32	0.00484235	0.00470393	0.00464131	0.02458838
64	0.00495739	0.00466561	0.00469984	0.00467254

Table 5.7.: Losses of trained models on the test set (4/2 stochastic volatility model with trading in second option)

achieves the fourth lowest overall test loss. Judging by this, the poor performance of the other "8-layer" models may stem from the fact that the gradient descent algorithm only achieves slight loss improvements for larger models, which can in turn lead to the validation loss not decreasing for several epochs. A larger choice of the number of epochs without validation loss to stop training (also referred to as "patience") may cause these models to achieve better performance. However, failure to converge properly for an already relatively large patience parameter of 15 indicates that these models might not be robust enough to be considered.

The following discussions will only consider the "4-layer 32-unit" model.
Figure 5.7 compares the distribution of the hedge errors between the benchmark and RNN strategies. As one can easily see, the RNN strategy achieves significantly lower hedge errors than the benchmark delta-sigma strategy. This is confirmed by the test losses, which are 0.010548 for the benchmark strategy and only 0.004641 for the RNN strategy. The delta-sigma strategy still outperforms the delta strategy from section 5.3 which achieved a test loss of 0.028832. The reason for the significantly better performance of the RNN strategy may stem from the fact, that the strategies are compared in a discrete time setting This possible affects the performance of the delta-sigma strategy, which, in reality, is a continuous time strategy, more than the RNN strategy. Further analysis of these two strategies with more frequent rebalancing (and therefore smaller discretization error) should be conducted.

Both approaches yield very similar option prices: 0.827093 for the benchmark approach and 0.827172 for the RNN approach. Unsurprisingly given the significantly lower test loss, the RNN strategy also achieves significantly lower VaRs for the absolute hedge errors. Losses, prices and 95% and 99% Value-at-Risk of the absolute hedge error are summarized in Table 5.8.

5.5. Hardware, Software and Runtimes

This section contains information about the used hardware, software and the runtime of the scripts used in for the numerical experience.

All numerical experiments were run on a Linux desktop PC with an Intel i5 3750K CPU running the operating system Manjaro Linux.
The code for the experiments was written mostly in Python, the only exception being a

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

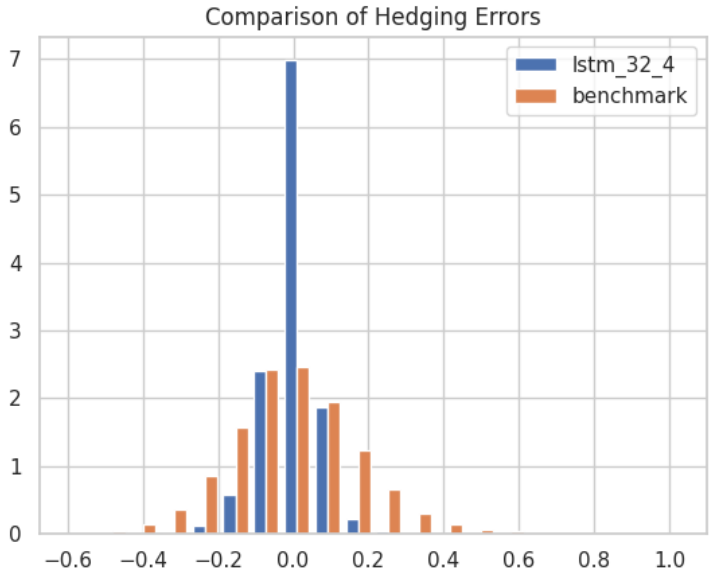

Figure 5.7.: Hedging error comparison between mean-variance hedging benchmark and 4-layer 32-unit stacked LSTM model (4/2 stochastic volatility model with trading in second option).

	Benchmark	RNN
c	0.827093	0.827172
Loss	0.010548	0.004641
95% VaR	0.224016	0.145148
99% VaR	0.334520	0.223310

Table 5.8.: Comparison of price, loss, and 95%/99% VaR of absolute hedge errors between the benchmark and RNN strategies on the test data set (4/2 stochastic volatility model with trading in second option).

small library written in C for the numerical integration in the pricing formula 5.11. Python version 3.10.8 was used with the following versions of major packages

- keras==2.11.0
- numpy==1.22.3
- scipy==1.8.0
- tensorflow==2.11.0

The following table contains the runtimes for different tasks for the numerical experiments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Task	Runtime
NIG RNN Training	4:23:53
NIG RNN Training (transaction costs)	6:37:01
4/2 RNN Training	5:20:51
4/2 RNN Training (additional option)	5:11:18
NIG Benchmark	2:53:55
4/2 Benchmark	27:06:22

Table 5.9.: Runtimes of numerical experiment tasks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6. Conclusion

This thesis illustrates that deep hedging approaches, specifically ones utilizing findings from the field of recurrent neural networks, are a viable alternative to traditional hedging strategies. Neural-network based approaches are not only greek-free but also do not require prior asset price model selection and calibration. Due to widely available libraries like TensorFlow, Keras and others, model training and prediction can be achieved in a highly efficient and scalable manner.

From the numerical experiments it is apparent that the stacked LSTM networks can achieve comparable, if not better, hedging performance than widely used benchmark strategies, depending on the specific setting. A key observation from these experiments is that hierarchically organizing multiple LSTM cells does improve the model's performance, as the optimal model topology for each experiment consisted of either two or four LSTM layers. However, this effect appears to be capped, as the models with eight LSTM layers did not outperform the optimal models and in some cases even failed to properly minimize the loss function. Additionally, more hidden units per layer seem to be beneficial for the models performance, as the optimal models used either 32 or 64 hidden units, which are the two largest numbers of hidden units tested in the numerical experiments.
A surprising result from section 5.2 was that introducing transaction costs to the setting allowed the RNN strategy to achieve a significantly lower loss than when no transaction costs were present, which was not replicated by the benchmark strategy.

Further research could focus on applying the techniques introduced in this thesis to real world markets. While real world phenomena like trading costs and other market friction can be included into the RNN model, other problems arise, mainly the availability of sample paths for model training, testing and validation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

A. Code

The code used for the numerical experiments was mostly written in Python. The implementation of the Stacked LSTM model, option pricers for benchmark calculation, sample asset price path generators and more were all combined into a Python package called mvhrnn. The structure of the mvhrnn source code is as follows:

```
mvhrnn
├── __init__.py
├── assetmodels.py
├── options.py
├── stackedlstm.py
└── utils.py
```

The code in each of these files is listed below.

Listing A.1: Source code of assetmodels.py

```python
from abc import ABC, abstractmethod
import numpy as np
from tqdm import tqdm
from scipy.stats import norminvgauss

class AssetModel(ABC):

    @property
    @abstractmethod
    def expected_params(self):
        pass

    def __init__(self, params, expected_params):
        if params.keys() != expected_params:
            raise ValueError("The expected parameters were not provided.")
        self.params = params

    @abstractmethod
    def simulate_paths(self, n, N, T):
        pass


class FourHalvesModel(AssetModel):

    expected_params = {"S0",
                       "V0",
                       "r",
                "kappa",
                "theta",
                "sigma",
                "a",
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
32                 "b",
33                 "corr",
34                 "d"}
35
36     def __init__(self, params):
37         super().__init__(params, self.expected_params)
38
39         if params["corr"].shape != (2 * params["d"], 2* params["d"]):
40             raise ValueError("corr does not have the expected shape (2d, 2d).")
41
42     def simulate_paths(self, n, N, T):
43         v_paths = np.zeros((N, n+1, self.params["d"]), 'float32')
44         s_paths = np.zeros((N, n+1, self.params["d"]), 'float32')
45
46         dt = T/n
47
48         S_t = self.params["S0"] * np.ones((N,self.params["d"]), 'float32')
49         V_t = self.params["V0"] * np.ones((N,self.params["d"]), 'float32')
50
51         for t in tqdm(range(n+1)):
52             if t == 0:
53                 v_paths[:,t, :] = V_t
54                 s_paths[:,t, :] = S_t
55                 continue
56             Z = np.random.multivariate_normal(
57                                     mean=np.zeros(2*self.params["d"]),
58                                     cov=self.params["corr"],
59                                     size=(N)
60                                     ) * np.sqrt(dt)
61
62             ab_term = self.params["a"]*np.sqrt(V_t) \
63                       + self.params["b"]/np.sqrt(V_t)
64
65             S_t = S_t*np.exp((self.params["r"]-0.5* ab_term*ab_term)*dt \
66                   + (ab_term)*Z[:,0:self.params["d"]])
67
68             V_t = V_t + self.params["kappa"]*(self.params["theta"] -  V_t)*dt \
69                   + self.params["sigma"] * np.sqrt(V_t)*Z[:,self.params["d"]:]
70
71             s_paths[:,t, :] = S_t
72             v_paths[:,t, :] = V_t
73
74         return s_paths, v_paths
75
76
77
78 class NormInvGaussModel(AssetModel):
79
80     expected_params = {"S0", "alpha", "delta", "mu", "beta"}
81
82     def __init__(self, params):
83         super().__init__(params, self.expected_params)
84
85     def simulate_paths(self, n, N, T):
86         a = self.params["alpha"] * self.params["delta"]
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
87          b = self.params["beta"] * self.params["delta"]
88          loc = self.params["mu"]
89          scale = self.params["delta"]
90
91          r = norminvgauss.rvs(a*(T/n),
92                               b*(T/n),
93                               loc*(T/n),
94                               scale*(T/n),
95                               size=n*N,
96                               random_state=None
97                              ).reshape(N,n,1).astype('float32')
98
99          r = np.cumsum(r, axis=1)
100         r = np.concatenate((np.zeros(shape=(N,1,1),dtype="float32"), r),axis=1)
101
102         return self.params["S0"] * np.exp(r)
103
104     def kappa(self, z):
105         return self.params["mu"]*z \
106                + self.params["delta"]*(np.sqrt(self.params["alpha"]**2 \
107                - self.params["beta"]**2) - np.sqrt(self.params["alpha"]**2 \
108                - (self.params["beta"] + z)**2))
109
110     def gamma(self, z):
111         return (self.kappa(z+1) - self.kappa(z)) / self.kappa(2)
112
113 nig_default_params = {
114     "S0" : 100,
115     "alpha" : 75.49,
116     "delta" : 3.024,
117     "mu" :  -0.04,
118     "beta" : 0.4984357
119 }
120
121 fh_default_params = {
122     "S0": 100,
123     "V0": 0.02,
124     "r": 0,
125     "kappa": 10,
126     "theta": 0.02,
127     "sigma": 0.2,
128     "a": 0.5,
129     "b": 0.0002,
130     "corr" :np.array(
131         [[1, -0.7],
132         [-0.7, 1]]
133     ),
134     "d": 1
135 }
```

Listing A.2: Source code of options.py

```
1 from abc import ABC, abstractmethod
2 import numpy as np
3 from mvhrnn.assetmodels import FourHalvesModel, fh_default_params
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
from mvhrnn.assetmodels import NormInvGaussModel, nig_default_params
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf


from scipy import LowLevelCallable
from scipy.integrate import quad
from ctypes import CDLL, c_double, c_int, POINTER

from pydantic.dataclasses import dataclass

import matplotlib.pyplot as plt
import seaborn as sns

from pathos.multiprocessing import Pool
from tqdm import tqdm

class EuropeanOption(ABC):
    @abstractmethod
    def payoff(self, x):
        pass

    @abstractmethod
    def tf_payoff(self, x):
        pass

class EuropeanCallOption(EuropeanOption):
    def __init__(self, T, strike):
        super().__init__()
        self.T = T
        self.strike = strike

    def payoff(self, x):
        x = np.array(x)
        if len(x.shape) > 1:
            return np.maximum(x[:,-1] - self.strike,0)
        else:
            return np.maximum(x - self.strike,0)

    def tf_payoff(self, x):
        return tf.math.maximum(x[:,-1] - self.strike,0)

    def __repr__(self):
        return f"EuropeanCallOption(T={self.T}, strike={self.strike})"

class EuropeanPutOption(EuropeanOption):
    def __init__(self, T, strike):
        super().__init__()
        self.T = T
        self.strike = strike

    def payoff(self, x):
        x = np.array(x)
        if len(x.shape) > 1:
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
59              return np.maximum(self.strike - x[:,-1],0)
60          else:
61              return np.maximum(self.strike - x,0)
62
63      def tf_payoff(self, x):
64          return tf.math.maximum(self.strike - x[:,-1],0)
65
66      def __repr__(self):
67          return f"EuropeanPutOption(T={self.T},␣strike={self.strike})"
68
69  @dataclass(config=dict(arbitrary_types_allowed=True))
70  class OptionPortfolio:
71      options: list[EuropeanOption] | EuropeanOption
72      underlying_idx: list[int] | None = None
73
74      def __post_init__(self):
75          if isinstance(self.options, EuropeanOption):
76              self.options = [self.options]
77          if self.underlying_idx is None:
78              self.underlying_idx = list(range(len(self.options)))
79
80          assert len(self.options) == len(self.underlying_idx)
81
82  class OptionPricer(ABC):
83      @abstractmethod
84      def price(self):
85          pass
86
87      def __init__(self):
88          self.price = np.vectorize(self.price)
89
90  class FourHalvesPricer(OptionPricer):
91      def __init__(self, _option, _model, h_s = 1E-4, h_v=1E-6):
92          super().__init__()
93
94          assert isinstance(_model, FourHalvesModel)
95          assert (_model.params["d"] == 1)
96
97          self.lib_path = os.path.abspath('/path/to/libfourhalvesintegrand.so')
98
99          self.h_s = h_s
100         self.h_v = h_v
101
102         self.option = _option
103         self.model = _model
104
105     def get_integrand(self):
106         lib = CDLL(self.lib_path)
107         if isinstance(self.option, EuropeanCallOption):
108             integrand = lib.integrand_call
109         elif isinstance(self.option, EuropeanPutOption):
110             integrand = lib.integrand_call
111
112         integrand.restype = c_double
113         integrand.argtypes = (c_int, POINTER(c_double))
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
114         return integrand
115
116     def price(self, t, s, v):
117         return quad(
118             LowLevelCallable(self.get_integrand()),
119             a = -5000,
120             b = 5000,
121             limit = 500,
122             args =(t,
123                    self.option.T,
124                    s,
125                    v,
126                    self.model.params["kappa"],
127                    self.model.params["theta"],
128                    self.model.params["sigma"],
129                    self.model.params["a"],
130                    self.model.params["b"],
131                    self.model.params["corr"][0,1],
132                    self.model.params["r"],
133                    self.option.strike
134             )
135         )[0] \
136         * (np.exp(-self.model.params["r"]*(self.option.T - t)) / (2*np.pi))
137
138     def map_price(self, arr):
139         return self.price(arr[0], arr[1], arr[2])
140
141     def mp_map_price(self, arr):
142         with Pool() as pool:
143             results_value = list(tqdm(pool.imap(self.map_price, arr),
144                                       total=len(arr),
145                                       smoothing=0
146                                       )
147                                  )
148
149         return np.stack(results_value)
150
151     def delta(self, t, s, v):
152         return (self.price(t, s + self.h_s, v) \
153                 - self.price(t ,s - self.h_s, v) \
154                 ) / (2*self.h_s)
155
156     def vega(self, t, s, v):
157         return (self.price(t, s , v + self.h_v) \
158                 - self.price(t, s, v - self.h_v) \
159                 ) / (2*self.h_v)
160
161     def greeks(self, t, s, v):
162         return np.array([self.delta(t,s,v), self.vega(t,s,v)])
163
164     def map_greeks(self, arr):
165         return self.greeks(arr[0],arr[1],arr[2])
166
167     def mp_map_greeks(self, arr):
168         with Pool() as pool:
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
            results_value = list(tqdm(pool.imap(self.map_greeks, arr),
                                      total=len(arr),
                                      smoothing=0
                                      )
                                 )

        return np.stack(results_value)


class NormInvGaussPricer(OptionPricer):
    def __init__(self, _option, _model):
        super().__init__()
        assert isinstance(_model, NormInvGaussModel)
        self.hedge_position = np.vectorize(self.hedge_position)
        self.option = _option
        self.model = _model

    def price(self, t, s):
        def integrand(v):
            z = 1.1 + 1j*v
            return np.real(np.exp(self.model.kappa(z)*(self.option.T-t)) \
                           *s**(z) \
                           *(self.option.strike**(1-z)/(2*np.pi*z*(z-1))) \
                          )

        return quad(integrand, -np.inf,np.inf, limit = 500)[0]

    def hedge_position(self, t, s):
        def integrand(v):
            z = 1.1 + 1j*v
            return np.real(self.model.gamma(z) \
                           *np.exp(self.model.kappa(z) \
                           *(self.option.T-t))*s**(z-1) \
                           *(self.option.strike**(1-z)/(2*np.pi*z*(z-1)))
                          )

        return quad(integrand, -np.inf,np.inf, limit = 500)[0]
```

Listing A.3: Source code of stackedlstm.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from functools import partial
import numpy as np

from mvhrnn.utils import pnl, trading_costs
from mvhrnn.utils import G_default, set_end_zero, portfolio_payoff


# Calculate the Loss for a payoff as defined above,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
# initial capital c and price and position time series

def mvh_loss(payoff, c, prices, positions, eps):
    return tf.math.square( payoff(prices) - c \
           - tf.math.reduce_sum(pnl(prices, positions[:,:-1])[:,-1,:],1) \
           + trading_costs(prices, positions[:,:-1], eps) )

# Subclass a Layer to be able to add mvh_loss to a model
# This layer also adds the initial capital c as a trainable variable

class LossLayer(layers.Layer):
    def __init__(self, payoff, eps, G, tradeidx, **kwargs):
        self.payoff = payoff
        self.eps = eps
        self.G=G
        self.tradeidx = tradeidx

        c_init = tf.zeros_initializer()
        self.c = tf.Variable(initial_value=c_init(shape=(1,),
                             dtype="float32"),
                             name="c",
                             trainable=True
                            )

        super(LossLayer, self).__init__(**kwargs)


    def call(self, inputs):
        proj_pos = self.G(inputs[0])
        prices = tf.transpose(tf.gather_nd(tf.transpose(inputs[1]),
                                           [[x] for x in self.tradeidx]
                                           )
                              )
        self.add_loss(mvh_loss(self.payoff,
                               self.c,
                               prices,
                               proj_pos,
                               self.eps
                               )
                      )
        return set_end_zero(proj_pos)  # Pass-through model outputs

    def get_config(self):

        config = super().get_config().copy()
        config.update({
            'payoff': self.payoff,
            'eps': self.eps,
            'G': self.G,
            'trade_idx': self.tradeidx
        })
        return config


```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
71
72 class StackedLSTM(tf.keras.Model):
73
74     def __init__(self,
75                  inputshape,
76                  hiddenlayers,
77                  hiddenunits,
78                  optionportfolio,
79                  tradeidx=None,
80                  eps=0,
81                  G=G_default,
82                  **kwargs):
83         self.optionportfolio = optionportfolio
84         self.eps = eps
85
86         if tradeidx:
87             self.tradeidx = tradeidx
88         else:
89             self.tradeidx = list(range(inputshape[1]))
90
91         inputs = keras.Input(shape=inputshape, name="input_layer")
92
93         temp = inputs
94         for i in range(hiddenlayers):
95             temp = layers.LSTM(hiddenunits,
96                                return_sequences=True,
97                                unroll = True,
98                                name=f"lstm_layer_{i}"
99                               )(temp)
100
101        dense = layers.Dense(len(self.tradeidx),
102                             name="dense_layer",
103                             activation='linear'
104                            )(temp)
105
106        outputs = LossLayer(name="loss_layer",
107                            payoff=self.payoff,
108                            eps=self.eps,
109                            G=G,
110                            tradeidx=self.tradeidx
111                           )([dense, inputs])
112        super(StackedLSTM, self).__init__(inputs=inputs,
113                                          outputs=outputs,
114                                          **kwargs)
115
116    def payoff(self,tensor):
117        return portfolio_payoff(self.optionportfolio, tensor)
```

Listing A.4: Source code of utils.py

```
1 import os
2 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
3
4 import tensorflow as tf
5
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
# Calculate trading pnl from price and position time series
def pnl(prices, positions):
    return tf.math.cumsum(
                           tf.math.multiply(
                            tf.experimental.numpy.diff(prices, axis=1),positions
                           )
                           ,axis=1
                          )

#Proportional Trading costs
def trading_costs(prices, positions, eps):
    zeros_dims = tf.stack([tf.shape(positions)[0], 1, tf.shape(positions)[2]])
    zeros = tf.fill(zeros_dims, 0.0)
    n = tf.experimental.numpy.diff(tf.concat([zeros, positions], 1), axis = 1)
    costs = tf.math.multiply(tf.math.abs(n), prices[:,:-1,:]) * eps
    return tf.math.reduce_sum(costs, axis=[1,2])

def G_default(tensor):
    return tensor


def set_end_zero(tensor):
    zeros_dims = tf.stack([tf.shape(tensor)[0], 1, tf.shape(tensor)[2]])

    zeros = tf.fill(zeros_dims, 0.0)

    return tf.concat([tensor[:,:-1,:], zeros], 1)

def portfolio_payoff(portfolio, prices):
    payoff_tensor = tf.zeros(tf.shape(prices)[0])
    for i,option in enumerate(portfolio.options):
        payoff_tensor = tf.add(
                               payoff_tensor,option.tf_payoff(
                                   prices[:,:,portfolio.underlying_idx[i]]
                               )
                              )
    return payoff_tensor


def hedge_loss(payoff, prices, pos, c, eps):
    return (payoff - c \
            - tf.math.reduce_sum(pnl(prices,pos)[:,-1,:],1) \
            + trading_costs(prices, pos ,eps)
           ).numpy()

def hedge_error_from_model(model, prices):
    pos = model.predict(prices, verbose = 0)
    return hedge_loss(model.payoff(prices),
                      prices, pos[:,:-1],
                      model.trainable_variables[-1],
                      model.eps
                     )

def hedge_error(positions, prices, portfolio, c, eps = 0):
    return hedge_loss(portfolio_payoff(portfolio, prices),
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                        prices,
                        positions,
                        c,
                        eps
                       )
```

As the integrand of the 4/2 model pricing formula 5.11 has several properties that make numerical integration very computationally intensive, and several tens of thousands of integrals had to be calculated for the benchmark strategies, this integrand was implemented in C to not only speed this process up, but make it feasible w.r.t. runtime in the first place. The implementation is shown in Listing A.5.

Listing A.5: Source code of fourhalvesintegrand.c

```c
#include <acb.h>
#include <arb.h>
#include <stdio.h>
#include <acb_hypgeom.h>
#include <complex.h>
#include <math.h>
#include <gsl/gsl_sf_gamma.h>
#include <assert.h>

double complex hypgeom1f1(double complex a,
                          double complex b,
                          double complex z
                         ){
    acb_t res_acb;
    acb_init(res_acb);

    acb_t a_acb;
    acb_init(a_acb);
    acb_set_d_d(a_acb,creal(a),cimag(a));

    acb_t b_acb;
    acb_init(b_acb);
    acb_set_d_d(b_acb,creal(b),cimag(b));

    acb_t z_acb;
    acb_init(z_acb);
    acb_set_d_d(z_acb,creal(z),cimag(z));


    acb_hypgeom_1f1(res_acb,a_acb,b_acb,z_acb,0,53);

    arb_t re_arb;
    arb_init(re_arb);
    acb_get_real(re_arb, res_acb);

    arb_t im_arb;
    arb_init(im_arb);
    acb_get_imag(im_arb, res_acb);

    double complex res = arf_get_d(arb_midref(re_arb),ARF_RND_NEAR)
                       + arf_get_d(arb_midref(im_arb),ARF_RND_NEAR) * I;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
42
43      return res;
44  }
45
46  double complex cgamma(double complex z){
47      gsl_sf_result lnr;
48      gsl_sf_result arg;
49      gsl_sf_lngamma_complex_e(creal(z), cimag(z), &lnr, &arg);
50
51      return exp(lnr.val)*cexp(I*arg.val);
52  }
53
54  double complex A(double complex u,
55                   double kappa,
56                   double theta,
57                   double sigma,
58                   double a,
59                   double b,
60                   double rho
61                  ){
62
63      return kappa*kappa
64             - 2*sigma*sigma*(u*( (a*rho*kappa)/sigma - 0.5*a*a)
65             + 0.5*u*u*(1 - rho*rho)*a*a);
66  }
67
68  double complex m(double complex u,
69                   double kappa,
70                   double theta,
71                   double sigma,
72                   double a,
73                   double b,
74                   double rho
75                  ){
76      double sigma_squared = sigma*sigma;
77      return (2/sigma_squared)*csqrt(pow(kappa*theta - 0.5*sigma_squared, 2)
78             -2*sigma_squared*(u*((b*rho/sigma)*(0.5*sigma_squared -kappa*theta)
79             -0.5*b*b) +
80             0.5*u*u*(1-rho*rho)*b*b) );
81  }
82
83  double complex K(double complex u,
84                   double t,
85                   double kappa,
86                   double theta,
87                   double sigma,
88                   double a,
89                   double b,
90                   double rho
91                  ){
92      double complex sqrt_A = csqrt(A(u, kappa, theta, sigma, a, b, rho));
93      return (1/(sigma*sigma)) * ( sqrt_A*(1/ctanh(0.5*sqrt_A*t)) + kappa);
94  }
95
96  double complex psi(double complex u,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
                    double t,
                    double T,
                    double y,
                    double v,
                    double kappa,
                    double theta,
                    double sigma,
                    double a,
                    double b,
                    double rho,
                    double r
                    ){
    double sigma_squared = sigma*sigma;
    double complex sqrt_A = csqrt(A(u, kappa, theta, sigma, a, b, rho));
    double complex m_u = m(u, kappa, theta, sigma, a, b, rho);


    double complex a1 = (u*y + ((kappa*kappa * theta)/sigma_squared)*(T-t)
                        + u*(r - a*b - (a*rho*kappa*theta)/(sigma)
                        + (b*rho*kappa)/(sigma) )*(T-t)
                        + u*u*(1 - rho*rho)*a*b*(T-t));

    double complex a2 = clog((sqrt_A/(sigma_squared*csinh(0.5*sqrt_A*(T-t)))))
                        *(m_u+1);

    double complex a3 = log(v)
                        * (0.5 + 0.5*m_u - (kappa * theta)/(sigma_squared)
                        - (u*b*rho)/(sigma) );

    double complex a4 = clog( K(u,T-t, kappa, theta, sigma, a, b, rho)
                        - (u*a*rho)/(sigma) )
                        *(-(0.5 + 0.5*m_u + (kappa * theta)/(sigma_squared)
                        + (u*b*rho)/(sigma) ));

    double complex a5 = ((v/sigma_squared)*(-sqrt_A*(1/ctanh(0.5*sqrt_A*(T-t))))
                        + kappa - u*a*rho*sigma) );

    double complex a6 = clog(cgamma(0.5+0.5*m_u+(kappa*theta)/(sigma_squared)
                        +(u*b*rho)/(sigma)) / cgamma(m_u + 1));

    double complex a7 = hypgeom1f1(0.5 + 0.5*m_u +
                        (kappa * theta)/(sigma_squared)
                        + (u*b*rho)/(sigma),
                        m_u + 1,
                        ((A(u, kappa, theta, sigma, a, b, rho)*v)
                        /(sigma_squared*sigma_squared
                        * cpow(csinh(0.5 * sqrt_A * (T-t) ),2)
                        * (K(u,T-t, kappa, theta, sigma, a, b, rho)
                        * - (u*a*rho)/(sigma) ))) );
    return    cexp(a1+a2+a3+a4+a5+a6)*(a7);
}

double complex Fhat(double complex z, double strike){
    return -cpow(strike,(I*z + 1))/(z*z - I*z);
}
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
152
153
154 double integrand_call(int n, double *xx){
155     /*
156     x               xx[0]
157     t               xx[1]
158     T               xx[2]
159     s               xx[3]
160     v               xx[4]
161     kappa           xx[5]
162     theta           xx[6]
163     sigma           xx[7]
164     a               xx[8]
165     b               xx[9]
166     rho             xx[10]
167     r               xx[11]
168     strike          xx[12]
169     */
170     double complex z = xx[0] + 1.1*I;
171     return creal(psi(-I * z,
172                      xx[1],
173                      xx[2],
174                      log(xx[3]),
175                      xx[4],
176                      xx[5],
177                      xx[6],
178                      xx[7],
179                      xx[8],
180                      xx[9],
181                      xx[10],
182                      xx[11]
183                      ) * Fhat(z,xx[12]));
184
185 }
186
187 double integrand_put(int n, double *xx){
188     /*
189     x               xx[0]
190     t               xx[1]
191     T               xx[2]
192     s               xx[3]
193     v               xx[4]
194     kappa           xx[5]
195     theta           xx[6]
196     sigma           xx[7]
197     a               xx[8]
198     b               xx[9]
199     rho             xx[10]
200     r               xx[11]
201     strike          xx[12]
202     */
203     double complex z = xx[0] - 0.1*I;
204     return creal(psi(-I * z,
205                      xx[1],
206                      xx[2],
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
207                     log(xx[3]),
208                     xx[4],
209                     xx[5],
210                     xx[6],
211                     xx[7],
212                     xx[8],
213                     xx[9],
214                     xx[10],
215                     xx[11]) * Fhat(z,xx[12]));
216
217 }
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Source Codes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] Jean-Philippe Aguilar. *Explicit Option Valuation in the Exponential NIG Model*. Oct. 4, 2020. arXiv: 2006.04659 [q-fin]. URL: http://arxiv.org/abs/2006.04659 (visited on 09/28/2022).

[2] Ole E. Barndorff-Nielsen. „Normal Inverse Gaussian Distributions and Stochastic Volatility Modelling". In: *Scandinavian Journal of Statistics* 24.1 (Mar. 1997), pp. 1–13. ISSN: 0303-6898, 1467-9469. DOI: 10.1111/1467-9469.00045. URL: https://onlinelibrary.wiley.com/doi/10.1111/1467-9469.00045 (visited on 09/28/2022).

[3] Hans Bühler et al. „Deep Hedging". Feb. 8, 2018. arXiv: 1802.03042 [math, q-fin]. URL: http://arxiv.org/abs/1802.03042 (visited on 03/04/2022).

[4] Edwin Kah Pin Chong and Stanislaw H. Żak. *An Introduction to Optimization*. Fourth edition. Wiley Series in Discrete Mathematics and Optimization. Hoboken, New Jersey: Wiley, 2013. 622 pp. ISBN: 978-1-118-27901-4.

[5] Samuel N. Cohen and Robert J. Elliott. *Stochastic Calculus and Applications*. 2. ed. Probability and Its Applications. Basel: Birkhauser, 2015. 666 pp. ISBN: 978-1-4939-2867-5. DOI: 10.1007/978-1-4939-2867-5.

[6] Freddy Delbaen and Walter Schachermayer. „A General Version of the Fundamental Theorem of Asset Pricing". In: *Mathematische Annalen* 300.1 (Sept. 1994), pp. 463–520. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01450498. URL: http://link.springer.com/10.1007/BF01450498 (visited on 08/19/2022).

[7] TensorFlow Developers. *TensorFlow*. Version v2.8.2. Zenodo, May 23, 2022. DOI: 10.5281/ZENODO.4724125. URL: https://zenodo.org/record/4724125 (visited on 06/28/2022).

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016. 775 pp. ISBN: 978-0-262-03561-3.

[9] Martino Grasselli. „The 4/2 Stochastic Volatility Model". In: *SSRN Electronic Journal* (2014). ISSN: 1556-5068. DOI: 10.2139/ssrn.2523635. URL: http://www.ssrn.com/abstract=2523635 (visited on 08/10/2022).

[10] Steven Heston. „A Simple New Formula for Options with Stochastic Volatility". In: 1997.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[11] Steven L. Heston. „A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options“. In: *Review of Financial Studies* 6.2 (Apr. 1993), pp. 327–343. ISSN: 0893-9454, 1465-7368. DOI: 10.1093/rfs/6.2.327. URL: https://academic.oup.com/rfs/article-lookup/doi/10.1093/rfs/6.2.327 (visited on 05/15/2022).

[12] Hannes Hirber. „Über semistatisches Hedging von Derivaten“. In: (2018). In collab. with Friedrich Hubalek, 58 pages. DOI: 10.34726/HSS.2018.55990. URL: https://repositum.tuwien.at/handle/20.500.12708/1804 (visited on 09/27/2022).

[13] Kurt Hornik. „Approximation Capabilities of Multilayer Feedforward Networks“. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 08936080. DOI: 10.1016/0893-6080(91)90009-T. URL: https://linkinghub.elsevier.com/retrieve/pii/089360809190009T (visited on 06/14/2022).

[14] Friedrich Hubalek, Jan Kallsen, and Leszek Krawczyk. „Variance-Optimal Hedging for Processes with Stationary Independent Increments“. In: *The Annals of Applied Probability* 16.2 (May 1, 2006). ISSN: 1050-5164. DOI: 10.1214/105051606000000178. URL: https://projecteuclid.org/journals/annals-of-applied-probability/volume-16/issue-2/Variance-optimal-hedging-for-processes-with-stationary-independent-increments/10.1214/105051606000000178.full (visited on 08/23/2022).

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 06/21/2022).

[16] Hiroshi Kunita and Shinzo Watanabe. „On Square Integrable Martingales“. In: *Nagoya Mathematical Journal* 30 (Aug. 1967), pp. 209–245. ISSN: 0027-7630, 2152-6842. DOI: 10.1017/S0027763000012484. URL: https://www.cambridge.org/core/product/identifier/S0027763000012484/type/journal_article (visited on 08/10/2022).

[17] Alan L. Lewis. *Option Valuation under Stochastic Volatility: With Mathematica Code*. Newport Beach, CA: Finance Press, 2000. 350 pp. ISBN: 978-0-9676372-0-4.

[18] Thorsten Rheinländer and Jenny Sexton. *Hedging Derivatives*. Advanced Series on Statistical Science and Applied Probability v. 15. Singapore ; Hackensack, NJ: World Scientific, 2011. 233 pp. ISBN: 978-981-4338-79-0.

[19] Walter Rudin. *Real and Complex Analysis*. 3rd ed. New York: McGraw-Hill, 1987. 416 pp. ISBN: 978-0-07-054234-1.

[20] Fathi M Salem. *Recurrent Neural Networks: From Simple to Gated Architectures*. 2022. ISBN: 978-3-030-89929-5. URL: https://doi.org/10.1007/978-3-030-89929-5 (visited on 04/28/2022).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN