School of Computation, Information and Technology
Department of Mathematics
Technical University of Munich

Integrating D-vine regression models and neural network approaches for conditional quantile estimation with financial application

Moritz Jaud

Thesis for the attainment of the academic degree

Master of Science (M.Sc.)

at the Department of Mathematics of the Technical University of Munich.

Study program:
Mathematical Finance and Actuarial Science

Examiner:
Prof. Claudia Czado, Ph.D.

Supervisor:
Prof. Claudia Czado, Ph.D.

Submitted:
Munich, 13.06.2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 13.06.2024 Moritz Jaud

Abstract

Quantile regression has become increasingly important in statistical modeling, finding applications across various fields as a complementary approach to linear regression. Issues such as quantile crossings and the linearity assumption are among the drawbacks of the classical linear quantile regression introduced by Koenker and Bassett Jr (1978). Kraus and Czado (2017) addressed these problems by introducing the so called D-vine quantile regression. On the other hand, Cannon (2011) introduced a quantile regression neural network employing the pinball loss as the risk function. This approach leverages the advantages of neural networks, such as non-linearity and scalability to handle large datasets, for quantile regression. We improve the quantile regression neural networks by accumulating quasi quantile regression neural networks in order to guarantee the absence of quantile crossings while maintaining the advantages of the neural networks. The resulting cQRNN model eliminates quantile crossings, whereas the original model exhibited quantile crossings in up to 15 percent of the test set estimations in our simulation study. Further, the importance of the uncertainty of neural networks is addressed by introducing the idea of Tagasovska, Ozdemir, and Brando (2023). We enhance the algorithm by fitting a D-vine quantile regression to both the last hidden layer and the response variable of a QRNN model. The resulting D-vine copula is utilized to generate data, effectively "bootstrapping" the underlying model. This is achieved by retraining the last hidden layer with copies of the original QRNN, utilizing the generated data. Lastly, the models are compared in a simulation study as well as a real world application where we investigate the conditional Value at Risk, using the five Fama and French factors as the predictors (Fama and French 2014). We conclude that the cumulative quantile regression neural network (cQRNN) is a superior model compared to the original quantile regression neural network (QRNN) because it eliminates the occurrence of quantile crossings while maintaining high accuracy.

Zusammenfassung

Die Quantilregression hat in der statistischen Modellierung zunehmend an Bedeutung gewonnen und findet in verschiedenen Bereichen Anwendung als ergänzender Ansatz zur linearen Regression. Probleme wie quantile crossing und die Annahme der Linearität gehören zu den Nachteilen der klassischen linearen Quantilregression, die von Koenker and Bassett Jr (1978) entwickelt wurde. Kraus and Czado (2017) haben diese Probleme durch die Entwicklung der sogenannten D-vine quantile regression gelöst. Weiter hat Cannon (2011) ein neuronales Netz, namens quantile regression neural network (QRNN), entwickelt, welches den pinball loss als Risikofunktion verwendet. Dieser Ansatz nutzt die Vorteile von neuronalen Netzen wie Nichtlinearität und Skalierbarkeit zur Bewältigung großer Datensätze für die Quantilregression. Wir verbessern diese QRNN Modelle, indem wir sogenannte quasi quantile regression neural networks verketten, um das Fehlen von quantile crossing zu garantieren, während wir die Vorteile der neuronalen Netze erhalten. Das resultierende cQRNN-Modell eliminiert das Aufkommen von quantile crossings, während das ursprüngliche Modell in unserer Simulationsstudie das Phänomen von quantile crossings in bis zu 15 Prozent der Schätzungen im Testset zeigte. Darüber hinaus wird die Bedeutung der Unsicherheit von neuronalen Netzen durch die Einführung des Konzepts von Tagasovska, Ozdemir, and Brando (2023) angesprochen. Wir verbessern den Algorithmus, indem wir eine D-vine quantile regression sowohl auf den letzten hidden layer als auch auf die responsvariable eines QRNN-Modells anpassen. Der resultierende D-Vine Copula wird verwendet, um Daten zu generieren, wodurch das zugrunde liegende Modell effektiv "gebootstrapt" wird. Dies wird erreicht, indem der letzte hidden layer von Kopien des originalen QRNN unter Verwendung der generierten Daten erneut trainiert wird. Schließlich werden die Modelle in einer Simulationstudie sowie in einer realen Anwendung verglichen, bei der wir den bedingten

conditional value at risk, unter Verwendung der fünf Fama- und French-Faktoren (Fama and French 2014) als Variablen, schätzen. Wir kommen zu dem Schluss, dass dass das cumulative quantile regression neural network (cQRNN) ein überlegenes Modell im Vergleich zum ursprünglichen quantile regression neural network (QRNN) darstellt, da es das Aufkommen von quantile crossings eliminiert und gleichzeitig eine hohe Genauigkeit beibehält.

Contents

1 Introduction

This master's thesis aims to integrate and compare models derived from both vine copula theory and neural network theory. The two fundamental models for predicting the conditional quantile of a response variable, given a set of predictive variables, examined in this thesis are the **D-vine quantile regression model** introduced by Kraus and Czado (2017), which is based on vine copula theory, and the **quantile regression neural network (QRNN)**, initially studied by Cannon (2011). Zhang, Quan, and Srinivasan (2019) continued the theory around QRNN's by introducing the early stopping method (Prechelt 2012) and various regularization methods like the dropout method (Hinton et al. 2012) and the L^2 regularization of the underlying neural network (Ng 2004) to further improving the accuracy of the estimations of the quantile regression neural networks. On the other hand, Sahin and Czado (2022) developed a refined D-vine quantile regression algorithm named **sparse D-vine quantile regression** which shows an enhanced performance for higher dimensional regression settings (i.e. the set of predictive variables incorporates more than 10 variables).

In practical regression settings, we often encounter high dimensional data with many covariates which can result in lengthy computing times. As a result, we apply the forward selection algorithm of the D-vine quantile regression to the set of predictive variables of the QRNN model in order to reduce the number of covariates for the resulting quantile regression neural network model. This model will be called **reduced quantile regression** model or **redQRNN**.

A commonly known issue with quantile regression neural networks is the existing of quantile crossing of the estimated conditional quantiles (Bernard and Czado 2015). We try to address this issue by introducing various concepts, starting by modifying the loss function of the QRNN by adding an error term related to quantile crossing. Moreover, we introduce a new model named **cumulative quantile regression model** (or short **cQRNN**) which accumulates so-called quasi quantile regression neural network models in order to give a prediction of the conditional quantile at predefined levels. By definition, it will guarantee the absence of quantile crossing of the estimated conditional quantiles.

Furthermore, we study the uncertainties for quantile regression neural networks by applying the concept first introduced by Tagasovska, Ozdemir, and Brando (2023). While the original algorithm fits an R-vine copula to the last hidden layer of the trained neural network, our approach involves fitting a D-vine quantile regression. This modification reduces the number of neurons in the last hidden layer and consequently lowers the computing time of the algorithm.

All the quantile regression models are then compared in a simulation study as well as a real-data example, in which we predict the conditional Value at Risk using the five Fama and French factors Fama and French (2014).

2 Mathematical background and definitions

2.1 Supervised learning setup

For the supervised learning setup, we follow the notation of Wolf (2023). First, we define the **training data** set as $D = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \dots, n\}$ where we assume that the pairs $(y_i, \mathbf{x}_i)$ are treated as independent values of $(Y, \mathbf{X}) \sim P_0$ and P_0 is some probability measure over $\mathbb{R} \times \mathbb{R}^d$. The training data set acts as the input in our learning algorithm.

As the output, we obtain a function $\Phi : \mathbb{R}^d \to \mathbb{R}$ (target function) that tries predicting $y \in \mathbb{R}$ given arbitrary $\mathbf{x} \in \mathbb{R}^d$. The learning algorithm can be characterized as the map

$$\mathcal{A} : \bigcup_{n \in \mathbb{N}} \left(\mathbb{R} \times \mathbb{R}^d\right)^n \to \mathbb{R}^{\mathbb{R}^d}.$$

The set of functions which can be outputted by the learning algorithm will be denoted by $\mathcal{F}$ and is referred to as the function class of the learning algorithm.

The objective of the learning algorithm is to identify a suitable function Φ that minimizes a pre-specified **loss function** $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The loss function measures the distance between the predicted value $h(\mathbf{x})$ and the observed y. By minimizing the average loss, called risk, we obtain a suitable choice for Φ:

$$\Phi^* = \min_{\Phi \in \mathcal{F}} R(\Phi) = \min_{\Phi \in \mathcal{F}} \int_{\mathbb{R} \times \mathbb{R}^d} L\left(y, \Phi(\mathbf{x})\right) dP_0(y, \mathbf{x}).$$

The most common loss which is used in regression settings is the **quadratic loss** $L(y, \Phi(\mathbf{x})) = \|y - \Phi(\mathbf{x})\|$ that leads to the $\boldsymbol{L^2}$**-risk**

$$R(\Phi) = \mathbb{E}_{P_0}\left[\|Y - \Phi(\mathbf{X})\|^2\right],$$

which is also known as the **mean-squared error**.

Since the probability measure P_0 is unknown, we cannot calculate the risk directly. Therefore, the task is to minimize the risk by only using the training data set D_{train}. As a result, we define the empirical risk by

$$\hat{R}(\Phi) = \frac{1}{|D|} \sum_{(y, \mathbf{x}) \in D} L\left(y, \Phi(\mathbf{x})\right).$$

Note that $\hat{R}(\Phi) \xrightarrow{|D| \to \infty} R(\Phi)$ for $\Phi \in \mathcal{F}$, by the law of large numbers. Choosing the squared error loss leads to the minimization problem

$$\hat{\Phi} = \min_{\Phi \in \mathcal{F}} \quad \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \Phi(\mathbf{x}_i)\right)^2.$$

The above optimization problem can only be solved approximately and is called **training** (Gühring, Raslan, and Kutyniok 2020). For our purposes, we study the **function class** $\boldsymbol{\mathcal{F}}$, which consists of **feed forward neural networks** characterized by a **parameter vector** $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^M$, where $\boldsymbol{M}$ **is the number of parameters** of the neural network, i.e. $M = |\boldsymbol{\theta}|$. This leads to the minimization problem

$$\hat{\boldsymbol{\theta}} = \min_{\boldsymbol{\theta} \in \Theta} \hat{R}(\Phi_{\boldsymbol{\theta}}) = \min_{\boldsymbol{\theta} \in \Theta} \quad \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \Phi(\mathbf{x}_i; \boldsymbol{\theta})\right)^2, \tag{2.1}$$

which we approximately try to solve using $(y_i, \mathbf{x}_i) \in D$, $i = 1, \dots, n$ and a pre-defined learning algorithm. In practice, the available **training data** D is divided into two disjoint sets: the **training set** and the **test**

set. The training set is used to optimize (2.1) and output a model $\Phi_{\hat{\theta}}$, while the test set is used to evaluate the performance of $\Phi_{\hat{\theta}}$. Sometimes, the data is further split into three disjoint sets: the training set, the test set, and a **validation set**, which is used for fine-tuning the hyperparameters of the learning algorithm. We will denote these sets as D_{train}, D_{test}, and D_{val}.

2.2 Statistical framework

The general definitions and notation used in the statistical framework of our thesis are based on Czado (2019) and Tepegjozova (2019). Throughout this thesis, random variables will be represented by capital letters, while their observed values will be indicated by lowercase letters. For example, we write $X = x$ for a random variable $X \in (\Omega, \mathcal{F}, P)$, where $(\Omega, \mathcal{F}, P)$ is a probability space. Furthermore, we assume that the random variables X are absolute continuous which will ensure that the corresponding density function f exists. Additionally, we will use F for the distribution function.

We continue by mentioning the terminology of the multivariate distributions case. Bold letters will indicate, that we are in a multivariate distribution setting.

Definition 2.2.1. *For a random vector $\mathbf{X} = (X_1, \dots, X_d)^T$, we define:*

- *The joint distribution and density of $\mathbf{X}$ as $F(\mathbf{x})$ and $f(\mathbf{x})$ for $\mathbf{x} = (x_1, \dots, x_d)^T$.*
- *The marginal distribution and density function of X_j as $F_j(x_j)$ and $f_j(x_j)$, for $j = 1, \dots, d$.*
- *Conditional distribution and density function of X_j given $\mathbf{X}_D$ where $D = \{1, \dots, d\} \backslash j$: $F_{j|D}(x_j|\mathbf{x}_D)$ and $f_{j|D}(x_j|\mathbf{x}_D)$ for $j \neq i$.*

Next, to characterize the interdependence among the random variables, it is necessary to standardize them. Hence, each random variable X_j, where $j = 1, \dots, d$, is standardized using the probability integral transform or PIT, which is defined as follows:

Definition 2.2.2. (Probability Integral Transform) *If $X \sim F$ is a continuous random variable and x is an observed value of X, then the transformation*

$$u := F(x)$$

is called the probability integral transform (PIT) at x.

Note that if $X \sim F$ it follows that $U := F(X)$ is uniformly distributed.

Definition 2.2.3. (Quantile function) *Let $F_X : \mathbb{R} \to [0, 1]$ be the distribution function of a random variable X. Then*

$$F_X^{-1} : (0, 1) \to \mathbb{R}, \quad y \mapsto \inf\{x \in \mathbb{R} : F_X(x) \geq y\}$$

is the quantile function.

In the following sections, we will often work in the setting of **quantile regression** for which the target function is the so-called **conditional quantile function**.

Definition 2.2.4. (Conditional quantile function) *The conditional quantile function at level $\alpha \in (0, 1)$, for a continuous response variable Y given the outcome of some predictor variables $X_1, ..., X_d$ for some number of predictors $d \geq 1$ is given by*

$$q_\alpha(x_1, ..., x_d) := F^{-1}_{Y|X_1,...,X_d}(\alpha|X_1 = x_1, ..., X_d = x_d). \tag{2.2}$$

We often use $q_\alpha(\mathbf{x})$ or even shorter q_α as an abbreviation. In general, regression techniques try to explain the behavior of the dependent variable Y as a deterministic function of predictor realizations $x_1, \ldots, x_d$, of the independent variables $X_1, \ldots, X_d$, along with some additional random noise ϵ. This is represented by the equation $Y = h(x_1, \ldots, x_d, \epsilon)$, where h is our target function that captures the relationship between Y and $x_1, \ldots, x_d$, incorporating the random noise ϵ. As we have seen in the previous section, we introduce a loss function such that the function that minimizes the expected loss function is the target function. For our purposes we strive for the target function which is the conditional quantile function. The corresponding loss is defined as follows.

Definition 2.2.5. (Pinball loss) *Let $\alpha \in (0,1)$, $(y, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^d$ and $h : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary model that tries to predict $q_\alpha(\mathbf{x})$ given an arbitrary $\mathbf{x} \in \mathbb{R}^d$. The pinball loss is defined as*

$$\rho_\alpha : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \quad \rho_\alpha(y, h(\mathbf{x})) := \begin{cases} \alpha(y - h(\mathbf{x})) & \text{if} \quad (y - h(\mathbf{x})) \geq 0 \\ (\alpha - 1)(y - h(\mathbf{x})) & \text{if} \quad (y - h(\mathbf{x})) < 0 \end{cases}. \tag{2.3}$$

We observe, however, that the pinball loss ρ_α is not differentiable at the origin. Therefore, we introduce a hybrid L^1/L^2 norm to provide a smooth transition between absolute and squared errors around the origin (Cannon 2011). The resulting norm is called **Huber norm**

$$H_\epsilon(x) := \begin{cases} \frac{x^2}{2\epsilon} & \text{if} \quad 0 \leq |x| \leq \epsilon \\ |x| - \frac{\epsilon}{2} & \text{if} \quad |x| > \epsilon \end{cases}, \tag{2.4}$$

and can be used to approximate the pinball loss and make it differentiable.

Definition 2.2.6. (Huber pinball loss) *Let $\alpha \in (0,1)$, $\epsilon > 0$, H_ϵ be the Huber norm, $(y, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^d$ and $h : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary model that tries to predict $y \in \mathbb{R}$ given an arbitrary $\mathbf{x} \in \mathbb{R}^d$. The Huber pinball loss is defined as*

$$\rho^{(H)}_{\alpha,\epsilon} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \quad \rho^{(H)}_{\alpha,\epsilon}(y, h(\mathbf{x})) := \begin{cases} \alpha H_\epsilon(y - h(\mathbf{x})) & \text{if} \quad (y - h(\mathbf{x})) \geq 0 \\ (\alpha - 1) H_\epsilon(y - h(\mathbf{x})) & \text{if} \quad (y - h(\mathbf{x})) < 0 \end{cases}. \tag{2.5}$$

2.2.1 Time series definitions

In this section we briefly introduce time series analysis definitions and the ARMA-GARCH model which we will use in later chapters. We follow the book of Shumway and Stoffer (2017) for the definitions.

The stochastic process (i.e a sequence of random variables) $(X_t)_{t \in \mathbb{N}}$ is called a **time series**, where the index t often represents time.

Definition 2.2.7. (Autocovariance Function) *The autocovariance function is defined as the second moment product:*

$$\mathit{Cov}(X_s, X_t) = E[(X_s - E[X_s])(X_t - E(X_t)], \tag{2.6}$$

for all s and t.

We will denote $\mathit{Cov}(X_t, X_{t-k})$ as the **lag k autocovariance of** $(X_t)_{t \in \mathbb{N}}$.

Definition 2.2.8. (White noise process) *A time series $(W_t)_{t \in \mathbb{N}}$ is called a white noise if it is a sequence of independent and identically distributed random variables with zero mean and finite constant variance σ^2:*

(i) $E[W_t] = 0$ *for all* t.

(ii) $Var(W_t) = \sigma^2$ *for all* t.

(iii) $Cov(W_t, W_{t+k}) = 0$ *for all* t, k.

Furthermore, we introduce the notion of regularity of a time series using the concept called **(weak) stationarity**.

Definition 2.2.9. (Weak stationarity) *A weakly stationary time series,* $(X_t)_{t\in\mathbb{N}}$*, is a finite variance process such that*

(i) $E[X_t] = \mu$ *for all* t.

(ii) The covariance function, $Cov(X_s, X_t)$*, defined in* (2.6) *depends on* s *and* t *only through their difference* $|s - t|$.

Henceforth, we will use the term stationarity to mean weak stationarity.

Definition 2.2.10. (Autocorrelation function) *The autocorrelation function (ACF) of a stationary time series* $(X_t)_{t\in\mathbb{N}}$ *is defined as*

$$\rho(h) = \frac{Cov(X_{t+h}, X_t)}{\sqrt{Var(X_{t+h})\,Var(X_t)}} \tag{2.7}$$

It follows that $-1 < \rho(h) < 1$ for all h by the Cauchy-Schwarz inequality (Shumway and Stoffer 2017). In the following, the ACF $\rho(h)$ of a stationary time series $(X_t)_{t\in\mathbb{N}}$ is called the **lag h autocorrelation** of $(X_t)_{t\in\mathbb{N}}$. As we have seen in the first section, we have no access to the true underlying distribution of the time series $(X_t)_{t\in\mathbb{N}}$ which leads us working with sample data $(x_t)_{t=1,\dots,n}$. The data is used to estimate the autocorrelation function (2.7).

Definition 2.2.11. (Sample autocorrelation function) *The sample autocorrelation function is defined as*

$$\hat{\rho}(h) = \frac{\sum_{t=1}^{n-h}(x_{t+h} - \overline{x})(x_t - \overline{x})}{\sum_{t=1}^{n}(x_t - \overline{x})^2}, \tag{2.8}$$

for $h = 0, \dots, n-1$ *where* $\overline{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$ *is the sample mean.*

Next, we briefly introduce the **ARMA-GARCH** model. For more details regarding the components of the model we refer to Shumway and Stoffer (2017). Given a time series $(X_t)_{t=1,\dots,T}$, a **generalized autoregressive conditional heteroscedasticity** (GARCH) model class splits the formula into two components:

$$X_t = \mu_t + W_t, \quad t = 1, \dots, T \tag{2.9}$$

where the conditional mean μ_t is modeled by an *ARMA*(p, q) model, given by

$$\underbrace{\mu_t = \phi_0 + \sum_{l=1}^{p} \phi_l X_{t-l} + \sum_{k=1}^{q} \theta_k W_{t-k}}_{ARMA(p,q)}, \tag{2.10}$$

with $W_t = X_t - \mu_t$ is called the **innovation** at time t. It satisfies

$$W_t = \sigma_t \epsilon_t, \qquad \underbrace{\sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i W_{t-i}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2}_{GARCH(m,s)}, \tag{2.11}$$

where ϵ_t is an i.i.d. distributed white noise process with mean 0 and variance 1 and is called the **innovation distribution**. The variance σ_t^2 is modeled by an *GARCH*(m, s) process. The parameters of the *ARMA*(p, q) model are called

- ϕ_0: mean value

- ϕ_l, $l = 1, \ldots, p$: autoregressive (ar) ARMA coefficients
- θ_k, $k = 1, \ldots, q$: moving average (ma) ARMA coefficients

and the coefficients of the *GARCH*(m, s) model are called

- α_0: constant coefficient of the variance equation
- α_i, $i = 1, \ldots, m$: value of autoregressive coefficients
- β_j, $j = 1, \ldots, s$: value of variance coefficients

The overall model is called *ARMA*(p, q)-*GARCH*(m, s).

After fitting an *ARMA*(p, q)-*GARCH*(m, s) model with a pre-defined innovation distribution, we want to access the goodness-of fit of the model. For this we calculate the **standardized residuals**

$$\hat{r}_t := \frac{X_t - \hat{\mu}_t}{\hat{\sigma}_t}, \qquad t = 1, \ldots, T \tag{2.12}$$

The standardized residuals $\hat{r}_t$, $t = 1, \ldots, T$, are expected to be independently and identically distributed and follow the selected innovation distribution. In order to check the distribution choice, we utilize Q-Q plots which are scatter plots of the empirical sample quantiles versus the theoretical quantiles. A good fit is achieved if the points follow the 45 degree line.

Moreover we check if the first H autocorrelations of the standardized residuals are zero using the **Ljung-Box test**:

$$H_0 : \ \rho(1) = \cdots = \rho(H) = 0 \quad \textit{vs.} \quad H_1 : \ \exists h \in \{1, \ldots, H\} : \ \rho(h) \neq 0.$$

The **Ljung-Box-Pierce Q-statistic** is calculated as

$$Q(H) = T(T+2) \sum_{h=1}^{H} \frac{\hat{\rho}(h)}{T-h} \tag{2.13}$$

is χ^2_H distributed under the null hypothesis H_0. Therefore, the null hypothesis would be rejected at level $\alpha \in (0, 1)$ if the value $Q(H)$ exceeds the $(1-\alpha)$-quantile of χ^2_H.

2.2.2 Financial time series and value at risk

In this section, we will briefly explain the empirical properties of asset prices. Given an asset **price process** $(S_t)_{t=1,\ldots,T}$ we compute the **log-return** of the asset

$$X_t = \ln\left(\frac{S_t}{S_{t-1}}\right), \tag{2.14}$$

for $t = 1, \ldots, T$. Some well-studied **stylized statistical properties** of asset returns are stated in Cont (2001):

- **Autocorrelations**: Autocorrelations are often insignificant
- **Heavy tails**: Asset return distributions often have heavier tails than a Gaussian distribution.
- **Volatility clustering**: High volatility events tend to cluster in time.
- **Leverage effect**: Typically, negative asset returns tend to increase volatility more than positive asset returns.

- **Gain/loss asymmetry**: Large downward shifts in stock prices are more frequent than upward shifts.

Definition 2.2.12. (Value at Risk) *The value at risk at level α on a financial position $X : \Omega \to \mathbb{R}$ is defined as*

$$\mathit{VaR}_\alpha(X) := \inf\{m \in \mathbb{R} : P(m + X < 0) \leq \alpha\}. \tag{2.15}$$

The value at risk at level α is the smallest amount of capital m, which, if added to the financial position X and invested in the risk-free asset keeps the probability of a loss below the level α (Makariou, Barrieu, and Tzougas 2021). Using Definition 2.2.3 we can follow for continuous distributions

$$\begin{aligned}
\mathit{VaR}_\alpha(X) &:= \inf\{m \in \mathbb{R} : P(m + X < 0) \leq \alpha\} \\
&= \inf\{m \in \mathbb{R} : P(X < -m) \leq \alpha\} \\
&= \inf\{m \in \mathbb{R} : P(X \geq -m) \geq 1 - \alpha\} \\
&= \inf\{m \in \mathbb{R} : P(-X \leq m) \geq 1 - \alpha\} \\
&= F^{-1}_{-X}(1 - \alpha).
\end{aligned}$$

3 Neural network theory

3.1 Stochastic gradient decent

One of the most frequently applied learning algorithm, which is used to solve the optimization problem (2.1), is the so-called **gradient descent** (Netrapalli 2019). The algorithm can be used if one is able to calculate the **gradient** $\nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}})$ at any given point $\boldsymbol{\theta} \in \Theta$. Note that

$$\nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \left[\frac{1}{n} \sum_{i=1}^{n} L(y_i, h(\mathbf{x}_i; \boldsymbol{\theta})) \right] = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \left[L(y_i, h(\mathbf{x}_i; \boldsymbol{\theta})) \right] \tag{3.1}$$

for any $\boldsymbol{\theta} \in \Theta$ and $(y_i, \mathbf{x}_i) \in D = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$, by the linearity of the gradient operator. Starting by initializing starting parameters $\boldsymbol{\theta}_0$, in each iteration t the parameters are updated according to

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}_t}) \quad \text{for } t = 1, \ldots, T \quad ,$$

where $\alpha_t \in \mathbb{R}$ are the step sizes also called **learning rates** and T is the **pre-defined number of iterations**. The idea of gradient descent is to find the minima of the risk function by going in the negative direction of the risk function's gradient.

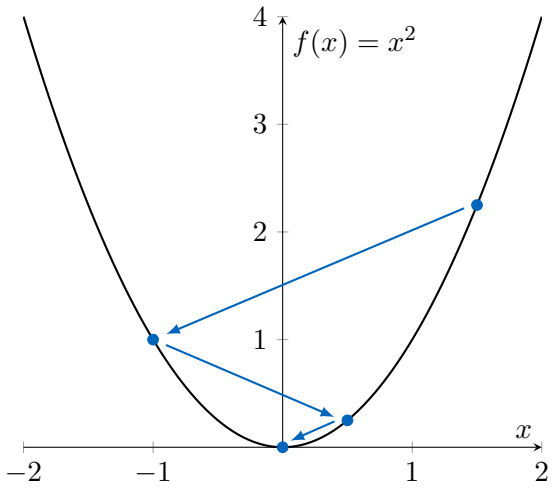


Figure 3.1 Visualization of the idea of gradient descent for finding the global minimum of the convex function $f(x) = x^2$.

Some well known issues of the gradient descent method are convergence of the algorithm, non-global minima and saddle points. In addition, the algorithm faces the problem of computational costs if we have a high dimensional Θ or a very large training set D. One way to reduce the number of evaluations we have to perform is to use the gradients $\nabla_{\boldsymbol{\theta}} \hat{R}_{i_t}(h_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \left[L(y_{i_t}, h(\mathbf{x}_{i_t}; \boldsymbol{\theta})) \right]$, with i_t sampled from $U \sim unif\{1, \ldots, n\}$ for every iteration t. We observe that $\nabla_{\boldsymbol{\theta}} \hat{R}_{i_t}(h_{\boldsymbol{\theta}})$ is an unbiased estimator of $\nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}})$ since

$$\mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \hat{R}_{i_t}(h_{\boldsymbol{\theta}}) \right] = \nabla_{\boldsymbol{\theta}} \mathbb{E}\left[\hat{R}_{i_t}(h_{\boldsymbol{\theta}}) \right] = \nabla_{\boldsymbol{\theta}} \left[\sum_{j=1}^{n} P(i_t = j) L(y_j, h(\mathbf{x}_j; \boldsymbol{\theta})) \right] =$$

$$\nabla_{\boldsymbol{\theta}} \left[\frac{1}{n} \sum_{j=1}^{n} L(y_j, h(\mathbf{x}_j; \boldsymbol{\theta})) \right] \overset{(1.2)}{=} \nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}}), \tag{3.2}$$

for any $\boldsymbol{\theta} \in \Theta$ and all $(y_i, \mathbf{x}_i) \in D$, $i = 1, \ldots, n$. For each iteration t, we define the update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} \hat{R}_{i_t}(h_{\boldsymbol{\theta}_t}) \quad \text{for } t = 1, \ldots, T, \tag{3.3}$$

which is called **stochastic gradient descent** (SGD). This learning algorithm is commonly used for solving optimization problems for large-scale problems. A proof of the convergence of the algorithm is shown in Turinici (2021). Since we only use one randomly chosen sample in the SGD, the estimation of the parameters can be noisy. In order to get more stable estimates, the concept of **mini-batch stochastic gradient descent** is the most commonly used algorithm. The algorithm is used as followed:

(i) Pre-define the learning rate $\alpha_t > 0$ for all iterations $t = 1, \ldots, T$.

(ii) Choose a batch size $1 < B_t < |D| = n$, where $|D|$ is the cardinality of the training set, for all iterations $t = 1, \ldots, T$. Initialize the parameter vector $\boldsymbol{\theta}_0 \in \Theta$, $t = 0$.

  (i) For each iteration t, uniformly sample B_t integers from the set $\{1, \ldots, |D|\}$ without replacement, and store the integers in the set $\mathcal{B}_t$.

  (ii) Next, the parameters are updated in the following way:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} \hat{R}_{\mathcal{B}_t}(h_{\boldsymbol{\theta}_t}), \tag{3.4}$$

  where

$$\nabla_{\boldsymbol{\theta}} \hat{R}_{\mathcal{B}_t}(h_{\boldsymbol{\theta}_t}) = \nabla_{\boldsymbol{\theta}} \left[\frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} L(y_i, h(\mathbf{x}_i; \boldsymbol{\theta}_t)) \right].$$

We again see, that $\nabla_{\boldsymbol{\theta}} \hat{R}_{\mathcal{B}_t}(h_{\boldsymbol{\theta}})$ is an unbiased estimator of $\nabla_{\boldsymbol{\theta}} \hat{R}(h_{\boldsymbol{\theta}})$, by following the same reasoning as for (3.2). As a result, mini-batch stochastic gradient descent capitalizes on the benefits of both the gradient descent algorithm and SGD (Robbins and Monro 1951) meaning that the algorithm can handle large-scale problems like SGD but reduces the noise in the estimation process. Note that gradient descent and stochastic gradient descent are two special cases of the mini-batch SGD. Setting $B_t = |D|$ (using the entire data set for updating the parameter vector) and $B_t = 1$ (using a uniformly sampled data point for updating the parameter vector) for all iterations $t = 1, \ldots, T$, respectively, recovers the gradient descent and stochastic gradient descent.

3.2 Feed forward neural network

We continue defining the mathematical framework of a feed forward neural network (or **multilayer perceptron**) following the notation of Gühring, Raslan, and Kutyniok (2020).

Definition 3.2.1. (Feed forward neural network) *Let $d, s, H \in \mathbb{N}$. A **feed forward neural network** Φ with **input dimension** d, **output dimension** s and H **hidden layers** is a sequence of matrix-vector tuples*

$$\Phi = \left(\left(\mathbf{W}^{[1]}, \mathbf{b}^{[1]} \right), \left(\mathbf{W}^{[2]}, \mathbf{b}^{[2]} \right), \ldots, \left(\mathbf{W}^{[H+1]}, \mathbf{b}^{[H+1]} \right) \right),$$

*where $k_0 = d$, $k_{H+1} = s$ and $k_1, \ldots, k_H \in \mathbb{N}$, and where each $\mathbf{W}^{[h]}$ is a $k_h \times k_{h-1}$ matrix, and $\mathbf{b}^{[h]} \in \mathbb{R}^{k_h}$. If Φ is a neural network as above, $K \subset \mathbb{R}^d$, and if $a : \mathbb{R} \to \mathbb{R}$ is arbitrary, then we define the associated realization of Φ with **activation function** a over K as the map $\tilde{\Phi} : K \to \mathbb{R}^s$ such that*

$$\tilde{\Phi}(\mathbf{x}) = \mathbf{h}^{[H+1]},$$

where $\mathbf{h}^{[H+1]}$ results from the following scheme:

$$\begin{aligned} \mathbf{h}^{[0]} &:= \mathbf{x}, \\ \mathbf{h}^{[h]} &:= a\left(\mathbf{W}^{[h]} \mathbf{h}^{[h-1]} + \mathbf{b}^{[h]} \right), \quad for\ h = 1, \ldots, H, \\ \mathbf{h}^{[H+1]} &:= \mathbf{W}^{[H+1]} \mathbf{h}^{[H]} + \mathbf{b}^{[H]}, \end{aligned}$$

where $\mathbf{x} \in K \subset \mathbb{R}^d$ and a acts componentwise, that is, $a(v) = (a(v_1), \ldots, a(v_m))$ for every $v = (v_1, \ldots, v_m) \in \mathbb{R}^m$ and $\mathbf{h}^{[h]}$ is the value vector of the neurons in h-th layer for $h = 1, \ldots, H$. For $h \leq H$ we call $M_h(\Phi) := \|\mathbf{W}^{[h]}\|_0 + \|\mathbf{b}^{[h]}\|_0$ the number of weights in the h-th layer where $\|\cdot\|_0$ is the total number of non-zero elements of a matrix, and we define $M(\Phi) := \sum_{h=1}^{H+1} M_h(\Phi)$, which we call the number of weights of Φ. Let $\boldsymbol{\theta} \in \mathbb{R}^{M(\Phi)}$ be the vector of the weights of Φ.

Since we only have a one dimensional output $y \in \mathbb{R}$ in the common regression setting, s is set to 1 if it is not stated otherwise.

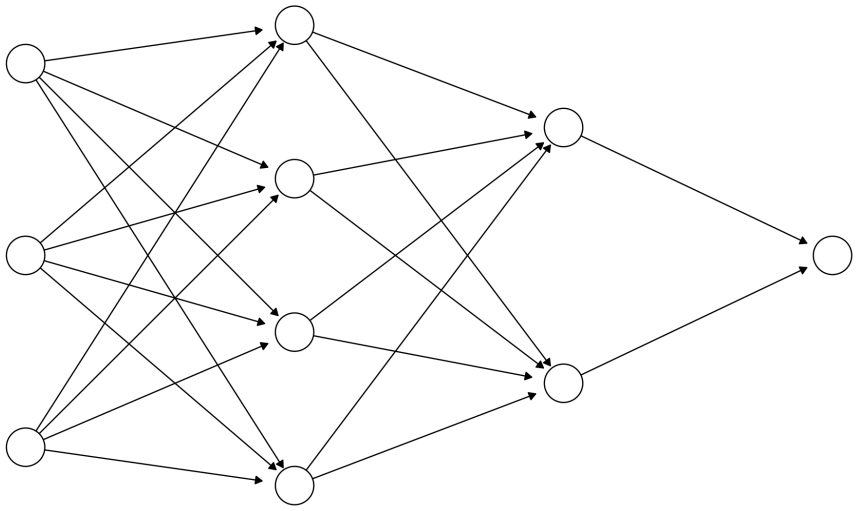


Figure 3.2 Visualization of a feed forward neural net with input dimension $d = 3$, two hidden layers $H = 2$ and output dimension $s = 1$.

The realization of the neural network Φ described in Figure 3.2 for an input vector $\mathbf{x} \in \mathbb{R}^3$ is calculated as

$$\tilde{\Phi}(\mathbf{x}) = \mathbf{h}^{[3]} = \mathbf{W}^{[3]} \left(a\left(\mathbf{W}^{[2]} a(\mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]} \right) \mathbf{b}^{[3]} \right),$$

where $\mathbf{W}^{[1]} \in \mathbb{R}^{4\times3}$, $\mathbf{W}^{[2]} \in \mathbb{R}^{2\times4}$, $\mathbf{W}^{[3]} \in \mathbb{R}^{1\times2}$, $\mathbf{b}^{[1]} \in \mathbb{R}^4$, $\mathbf{b}^{[2]} \in \mathbb{R}^2$, $\mathbf{b}^{[3]} \in \mathbb{R}$ and $a : \mathbb{R} \to \mathbb{R}$ is an arbitrary activation function.

Definition 3.2.2. (Set of neural networks) Let $d, k_1, \ldots, k_H, s \in \mathbb{N}$ for some $H \in \mathbb{N}$, $a : \mathbb{R} \to \mathbb{R}$ an activation function and $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ a loss function, then we define

$$\mathcal{K}^{a,L}_{(d,k_1,\ldots,k_H,s)} := \{\Phi : \quad \text{neural network with input dimension } d, H \text{ hidden layers, } k_h \text{ neurons in layer } h \text{ and output dimension } s; \Phi \text{ is equiped with the activation function } a \text{ and the loss function } L\}. \quad (3.5)$$

In the following we fix the input dimension $d \in \mathbb{N}$ and output dimension $s \in \mathbb{N}$. If it is not stated otherwise, we set a to be the sigmoid function and L to be the L^2 loss and shorten the notation to $\mathcal{K}_{(d,k_1,\ldots,k_H,s)}$.

This definition gives us the set of neural networks with input dimension d, output dimension s, H hidden layers equipped with the activation function a and loss function L. These networks are further characterized with the weight matrices between the layers, seen in Definition 3.2.1. To account for the weights and biases we introduced the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{M(\Phi)}$, which stores all matrices- as well as bias entries in one column vector, where $M(\Phi)$ is the number of weights and biases of the network Φ. This leads to the following notation:

Notation 3.2.1. *Let $d, k_1, \ldots, k_H, s \in \mathbb{N}$ for some $H \in \mathbb{N}$, $a : \mathbb{R} \to \mathbb{R}$ an activation function and $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ a loss function. Let*

$$\Phi^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)} \in \mathcal{K}^{a,L}_{(d,k_1,\ldots,k_H,s)} \quad (3.6)$$

donate the neural network with input dimension d, output dimension s, H hidden layers, where k_h is the number of neurons in the h-th layer. The neural network is equipped with the activation function a, loss function L and parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{M(\Phi)}$. If it is not stated otherwise, we set a to be the sigmoid function and L to be the L^2 loss. We often drop the parameter vector $\boldsymbol{\theta}$ and the vector $(d, k_1, \ldots, k_H, s)$ and work with the shortened notation $\Phi \in \mathcal{K}_{(d,k_1,\ldots,k_H,s)}$.

Note that $\Phi^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)}$ and the abbreviation Φ only describe the neural net. The realization of $\Phi^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)}$ and Φ are described by the maps

$$\tilde{\Phi}^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H)} : \mathbb{R}^d \to \mathbb{R}^s, \ \tilde{\Phi}^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)}(\mathbf{x}) = \mathbf{h}^{[H+1]} \quad (3.7)$$

and

$$\tilde{\Phi} : \mathbb{R}^d \to \mathbb{R}^s, \ \tilde{\Phi}(\mathbf{x}) = \mathbf{h}^{[H+1]}, \quad (3.8)$$

where $\mathbf{h}^{[H+1]}$ results from the calculation scheme in Definition 3.2.1.
In Definition 3.2.1 the choice of the activation function is arbitrary. However, in practice, certain commonly used activation functions serve specific purposes. The primary objective of these activation functions is to introduce non-linearity into neural network models. It is essential because linear transformations alone would result in a limited capacity to represent complex relationships. Note that if we choose the identity function to be the identity function, the neural networks will collapse to a concatenation of multivariate regression models. The most used activation functions and their properties are listed in the Table 3.1.

Name	Mathematical Expression	Properties
ReLU (Rectified Linear Unit)	$a(z) = \max(0, z)$	• Not differentiable at the origin • Not bounded • $\frac{d}{dz}a(z) = \mathbb{1}_{z>0}$
Sigmoidal	$a(z) = \frac{e^z}{1+e^z}$	• Differentiable • Monotonically increasing • Bounded: range $(0, 1)$ • $\frac{d}{dz}a(z) = \frac{e^z}{(1+e^z)^2}$
Leaky ReLU	$a(z) = z\mathbb{1}_{z>0} + 0.01z\mathbb{1}_{z\leq 0}$	• Not differentiable at the origin • Not bounded • $\frac{d}{dz}a(z) = \mathbb{1}_{z>0} + 0.01\mathbb{1}_{z\leq 0}$
Hyperbolic tangent	$a(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	• Differentiable • Monotonically increasing • Bounded: range $(0, 1)$ • $\frac{d}{dz}a(z) = \frac{1}{\cosh z^2}$

Table 3.1 Frequently used activation functions $a : \mathbb{R} \to \mathbb{R}$ and their properties.

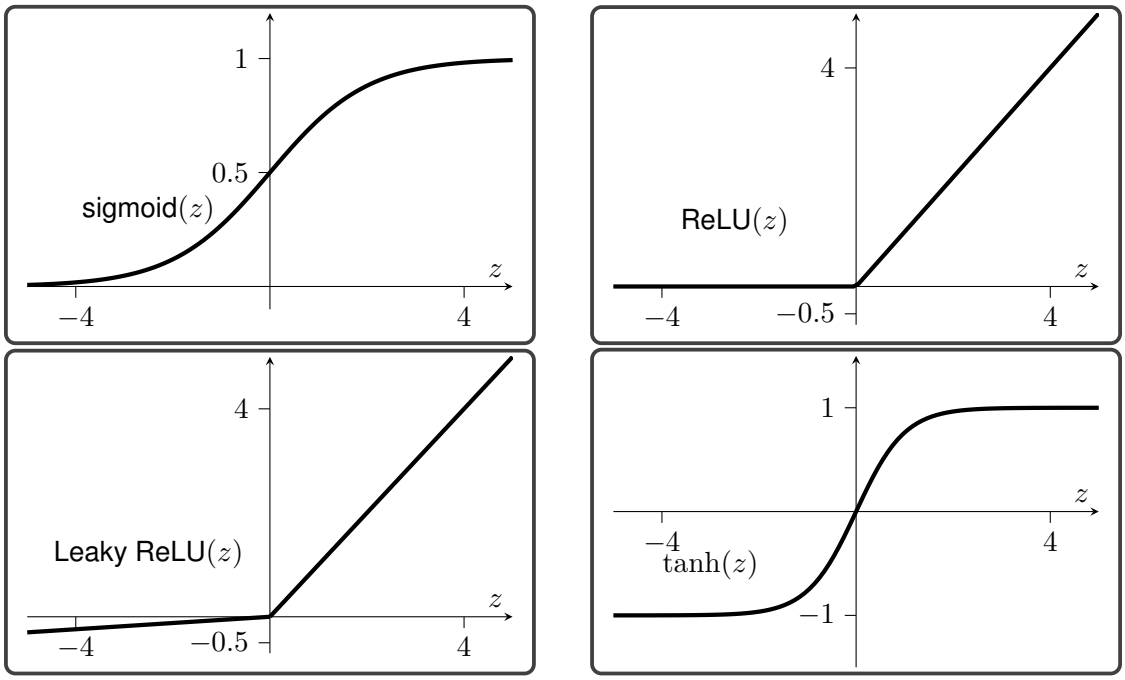


Figure 3.3 Graphs of the activation functions in Table 3.1

3.3 Back propagation

In the following chapter, the process of training a feed forward neural network is described.

(i) Firstly, the architecture of the neural network which is appropriate for the training data set $D = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$ has to be defined. This means we choose the appropriate input dimension d, starting weights $\boldsymbol{\theta}_0$, the number of hidden layers H and the number of neurons in each layer i.e. $k_1, \ldots, k_H$.

(ii) After defining the neural network $\Phi^{\boldsymbol{\theta}_0} \in \mathcal{K}_{(d,k_1,\ldots,k_H,s)}$, we insert the samples $(y_i, \mathbf{x}_i) \in D$ for $i = 1, \ldots, n$, in the model meaning we plug $\mathbf{x}_i$ into the model and calculate the output $\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)$ (*forward pass*).

(iii) Then we calculate the associated loss $\hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)) := L(y_i, \tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i))$ for every $i = 1, \ldots, n$ in order to calculate the empirical risk $\hat{R}(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)) = \sum_{i=1}^n \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)) = \sum_{i=1}^n L(y_i, \tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i))$.

(iv) After calculating the empirical risk of $\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)$, the weights $\boldsymbol{\theta}_0$ get adjusted using the learning algorithm **stochastic gradient descent** described in Section 3.1. We do this by computing the successive derivatives of the error with respect to the weights in each layer of the network, starting from the output layer and moving backward through the network (**backward pass**).

The procedure is called **back propagation** and is a direct application of the chain rule. The method is illustrated in the following example.

Example 1.1 Define a feed forward neural $\Phi_{2,3,1}^{\boldsymbol{\theta},a,L} \in \mathcal{K}_{(2,3,1)}^{;a,L}$ with the sigmoid activation function $a(x) = 1/(1 + \exp(-x))$, training set $D = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$ and the L^2 loss as the loss func-

tion. We again abbreviate the notation by setting $\Phi^{\boldsymbol{\theta}_0} := \Phi_{2,3,1}^{\boldsymbol{\theta},a,L}$, similar as in Notation 3.2.2. The weight matrices are then given by

$$\mathbf{W}^{[1]} = \begin{pmatrix} \omega_{11}^{[1]} & \omega_{12}^{[1]} \\ \omega_{21}^{[1]} & \omega_{22}^{[1]} \\ \omega_{31}^{[1]} & \omega_{31}^{[1]} \end{pmatrix}, \qquad \mathbf{b}^{[1]} = \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix};$$
$$\mathbf{W}^{[2]} = \begin{pmatrix} \omega_{11}^{[2]} & \omega_{12}^{[2]} & \omega_{13}^{[2]} \end{pmatrix}, \quad \mathbf{b}^{[2]} = b^{[2]}$$

and the weight vector takes the form of

$$\boldsymbol{\theta} = \begin{pmatrix} \omega_{11}^{[1]}, & \omega_{12}^{[1]}, & \omega_{21}^{[1]}, & \omega_{22}^{[1]}, & \omega_{31}^{[1]}, & \omega_{32}^{[1]}, & b_1^{[1]}, & b_2^{[1]}, & b_3^{[1]}, & \omega_{11}^{[2]}, & \omega_{12}^{[2]}, & \omega_{13}^{[2]}, & b^{[2]} \end{pmatrix}^T$$

Fixing $\boldsymbol{\theta}$, the **forward pass** can be illustrated by

$$\mathbf{h}^{[0]} := \mathbf{x}_i \in \mathbb{R}^2 \xrightarrow{a\left(\mathbf{W}^{[1]}\mathbf{h}^{[0]}+\mathbf{b}^{[1]}\right)} \mathbf{h}^{[1]} \in \mathbb{R}^3 \xrightarrow{\mathbf{W}^{[2]}\mathbf{h}^{[1]}+\mathbf{b}^{[2]}} \tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i) \in \mathbb{R} \xrightarrow{\left(y_i-\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)\right)^2} \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)).$$

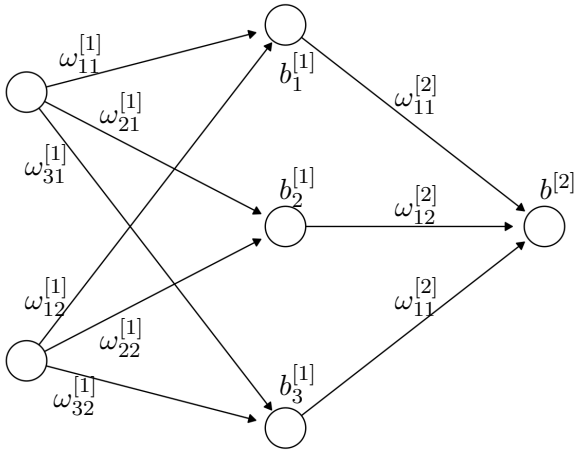


Figure 3.4 Visualization of the example feed forward neural network.

In order to train the network, we need to calculate the partial derivatives of $\hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i)$ for $i = 1, \ldots, n$ with respect to the parameters $\boldsymbol{\theta}$. To further simplifying the notation, denote $\hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_0}(\mathbf{x}_i) := \tilde{\Phi}(\mathbf{x}_i)$ and $\mathbf{h}_{\boldsymbol{\theta}}^{[H]} := \mathbf{h}^{[H]}$ for $H \in \{0, 1\}$ and $\mathbf{h}_{il}^{[H]}$ is the value of the l-th neuron in th H-th layer of the i-th sample. Differentiation yields

$$\begin{aligned} \frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial \omega_{1l}^{[2]}} = \frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial \tilde{\Phi}(\mathbf{x}_i)} \frac{\partial \tilde{\Phi}(\mathbf{x}_i)}{\partial \omega_{1l}^{[2]}} &= \frac{\partial \left((y_i - \tilde{\Phi}(\mathbf{x}_i))^2\right)}{\partial \tilde{\Phi}(\mathbf{x}_i)} \frac{\partial (\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]})}{\partial \omega_{1l}^{[2]}} \\ &= -2(y_i - \tilde{\Phi}(\mathbf{x}_i)) \frac{\partial (\omega_{11}^{[2]}\mathbf{h}_{i1}^{[1]} + \omega_{12}^{[2]}\mathbf{h}_{i2}^{[1]} + \omega_{13}^{[2]}\mathbf{h}_{i3}^{[1]} + \mathbf{b}^{[2]})}{\partial \omega_{1l}^{[2]}} \\ &= -2(y_i - \tilde{\Phi}(\mathbf{x}_i))\mathbf{h}_{il}^{[1]} = \delta_{1i}\mathbf{h}_{il}^{[1]}, \end{aligned} \tag{3.9}$$

where $l \in \{1, 2, 3\}$ and $\delta_{1i} := -2(y_i - \tilde{\Phi}(\mathbf{x}_i))$ for $i = 1, \ldots, n$. The derivative of the empirical risk function with respect to the weights in the first layer is derived as

$$
\begin{aligned}
\frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial \omega_{lj}^{[1]}} &= \frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial \tilde{\Phi}(\mathbf{x}_i)} \frac{\partial \tilde{\Phi}(\mathbf{x}_i)}{\partial \mathbf{h}_{il}^{[1]}} \frac{\partial \mathbf{h}_{il}^{[1]}}{\partial \left(\mathbf{W}^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}^{[1]}\right)_l} \frac{\partial \left(\mathbf{W}^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}^{[1]}\right)_l}{\partial \omega_{lj}^{[1]}} \\
&= -2(y_i - \tilde{\Phi}(\mathbf{x}_i)) \frac{\partial(\omega_{11}^{[2]}\mathbf{h}_{i1}^{[1]} + \omega_{12}^{[2]}\mathbf{h}_{i2}^{[1]} + \omega_{13}^{[2]}\mathbf{h}_{i3}^{[1]} + \mathbf{b}^{[2]})}{\partial \mathbf{h}_{il}^{[1]}} \times \\
&\qquad \frac{\partial\, a\left(\left(\mathbf{W}^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}^{[1]}\right)_l\right)}{\partial \left(\mathbf{W}^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}^{[1]}\right)_l} \frac{\partial \left(\omega_{l1}\mathbf{h}_{i1}^{[0]} + \omega_{l2}\mathbf{h}_{i2}^{[0]} + \mathbf{b}_l^{[1]}\right)}{\partial \omega_{lj}^{[1]}} \\
&= -2(y_i - \tilde{\Phi}(\mathbf{x}_i))\omega_{1l}^{[2]} a'\left(\mathbf{w}_l^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}_l^{[1]}\right)\mathbf{h}_{ij}^{[0]} \\
&= \gamma_{li}\mathbf{h}_{ij}^{[0]}
\end{aligned} \tag{3.10}
$$

where $\frac{\partial a(x)}{\partial x} = \exp(-x)/(1+exp(x))^2$, $\gamma_{li} := -2(y_i - \tilde{\Phi}(\mathbf{x}_i))\omega_{1l}^{[2]} a'\left(\mathbf{w}_l^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}_l^{[1]}\right)$ and $\mathbf{w}_l^{[1]}$ is the l-th row of $\mathbf{W}^{[1]}$, $j \in \{1,2\}$, $l \in \{1,2,3\}$. The introduced variables δ_{1i} and γ_{li} are called **errors** and satisfy the equations

$$
\gamma_{li} = a'\left(\mathbf{w}_l^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}_l^{[1]}\right)\omega_{1l}^{[2]}\delta_{1i}, \qquad \text{where } \delta_{1i} := -2(y_i - \tilde{\Phi}(\mathbf{x}_i)), \tag{3.11}
$$

for $l \in \{1,2,3\}$ and $i = 1,\ldots,n$. Similar calculations yield the expressions

$$
\frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial b^{[2]}} = -2(y_i - \tilde{\Phi}(\mathbf{x}_i))
$$

and

$$
\frac{\partial \hat{R}_i(\tilde{\Phi}(\mathbf{x}_i))}{\partial b_l^{[1]}} = -2(y_i - \tilde{\Phi}(\mathbf{x}_i))\omega_{1l}^{[2]} a'\left(\mathbf{w}_l^{[1]}\mathbf{h}_i^{[0]} + \mathbf{b}_l^{[1]}\right)
$$

for $l \in \{1,2,3\}$.

A two-pass algorithm can be used to implement the updates in stochastic gradient descent using these equations (Hastie, Tibshirani, and Friedman 2016). Using gradient descent, the **two pass algorithm** proceeds as follows:

(i) Let $\boldsymbol{\theta}_t$ be the parameter vector obtained from the t-th iteration (or the starting parameters if we initiate the algorithm). First, compute the output of the forward pass of the neural network, namely $\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i)$ for $i = 1,\ldots,n$.

(ii) We continue with calculating $\frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial \omega_{1l}^{[2]}}$ for all $i = 1,\ldots,n$ and $l = 1,2,3$ using (3.9). In this step the *errors* δ_{1i} get calculated which are used to back propagate the error which yields γ_{li} for $i = 1,\ldots,n$ and $l = 1,2,3$ by using Equation (3.11).

(iii) The calculated errors γ_{li} are then utilized to get the values $\frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial \omega_{lj}^{[1]}}$ for $i = 1,\ldots,n$, $j = 1,2$ and $l = 1,2,3$, using (3.10).

(iv) Using the idea of gradient descent (discussed in Section 3.1), the weights in the t-th iteration are adjusted in the following way:

$$\omega_{lj}^{[1],t+1} = \omega_{lj}^{[1],t} - \alpha_t \sum_{i=1}^{n} \frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial \omega_{lj}^{[1]}},$$

$$\omega_{1l}^{[2],t+1} = \omega_{1l}^{[2],t} - \alpha_t \sum_{i=1}^{n} \frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial \omega_{1l}^{[2]}},$$

$$b^{[2],t+1} = b^{[2],t} - \alpha_t \sum_{i=1}^{n} \frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial b^{[2]}},$$

$$b_l^{[1],t+1} = b_l^{[1],t} - \alpha_t \sum_{i=1}^{n} \frac{\partial \hat{R}_i(\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))}{\partial b_l^{[1]}}$$

for $j \in \{1, 2\}$ and $l \in \{1, 2, 3\}$ and $\alpha_t \in \mathbb{R}$ a pre-defined learning rate. Overall, we obtain the updated parameter vector $\boldsymbol{\theta}_{t+1}$.

3.4 Regularization of neural networks

Training neural networks might result in overfitting the training data, meaning that we fit the model to the noise rather than the signal. A common approach is to add a penalty term to the loss function. In this thesis we use the L^2 error term, resulting in the empirical risk function

$$\hat{R}_{\text{reg}}(\Phi) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{\Phi}(\mathbf{x}_i))^2 \quad + \\ \frac{\lambda}{dk_1 \dots k_H}\left(\sum_{j=1}^{d}\sum_{j_1=1}^{k_1}(\omega_{j_1 j}^{[1]})^2 + \sum_{j_1=1}^{k_1}\sum_{j_2=1}^{k_2}(\omega_{j_2 j_1}^{[2]})^2 + \dots + \sum_{j_{H-1}=1}^{k_{H-2}}\sum_{j_H=1}^{k_H}(\omega_{j_{L-1} j_{L-2}}^{[L-1]})^2\right),$$

where $\lambda \in \mathbb{R}$ is a pre-defined hyperparameter, $\Phi \in \mathcal{K}_{(d,k_1,\dots,k_H,1)}$ and $(y_i, \mathbf{x}_i) \in D = \{(y_i, \mathbf{x}_i) \in \mathbb{R}\times\mathbb{R}^d | i = 1, \dots, n\}$. The penalty term encourages the sum of squares of the weights of the neural network to be small, which may prevent overfitting (Ng 2004).

In addition, we will apply the **dropout** method to neural networks in order to prevent overfitting. Dropout is a technique used in training neural networks, first introduced by Hinton et al. (2012). Essentially, during the training process, for each layer of the network, there's a pre-defined rate p that neurons will be removed for the respective iteration of the training process. The weights of the remaining neurons are then adjusted using backpropagation. The main motivation behind the algorithm is to prevent overfitting, by forcing neurons to be robust and rely on population behavior, rather than on the activity of other specific neurons (Baldi and Sadowski 2014). We demonstrate the dropout method through its application in Example 1.1.

Example 1.1 (continued) The dropout method is applied to the forward pass of the training iteration t in the following manner:

(i) First, set the drop rate p

(ii) For each training iteration t, sample $Y_l \sim \text{Ber}(p)$ for $l = 1, \dots, 3$, where Y_l are i.i.d. for every $l = 1, \dots, 3$

(iii) Next, calculate the dropout forward pass output for every sample $(y_i, \mathbf{x}_i) \in D = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^2 | i = 1, \dots, n\}$:

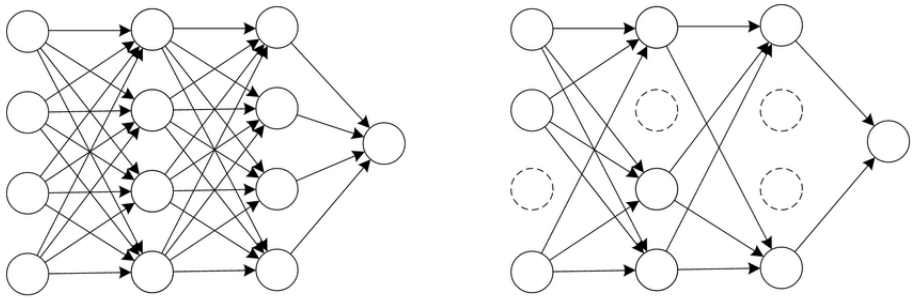
$$\tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i) = a\Bigg(\sum_{k=1}^{3}\omega_{1k}^{[2],t}\, a\Bigg(\sum_{l=1}^{3}\underbrace{Y_l\sum_{j=1}^{2}\omega_{lj}^{[1],t}(\mathbf{x}_i)_j}_{\text{value of the } l\text{-th neuron}} + b_l^{[1],t}\Bigg) + b^{[2],t}\Bigg)$$

(iv) Use **back propagation** to adjust the weights and biases of Φ and start the next training iteration $t+1$ by repeating the above procedure.

Depending on the drop rate, the training neural network can be sparse. Once the neural network has been trained, the Bernoulli variables Y_l, $\{1, 2, 3\}$ are eliminated, enabling the model to be utilized with new data which gives the realization:

$$\tilde{\Phi}^{\boldsymbol{\theta}}(\mathbf{x}^{\text{new}}) = a\Bigg(\sum_{k=1}^{3}\omega_{1k}^{[2]}\, a\Bigg(\sum_{l=1}^{3}\sum_{j=1}^{2}\omega_{lj}^{[1]}\mathbf{x}_j^{\text{new}} + b_l^{[1]}\Bigg) + b^{[2]}\Bigg),$$

with $\mathbf{x}^{\text{new}} \in \mathbb{R}^2$.



(a) Standard Neural Network (b) Network after Dropout

Figure 3.5 Dropout neural network model. (a) shows a standard neural network. (b) illustrates the same network with dropout applied during a training iteration. Dotted lines indicate nodes that have been dropped.(Srivastava et al. 2014)

3.5 Early stopping method

As we have seen in the first chapter, training a neural net is often done by utilizing the disjoint sets $D_{train} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{train}\}$, $D_{val} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{val}\}$ and $D_{test} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{test}\}$. The former two sets are used for the training algorithm (for instance mini-batch SGD) while the test set validates the model after the training is finished. The **training, validation and test errors of a neural network** $\Phi^{\boldsymbol{\theta}_t}$ at the iteration t are denoted as

$$\hat{R}^{\text{train}}(\Phi^{\boldsymbol{\theta}_t}) := \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} (y_i - \tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))^2, \quad (y_i, \mathbf{x}_i) \in D_{train},\ i = 1, \ldots, n_{train}\ , \tag{3.12}$$

$$\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t}) := \frac{1}{n_{val}} \sum_{i=1}^{n_{val}} (y_i - \tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))^2, \quad (y_i, \mathbf{x}_i) \in D_{val},\ i = 1, \ldots, n_{val} \quad \text{and} \tag{3.13}$$

$$\hat{R}^{\text{test}}(\Phi^{\boldsymbol{\theta}_t}) := \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i - \tilde{\Phi}^{\boldsymbol{\theta}_t}(\mathbf{x}_i))^2, \quad (y_i, \mathbf{x}_i) \in D_{test},\ i = 1, \ldots, n_{test}\ , \tag{3.14}$$

respectively. In the beginning of the chapter, we have already mentioned the **problem of overfitting** the neural net to the training data D_{train}. Overfitting a neural network Φ might result in an increasing value of the error term $\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t})$ for D_{val}, even though the error $\hat{R}^{\text{train}}(\Phi^{\boldsymbol{\theta}_t})$ of the training set D_{train} is still decreasing, where $1 \leq t \leq T$, $T \in \mathbb{N}$ is the maximum number of iterations of the training algorithm. The training error as well as the validation error can be visualized graphically, where we plot the errors against the number of iteration.
We observe, that the validation error has a turning point at a certain iteration. This simple visualization would suggest an easy early stopping rule:

(i) Train the neural network with the training set and evaluate the validation error after one iteration.

(ii) Stop training when the error term of the validation set is higher than the last one, meaning we have the scenario $\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t}) < \hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_{t+1}})$, where $1 \leq t \leq t+1 \leq T$, $T \in \mathbb{N}$.

(iii) Use the parameter vector $\boldsymbol{\theta}_t$ for the final model: $\Phi^{\boldsymbol{\theta}_t}$.

(iv) Lastly, evaluate the value by calculating the error on the test set, namely $\hat{R}^{\text{test}}(\Phi^{\boldsymbol{\theta}_t})$.

In reality, however, the validation set error is not as smooth as in Figure 3.6. In practice, validation error curves often exhibit multiple local minima. To address this problem, Demuth et al. (2014) introduced several **stopping criteria for training neural networks**. We will use the following stopping criteria for training neural networks. Before training, define a stopping threshold $\alpha > 0$ and the number of maximum iterations $T \in \mathbb{N}$. For each iteration $1 \leq t \leq T$, follow the below procedure

Figure 3.6 Number of iterations t vs. training error $\hat{R}^{\text{train}}(\Phi^{\boldsymbol{\theta}_t})$ and validation error $\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t})$ (Prechelt 2012).

(i) First, compute the validation error $\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t})$ using the validation set and the model $\Phi^{\boldsymbol{\theta}_t}$ which was computed in the t-th iteration, utilizing the training set.

(ii) Update the parameter vector $\boldsymbol{\theta}_t^{\text{opt}} := \min_{\boldsymbol{\theta}_{t'}, t' \in \{1,\ldots,t\}} \hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}'_t})$

(iii) Next, we calculate the **generalization loss** at iteration t as the relative increase of the validation error over the minimum validation error so far

$$GL(t) = 100 \cdot \left(\frac{\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t})}{\hat{R}^{\text{val}}(\Phi^{\boldsymbol{\theta}_t^{\text{opt}}})} - 1 \right) \tag{3.15}$$

(iv) Stop the training algorithm after the first iteration t^* when $GL(t^*) > \alpha$ and use the vector $\boldsymbol{\theta}_{t^*}^{\text{opt}}$ as the parameter vector for the final model.

4 D-vine copula theory

4.1 Notation and Definitions

We start by defining the copula approach to multivariate data, following the definition and notation of Czado (2019).

Definition 4.1.1. (Copula and copula density)

- *A d-dimensional copula C is a multivariate distribution function on the d-dimensional hypercube $[0,1]^d$ with uniformly distributed marginals.*
- *The corresponding copula density for an absolutely continuous copula we denote by c can be obtained by partial differentiation, i.e., $c(u_1,...,u_d) := \frac{\partial^d}{\partial u_1 ... \partial u_d} C(u_1,...,u_d)$ for all $\mathbf{u} \in [0,1]^d$.*

Theorem 4.1.1. (Sklar's Theorem)
Let $\mathbf{X}$ be a d-dimensional random vector with joint distribution F and marginal distributions F_i, $i = 1,\dots,d$, then the joint distribution can be expressed as

$$F(x_1,\dots,x_d) = C(F_1(x_1),\dots,F_d(x_d)) \tag{4.1}$$

with associated density

$$f(x_1,\dots,x_d) = c(F_1(x_1),\dots,F_d(x_d)) f_1(x_1)\dots f_d(x_d) \tag{4.2}$$

for some d-dimensional copula C with copula density c. For absolutely continuous distributions, the copula C is unique.

If we are primarily interested in the dependence structure of the random vector $\mathbf{X} \in \mathbb{R}^d$, we can examine it on the *u-scale* (or copula scale) by applying the probability integral transform (PIT) to its marginals: $U_j := F_j(X_j)$, for $j = 1,\dots,d$.

Lemma 4.1.1. (Conditional densities and distribution functions of bivariate distributions in terms of their copula) *The conditional density and distribution function can be rewritten as*

$$\begin{aligned} f_{X_1|X_2}(x_1|x_2) &= c_{U_1U_2}(F_1(x_1), F_2(x_2)) f_2(x_2) \\ F_{X_1|X_2}(x_1|x_2) &= \frac{\partial}{\partial u_2} C_{U_1U_2}(F_1(x_1), u_2)|_{u_2=F_2(x_2)} \\ &=: \frac{\partial}{\partial F_2(x_2)} C_{U_1U_2}(F_1(x_1), F_2(x_2)) \end{aligned}$$

Proof. The proof can be found on page 20 in the book of Czado (2019).
Note that we can apply Lemma (4.1.1) to the bivariate copula distribution $C_{U_1U_2}$ which yields

$$C_{U_1|U_2}(u_1|u_2) = \frac{\partial}{\partial u_2} C_{U_1U_2} \quad \forall u_1 \in [0,1] \tag{4.3}$$

We continue by introducing notation following Kraus and Czado (2017). For a set $D \subset \{1,\dots,d\}$ and $i,j \in \{1,\dots,d\}\backslash D$ we write

- Let $C_{X_1X_2;\mathbf{X}_D}(\cdot,\cdot;\mathbf{x}_D)$ denote the copula associated with the conditional distribution $(X_i, X_j)^T$ given $\mathbf{X}_D = \mathbf{x}_D$. We use the abbreviation $C_{ij|D}(\cdot,\cdot|\mathbf{x}_D)$.
- Let $F_{X_i|\mathbf{X}_D(\cdot|\mathbf{x}_D)}$ denote the conditional distribution of the random variable X_i given $\mathbf{X}_D = \mathbf{x}_D$. We use the abbreviation $F_{i|\mathbf{x}_D}$.
- Let $C_{U_i|\mathbf{U}_D}(\cdot|\mathbf{u}_D)$ denote the conditional distribution of the PIT random variable U_i given $\mathbf{U}_D = \mathbf{u}_D$. We use $C_{i|D}(\cdot|\mathbf{u}_D)$ as an abbreviation.

We continue by defining the so called **D-vine** copula following Czado (2010).

Definition 4.1.2. (D-vine density and copula)
Let $\mathbf{X}$ be a d-dimensional continuously distributed random vector. The joint density f of $\mathbf{X}$ can be composed as

$$f(x_1,\dots,x_d) = \prod_{k=1}^{d} f_k(x_k) \prod_{i=1}^{d}\prod_{j=i+1}^{d} c_{ij;i+1,\dots,j-1}\big(F_{i|i+1,\dots,j-1}(x_i|x_{i+1},\dots,x_{j-1}), F_{j|i+1,\dots,j-1}(x_j|x_{i+1},\dots,x_{j-1}); x_{i+1},\dots,x_{j-1}\big) \quad (4.4)$$

*This **pair-copula construction (PCC)** is called a **D-vine density** with order $X_1 - X_2 - \cdots - X_d$. The copula associated with the density is called a **D-vine copula**.*

Bedford and Cooke (2002) introduced a graphical representation of the D-vine copula, where each edge of the graph corresponds to a pair-copula, see Figure 4.1. In the theory of vine copulas, the following assumption is often made.

Definition 4.1.3. (Simplifying assumption) *Let $D \subset \{1,\dots,d\}$. The D-vine copula is called simplified, if*

$$c_{ij;D}\left(F_{i|D}(x_i|\mathbf{x}_D), F_{j|D}(x_j|\mathbf{x}_D); \mathbf{x}_D\right) = c_{ij,D}\left(F_{i|D}(x_i|\mathbf{x}_D), F_{j|D}(x_j|\mathbf{x}_D)\right) \quad (4.5)$$

for all $i, j \in \{1,\dots,d\}\backslash D$.

As an example we give the density of a 4-dimensional simplified D-vine density

$$\begin{aligned} f_{1,2,3,4}(x_1,x_2,x_3,x_4) = \left[\prod_{i=1}^{4} f_i(x_i)\right] &\cdot c_{12}(x_1,x_2)\cdot c_{23}(x_2,x_3)\cdot c_{34}(x_3,x_4) \\ &\cdot c_{13;2}\left(F_{1|2}(x_1|x_2), F_{3|2}(x_3|x_2)\right)\cdot c_{24;3}\left(F_{2|3}(x_2|x_3), F_{4|3}(x_4|x_3)\right) \\ &\cdot c_{14;23}\left(F_{1|23}(x_1|x_2,x_3), F_{4|23}(x_4|x_2,x_3)\right). \end{aligned}$$

Note that in the setting of a D-vine, the first order $X_1 - \cdots - X_d$ fully determines the structure of the graphical representation. This is called the **proximity**.

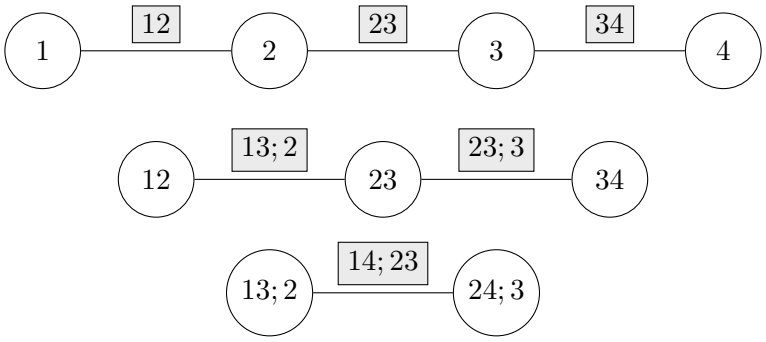


Figure 4.1 Graphical representation of a 4-dimensional D-vine copula. The squares above the edges represent the pair copulas.

Definition 4.1.4. (h-functions) *For the bivariate copula $C_{ij}(u_i, u_j; \theta_{ij})$ with parameter θ_{ij}, we define the h-functions*

$$h_{i|j}(u_i|u_j;\theta_{ij}) = \frac{\partial}{\partial u_j} C_{ij}(u_i, u_j;\theta_{ij}) \tag{4.6}$$

$$h_{j|i}(u_j|u_i;\theta_{ij}) = \frac{\partial}{\partial u_i} C_{ij}(u_i, u_j;\theta_{ij}) \tag{4.7}$$

For the parametric pair copula $C_{e_a,e_b;D_e}(w_1, w_2, \theta_{e_a,e_b;D_e})$ in a simplified D-vine corresponding to the edge $e_a, e_b; D_e$, we introduce the notation

$$h_{e_a|e_b;D_e}(w_1|w_2;\theta_{e_a,e_b;D_e}) := \frac{\partial}{\partial w_2} C_{e_a,e_b;D_e}(w_1, w_2;\theta_{e_a,e_b;D_e}) \tag{4.8}$$

$$h_{e_b|e_a;D_e}(w_2|w_1;\theta_{e_a,e_b;D_e}) := \frac{\partial}{\partial w_1} C_{e_a,e_b;D_e}(w_1, w_2;\theta_{e_a,e_b;D_e}) \tag{4.9}$$

We are able to describe the conditional distribution $F_{i|D}(x_i|\mathbf{x}_D)$ with the bivariate copulas of the lower trees of the D-vine using the recursion formula first stated in Joe (1997):

$$F_{i|D}(x_i|\mathbf{x}_D) = h_{i|l;D_{-l}}(F_{i|D_{-l}}(x_i|\mathbf{x}_{D_{-l}})|F_{l|D_{-l}}(x_l|\mathbf{x}_{D_{-l}})), \tag{4.10}$$

where $l \in D$ and $D_{-l} := D\backslash\{l\}$.

In order to quantify the dependency between two random variables, we introduce the dependence measure Kendall's tau.

Definition 4.1.5. (Kendall's Tau) *The Kendall's τ between the continuous random variables X_1 and X_2 is defined as*

$$\tau(X_1, X_2) = P((X_{11} - X_{21})(X_{12} - X_{22} > 0)) - P((X_{11} - X_{21})(X_{12} - X_{22} < 0)),$$

where (X_{11}, X_{12}) and (X_{21}, X_{22}) are independent and identically distributed copies of (X_1, X_2).

Example 4.1.1. In the following example, the structure and formulas of a D-Vine copula with order $V - U_1 - U_2$ will be studied. The bivariate copulas C_{VU_1}, $C_{U_1U_2}$ and $C_{VU_2|U_1}$ are given by a Clayton copula with parameter δ_{V1}, δ_{12} and $\delta_{V2;1}$, respectively. The Clayton copula is defined as

$$C_{U_1U_2}(u_1, u_2) = (u_1^{-\delta} + u_2^{-\delta} - 1)^{-\frac{1}{\delta}},$$

for $0 \leq \delta \leq \infty$. For this copula the h-function is described by

$$\begin{aligned} h_{U_1|U_2}(u_1, u_2) &= F_{U_1|U_2}(u_1|u_2) \\ &= \frac{\partial}{\partial u_2} C_{U_1U_2}(u_1, u_2) \\ &= u_2^{-\delta-1}\left[\left(u_1^{-\delta} + u_2^{-\delta} - 1\right)^{-1-\frac{1}{\delta}}\right] \end{aligned}$$

with its corresponding inverse

$$h^{-1}_{U_1|U_2}(\alpha|u_2) = \left(\alpha^{-\frac{\delta}{1+\delta}} u_2^{-\delta} - u_2^{-\delta} + 1\right)^{-\frac{1}{\delta}}.$$

We are interested in the conditional distribution of V given (U_1, U_2) which can recursively be written as

$$\begin{aligned} C_{V|U_1,U_2}(v|u_1, u_2) &= h_{V|U_2;U_1}\left(C_{V|U_1}(v|u_1)|C_{U_2|U_1}(u_2|u_1)\right) \\ &= h_{V|U_2;U_1}\left(h_{V|U_1}(v|u_1)|h_{U_2|U_1}(u_2|u_1)\right), \end{aligned} \tag{4.11}$$

where we used (4.10). Inversion yields

$$C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2) = h^{-1}_{V|U_1}(h^{-1}_{V|U_2;U_1}(\alpha|h_{U_2|U_1}(u_2|u_1))|u_1),$$

for $\alpha \in [0,1]$. Plugging in the formulas of our example D-vine copula leads to

$$\begin{aligned}
C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2) &= h^{-1}_{V|U_1}(h^{-1}_{V|U_2;U_1}(\alpha|h_{U_2|U_1}(u_2|u_1))|u_1) \\
&= h^{-1}_{V|U_1}(h^{-1}_{V|U_2;U_1}(\alpha|u_2^{-\delta_{V2;1}-1}(u_1^{-\delta_{V2;1}-1}+u_2^{-\delta_{V2;1}-1}-1)^{-1-\frac{1}{\delta_{V2;1}}})|u_1) \\
&= h^{-1}_{V|U_1}\left[\left((u_1^{\delta_{V2;1}(\delta_{12}+1)}(u_2^{-\delta_{12}}+u_1^{-\delta_{12}}-1)^{1+\frac{\delta_{V2;1}}{\delta_{12}}})(\alpha^{-\frac{\delta_{V2;1}}{\delta_{V2;1}+1}}-1)+1\right)^{-\frac{1}{\delta_{V2;1}}}\middle| u_1\right] \\
&= \left[\left((u_1^{\delta_{V2;1}(\delta_{12}+1)}(u_2^{-\delta_{12}}+u_1^{-\delta_{12}}-1)^{1+\frac{\delta_{V2;1}}{\delta_{12}}})(\alpha^{-\frac{\delta_{V2;1}}{\delta_{V2;1}+1}}-1)+1\right)^{\frac{\delta_{V1}}{\delta_{V2;1}(\delta_{V1}+1)}} u_1^{-\delta_{V1}}-u_1^{-\delta_{V1}}+1\right]^{-\frac{1}{\delta_{V1}}},
\end{aligned}$$

for $\alpha \in [0,1]$.

The dependency between two random variables can be measured by Kendall's τ. For the bivariate copula families which we use in this thesis, there is a one to one correspondence between Kendall's tau and the family parameter. Following the example of a Clayton copula C_{VU_1} with family parameter δ_{VU_1}, the relationship is given

$$\tau = \frac{\delta_{VU_1}}{\delta_{VU_1}+2}, \tag{4.12}$$

where $\tau \in (0,1)$.

Algorithm 4.1.1. Simulating from a multivariate copula (using the inverse Rosenblatt transform)

- **Start**: *Sample i.i.d.* $w_j \sim U[0,1]$, $j = 1,...,d$
- **Then**:

$$\begin{aligned}
u_1 &:= w_1 \\
u_2 &:= C^{-1}_{2|1}(w_2|u_1), \\
&\vdots \\
u_d &:= C^{-1}_{d|d-1,...,1}(w_d|u_{d-1},...,u_1)
\end{aligned}$$

For more details regarding the **inverse Rosenblatt transform** look into Rosenblatt (1952) and Rüschendorf (1981).

4.2 D-Vine quantile regression

The following model and algorithm was introduced and studied in Kraus and Czado (2017). As we have seen in Equation (5.1), the conditional quantile function $F^{-1}_{Y|X_1,...,X_d}$ can be expressed by the inverse marginal function F_Y^{-1} and the conditional copula quantile function $C^{-1}_{V|U_1,...,U_d}(\alpha|u_1,\dots,u_d)$ conditioned on the PIT values of $\mathbf{x}$. Estimating the marginals F_Y and F_{X_j} for $j = 1,\dots,d$, as well as the copula $C_{V,U_1,...,U_d}$ and plugging the estimates into (5.1) gives us an estimation of the conditional α-quantile:

$$\hat{q}_\alpha(x_1,\dots,x_d) := \hat{F}_Y^{-1}\left(\hat{C}^{-1}_{V|U_1,...,U_d}(\alpha|\hat{u}_1,\dots,\hat{u}_d)\right), \tag{4.13}$$

where $\hat{u}_j := \hat{F}_j(x_j)$ is the estimated PIT of x_j, $j = 1, \ldots, d$. The marginals are estimated non-parametrically via the kernel smoothing estimator, which is defined as

$$\hat{F}_{X_j}(x) := \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{(x - x_{ij})}{h}\right), \quad x \in \mathbb{R}, \tag{4.14}$$

where $(x_{ij})_{i=1,\ldots,n}$ is an i.i.d. sample of $(X_j)_{j\in\{1,\ldots,d\}}$, $h > 0$ a bandwidth parameter and $K(x) := \int_{-\infty}^{x} k(t)dt$ with $k(\cdot)$ being a symmetric probability density function (Parzen 1962). This estimator is implemented in the R package `kde1d` (Nagler and Vatter 2022). The estimated marginal distribution functions $\hat{F}_Y$ and $\hat{F}_{X_j}$ are utilized to transform the i.i.d. samples $(y_i, \mathbf{x}_i) \sim (Y, X_1, \ldots, X_d)$, $i = 1, \ldots, n$, to pseudo copula data

$$(\hat{v}_i, \hat{\mathbf{u}}_i) = (\hat{v}_i, \hat{u}_{i1}, \ldots, \hat{u}_{id}) := \left(\hat{F}_Y(y_i), \hat{F}_{X_1}(x_{i1}), \ldots, \hat{F}_{X_d}(x_{id})\right)$$

for $i = 1, \ldots, n$. The data $((\hat{v}_i, \hat{\mathbf{u}}_i))_{i=1,\ldots,n}$ is an approximately i.i.d. sample from the random vector $(V, U_1, \ldots, U_d)$.
In the next step, the pseudo copula data is used to fit a D-vine copula $V - U_{l_1} - \cdots - U_{l_d}$ to the training data set. We aim to select an order $\mathbf{l} = (l_1, \ldots, l_d)^T$ that maximizes the explanatory power of the model. While one approach could be to evaluate all $d!$ potential sequences, this is not a practical solution. As a result, Kraus and Czado (2017) proposed a method that progressively chooses the most impactful covariates. At each stage, we include the covariate that improve the model's fit the most, with respect to the conditional log-likelihood measure (cll)

$$\text{cll}\left(\mathbf{l}, \hat{\boldsymbol{\mathcal{F}}}, \hat{\boldsymbol{\theta}}; \hat{\mathbf{v}}, \hat{\mathbf{U}}\right) := \sum_{i=1}^{n} \ln c_{V|U_{l_1},\ldots U_{l_d}}\left(\hat{v}_i | \hat{u}_{il_1} \ldots \hat{u}_{il_d}; \mathbf{l}, \hat{\boldsymbol{\mathcal{F}}}, \hat{\boldsymbol{\theta}}\right), \tag{4.15}$$

where $\hat{\boldsymbol{\mathcal{F}}}$ are the estimated parametric pair-copula families with the corresponding copula parameters $\hat{\boldsymbol{\theta}}$ and $\hat{\mathbf{v}} := (\hat{v}_1, \ldots, \hat{v}_n)^T$, $\hat{\mathbf{U}} = (\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_n)^T$. The density of the conditional copula $c_{V|U_{l_1},\ldots U_{l_d}}$ can be expressed as the multiplication of all pair-copulas in the D-vine that include V, as described in Killiches, Kraus, and Czado (2016)

$$\begin{aligned} c_{V|U_{l_1},\ldots,U_{l_d}}\left(\hat{v}_i | \hat{u}_{il_1} \ldots \hat{u}_{il_d}; \mathbf{l}, \hat{\boldsymbol{\mathcal{F}}}, \hat{\boldsymbol{\theta}}\right) &= c_{VU_{l_1}}(\hat{v}_i, \hat{u}_{il_1}; \hat{\mathcal{F}}_{VU_{l_1}}, \hat{\theta}_{VU_{l_1}}) \times \\ \prod_{j=2}^{d} c_{VU_{l_j};U_{l_1},\ldots,U_{l_{j-1}}}\Big(&\hat{C}_{V|U_{l_1},\ldots,U_{l_{j-1}}}(\hat{v}_i | \hat{u}_{il_1}, \ldots, \hat{u}_{il_{j-1}}), \hat{C}_{U_{l_j}|U_{l_1},\ldots,U_{l_{j-1}}}(\hat{u}_{il_j} | \hat{u}_{il_1}, \ldots, \hat{u}_{il_{j-1}}); \\ &\hat{\mathcal{F}}_{VU_{l_j};U_{l_1}\ldots,U_{l_{j-a}}}, \hat{\theta}_{VU_{l_j};U_{l_1}\ldots,U_{l_{j-a}}}\Big), \end{aligned}$$

where $\hat{\mathcal{F}}_I$ and $\hat{\theta}_I$ denote the estimated family and parameters(s) of the pair copula C_I.

4.2.1 Forward selection algorithm

The algorithm which sequentially constructs a D-vine while maximizing the model's conditional log-likelihood in each step was introduced in Kraus and Czado (2017) and operates as follows. At the start of the k-th step, the optimal D-vine already includes $k - 1$ predictors (see Figure 4.2 for a visual representation). For each remaining variable U_j that hasn't been selected, we fit the necessary pair-copulas to expand the current model to a D-vine with order $V - U_{l_1} - \cdots - U_{l_{k-1}} - U_j$ (represented by the gray circles) and calculate the resulting model's conditional log-likelihood.

The model is then updated by incorporating the variable with the highest conditional log-likelihood, completing the k-th step. This method allows for the sequential ordering of covariates based on their predictive power. If at the k-th step, none of the remaining covariates can improve the model's conditional log-likelihood, the algorithm halts. It then returns the model that includes only the $k - 1$ covariates selected up to that point.

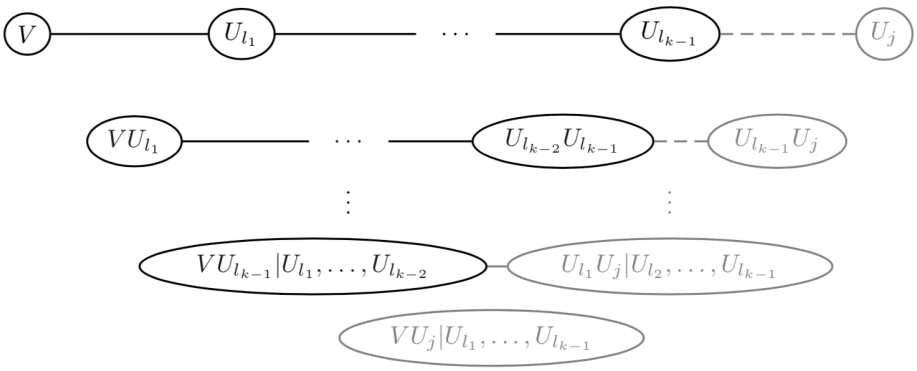


Figure 4.2 Extending the current D-vine (black) by incorporating U_j in the k-th step of the algorithm. The gray pair-copulas are the ones that need to be estimated. Visualization adapted from Kraus and Czado (2017).

The resulting estimated D-vine copula $\hat{C}_{V|U_{j_1},\ldots,U_{j_k}}$ for some $k \in \{1,\ldots,d\}$ is then used to calculate $\hat{C}^{-1}_{V|U_{j_1},\ldots,U_{j_k}}(\alpha|\hat{u}_{j_1}\ldots\hat{u}_{j_k})$ via nested inverse h-functions. Finally, Equation (4.13) is applied to obtain an estimate for the conditional α-quantile function.

4.3 Approximation of D-vines using neural networks

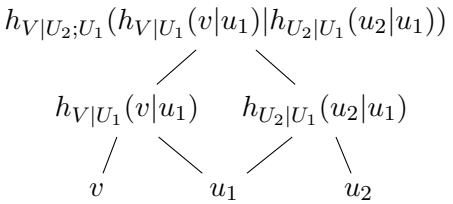
In the last section, it was demonstrated that the conditional copula quantile function $C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2)$ and the conditional copula distribution $C_{V|U_1,U_2}(v|u_1,u_2)$ are a concatenation of non-linear bivariate functions. Instead of directly approximating the functions, we aim to approximate $C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2)$ and $C_{V|U_1,U_2}(v|u_1,u_2)$ using the hierarchical structure. This method aims to increase the approximation rate and to decrease the weights required to achieve a universal approximation error $\epsilon > 0$, similar to the approach used in Gühring, Raslan, and Kutyniok (2020), and more theoretically in Mhaskar and Poggio (2016).
We introduce the concept of neural networks with one hidden layer (so-called **shallow neural networks**) as **universal approximators**.

Theorem 4.3.1. *For every $f \in C(K)$ (set of continuous functions on the domain K), $K \subset \mathbb{R}^d$, K is compact and every $\epsilon > 0$ as well as every non-constant, bounded and monotonically increasing continuous activation function $\sigma : \mathbb{R} \to \mathbb{R}$, there exists some $N \in \mathbb{N}$ and some shallow neural network Φ_ϵ with the number of weights $M_1(\Phi) = N$ such that*

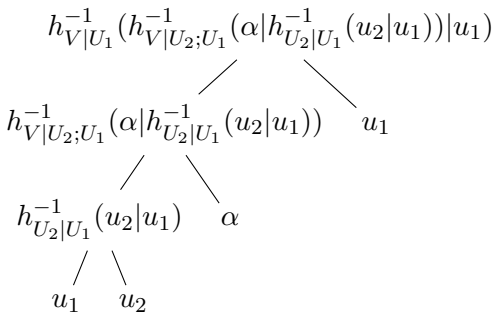
$$\max_{\mathbf{x}\in K} |f(\mathbf{x}) - \Phi_\epsilon(\mathbf{x})| \leq \epsilon \quad .$$

This was first proven in Funahashi (1989). Note that the *sigmoid* function is a non-constant, bounded and monotonically increasing continuous function. The universal approximation theorem can be extended to deep neural networks (neural networks with number of layers $L > 2$) with ReLU activation functions, which shown in Hanin (2019).

We continue by constructing the approximation of $C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2)$ and $C_{V|U_1,U_2}(v|u_1,u_2)$ of a D-Vine with structure $V - U_1 - U_2$ by concatenating shallow neural nets with activation function σ. The concatenating structure of $C_{V|U_1,U_2}(v|u_1,u_2)$ can be decomposed to the graph



using Equation (4.11). And similarly we draw the corresponding graph for $C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2)$



In case of approximating $C_{V|U_1,U_2}(v|u_1,u_2)$, the idea is to construct two shallow neural networks in order to estimate $h_{V|U_1}(v|u_1)$ and $h_{U_2|U_1}(u_2|u_1)$, respectively. The output of these neural networks is then fed into another neural network to estimate $h_{V|U_2;U_1}$, which gives us an approximation of $C_{V|U_1,U_2}(v|u_1,u_2)$. The issue with training such a model is that we only have given $(v,u_1,u_2)^T$ and no validation data (true data) for the target value $C_{V|U_1,U_2}(v|u_1,u_2)$. The question arises if we can approximate $C_{V|U_1,U_2}(v|u_1,u_2)$

and $C^{-1}_{V|U_1,U_2}(\alpha|u_1,u_2)$ by only using training data $(v,u_1,u_2)^T$. A possible solution is to adopt a similar approach to Sun, Cuesta-Infante, and Veeramachaneni (2018), where the copula configuration is determined by a reinforcement learning algorithm and the log-likelihood functions are used as the loss function. However, this approach does not guarantee the Markovian property. The paper proposes using an LSTM method to address this issue. An LSTM, or Long Short-Term Memory, introduced by Hochreiter and Schmidhuber (1997), is a type of neural network architecture specifically designed to handle sequence prediction problems with long term dependencies.

5 Quantile regression neural network (QRNN)

We are interested in the conditional quantile function $F^{-1}_{Y|X_1,\dots,X_d}$ which can be expressed as

$$q_\alpha(x_1,\dots,x_d) = F^{-1}_{Y|X_1,\dots,X_d}(\alpha|x_1,\dots,x_d) = F^{-1}_Y\left(C^{-1}_{V|U_1,\dots,U_d}(\alpha|u_1,\dots,u_d)\right), \tag{5.1}$$

where $X_j \sim F_j$, $Y \sim F_Y$ and $V := F_Y(Y)$, $U = F_j(X_j)$ are the PIT transformations for $j \in \{1,\dots,d\}$ and $\alpha \in (0,1)$. This representation was shown in Kraus and Czado (2017). We again use $q_\alpha(\mathbf{x})$ or even shorter q_α as an abbreviation. In this section, we want to estimate the vector

$$q_{\boldsymbol{\alpha}} = (q_{\alpha_1},\dots,q_{\alpha_K})^T \in \mathbb{R}^K,$$

where $\boldsymbol{\alpha} \in (0,1)^K$, using neural networks. We initially construct a neural network to estimate a single quantile q_α, $\alpha \in (0,1)$. Later, we extend the neural network to estimate the vector $q_{\boldsymbol{\alpha}}(x_1,\dots,x_d)$ for $\boldsymbol{\alpha} \in (0,1)^K$. The idea of a quantile regression neural network is to equip a neural network with the pinball loss function to get estimates for $q_\alpha(x_1,\dots,x_d)$ for a fixed $\alpha \in (0,1)$.

Definition 5.0.1. ***(QRNN)*** *Let $d, k_1,\dots,k_H \in \mathbb{N}$ for some $H \in \mathbb{N}$, $a : \mathbb{R} \to \mathbb{R}$ an activation function, $\alpha \in (0,1)$ and $\rho_\alpha : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ be the pinball loss function. Let*

$$\Phi^{\boldsymbol{\theta},a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)} \in \mathcal{K}^{a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)} \tag{5.2}$$

denote the quantile regression neural network with input dimension d, output dimension 1, H hidden layers, where k_h is the number of neurons in the h-th layer. The neural network is equipped with the activation function a, loss function ρ_α and parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{M(\Phi)}$, where $M(\Phi)$ is the number of weights of the quantile regression neural network. If it is not stated otherwise, we set a to be the sigmoid function. We often drop the parameter vector $\boldsymbol{\theta}$ and the vector $(d,k_1,\dots,k_H,1)$ and work with the shortened notation $\Phi^{\rho_\alpha} \in \mathcal{K}^{a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)}$.

We again denote the realization of the quantile regression neural network $\Phi^{\boldsymbol{\theta},a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)}$ and its abbreviation Φ^{ρ_α} by

$$\tilde{\Phi}^{\boldsymbol{\theta},a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)} : \mathbb{R}^d \to \mathbb{R}, \quad \tilde{\Phi}^{\boldsymbol{\theta},a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)}(\mathbf{x}) = \mathbf{h}^{[H+1]} \tag{5.3}$$

or shorter

$$\tilde{\Phi}^{\rho_\alpha} : \mathbb{R}^d \to \mathbb{R}, \quad \tilde{\Phi}^{\rho_\alpha}(\mathbf{x}) = \mathbf{h}^{[H+1]}, \tag{5.4}$$

where $\mathbf{h}^{[H+1]}$ results from the calculation scheme in Definition 3.2.1. Following the idea of stochastic gradient descent, we try to find the parameter vector $\boldsymbol{\theta}$ which minimizes the empirical risk function

$$\hat{R}(\Phi^{\boldsymbol{\theta},\rho_\alpha}) = \frac{1}{n}\sum_{i=1}^{n} \rho_\alpha(y_i, \tilde{\Phi}^{\boldsymbol{\theta},\rho_\alpha}(\mathbf{x}_i)), \tag{5.5}$$

for $\Phi^{\boldsymbol{\theta},\rho_\alpha} \in \mathcal{K}^{a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)}$, $\alpha \in (0,1)$ and $(y_i,\mathbf{x}_i) \in D = \{(y_i,\mathbf{x}_i) \in \mathbb{R}\times\mathbb{R}^d | i = 1,\dots,n\}$, $i = 1,\dots,n$. The resulting model is a nonlinear version of quantile regression, including interactions between predictors without prior specification of the form of the relationships, and is called quantile regression neural network or QRNN (Cannon 2011). We slightly deviate from Cannon (2011), who defined the QRNN to have only one hidden layer and only used the sigmoid function as the activation function a.

We now want to extend the model and simultaneously estimate the quantiles at levels $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_K)$.

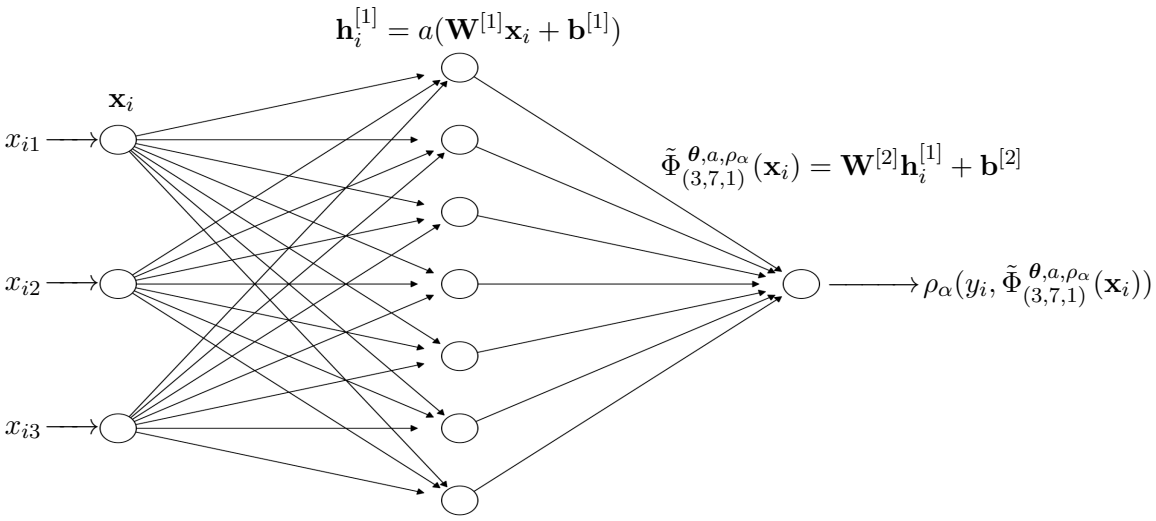


Figure 5.1 Visualization of the forward pass of a quantile regression neural network $\Phi^{\boldsymbol{\theta},a,\rho_\alpha}_{(3,7,1)} \in \mathcal{K}^{a,\rho_\alpha}_{(3,7,1)}$ with an arbitrary activation function a for estimating $q_\alpha(\mathbf{x}_i)$ for a fixed $\alpha \in \mathbb{R}$. We have $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})^T \in \mathbb{R}^3$, $\mathbf{W}^{[1]} \in \mathbb{R}^{7\times 3}$, $\mathbf{b}^{[1]} \in \mathbb{R}^7$, $\mathbf{W}^{[2]} \in \mathbb{R}^{1\times 7}$, $\mathbf{b}^{[2]} \in \mathbb{R}^1$ and $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$ and $\alpha \in (0, 1)$.

Definition 5.0.2. ***(K-QRNN)*** *Let $d, k_1, \ldots, k_H \in \mathbb{N}$ for some $H \in \mathbb{N}$, $a : \mathbb{R} \to \mathbb{R}$ an activation function, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^T \in (0, 1)^K$, $K \in \mathbb{N}$. Let*

$$\rho_{\boldsymbol{\alpha}} : \mathbb{R} \times \mathbb{R}^K \to \mathbb{R}, \quad \rho_{\boldsymbol{\alpha}}(y, h(\mathbf{x})) := \frac{1}{K}\sum_{k=1}^{K} \rho_{\alpha_k}(y, (h(\mathbf{x}))_k), \tag{5.6}$$

with $h : \mathbb{R}^d \to \mathbb{R}^K$ and ρ_{α_k} the pinball loss for α_k, $k = 1, \ldots, K$. Let

$$\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)} \tag{5.7}$$

denote the quantile regression neural network with input dimension d, output dimension K, H hidden layers, where k_h is the number of neurons in the h-th layer. The neural network is equipped with the activation function a, loss function $\rho_{\boldsymbol{\alpha}}$ and parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{M(\Phi)}$, where $M(\Phi)$ is the number of weights of the quantile regression neural network. If it is not stated otherwise, we set a to be the sigmoid function. We often drop the parameter vector $\boldsymbol{\theta}$ and the vector $(d, k_1, \ldots, k_H, K)$ and work with the shortened notation $\Phi^{\rho_{\boldsymbol{\alpha}}} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}$.

As in the one-dimensional QRNN, the realization of the network $\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}$ and its abbreviation $\Phi^{\rho_{\boldsymbol{\alpha}}}$ is notated as $\tilde{\Phi}^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}(\mathbf{x})$ and $\tilde{\Phi}^{\rho_{\boldsymbol{\alpha}}}(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$, respectively. Note that the realization of the K-QRNN network is of the from

$$\tilde{\Phi}^{\rho_{\boldsymbol{\alpha}}}(\mathbf{x}) = (\tilde{\Phi}^{\rho_{\alpha_1}}(\mathbf{x}), \ldots, \tilde{\Phi}^{\rho_{\alpha_K}}(\mathbf{x}))^T \in \mathbb{R}^K,$$

for $\mathbf{x} \in \mathbb{R}^d$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^T \in (0, 1)^K$ The empirical risk function, which we use in K-QRNN is given by

$$\hat{R}^{\text{QRNN}}(\Phi^{\rho_{\boldsymbol{\alpha}}}) = \frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n} \rho_{\alpha_k}(y_i, \Phi^{\rho_{\alpha_k}}(\mathbf{x}_i)),$$

where $\Phi^{\rho_{\boldsymbol{\alpha}}} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^T \in (0, 1)^K$ and $(y_i, \mathbf{x}_i) \in D$, $i = 1, \ldots, n$. The resulting model is a composite version of the QRNN model introduced and studied in Xu et al. (2017).

The dimension of the input variables d as well as the number of neurons $(k_1, \ldots, k_H)$, $H \in \mathbb{N}$, determine the complexity of the QRNN model. Setting the number of neurons too high might result in overfitting the training data. In order to prevent overfitting, L^2 regularization is applied to the QRNN model. In case of

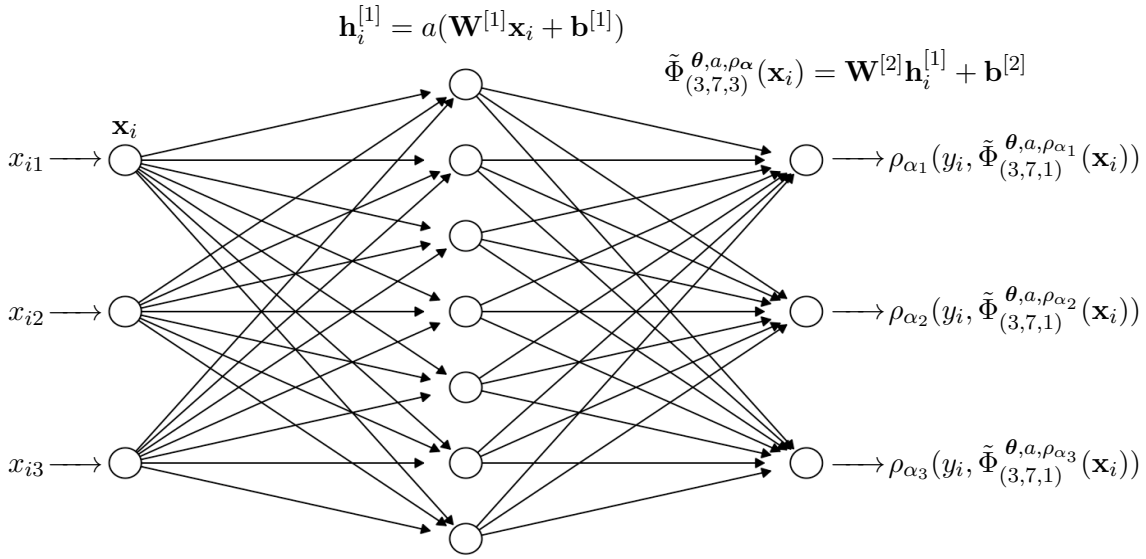


Figure 5.2 Visualization of the forward pass of a quantile regression neural network $\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(3,7,3)} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(3,7,3)}$ with an arbitrary activation function a. We have $\mathbf{x}_i \in \mathbb{R}^3$, $\mathbf{W}^{[1]} \in \mathbb{R}^{7\times 3}$, $\mathbf{b}^{[1]} \in \mathbb{R}^7$, $\mathbf{W}^{[2]} \in \mathbb{R}^{3\times 7}$, $\mathbf{b}^{[2]} \in \mathbb{R}^3$ and $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$ and $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3)^T \in (0,1)^3$.

using the sigmoid function, regularization leads to the reduction of the non-linearity of the model, since the function acts linear around the origin. The modified empirical risk function is defined as

$$\hat{R}^{\text{QRNN}}_{\text{reg}}(\Phi^{\rho_{\boldsymbol{\alpha}}}) = \frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\rho_{\alpha_k}(y_i, \tilde{\Phi}^{\rho_{\alpha_k}}(\mathbf{x}_i)) + \frac{\lambda}{dk_1\ldots k_H K}\left(\sum_{j=1}^{d}\sum_{j_1=1}^{k_1}(\omega^{[1]}_{j_1 j})^2 + \cdots + \sum_{j_{H-1}=1}^{k_{H-2}}\sum_{j_H=1}^{k_H}(\omega^{[H]}_{j_H j_{H-1}})^2 + \sum_{j_H=1}^{k_H}\sum_{j_{H+1}=1}^{K}(\omega^{[H+1]}_{j_{H+1} j_H})^2\right), \quad (5.8)$$

where $\lambda > 0$ controls the relative contribution of the penalty term. Training a QRNN model is executed by calculating the gradient of the empirical risk function $\hat{R}^{\text{QRNN}}(\Phi_{\boldsymbol{\alpha}})$ and back propagate it to adjust the weights in the training process, using the idea of gradient descent. As we have already observed, however, that the pinball loss ρ_α is not differentiable at the origin.

We therefore use the **Huber pinball loss** $\rho^{(H)}_{\alpha,\epsilon}(x)$ (Definition 2.5) as the loss function which results in the empirical risk function

$$\hat{R}^{\text{QRNN}}_{\text{reg}}(\Phi^{\rho^{(H)}_{\boldsymbol{\alpha},\epsilon}}) = \frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\rho_{\alpha_k}(y_i, \tilde{\Phi}^{\rho^{(H)}_{\alpha_k,\epsilon}}(\mathbf{x}_i)) + \frac{\lambda}{dk_1\ldots k_H K}\left(\sum_{j=1}^{d}\sum_{j_1=1}^{k_1}(\omega^{[1]}_{j_1 j})^2 + \cdots + \sum_{j_{H-1}=1}^{k_{H-2}}\sum_{j_H=1}^{k_H}(\omega^{[H]}_{j_H j_{H-1}})^2 + \sum_{j_H=1}^{k_H}\sum_{j_{H+1}=1}^{K}(\omega^{[H+1]}_{j_{H+1} j_H})^2\right), \quad (5.9)$$

where we chose $\epsilon > 0$ before training the model. In practice, we implement the quantile neural network using the python package `tensorflow`. We define the neural network architecture and the empirical risk function in Equation (5.9) which will be back propagated using the automatic differentiation feature implemented in `tensorflow`. In order to better account for overfitting the training data, we implement the dropout technique, which was discussed in Section 3.4 as well as the early stopping technique studied in Section 3.5.

5.0.1 Modified quantile regression neural network

A common problem of quantile regression is the occurrence of a so-called **quantile crossing**, which describes the violation of the non-decreasing property of the quantile function. In other words, for $\alpha_1 > \alpha_2$ there exists a $\mathbf{x}$ such that

$$\hat{q}_{\alpha_1}(\mathbf{x}) < \hat{q}_{\alpha_2}(\mathbf{x}). \tag{5.10}$$

In terms of the quantile regression neural network setting, we have some $(y, \mathbf{x}) \in D$ such that the output vector of the neural network, $\tilde{\Phi}^{\rho_{\boldsymbol{\alpha}}}(\mathbf{x}) = (\tilde{\Phi}^{\rho_{\alpha_1}}(\mathbf{x}), \dots, \tilde{\Phi}^{\rho_{\alpha_K}}(\mathbf{x}))^T \in \mathbb{R}^K$ violates the non-decreasing property. In other words, there exists a $k \in \{1, \dots, K-1\}$ such that

$$\tilde{\Phi}^{\rho_{\alpha_k}}(\mathbf{x}) > \tilde{\Phi}^{\rho_{\alpha_{k+1}}}(\mathbf{x}), \tag{5.11}$$

for $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)^T \in (0,1)^K$ with $\alpha_k < \alpha_{k+1}$ for $k \in \{1, \dots, K-1\}$.

We aim to address this issue by presenting two approaches to eliminate the problem of quantile crossing in terms of QRNN. The first approach is to add a penalty term to the empirical risk function of the QRNN, $\hat{R}^{\text{QRNN}_{reg}}$. The idea is to penalize outputs of a quantile neural network $\Phi^{\rho_{\boldsymbol{\alpha}}} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\dots,k_H,K)}$ which violate $\tilde{\Phi}^{\rho_{\alpha_1}}(\mathbf{x}) < \dots < \tilde{\Phi}^{\rho_{\alpha_K}}(\mathbf{x})$, $\mathbf{x} \in D$. Therefore, the penalty term is of the form

$$\frac{\gamma}{K+1} \sum_{k=1}^{K-1} \mathbb{1}(\tilde{\Phi}^{\rho_{\alpha_k}}(\mathbf{x}) > \tilde{\Phi}^{\rho_{\alpha_{k+1}}}(\mathbf{x})), \tag{5.12}$$

where $\gamma > 0$ controls the contribution of the penalty term and $\mathbf{x} \in D$. Overall, we obtain the regularized empirical risk function

$$\begin{aligned}\hat{R}^{\text{QRNN}}_{\text{reg}}(\Phi^{\rho_{\boldsymbol{\alpha}}}) = \frac{1}{nK} \sum_{k=1}^{K} \sum_{i=1}^{n} \rho_{\alpha_k}(y_i, \tilde{\Phi}^{\rho_{\alpha_k}}(\mathbf{x}_i)) + \\ \frac{\lambda}{dk_1 \dots k_H K} \left(\sum_{j=1}^{d} \sum_{j_1=1}^{k_1} (\omega^{[1]}_{j_1 j})^2 + \dots + \sum_{j_H=1}^{k_H} \sum_{j_{H+1}=1}^{K} (\omega^{[H+1]}_{j_{H+1} j_H})^2 \right) + \frac{\gamma}{K-1} \sum_{k=1}^{K-1} \mathbb{1}(\tilde{\Phi}^{\rho_{\alpha_k}}(\mathbf{x}) > \tilde{\Phi}^{\rho_{\alpha_{k+1}}}(\mathbf{x})),\end{aligned} \tag{5.13}$$

where $\lambda > 0$ and $\gamma > 0$ for a quantile regression neural network $\Phi^{\rho_{\boldsymbol{\alpha}}} \in \mathcal{K}^{a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\dots,k_H,K)}$. Note that while this method reduces the likelihood of quantile crossing within the model, it does not guarantee its complete absence.

5.1 Cumulative quantile regression neural network (cQRNN)

In the following, we develop a new model which eliminates quantile crossing in the setting of quantile regression neural networks. Given an ordered vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)^T \in (0,1)^K$, such that $\alpha_k < \alpha_{k+1}$ for $k \in \{1, \dots, K-1\}$, define a quantile neural network $\Phi^{\boldsymbol{\theta}_1, \rho_{\alpha_1}} \in \mathcal{K}^{a,\rho_{\alpha_1}}_{(d,k_1,1)}$. For a sample $\mathbf{x} \in D$ the QRNN $\Phi^{\rho_{\alpha_1}}$ estimates $q_{\alpha_1}(\mathbf{x})$, i.e. $\hat{q}_{\alpha_1}(\mathbf{x}) = \tilde{\Phi}^{\rho_{\alpha_1}}(\mathbf{x})$. We continue by defining **quasi quantile neural networks** of the form $\Phi_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}} \in \mathcal{K}^{a,\rho_{\alpha_k}}_{(d,k_1,1)}$, where the output $\tilde{\Phi}^{\boldsymbol{\theta}_k,\rho_{\alpha_k}}_{\text{qc}}(\mathbf{x})$ results from the following scheme:

$$\begin{aligned}&\mathbf{h}^{[0]} := \mathbf{x}, \\ &\mathbf{h}^{[1]} := a\left(\mathbf{W}^{[1]}\mathbf{h}^{[0]} + \mathbf{b}^{[1]}\right), \\ &\tilde{\Phi}_q^{\boldsymbol{\theta}_k,\rho_{\alpha_k}}(\mathbf{x}) = \mathbf{h}^{[2]} := \exp(\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[1]}),\end{aligned}$$

for an activation function a and $k \in \{2, \ldots, K\}$. Note that we apply the exponential function to the last layer of the neural net, ensuring that the output $\tilde{\Phi}_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}(\mathbf{x})$ is strictly positive. An estimation for $q_{\alpha_k}(\mathbf{x})$ is then derived by cumulatively adding quasi quantile neural networks to the estimates as the quantile level α_k increases, resulting in

$$\hat{q}_{\alpha_k}(\mathbf{x}) := \begin{cases} \tilde{\Phi}^{\boldsymbol{\theta}_1, \rho_{\alpha_1}}(\mathbf{x}), & k = 1, \\ \hat{q}_{\alpha_{k-1}}(\mathbf{x}) + \tilde{\Phi}_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}(\mathbf{x}), & k > 1, \end{cases} \tag{5.14}$$

for $\mathbf{x} \in D$. By construction, the estimates satisfy $\hat{q}_{\alpha_1}(\mathbf{x}) < \cdots < \hat{q}_{\alpha_K}(\mathbf{x})$ for all $\mathbf{x} \in D$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^T$ with $\alpha_k < \alpha_{k+1}$ for $k \in \{1, \ldots, K-1\}$. We call the resulting model **cumulative quantile regression neural network** or **cQRNN**.

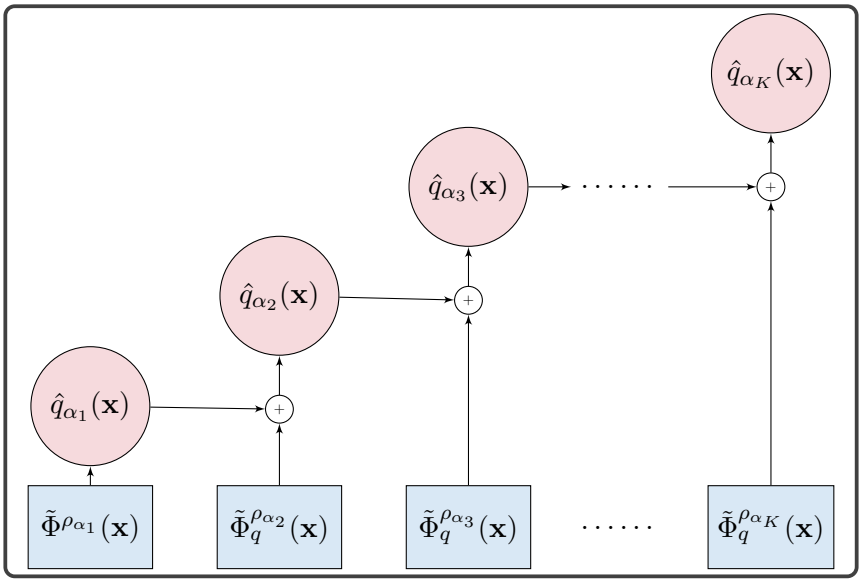


Figure 5.3 Neural network architecture of the cumulative quantile regression neural network (cQRNN).

The cQRNN model possesses the parameter vectors $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$ which get estimated sequentially, using mini-batch stochastic gradient descent. Following the structure of the cQRNN, the empirical risk functions of the submodels are defined as

$$\hat{R}^{\text{QRNN}}(\Phi^{\boldsymbol{\theta}_1, \rho_{\alpha_1}}) = \frac{1}{n} \sum_{i=1}^{n} \rho_{\alpha_1}(y_i, \tilde{\Phi}^{\boldsymbol{\theta}_1, \rho_{\alpha_1}}(\mathbf{x_i})), \qquad \text{for } k = 1, \tag{5.15}$$

$$\hat{R}_q^{\text{QRNN}}(\Phi_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}) = \frac{1}{n} \sum_{i=1}^{n} \rho_{\alpha_k}(y_i, \hat{q}_{\alpha_{k-1}}(x_i) + \tilde{\Phi}_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}(\mathbf{x_i})), \qquad \text{for } k > 1, \tag{5.16}$$

where $\mathbf{x}_i \in D_{train}$. The empirical risk function of the full cQRNN model is the sum of the empirical risk function of the submodels, namely

$$\hat{R}^{\text{cQRNN}} := \hat{R}^{\text{QRNN}}(\Phi^{\boldsymbol{\theta}_1, \rho_{\alpha_1}}) + \sum_{k=2}^{K} \hat{R}_q^{\text{QRNN}}(\Phi_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}). \tag{5.17}$$

As discussed in Section 3.1, we use mini-batch stochastic gradient descent to numerical solve the optimization problem

$$\min_{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K} \hat{R}^{\text{cQRNN}} = \min_{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K} \left(\hat{R}^{\text{QRNN}}(\Phi^{\boldsymbol{\theta}_1, \rho_{\alpha_1}}) + \sum_{k=2}^{K} \hat{R}_q^{\text{QRNN}}(\Phi_q^{\boldsymbol{\theta}_k, \rho_{\alpha_k}}) \right). \tag{5.18}$$

Note that we can regularize the cQRNN model by applying the dropout method as well as by adding a L^2 regularization term to the empirical risk functions of the sub models. In addition, the submodels can be extended to hold more than one hidden layer. We will use the `tensorflow` library in python to implement the cQRNN model.

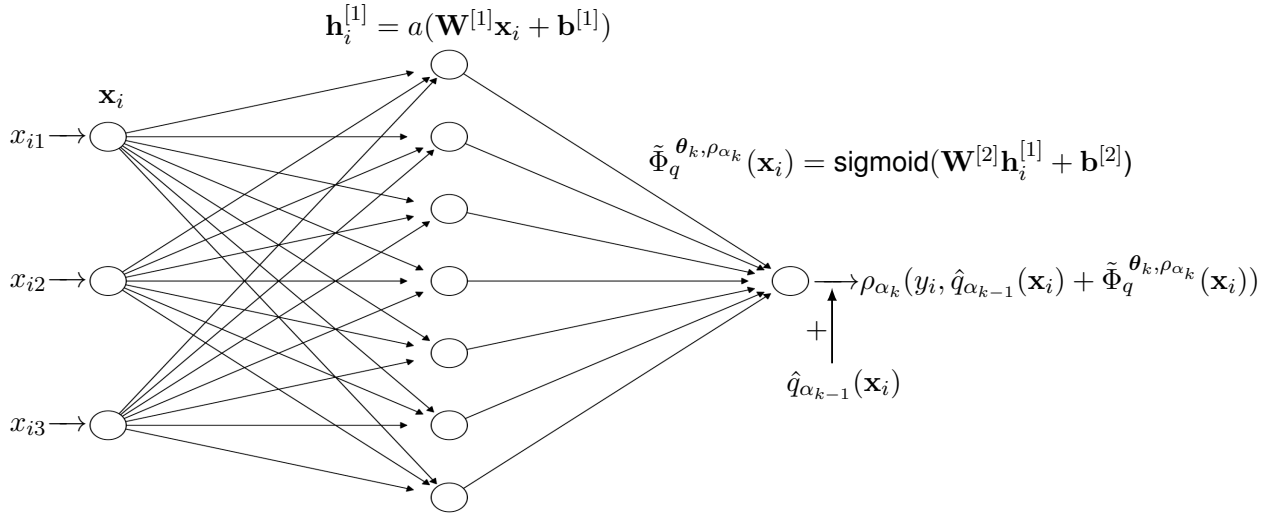


Figure 5.4 Visualization of the forward pass of a quasi quantile regression neural network $\Phi_q^{\boldsymbol{\theta}_k,\rho_{\alpha_k}} \in \mathcal{K}_{(3,7,1)}^{a,\rho_{\alpha_k}}$, $k \in \{2,\dots,K\}$ with an arbitrary activation function a. We have $\mathbf{x}_i \in \mathbb{R}^3$, $\mathbf{W}^{[1]} \in \mathbb{R}^{7\times 3}$, $\mathbf{b}^{[1]} \in \mathbb{R}^7$, $\mathbf{W}^{[2]} \in \mathbb{R}^{1\times 7}$, $\mathbf{b}^{[2]} \in \mathbb{R}^1$ and $y_i \in \mathbb{R}$ for $i = 1,\dots,n$ and $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_K)^T \in (0,1)^K$ with $\alpha_k < \alpha_{k+1}$.

5.2 Reduced quantile regression neural network (redQRNN)

In practical regression situations, we often encounter many covariates. The process of training a deep neural network with numerous input variables can result in lengthy computation times and imprecise point estimates. As a result, we aim to reduce the number of input variables. In this section, we employ the forward selection algorithm of the D-vine quantile regression model to pre-select covariates by fitting a D-vine copula to the training data. The selected covariates are then utilized as the input for training and testing a quantile neural network. Following this procedure, the resulting model is called **reduced quantile regression neural network** (or short **redQRNN**)

We start by describing the algorithm. Let $((y_i, \mathbf{x}_i))_{i=1,\dots,n}$ be an i.i.d. sample of the random vector $(Y, X_1, \dots, X_d)$. First, the marginals F_Y and F_{X_j} for $j = 1,\dots,d$ get estimated using the kernel smoothing estimator (4.14). The resulting estimates $\hat{F}_Y$ and $\hat{F}_{X_j}$ are then used to derive the pseudo copula data $((\hat{v}_i, \hat{\mathbf{u}}_i))_{i=1,\dots,n}$. Next, we fit a D-vine Copula $V - U_{l_1} - \cdots - U_{l_k}$, where $k \in \{1,\dots,d\}$, to the pseudo copula data using the forward selection algorithm described in Section 4.2.1. Note that $k = 1$ results in the D-vine copula $V - U_{l_1}$.

The conditional log-likelihood maximizing ordering $\mathbf{l} = (l_1,\dots,l_k)$ is used to select the k covariates of the samples $(x_{il_1},\dots,x_{il_k}) =: \mathbf{x}_i^{\text{red}}$, for $i = 1,\dots,n$. Finally, a quantile regression neural network model $\Phi_{(k,k_1,\dots,k_H,1)}^{\boldsymbol{\theta},a,\rho_\alpha,\text{red}} \in \mathcal{K}_{(k,k_1,\dots,k_H,1)}^{a,\rho_\alpha}$ is fitted using the samples $\left((y_i, \mathbf{x}_i^{\text{red}})\right)_{i=1,\dots,n}$. This is done using the algorithm studied in Section 3.1. Similar as in Section 5 we can extend the model and estimate K different quantiles at levels $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_K)$ resulting in the network $\Phi_{(k,k_1,\dots,k_H,1)}^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}},\text{red}} \in \mathcal{K}_{(k,k_1,\dots,k_H,K)}^{a,\rho_\alpha}$. In order to prevent overfitting, we again use the L^2 regularization, the dropout method and the early stopping technique. Note that the algorithm can easily be modified and used for a **cQRNN** approach. This is done by using the extracted ordering from the forward algorithm of the D-vine quantile regression in order to reduce the input variables of the **quasi quantile neural networks** (see Section 5.1). The re-

Algorithm 1 Pseudo Code for the **redQRNN**

1: **Input**: Target vector $\mathbf{y} = (y_i)_{i=1,\dots,n}$ and covariate matrix $\mathbf{X} = (x_{ij})_{i=1,\dots,n,\ j=1,\dots,d}$
2: Use the `vinereg` function to fit a D-vine to the data $(\mathbf{y}, \mathbf{X})$, resulting in the D-vine with order $V - U_1 - \cdots - U_{l_k}$, where $\mathbf{l} = (l_1, \dots, l_k)$ is the conditional log-likelihood maximizing ordering of the pseudo copula data $(\hat{\boldsymbol{v}}, \hat{\boldsymbol{U}})$.
3: *order* $\leftarrow (l_1, \dots, l_k)$ extract the ordering of the fitted D-vine $V - U_1 - \cdots - U_{l_k}$
4: $\mathbf{X}^{\text{red}} \leftarrow \mathbf{X}[, \textit{order}]$
5: Use $\mathbf{X}^{\text{red}}$ and $\mathbf{y}$ to fit a quantile regression neural network $\Phi^{\boldsymbol{\theta},a,\rho_\alpha,\text{red}}_{(k,k_1,\dots,k_H,1)} \in \mathcal{K}^{a,\rho_\alpha}_{(k,k_1,\dots,k_H,1)}$.
6: **Output**: $\Phi^{\boldsymbol{\theta},a,\rho_\alpha,\text{red}}_{(k,k_1,\dots,k_H,1)}$

sulting model will be called **red-cQRNN**. Additionally, we will utilize the modified forward algorithm of the sparse D-vine quantile regression introduced by Sahin and Czado (2022) to achieve faster computation times. This algorithm is implemented in the R package `sparsevinereg`, which is accessible at github.com/oezgesahin/sparsevinereg. The models utilizing the sparse D-vine quantile regression algorithm for dimensionality reduction will be referred to as **sredQRNN** and **sred-cQRNN**.

5.3 Uncertainties for quantile regression neural networks using D-vine copulas

As discussed in previous chapters, neural networks yield only point estimates when making predictions. To use neural networks as predictive models, it is crucial to consider uncertainties, leading to the generation of confidence estimates. In this chapter, we present a modified version of the **vine copula neural network (VCNN)** introduced by Tagasovska, Ozdemir, and Brando (2023), which is designed to handle the predictive uncertainties of neural networks. These uncertainties stem from two probabilistic sources:

- *epidimistic* - The training data is incomplete, meaning that some input areas are not represented in the training data, or the model does not have the ability to accurately represent the actual function. This can be captured by confidence intervals.
- *aleatoric* - It arises from the intrinsic noise in the data and is therefore an irreducible factor. This can be captured by prediction intervals.

The vine copula neural network is designed to handle both types of uncertainties. It is important to note that the VCNN can be implemented retrospectively on any previously trained neural network, eliminating the need for a complete retraining of the main model.

5.3.1 Simulation-based confidence intervals for QRNN

To predict confidence intervals based on simulations, we utilize the generative nature of vine copulas (Dissmann et al. 2013) to "bootstrap" the final hidden layer of the fully trained neural network. The main idea was introduced by Tagasovska, Ozdemir, and Brando (2023). We modify the procedure by fitting a sparse D-vine regression (Sahin and Czado 2022) to the values of the last hidden layer instead of a R-vine. This will result in a dimensionality reduction of the last layer as well as a faster computation time. The sparse D-vine regression was developed by Sahin and Czado (2022). The simulations and the estimation of the simultaneous confidence intervals is described in the quantile neural network setting. Let $\Phi \in \mathcal{K}^{a,\rho_\alpha}_{(d,k_1,\dots,k_H,1)}$ be the trained quantile regression neural network predicting q_α, $\alpha \in (0,1)$. We use the training set $D_{train} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \dots, n\}$ and the test set $D_{test} = \{(y_i^*, \mathbf{x}_i^*) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \dots, n_{test}\}$.

(i) First, we obtain the values from the neurons of the last hidden layer before applying the activation function. This is done by using the calculating scheme described in the definition of neural networks

for every sample $\mathbf{x}_i \in D_{train}$, $i = 1, \ldots, n$. We calculate the forward pass of the trained neural network Φ for all samples: $(y_i, \mathbf{x}_i)$

$$\begin{aligned}
&\mathbf{h}_i^{[0]} = \mathbf{x}_i, \\
&\mathbf{h}_i^{[h]} = a\left(\mathbf{W}^{[h]}\mathbf{h}_i^{[h-1]} + \mathbf{b}^{[h]}\right), \quad \textit{for } h = 1, \ldots, H-1, \\
&\tilde{\mathbf{h}}_i^{[H]} = \mathbf{W}^{[H]}\mathbf{h}_i^{[H-1]} + \mathbf{b}^{[H]}, \\
&\mathbf{h}_i^{[H]} = a\left(\tilde{\mathbf{h}}_i^{[H]}\right), \\
&\tilde{\Phi}(\mathbf{x}_i) = \mathbf{h}_i^{[H+1]} = \mathbf{W}^{[H+1]}\mathbf{h}^{[H]} + \mathbf{b}^{[H]},
\end{aligned}$$

and extract the vectors $\tilde{\mathbf{h}}_i^{[H]} \in \mathbb{R}^{k_H}$ for $i = 1, \ldots, n$, where k_H is the number of neurons of the last hidden layer of the trained neural network Φ. We further work with the vector

$$\left(y_i, \tilde{\mathbf{h}}_i^{[H]}\right) \in \mathbb{R}^{k_H+1}, \tag{5.19}$$

where $(y_i, \mathbf{x}_i) \in D_{train}$ for $i = 1, \ldots, n$.

(ii) In order to generate samples from the last hidden layer, we perform a sparse D-vine regression on the data $\left\{\left(y_i, \tilde{\mathbf{h}}_i^{[H]}\right)\right\}_{i=1}^{n}$, resulting in a D-vine copula C of the structure $V - U_{l_1} - \cdots - U_{l_j}$, where $\mathbf{l} = (l_1, \ldots, l_j)$, $1 \leq j \leq k_H$ is the cll maximizing order discussed in Section 1.8. For implementing the sparse D-vine regression algorithm, we use the R package `sparsevinereg`, available at github.com/oezgesahin/sparsevinereg, and import it to the python environment with the `rpy2` package.

(iii) For repetition $s \in \{1, \ldots, S\}$, we "bootstrap" the neural network by copying the neural network Φ and retrain the last hidden layer of the copies $\Phi^1, \ldots, \Phi^S$ with generated D-vine copula data. Depending on the fitted D-vine copula C, some neurons of the last hidden layer are set to zero, which results in a dimensionality reduction of the last hidden layer of the copies Φ^s, $s = 1, \ldots, S$. For $s = 1, \ldots, S$:

a) Sample K random observations from the D-vine copula $C^{D_{train},[H]}$, which we use to retrain the last hidden layer of the copies Φ^s:

$$D_{train}^{s,[H]} := \left\{\left(y_k^s, \tilde{\mathbf{h}}_k^{[H],s}\right)\right\}_{k=1}^{K}.$$

Overall, this gives us $K \cdot S$ realizations from $C^{D_{train},[H]}$, where $|D_{train}^{s,[H]}| = K$ for each repetition $s \in \{1, \ldots, S\}$.

b) Retrain the last hidden layer and the output layer of the copied neural network Φ^s utilizing the generated training set $D_{train}^{s,[H]}$, i.e. calculate the forward pass of the last hidden layer

$$\begin{aligned}
&\mathbf{h}_k^{[H],s} = a\left(\tilde{\mathbf{h}}_k^{[H],s}\right), \\
&\mathbf{h}_k^{[H+1],s} = \mathbf{W}^{[H+1],s}\,\mathbf{h}_k^{[H],s} + \mathbf{b}^{[H],s},
\end{aligned} \tag{5.20}$$

with $k = 1, \ldots, K$ and use back propagation and an optimization algorithm to retrain the last hidden layer of the neural network copy Φ^s with the respective generated training set $D_{train}^{s,[H]}$, $s = 1, \ldots, S$. Observe that the retraining of the last hidden layer of the neural network Φ^s is equivalent to training a neural network with input dimension l_j (dimension of the fitted D-vine copula $C^{D_{train},[H]}$) and the calculation Scheme 5.20 using training data $D_{train}^{s,[H]}$.

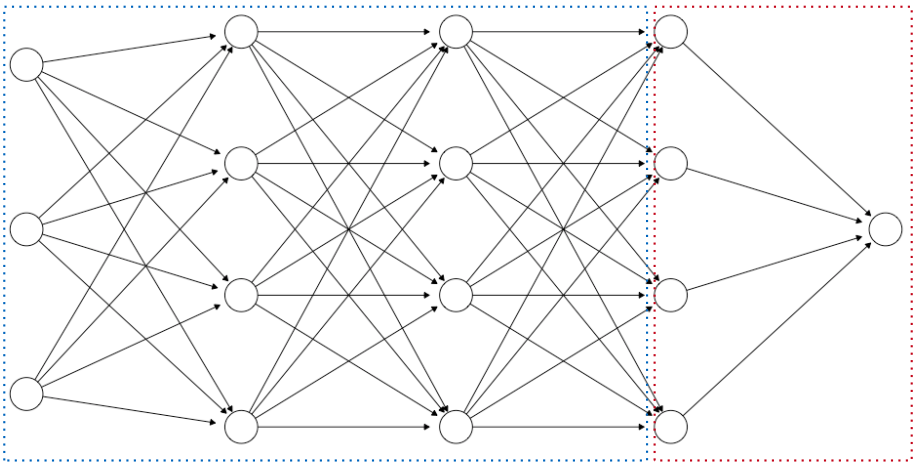


Figure 5.5 Visualization of a copy Φ^s, $s \in \{1, \dots, S\}$. The weights in the blue frame are not changed while the parameters in the last hidden layer (red frame) get retrained with the generated training set $D_{train}^{s,[H]}$. Note that the neurons of the last hidden layer might vary due to the dimensionality reduction of the fitted D-vine copula $C^{D_{train},[H]}$.

Overall, we get the neural networks $\Phi^s \in \mathcal{K}^{a,L}_{(d,k_1,\dots,l_j,1)}$ for $s = 1, \dots, S$. Note that the neural network copies Φ^s, $s = 1, \dots, S$ have the same input and output dimension as the neural network Φ. However, the number of neurons in their last hidden layer H may vary due to the dimensionality reduction of the fitted D-vine copula $C^{D_{train},[H]}$. The values of the neurons in the last hidden layer of the Φ^s is always different then Φ. Note that the values of the neurons in the last layer also vary between the copies Φ^s due to the sampling of the training data $D_{train}^{s,[H]}$, where $s = 1, \dots, S$.

(iv) Next, we compute the vector

$$\mathbf{H}_i := \left(\tilde{\Phi}(\mathbf{x}_i^*) - \tilde{\Phi}^1(\mathbf{x}_i^*), \dots, \tilde{\Phi}(\mathbf{x}_i^*) - \tilde{\Phi}^S(\mathbf{x}_i^*)\right),$$

where $\tilde{\Phi}(\mathbf{x}_i^*)$ is the forward pass of the main model Φ and $\tilde{\Phi}^s(\mathbf{x}_i^*)$ is the forward pass of the copied and retrained neural networks Φ^s, $s = 1, \dots, S$, for all $(y_i^*, \mathbf{x}_i^*) \in D_{test}$. We continue by calculating the sample standard deviation of the vectors $\mathbf{H}_i$ for $i = 1, \dots, n_{test}$ resulting in the vector

$$\boldsymbol{V} := (\hat{\sigma}(H_1), \dots, \hat{\sigma}(H_{n_{test}})),$$

where

$$\hat{\sigma} : \mathbb{R}^n \to (0, \infty), \quad \hat{\sigma}(\mathbf{x}) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \frac{1}{n}\sum_{i=1}^{n} x_i)^2} \tag{5.21}$$

is the sample standard deviation. Moreover, we calculate the **empirical** $1-\alpha$ **quantile** of the vector $\boldsymbol{V}$, which we denote by $\hat{q}_{1-\alpha}^{emp}$.

(v) Finally, the upper and lower bound of the $100(1-\alpha)\%$ simultaneous **prediction interval** (PI) can be estimated as

$$\widehat{\mathsf{PI}}_{1-\alpha} = \left[\begin{pmatrix} \tilde{\Phi}(\mathbf{x}_1^*) \\ \vdots \\ \tilde{\Phi}(\mathbf{x}_{n_{test}}^*) \end{pmatrix} - \hat{q}_{1-\alpha}^{emp} \begin{pmatrix} \hat{\sigma}(H_1) \\ \vdots \\ \hat{\sigma}(H_{n_{test}}) \end{pmatrix}, \begin{pmatrix} \tilde{\Phi}(\mathbf{x}_1^*) \\ \vdots \\ \tilde{\Phi}(\mathbf{x}_{n_{test}}^*) \end{pmatrix} + \hat{q}_{1-\alpha}^{emp} \begin{pmatrix} \hat{\sigma}(H_1) \\ \vdots \\ \hat{\sigma}(H_{n_{test}}) \end{pmatrix}\right], \tag{5.22}$$

for $(y_i^*, \mathbf{x}_i^*) \in D_{test}$, $1 \leq i \leq n_{test}$ and $\alpha \in (0, 1)$. Note that the prediction interval is **only valid** for i.i.d. input data $(y_i^*, \mathbf{x}_i^*)$, $1 \leq i \leq n_{test}$. By estimating the simultaneous PI we will account for the epidimistic uncertainty of the QRNN model Φ (Tagasovska, Ozdemir, and Brando 2023).

By using this procedure, we will account for the epidimistic uncertainty of our model.

6 Simulation study

6.1 Setup of underlying true models

In this section, the accuracy of approximating quantiles via D-vine quantile regression and quantile regression neural networks will be compared. We will proceed as follows:

(i) First, we generate n training- and $n_{test} := n/2$ test samples from a random vector $(Y, X_1, \ldots, X_d)^T$ and set the quantile level $\boldsymbol{\alpha} = (0.05, 0.5, 0.95)^T$, which we want to predict.

(ii) Then we fit the models

- **D-vine quantile regression** (See Section 4.2)
- **QRNN**: Quantile regression neural network $\Phi_{(d,200,200,3)}^{(\text{ReLU, sigmoid}),\rho_{\boldsymbol{\alpha}}}$ with input dimension d, output dimension $|\boldsymbol{\alpha}| = 3$ and two hidden layers. Each of these layers contains 200 neurons. The activation function applied to the first layer is the ReLU function, while for the second layer, we utilize the sigmoid function. (See Section 5)
- **cQRNN**: Cumulative quantile regression model with three **quasi quantile regression neural network** where each network consists of one hidden layer with 200 neurons and the sigmoid activation function. (See Section 5.1)
- **redQRNN**: Quantile regression neural network $\Phi_{(k,200,200,3)}^{(\text{ReLU, sigmoid}),\rho_{\boldsymbol{\alpha}},\text{red}}$ with reduced input dimension $1 \leq k \leq d$, output dimension $|\boldsymbol{\alpha}| = 3$ and two hidden layers. Each of the hidden layers contains 200 neurons. The ReLU function is applied to the first hidden layer while the sigmoid function is applied to the second hidden layer. (See Section 5.2) We only fit this model for the high dimensional simulation scenarios D6 and G20.
- **sparsevinereg**: Sparse D-vine quantile regression introduced and studied in Sahin and Czado (2022). This model is fitted for the high dimensional simulation scenarios D6 and G20.
- **sredQRNN**: Quantile regression neural network $\Phi_{(k,200,200,3)}^{(\text{ReLU, sigmoid}),\rho_{\boldsymbol{\alpha}},\text{red}}$ with reduced input dimension $1 \leq k \leq d$, output dimension $|\boldsymbol{\alpha}| = 3$ and two hidden layers. The reduced input dimension is computed with the forward selection algorithm of the **sparse D-vine quantile regression**. The algorithm is described in Sahin and Czado (2022). Each of the hidden layers of the model contains 200 neurons. The ReLU function is applied to the first hidden layer while the sigmoid function is applied to the second hidden layer. We only fit this model for the high dimensional simulation scenarios D6 and G20.
- **red-cQRNN**: Cumulative quantile regression neural network with reduced input dimension $1 \leq k \leq d$, where each quasi quantile regression neural network consists of one hidden layer with 200 neurons and the sigmoid activation function. (See Section 5.1 and Section 5.2) We only fit this model for the high dimensional simulation scenarios D6 and G20.

The training sets in the form of $D_{train} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$ are used to train the listed models. In order to prevent overfitting, we utilize the early stopping technique, which is detailed in Section 3.5. Additionally, we implement the dropout method with a dropout rate of $p = 0.1$ and L^2 regularization with a learning rate of $\lambda = 0.001$ (Section 3.4).

We consider the following distributions for $(Y, X_1, \ldots, X_d)^T$:

- **D3**: $(Y, X_1, X_2)^T$ follows a three-dimensional D-vine copula with order $V - U_1 - U_2$ and marginal distributions given in the set $\mathcal{M}$. The bivariate copulas of the D-vine on the u-scale $C_{VU_1}, C_{U_1U_2}$ and $C_{VU_2;U_1}$ are from the Clayton family with copula parameter δ_{V1}, δ_{12} and $\delta_{V2;1}$, respectively. The copula parameters as well as the margin set $\mathcal{M}$ are given in Table 6.1.

Kendall's τ	Copula parameter	Marginal
$\tau_{V1} = 0.70$ $\tau_{12} = 0.57$ $\tau_{V2;1} = 0.37$	$\delta_{V1} = 4.68$ $\delta_{12} = 2.68$ $\delta_{V2;1} = 1.2$	$\begin{array}{\|c\|c\|c\|c\|} \hline & Y & X_1 & X_2 \\ \hline \mathcal{M} & \mathcal{N}(0,1) & t_4(0,1) & \mathcal{N}(1,4) \\ \hline \end{array}$

Table 6.1 Clayton copula parameters and marginal distribution set of simulation scenario **D3**.

- **D6**: $(V, U_1, U_2, U_3, U_4, U_5)$ follows a D-vine copula with order $V - U_1 - \cdots - U_5$ and the parameters given in Table 6.2.

Tree	Edge	Conditioned	Conditioning	Family	Kendall's τ
1	1	VU_1 ;		Clayton	0.70
1	2	U_1U_2 ;		Gumbel	0.82
1	3	U_2U_3 ;		Gauss	0.54
1	4	U_3U_4 ;		Joe	0.21
1	5	U_4U_5 ;		Indep.	0.00
2	1	VU_2 ;	U_1	Gumbel(90)	0.73
2	2	U_1U_3 ;	U_2	Clayton	0.59
2	3	U_2U_4 ;	U_3	Joe	0.31
2	4	U_1U_3 ;	U_2	Gauss	0.13
3	1	VU_3 ;	U_1U_2	Frank	0.6
3	2	U_1U_4 ;	U_2U_3	Clayton	0.45
3	3	U_2U_5 ;	U_3U_4	Gumbel	0.15
4	1	VU_4 ;	$U_1U_2U_3$	Gauss	0.50
4	2	U_1U_5 ;	$U_2U_3U_4$	Frank	0.10
5	1	VU_5 ;	$U_1U_2U_3U_4U_5$	Indep.	0.00

Table 6.2 Copula families and their parameters of the six dimensional D-vine copula of simulation scenario **D6**.

- **NH4**: Nonlinear and heteroscedastic (variance is of residuals of a regression model and is not constant)(Tepegjozova et al. 2022): $Y = U_1U_2 \exp(1.8U_3U_4) + 0.5(U_1 + U_2 + U_3 + U_4)\epsilon$, where U_i, $j = 1, \ldots, 4$ are obtained from the PIT from $N_4(0, \Sigma)$, $\Sigma_{i,j} = 0.5^{\mathbb{1}(\{i \neq j\})}$, and $\epsilon \sim N(0, 0.5)$.

- **LH4**: Linear and heteroscedastic (Tepegjozova et al. 2022): $Y = 5(X_1 + X_2 + X_3 + X_4) + 0.5(U_1 + U_2 + U_3 + U_4)\epsilon$, where $(X_1, X_2, X_3, X_4)^T \sim N_4(0, \Sigma)$, $\Sigma_{i,j} = 0.5^{\mathbb{1}(\{i \neq j\})}$, and U_i are the PIT random variables from X_i, $i = 1, \ldots, 4$, $\epsilon \sim N(0, 0.5)$.

- **G20**: high dimensional sparse D-vine copula: $(V, U_1, \ldots, U_{19})^T$ follows a 20-dimensional D-vine copula *truncated* at the third level. The pair copula family is drawn uniformly from the set

$$\mathcal{F} = \{\text{Gaussian, Clayton, Gumbel, Joe, Frank}\}.$$

The family parameter of the bivariate copulas is obtained by using the one-to-one relationship between Kendall's τ and the family parameter, where Kendall's τ is uniformly drawn from different sets (see Table 6.3).The last four pair copulas of each tree are set to be the independent copula.

Tree	Edge	Conditioned	Conditioning	Family	Kendall's τ
1	1	VU_1 ;		$F_1^{(1)} \sim \mathcal{F}$	$\tau_1^{(1)} \sim U(0.6, 0.9)$
1	2	U_1U_2 ;		$F_2^{(1)} \sim \mathcal{F}$	$\tau_2^{(1)} \sim U(0.6, 0.9)$
⋮	⋮	⋮		⋮	⋮
1	15	$U_{14}U_{15}$;		$F_{15}^{(1)} \sim \mathcal{F}$	$\tau_{15}^{(1)} \sim U(0.6, 0.9)$
1	16	$U_{15}U_{16}$;		Indep.	0
⋮	⋮	⋮		⋮	⋮
1	19	$U_{18}U_{19}$;		Indep.	0
2	1	VU_2 ;	U_1	$F_1^{(2)} \sim \mathcal{F}$	$\tau_1^{(2)} \sim U(0.4, 0.6)$
2	2	U_1U_3 ;	U_2	$F_2^{(2)} \sim \mathcal{F}$	$\tau_2^{(2)} \sim U(0.4, 0.6)$
⋮	⋮	⋮	⋮	⋮	⋮
1	14	$U_{13}U_{15}$;	U_{14}	$F_{14}^{(2)} \sim \mathcal{F}$	$\tau_{14}^{(2)} \sim U(0.4, 0.6)$
1	15	$U_{14}U_{16}$;	U_{15}	Indep.	0
⋮	⋮	⋮	⋮	⋮	⋮
2	18	$U_{17}U_{19}$;	U_{18}	Indep.	0
3	1	VU_3 ;	U_1U_2	$F_1^{(3)} \sim \mathcal{F}$	$\tau_1^{(3)} \sim U(0.2, 0.5)$
3	2	U_1U_4 ;	U_2U_3	$F_2^{(3)} \sim \mathcal{F}$	$\tau_2^{(3)} \sim U(0.2, 0.5)$
⋮	⋮	⋮	⋮	⋮	⋮
3	13	$U_{12}U_{15}$;	$U_{13}U_{14}$	$F_{13}^{(3)} \sim \mathcal{F}$	$\tau_{13}^{(3)} \sim U(0.2, 0.5)$
3	14	$U_{13}U_{16}$;	$U_{14}U_{15}$	Indep.	0
⋮	⋮	⋮	⋮	⋮	⋮
3	17	$U_{16}U_{19}$;	$U_{17}U_{18}$	Indep.	0

Table 6.3 Copula families and their parameters of the 20-dimensional D-vine copula of simulation scenario **G20**.

6.2 Evaluation measures

In the next step, the trained models $m \in \{$D-vinereg, QRNN,cQRNN, redQRNN$\}$ are applied to the samples of the test set $(y_i, \mathbf{x}_i) \in D_{test} = \{(y_i, \mathbf{x}_i) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{test}\}$ in order to predict the true quantiles $q_\alpha(\mathbf{x}_i)$ for $(y_i, \mathbf{x}_i) \in D_{test}$.

Following Tepegjozova et al. (2022), the *out of sample* performance of the model m is measured by calculating the sample mean pinball loss function, namely

$$\widehat{\mathsf{MPL}}_{m,\alpha} := \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \rho_\alpha(y_i, \hat{q}_{\alpha,m}(\mathbf{x}_i)),$$

with $\rho_\alpha(x) := \alpha x \mathbb{1}_{x \geq 0} + (\alpha - 1) x \mathbb{1}_{x<0}$ and $(y_i, \mathbf{x}_i) \in D_{test}$. This process is replicated $r = 1, \ldots, R = 100$ times, which leads to the **out of sample mean integrated pinball loss**

$$\widehat{\mathsf{MIPL}}_{m,\alpha} := \frac{1}{R} \sum_{r=1}^{R} \left[\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \rho_\alpha(y_{r,i} - \hat{q}^{(r)}_{\alpha,m}(\mathbf{x}_{r,i})) \right], \tag{6.1}$$

where $(y_{r,i}, \mathbf{x}_{r,i}) \in D^r_{test}$ and $D^r_{test} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{test}\}$ is the r-th simulated test set and $\hat{q}^{(r)}_{\alpha,m}$ is the α level conditional quantile using the model m which is fitted by using the r-th simulated training set $D^r_{train} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$.

In addition, we consider the **interval score for the $(1 - \alpha) \times 100\%$ prediction interval**, which is defined as

$$\begin{aligned} \widehat{\mathsf{IS}}_{m,\alpha} = \frac{1}{R} \sum_{r=1}^{R} \Bigg[\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \Bigg(& (\hat{q}^{(r)}_{\alpha/2,m}(\mathbf{x}_{r,i}) - \hat{q}^{(r)}_{1-\alpha/2,m}(\mathbf{x}_{r,i})) + \\ & \frac{2}{\alpha} \left((\hat{q}^{(r)}_{1-\alpha/2,m}(\mathbf{x}_{r,i}) - y_{r,i}) \mathbb{1}\{y_{r,i} \leq \hat{q}^{(r)}_{1-\alpha/2,m}(\mathbf{x}_{r,i})\} \right) + \\ & \frac{2}{\alpha} \left((y_{r,i} - \hat{q}^{(r)}_{\alpha/2,m}(\mathbf{x}_{r,i})) \mathbb{1}\{y_{r,i} > \hat{q}^{(r)}_{\alpha/2,m}(\mathbf{x}_{r,i})\} \right) \Bigg) \Bigg], \end{aligned} \tag{6.2}$$

where $(y_{r,i}, \mathbf{x}_{r,i}) \in D^r_{test}$ and $D^r_{test} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{test}\}$ is the r-th simulated test set and $\hat{q}^{(r)}_{\alpha,m}$ is the α level conditional quantile using the model m which is fitted by using the r-th simulated training set $D^r_{train} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$. A smaller interval score implies a better model fit (Grønneberg and Raftery 2007). The interval score will be used for validating the prediction intervals introduced in Section 5.3. Note that we take prediction intervals for quantiles into account, which means we must compute the true quantiles in order to calculate the interval score. We estimate the interval score for the high-dimensional scenarios **D6** and **G20**.

In addition, we will take into account the **average computation time**, which gives the duration of computer time required to execute the algorithm for fitting the model to the data. The variable will be called $\hat{\mathsf{T}}$.

For the higher dimensional scenarios **D6** and **G20**, we count the **average relative quantile crossings** (in %) described in Section 5.0.1 per model for the quantiles of level $\alpha \in \{0.01, 0.05\}$:

$$\widehat{\mathsf{ARQC}}_m := \left(\frac{1}{R \cdot n_{test}} \sum_{r=1}^{R} \sum_{i=1}^{n_{test}} \mathbb{1}\{\hat{q}^{(r)}_{0.05,m}(\mathbf{x}_{r,i}) < \hat{q}^{(r)}_{0.01,m}(\mathbf{x}_{r,i})\} \right) \cdot 100\%, \tag{6.3}$$

where $(y_{r,i}, \mathbf{x}_{r,i}) \in D^r_{test}$ and $D^r_{test} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n_{test}\}$ is the r-th simulated test set and $\hat{q}^{(r)}_{\alpha,m}$ is the α level conditional quantile using the model m which is fitted by using the r-th simulated training set $D^r_{train} = \{(y_{i,r}, \mathbf{x}_{i,r}) \in \mathbb{R} \times \mathbb{R}^d | i = 1, \ldots, n\}$.

6.3 Results

In our simulation study, we consider the quantiles of level $\alpha \in \{0.05, 0.5, 0.95\}$. We also compare the performance of the models with large vs. small sized training data by choosing $n_{train} \in \{400, 1000\}$. In order

to prevent over fitting, we implement the L^2 regularization with $\lambda = 0.01$ (Section 3.4) and the dropout method with drop rate 0.1 (Section 3.4) as well as the early stopping method (Section 3.5) for the different quantile regression neural networks QRNN, cQRNN and redQRNN. The implementation is done in python (Van Rossum and Drake 2009), where we use the `tensorflow` package to build the various quantile regression neural networks. The D-vine quantile regression model is implemented using the `vinereg` package from R (R Core Team 2021). The methods of this package can be called in python (Van Rossum and Drake 2009) through the use of the `rpy2` python package.

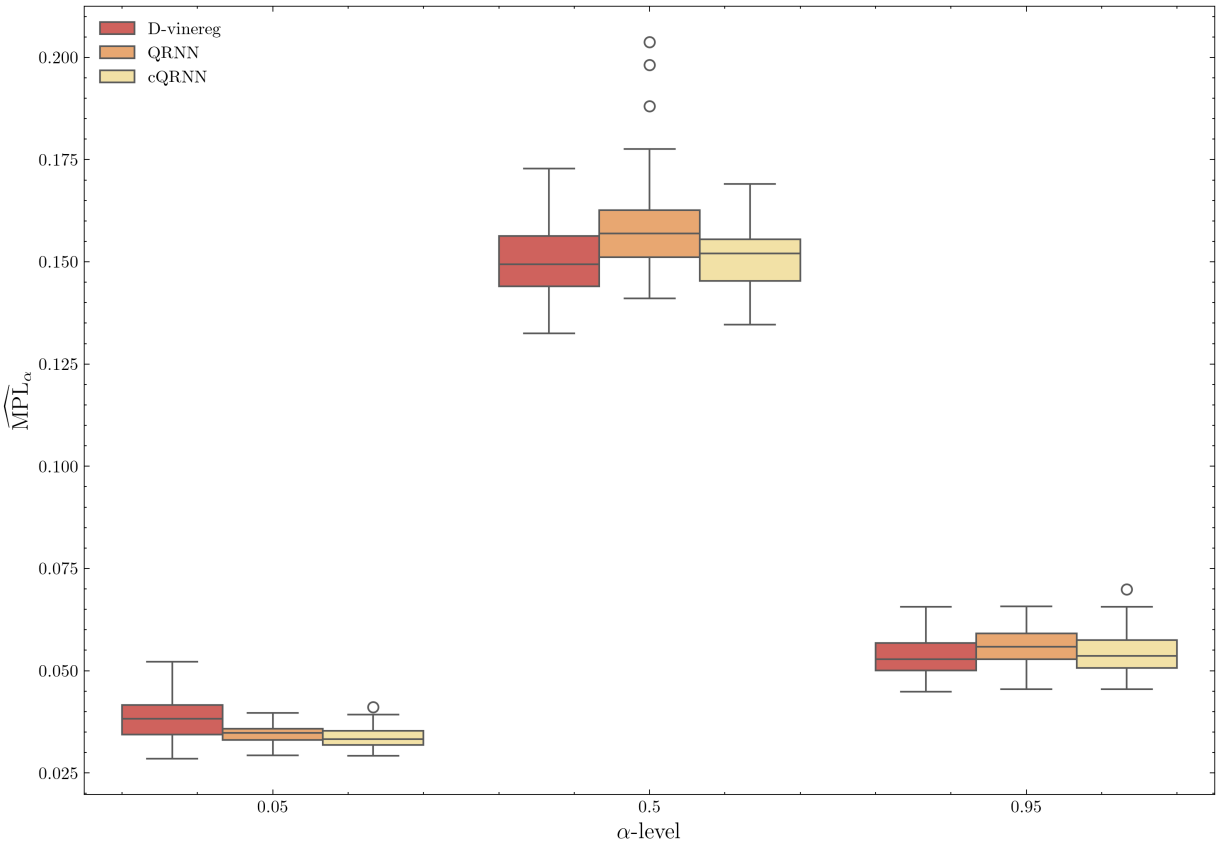
6.3.1 Low-dimensional simulation scenario results

First, we look at the low-dimensional scenario results.

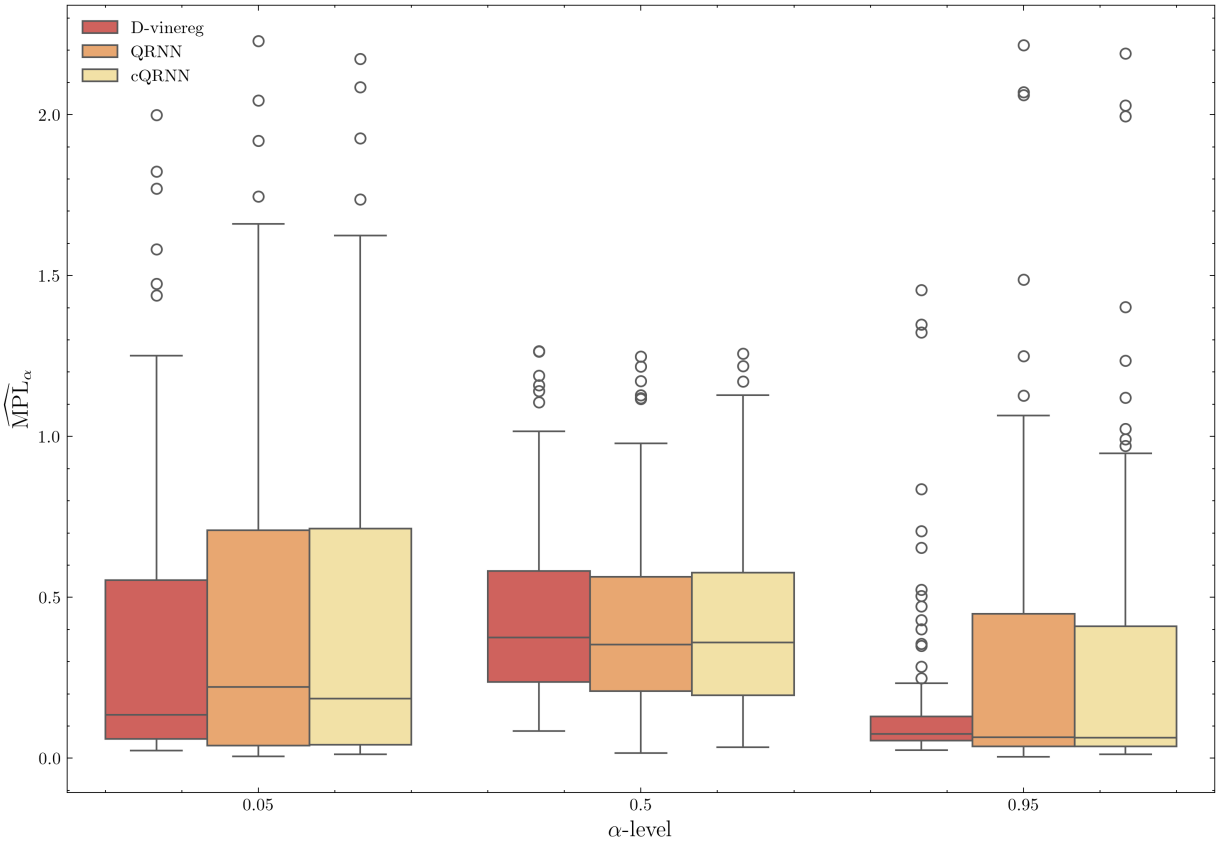
Scenario	Model	$\widehat{\text{T}}$	$\widehat{\text{MIPL}}_{0.05}$	$\widehat{\text{MIPL}}_{0.5}$	$\widehat{\text{MIPL}}_{0.95}$	$\widehat{\text{T}}$	$\widehat{\text{MIPL}}_{0.05}$	$\widehat{\text{MIPL}}_{0.5}$	$\widehat{\text{MIPL}}_{0.95}$
		$n_{train} = 400$				$n_{train} = 1000$			
D3	D-vinereg	**0.49**	**0.04**	**0.15**	**0.05**	**1.14**	0.04	**0.15**	**0.05**
	QRNN	23.55	0.05	**0.15**	**0.05**	17.27	**0.03**	**0.15**	0.06
	cQRNN	6.17	**0.04**	**0.15**	**0.05**	7.58	**0.03**	**0.15**	**0.05**
NH4	D-vinereg	**3.04**	**0.35**	0.47	**0.19**	**7.07**	**0.35**	0.45	**0.17**
	QRNN	9.21	0.41	**0.44**	0.33	15.69	0.43	**0.42**	0.30
	cQRNN	9.85	0.38	0.45	0.33	14.11	0.43	0.43	0.29
LH4	D-vinereg	**3.20**	**0.22**	0.51	**0.20**	**7.40**	0.24	0.46	0.26
	QRNN	14.71	0.24	0.54	0.21	14.80	**0.22**	0.49	**0.20**
	cQRNN	14.51	0.32	**0.39**	0.34	14.14	0.36	**0.40**	0.39

Table 6.4 Out of sample predictions of $\widehat{\text{MIPL}}_{m,\alpha}$, for $\alpha \in \{0.05, 0.5, 0.95\}$ of the different models $m \in \{\text{D-vinereg, QRNN, cQRNN}\}$ for the low dimensional scenarios D3, NH4 and LH4. Lowest values per evaluation measure and scenario are highlighted bold.

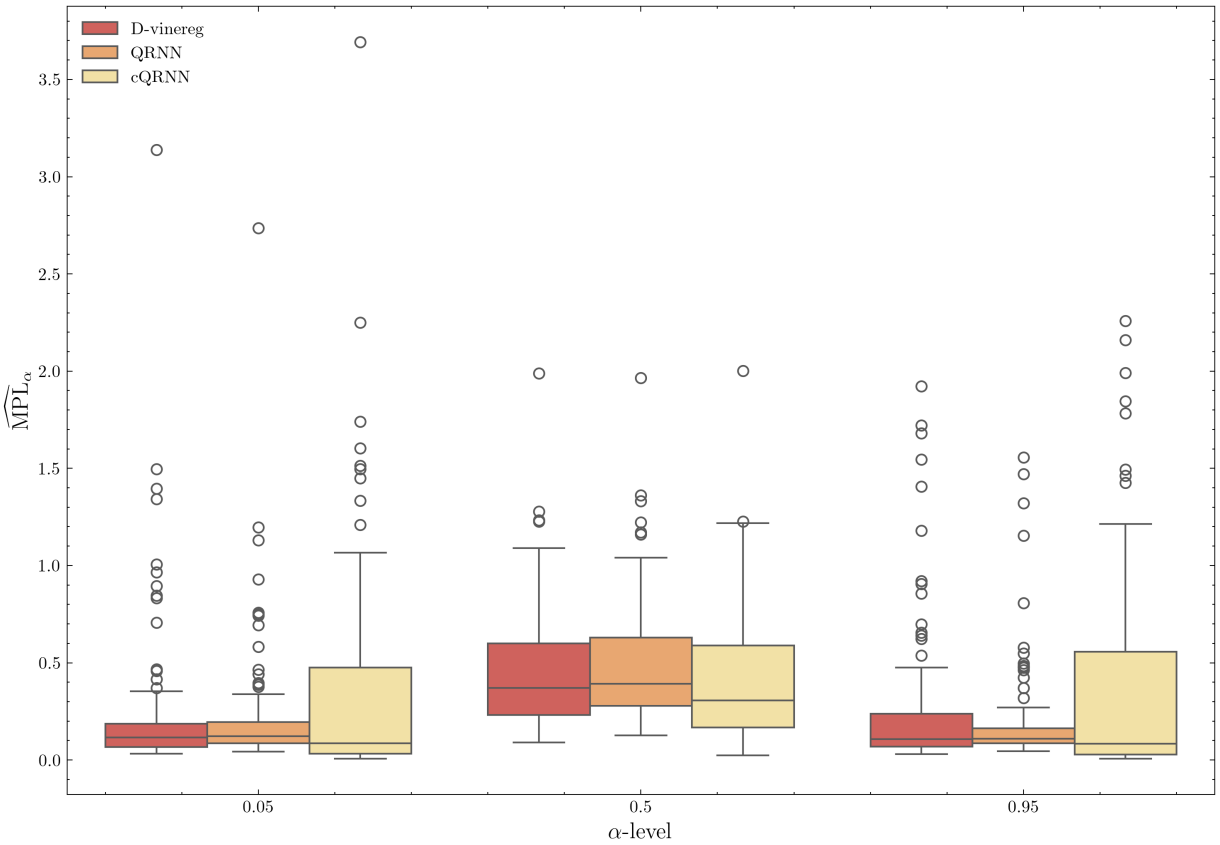
For the three-dimensional D-vine scenario **D3**, the performance of the models D-vine regression, QRNN and cQRNN are similar (see Table 6.4). We observe a small performance increase for the QRNN and cQRNN models when we have higher dimensional training data $n_{train} = 1000$. The QRNN and cQRNN models seem to be a more adequate model for estimating the 0.05 quantile if we are in the large sample training setting (see Figure 6.1a). In terms of the average computation time $\hat{T}$, the D-vine regression model clearly outperforms the quantile regression neural network models. Note that in our case, the average computation time $\hat{T}$ of the QRNN model is lower for small dimensional train data $n_{train} = 400$. This phenomena can arise since the stochastic gradient descent algorithm is not deterministic (Section 3.1).
In the nonlinear and heteroscedastic scenario NH4, the D-vine quantile regression clearly outperforms the quantile neural network models in terms of accuracy and time complexity for both training sizes $n_{train} \in \{400, 1000\}$ (see Figure 6.1b). For the setting linear and heteroscedastic LH4 however, we see that the QRNN performs better than the D-vine quantile regression and the cQRNN for higher dimensional training data $n_{train} = 1000$ (see Figure 6.1c). In summary, for scenarios with low dimensions, the **D-vine quantile regression** model appears to outperform the quantile regression neural network approaches **cQRNN**, **QRNN** in terms of accuracy and time efficiency. Furthermore, in a non-linear setting such as NH4 and D4, the D-vine quantile regression model is more effective at estimating conditional quantiles.



(a) Scenario D3



(b) Scenario NH4



(c) Scenario LH4

Figure 6.1 Boxplots of $\widehat{\text{MPL}}_{m,\alpha}$, $\alpha \in \{0.05, 0.5, 0.95\}$ for scenario D3, NH4 and LH4 with large sample training data size $n_{train} = 1000$ and $m \in \{$D-vinereg, QRNN, cQRNN $\}$.

6.3.2 High-dimensional simulation scenario results

We continue by presenting the results for the high dimensional simulation scenarios **D6** and **G20**. As for the low dimensional settings, we look at the out of sample mean integrated pinball loss $\widehat{\text{MIPL}}_{m,\alpha}$ for $\alpha \in \{0.05, 0.5, 0.95\}$ as well as the time complexity $\hat{\text{T}}$ and the average relative quantile crossings $\widehat{\text{ARQC}}_m$ per model $m \in \{$D-vinereg, QRNN, cQRNN, redQRNN, sparsevinereg, sredQRNN$\}$. **Note**, that we also tested the **reduced cQRNN** model, as introduced in Section 5.2. However, the results did not show any significant improvement over the standard cQRNN approach. Consequently, we have chosen to present only the results for the cQRNN, alongside with the models

$$m \in \{\text{D-vinereg, QRNN, cQRNN, sparsevinereg, sredQRNN}\}$$

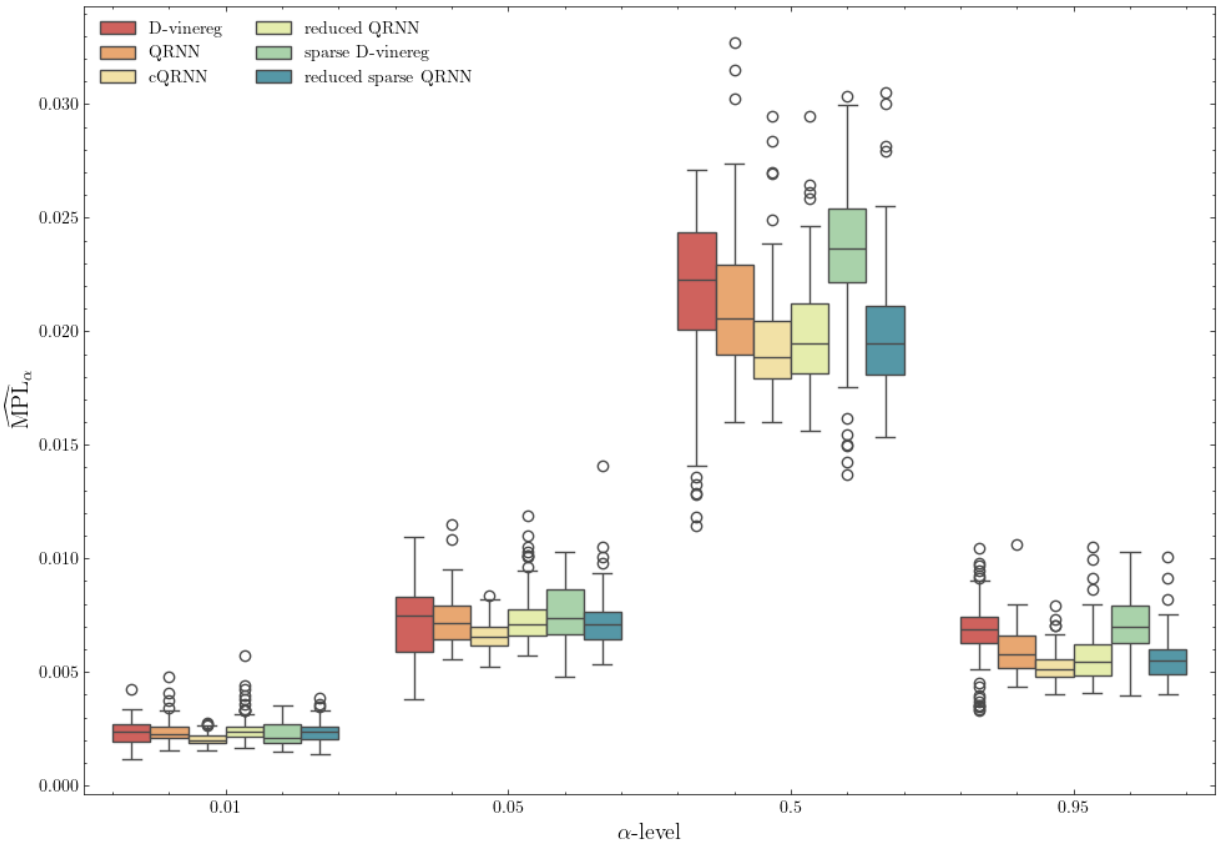
.

Scenario	Model	$\widehat{\mathsf{T}}$	$\widehat{\text{MIPL}}_{0.01}$	$\widehat{\text{MIPL}}_{0.05}$	$\widehat{\text{MIPL}}_{0.5}$	$\widehat{\text{MIPL}}_{0.95}$	$\widehat{\text{ARQC}}$
$n_{train} = 400$							
D6	D-vinereg	**4.6**	**0.002**	**0.007**	0.022	0.007	**0**
	QRNN	20.30	**0.002**	0.008	0.023	**0.006**	10.91
	cQRNN	27.82	**0.002**	**0.007**	0.023	**0.006**	**0**
	redQRNN	21.48	**0.002**	**0.007**	**0.020**	**0.006**	10.72
	sparsevinereg	7.87	**0.002**	0.008	0.025	0.007	**0**
	sredQRNN	21.70	**0.002**	0.008	0.022	**0.006**	10.04
G20	D-vinereg	100.46	0.009	0.022	0.069	0.021	**0**
	QRNN	**12.40**	**0.005**	0.020	0.073	0.019	11.18
	cQRNN	21.75	0.007	**0.018**	**0.065**	0.020	**0**
	redQRNN	16.70	**0.005**	0.019	0.071	**0.018**	11.65
	sparsevinereg	25.34	0.007	0.021	**0.065**	0.019	**0**
	sredQRNN	19.68	**0.005**	0.019	0.072	**0.018**	14.79
$n_{train} = 1000$							
D6	D-vinereg	**10.06**	0.002	0.007	0.022	0.007	**0**
	QRNN	17.29	0.002	0.007	0.021	0.006	8.75
	cQRNN	29.98	**0.001**	**0.006**	**0.019**	**0.005**	**0**
	redQRNN	18.15	0.002	0.007	0.019	**0.005**	9.13
	sparsevinereg	21.88	0.002	0.007	0.023	0.007	**0**
	sredQRNN	18.48	0.002	0.007	0.020	**0.005**	11.91
G20	D-vinereg	271.31	0.007	0.021	0.067	0.020	**0**
	QRNN	**11.75**	0.005	0.019	0.072	0.019	8.52
	cQRNN	21.09	0.007	**0.017**	**0.064**	0.019	**0**
	redQRNN	13.11	0.005	0.019	0.070	**0.018**	9.87
	sparsevinereg	64.84	0.006	0.019	**0.064**	0.019	**0**
	sredQRNN	17.44	**0.004**	0.019	0.068	**0.018**	12.73

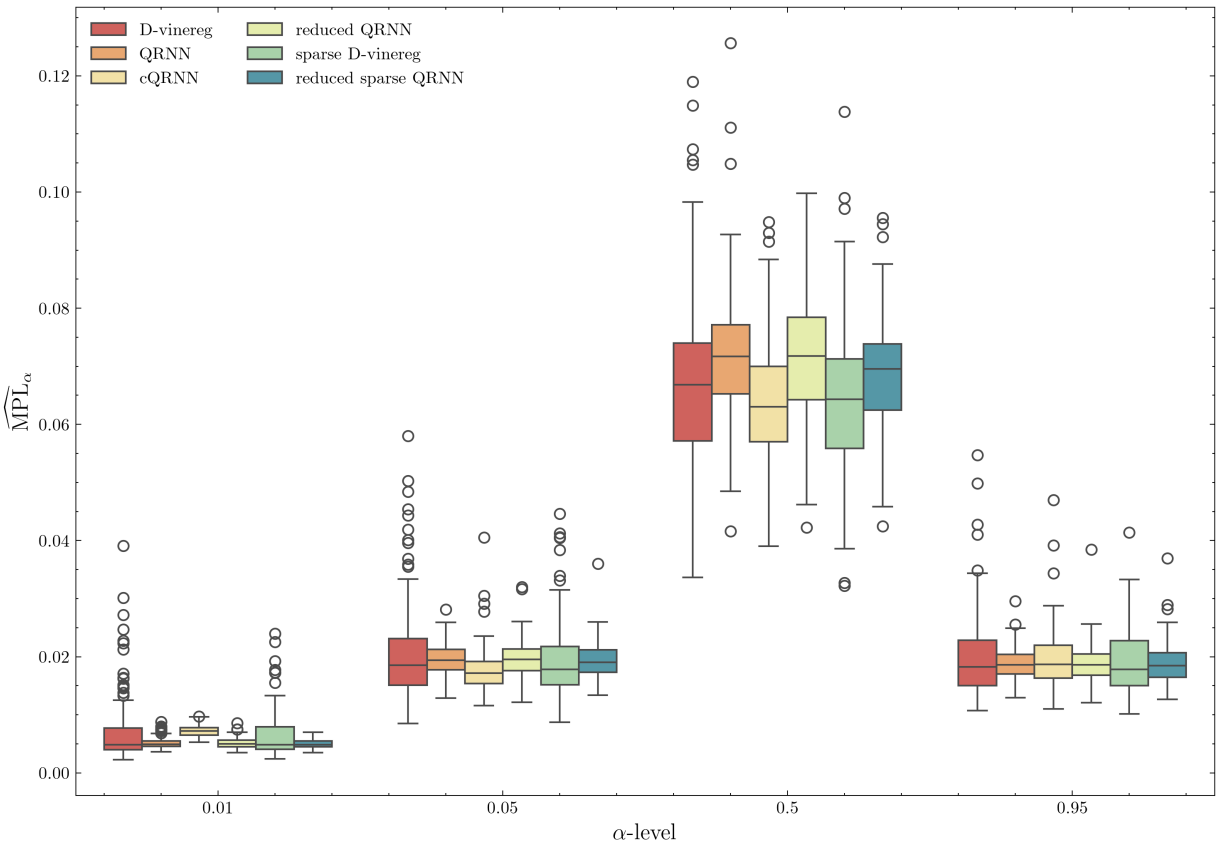
Table 6.5 Out of sample predictions of $\widehat{\text{MIPL}}_{m,\alpha}$, for $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ of the different models $m \in \{\text{D-vinereg, QRNN, cQRNN, redQRNN, sparsevinereg, sredQRNN}\}$ for the high dimensional scenarios D6 and G20. Lowest values per evaluation measure and scenario are highlighted bold.

For the simulation scenario **D6** with training setting $n_{train} = 400$, the **reduced QRNN** and the **cQRNN** are outperforming the other models for all levels $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ while the number of **average relative quantile crossings** of the **redQRNN** model is at 0.107 (or 10.7%) while the **cQRNN** model results inhabit zero quantile crossings (per definition; see Section 5.1). The computing time is similar to the low dimensional simulation scenarios for the quantile regression neural networks. While the D-vine quantile regression exhibits a marginal increase in the computing time, it continues to surpass the quantile regression neural network approaches in terms of computing time. For higher dimensional training data $n_{train} = 1000$, the **cQRNN** outperforms the other models for all levels $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$. See Figure 6.2a for the boxplot of the $\widehat{\text{MPL}}_{m,\alpha}$ for $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ of scenario **D6** and training size $n_{train} = 1000$.

Following with the high dimensional sparse D-vine copula scenario G20 we observe that the **cQRNN** outperforms the other models for the levels $\alpha \in \{0.05, 0.5\}$ and for both training dimensions $n_{train} \in \{400, 1000\}$. We also see a huge increase in the computing time for the D-vine quantile regression model, while the computing time for the quantile regression neural networks are similar than for the lower dimensional simulation scenarios. Note that the **sparsevinereg** model has a significantly lower computing time than the standard **D-vinereg** model, but is still slower compared to the quantile regression neural network approaches. In addition, we notice a small performance increase of the **reduced QRNN** compared to the standard **QRNN** for all alpha levels as well as training data sizes. It is noteworthy that the models QRNN, redQRNN, and sredQRNN exhibit a decreased average relative quantile crossing rate when the sample size is larger ($n_{train} = 1000$) as compared to the scenario with a smaller sample size ($n_{train} = 400$). The average relative quantile crossing rate for these models hovers around 10% which is still high. Therefore, for high-dimensional scenarios (for instance, when the number of dependent variables exceeds 10), a **cumulative quantile regression neural network (cQRNN)** approach is recommended. This method is particularly effective in handling high dimensional data while guaranteeing the absence of quantile crossings of the model estimates. See Figure 6.2b for the boxplot of the $\widehat{\text{MPL}}_{m,\alpha}$ for $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ of scenario **G20** and training size $n_{train} = 1000$.



(a) Scenario D6



(b) Scenario G20

Figure 6.2 Boxplots of $\widehat{\text{MPL}}_\alpha$, $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ for scenarios D6 and G20 with high dimensional training data $n_{train} = 1000$.

Moreover, we fit the 95% prediction interval introduced in Section 5.3 to the models **QRNN** and **redQRNN** for the simulation scenarios **D6** and **G20**. The accuracy of the prediction intervals is measured with the interval score $\widehat{\text{IS}}_{0.05}$ where we need to calculate the true quantiles in order to calculate the interval score. This is done by using the **inverse Rosenblatt transform** briefly discussed in Algorithm 4.1.1. For more details regarding the **inverse Rosenblatt transform** look into Rosenblatt (1952) and Rüschendorf (1981). We retrain the last hidden layer of the models for $S = 30$ times with $K = 400$ randomly sampled observations from the fitted D-vine copula $C^{D_{train},[H]}$ (See Section 5.3 for the notation). This process is repeated $R = 20$ times.

G20	$q_{0.01}$	$q_{0.05}$	$q_{0.5}$
QRNN	3.931	2.830	2.339
redQRNN	**3.133**	**2.491**	**1.827**

Table 6.6 $\widehat{\text{IS}}_{0.05}$ for scenario **G20**

D6	$q_{0.01}$	$q_{0.05}$	$q_{0.5}$
QRNN	**7.623**	**9.325**	1.329
redQRNN	7.744	9.487	**1.221**

Table 6.7 $\widehat{\text{IS}}_{0.05}$ for scenario **D6**

We observe a smaller interval score for the quantiles at level $\alpha \in \{0.01, 0.05, 0.5\}$ for the **reduced QRNN** model in the **G20** scenario. In the setting of scenario **D6**, however, the QRNN is outperforming the reduced QRNN at the levels $\alpha \in \{0.01, 0.05\}$. For the higher dimensional scenario **G20** we prefer the **reduced QRNN** model over the **QRNN** model. For future projects, it would be interesting to implement the prediction intervals introduced in Section 5.3 for the **cQRNN** model studied in Section 5.1. This could be achieved by fitting a prediction interval with retraining (see Section 5.3 for the procedure) to each **quasi-quantile regression neural network** within the cQRNN. The resulting prediction intervals can then be added up to obtain a simultaneous prediction interval, similar to the simultaneous prediction interval (5.22).

7 Real Data Example: Fama and French

The Fama and French five-factor model extends the classical capital asset pricing model (CAPM) and is designed to capture the relation between the average return $Y^{(i)}$ of an asset i and the factors

- **Mkt-RF** Market risk premium, which is calculated as the expected return on the market minus the risk-free rate.
- **SMB** The small minus big factor is based on the market capitalization of a company. The idea is that in the long run, small-capitalization companies tend to have higher returns than large-capitalization companies. The SMB factor is determined by computing the difference between the average returns of a portfolio composed of small-cap stocks and that of a portfolio consisting out of large-cap stocks.
- **HML** The high minus low factor quantifies the difference of returns between high book-to-market and low book-to-market stocks. In order to compute the HML factor, the difference in returns between a portfolio of high book-to-market firms and a portfolio of low book-to-market firms is calculated. The underlying assumption is that high book-to-market stocks are expected to yield higher returns with respect to low book-to-market firms.
- **CMA** The Conservative minus aggressive factor (CMA) is calculated as the difference in returns between a portfolio of conservatively investing firms and a portfolio of aggressively investing firms. The underlying assumption is that, all else being equal, companies that invest conservatively have higher expected returns than those that invest aggressively. Companies which invest more aggressive are those who frequently extend their asset bases through acquiring other businesses, launching new products, or entering new markets.
- **RMW** Robust minus weak factor measures the excess return of companies with high operating provability over those with low operating provability. The underlying assumption of this factor is the empirical observation that companies with higher profits tend to have higher returns.

The original idea of extending the CAPM model in order to capture the expected returns of stocks was introduced by the work of Fama and French (1993) and is called the Fama and French 3-factor model, or short **FF 3-factor model**. This model inc operates SMB and HML factors in addition to the market risk premium in order to capture the average return of an asset. It was then extended by Fama and French (2014), who added the CMA and RMW factors to the 3-factor model. Note that the factors are measured for a whole market, for example the US stock market. Given an asset i, we want to explain the expected return by the exposures to the FF factors. The Fama and French 5-factor model of an asset i is the regression model

$$Y_t^{(i)} - R_{f_t} = \alpha_i + \beta_i^{(1)}(R_{M_t} - R_{f_t}) + \beta_i^{(2)}\mathit{SMB}_t + \beta_i^{(3)}\mathit{HML}_t + \beta_i^{(4)}\mathit{CMA}_t + \beta_i^{(5)}\mathit{RWM}_t + \epsilon_i, \qquad (7.1)$$

where $Y_t^{(i)}$ is the return of an asset i for the period t, R_{f_t} is the risk-free return, R_{M_t} is the return on the market portfolio and ϵ_{it} is a zero-mean residual (Fama and French 2014). If the exposures to the five factors $\beta_{it}^{(1)}$, $\beta_{it}^{(2)}$, $\beta_{it}^{(3)}$, $\beta_{it}^{(4)}$ and $\beta_{it}^{(5)}$ capture all variation in the expected returns, the intercept α_i in (7.1) is zero for all securities and portfolios i (Fama and French 2014).

7.1 Explanatory analysis

In our study, we will consider the daily log-returns of 8 U.S. assets from the U.S. stock market. The risk-free rate will be approximated by the yield on a 3-month U.S. Treasury bill and the historical stock prices are collected from yahoo-finance, using the `ytfinance` package in Python. The daily Fama and French factors for the U.S. stock market can be downloaded from Kenneth French's data library. [1] For the observation period, we choose January 1, 2007 to December 1, 2023. The time series of the Fama and French factors are clearly non-stationary, which can be seen in Figure 7.2. The factor spikes observed in 2009 can again be attributed to the financial crisis that occurred in 2008. Additionally, noticeable fluctuations and increased volatility of the factors were evident in the years 2020 and 2023. These could potentially be a result of the 2020 Corona crisis and the Russian invasion of the Ukraine in 2022. The time series associated with the five Fama and French factors will be called $Y_t^{(f)}$ with $t = 1, \ldots, T$ and $f = 1, \ldots, 5$ where we have the following abbreviations

$$\{1 = \text{Mkt-RF}, 2 = \text{HML}, 3 = \text{SMB}, 4 = \text{CMA}, 5 = \text{RMW}\}$$

.

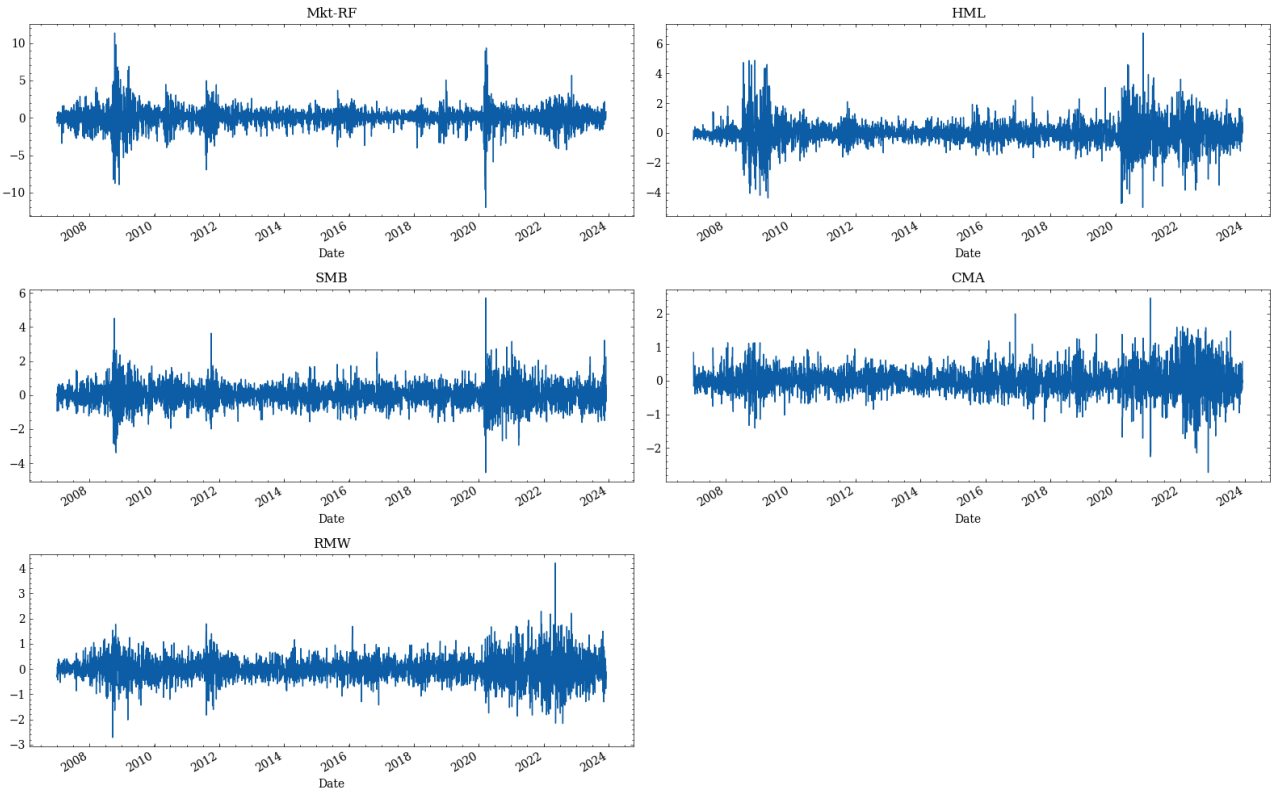


Figure 7.1 Time series $Y_t^{(f)}$ of the five Fama and French factors $f \in \{\text{Mkt-RF, HML, SMB, CMA, RMW}\}$ for the period January 1, 2007 to December 1, 2023.

Looking at the empirical histograms of the factors, we observe a symmetric distribution of all five factors around zero. The presence of outliers in the corresponding time series of the factors leads us to propose that the factors follow a Student's t-distribution, which is characterized by heavier tails compared to the normal distribution.

1. https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/

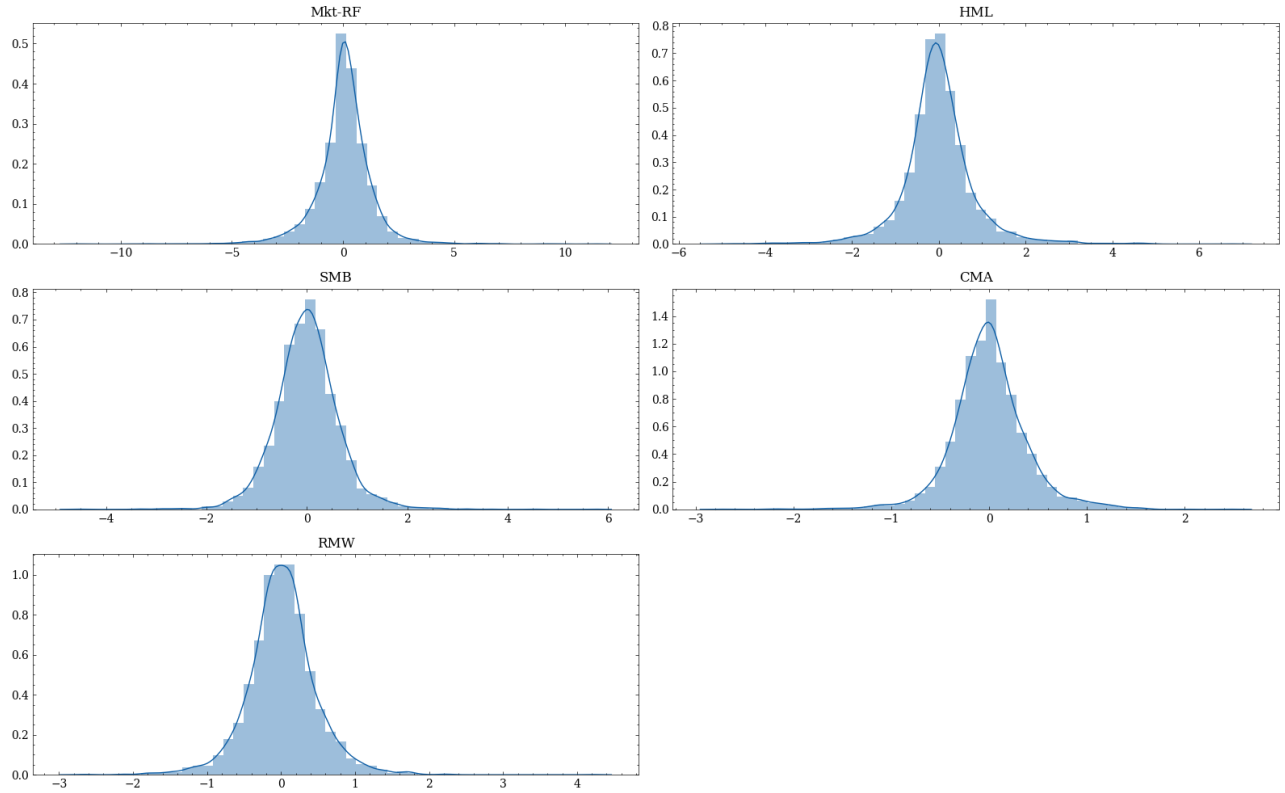


Figure 7.2 Histograms of the five Fama and French factors for the period January 1, 2007 to December 1, 2023.

We consider the daily log return in the observation period January 1, 2007 to December 1, 2023 of the following 8 stocks, resulting in working with a multivariate financial time series.

- **AAPL** Apple Inc. is a manufactor of smartphones, tablets and computers.
- **MSFT** Microsoft Corporation is a software company which offers and supplies software services and devices.
- **BIIB** Biogen Inc. is a healthcare company that develops medecine for neurological diseases.
- **COO** Cooper Companies Inc is a leading global medical device company.
- **KO** The Coca-Cola Company manufacts and sells nonalcoholic drinks worldwide.
- **INTC** Intel Corporation produces and sells computer products and technologies
- **UNH** UnitedHealth Group Incorporated is a healthcare company.
- **DIS** The Walt Disney Company is an entertainment company which produces movies.

Note that the chosen stocks are listed in the **S&P 500** index. The assets we have chosen are from the technology, healthcare, entertainment and food sectors. The spikes observed in the time series of daily log returns (Figure 7.3) in the years 2008/2009, 2020, and 2022 can be attributed to specific global events. These include the financial crisis of 2008, the Corona crisis in 2020, and the Russian invasion of Ukraine in 2022, respectively. Each of these events had significant impacts on global markets, which is reflected in the daily log returns for those periods. The time series of the log-returns of the chosen assets will be called $Y_t^{(a)}$ with $t = 1, \ldots, T$ and $a = 1, \ldots, 8$ where we have the following abbreviations:

$$\{1\text{=AAPL, } 2\text{ = MSFT, } 3\text{ = BIIB, } 4\text{ = COO, } 5\text{ = KO, } 6\text{ = INTC, } 7\text{ = UNH, } 8\text{ = DIS}\}$$

.

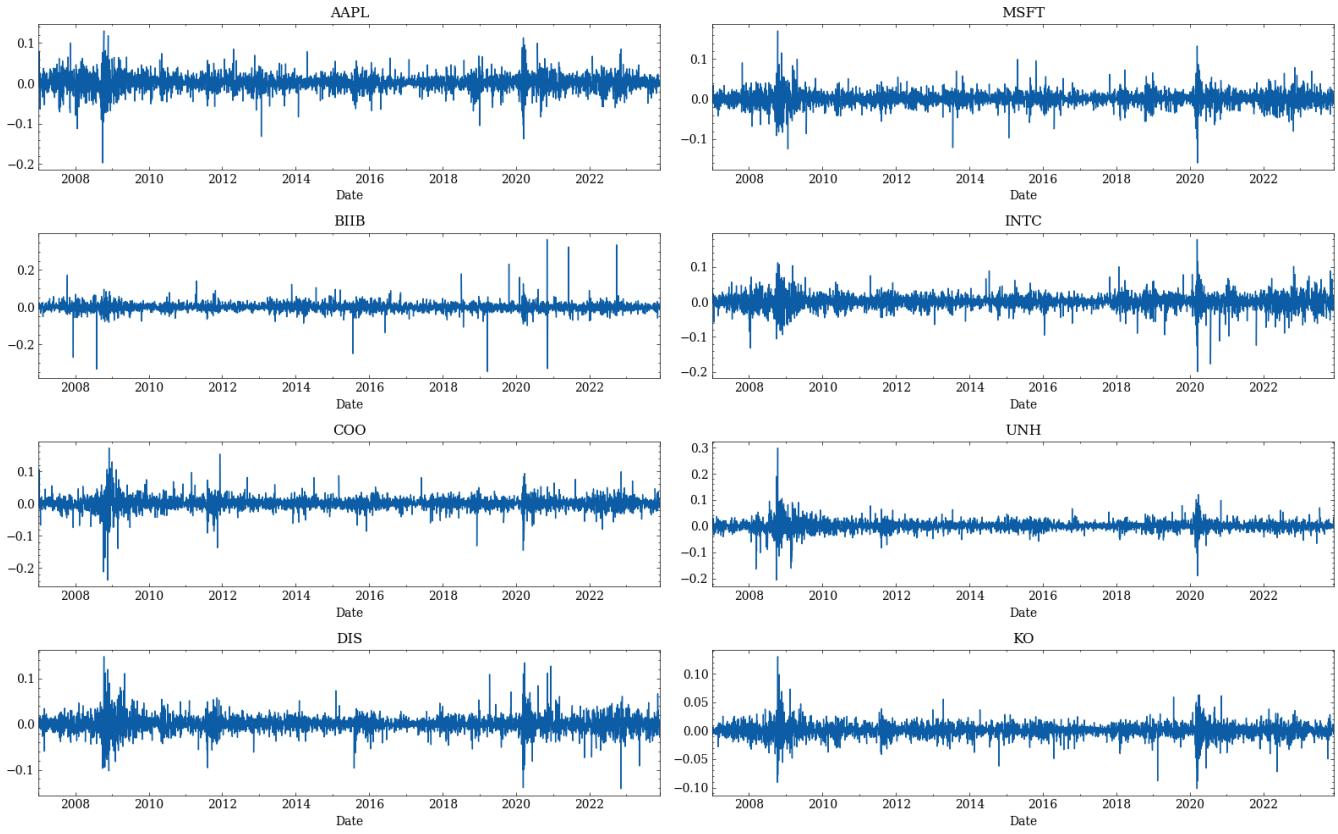


Figure 7.3 Daily log-returns of the AAPL, MSFT, BIIB, COO, KO, INTC, UNH and DIS for the period of January 1, 2007 to December 1, 2023.

7.2 Removing serial dependence for each Fama French and stock time series

As we have already discussed in Section 2.2.2 multivariate financial time series often exhibit interdependence due to their operation within the same market or sector. It is also observed that joint losses tend to occur more frequently than joint gains, indicating the need for models that can handle asymmetric dependence structures. While a Gaussian copula is unable to model this behavior, a D-vine copula is capable of doing so. In order to remove serial dependence, we first apply an ARMA-GARCH model (see Section 2.2.1) to the marginal time series of the assets log-returns $Y_t^{(a)}$, $a = 1, \dots, 8$ and the five Fama and Frech factors $X_t^{(f)}$, $f = 1, \dots, 5$ for $t = 1, \dots, T$. We choose the innovation function to be the standardized Student-t distribution with degree of freedom ν. Then we calculate the standardized residuals $\hat{r}^{(a)}$, $a = 1, \dots, 8$ and $\hat{r}^{(f)}$, $f = 1, \dots, 5$ of the log returns and the five Fama and French factors with Equation (2.12). The **innovation distribution function** is then used to transform the standardized residuals to approximate i.i.d. copula data.

Start by fitting a *ARMA*$(1,1)$-*GARCH*$(1,1)$ model with Student's t innovation to each time series $Y_t^{(a)}$ and $X_t^{(f)}$ resulting in conditional mean estimates $\hat{\mu}_t^{(a)}$ and $\hat{\mu}_t^{(f),\text{FF}}$ and conditional variance estimates $\hat{\sigma}_t^{2,(a)}$ and $\hat{\sigma}_t^{2,(f),\text{FF}}$ for $t = 1, \dots, T$, $a = 1, \dots, 8$ and $f = 1, \dots, 5$, respectively. The *ARMA*$(1,1)$-*GARCH*$(1,1)$ model is a good starting point for financial time series (Tsay 2010).

	ϕ_0	ϕ_1	θ_1	α_0	α_1	β_1	$\hat{\nu}$
AAPL	0.00	-0.30	0.30	0.00	0.09	0.90	5.03
MSFT	0.00	0.76	-0.80	0.00	0.09	0.89	4.68
BIIB	0.00	0.31	-0.38	0.00	0.16	0.60	3.26
INTC	0.00	-0.13	0.11	0.00	0.06	0.94	4.44
COO	0.00	0.50	-0.51	0.00	0.06	0.92	3.98
UNH	0.00	-0.25	0.23	0.00	0.06	0.93	4.44
DIS	0.00	0.79	-0.81	0.00	0.08	0.92	4.54
KO	0.00	0.92	-0.94	0.00	0.09	0.88	4.68
Mkt-Rf	0.02	0.84	-0.88	0.02	0.14	0.86	5.72
SMB	0.00	0.55	-0.56	0.01	0.07	0.92	10.00
HML	-0.01	0.65	-0.65	0.00	0.10	0.90	8.05
RMW	0.01	-0.46	0.47	0.00	0.05	0.94	10.00
CMA	-0.01	-0.10	0.14	0.00	0.05	0.94	9.35

Table 7.1 Estimates of the *ARMA*(1,1)-*GARCH*(1,1) coefficients for the eight log-return time series and the five Fama and French factors.

Next we calculate the **standardized residuals**

$$r_t^{(a)} := \frac{Y_t^{(a)} - \hat{\mu}_t^{(a)}}{\hat{\sigma}_t^{(a)}} \qquad \textit{and} \qquad r_t^{(f),\mathsf{FF}} := \frac{X_t^{(f)} - \hat{\mu}_t^{(f),\mathsf{FF}}}{\hat{\sigma}_t^{(f),\mathsf{FF}}}, \tag{7.2}$$

for all time series $Y_t^{(a)}$, $X_t^{(f)}$, $a = 1, \ldots, 8$, $f = 1, \ldots, 5$.

After calculating the standardized residuals, we check if the *ARMA*$(1,1)$-*GARCH*$(1,1)$ with Student-t innovations fits the log-returns and Fama and French time series by looking at the QQ-plots and conducting the Ljung-Box test as outlined in Section 2.2.1. We observe that the innovation function fits the diagonal line of the QQ-plots for most of the assets (see Figure 7.4). For the COO asset we might try to fit another innovation function. The QQ-plots of the standardized residual of the Fama and French factors time series (see Figure 7.5) appear to be a good.

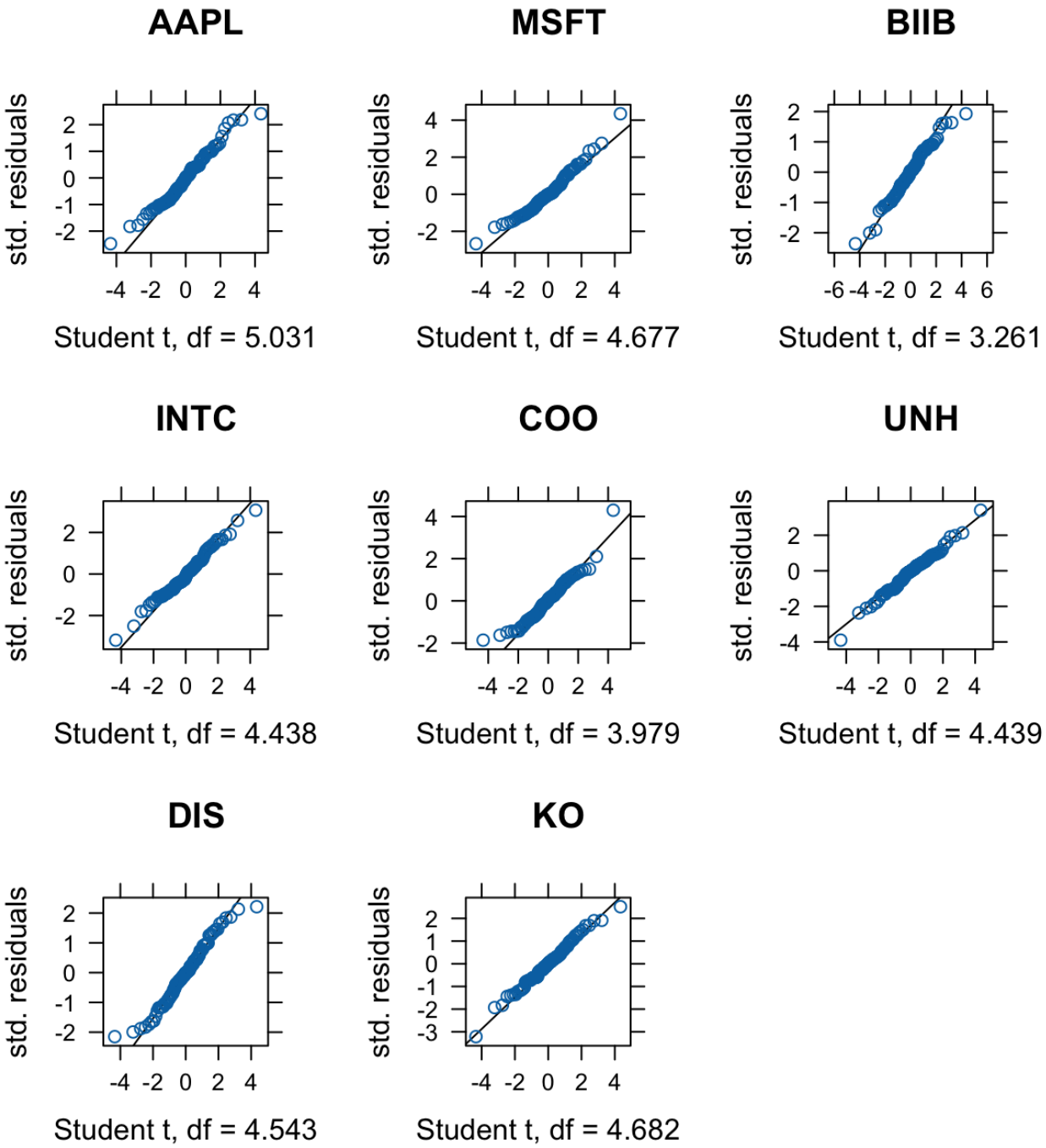


Figure 7.4 QQ-plots of the standardized residuals $r_t^{(a)}$ for $Y_t^{(a)}$, $a = 1, \ldots, 8$ after fitting an *ARMA*$(1,1)$-*GARCH*$(1,1)$ model with Student t innovations to each of the log-return time series.

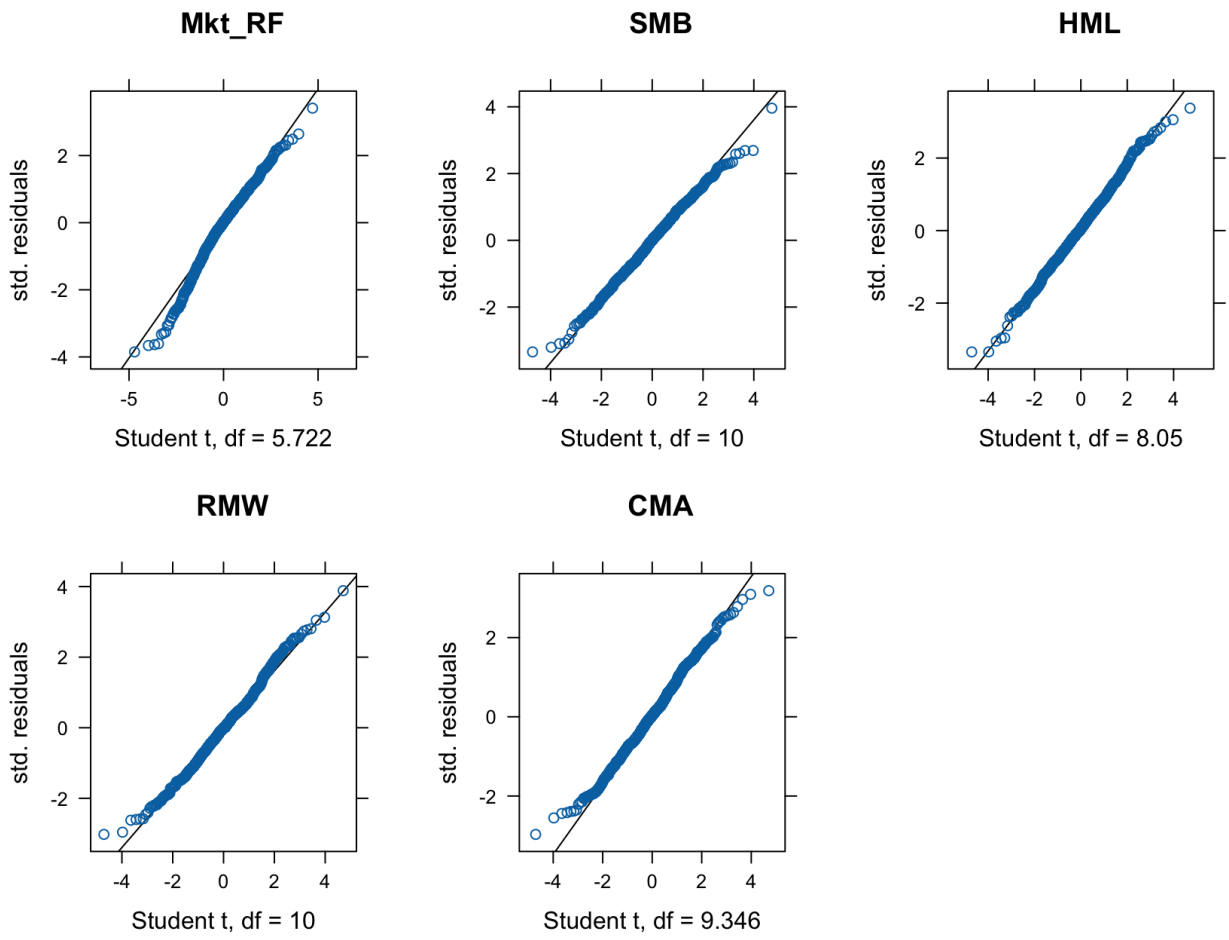


Figure 7.5 QQ-plots of the standardized residuals $r_t^{(f),\text{FF}}$ for $X_t^{(f)}$, $a = 1, \ldots, 8$ after fitting an *ARMA*$(1,1)$-*GARCH*$(1,1)$ model with Student t innovations to each of the Fama and French time series.

Next, we apply the Ljung-Box test described in Section 2.2.1 to each standardized residual of each univariate time series. The red colors of the visualization of the test (see Figure 7.6) represent a p-value < 0.05 which means that there are significant autocorrelation effects of the corresponding lag of the standardized residuals. Our observation indicates that significant autocorrelation effects are present only for lags greater than 7.

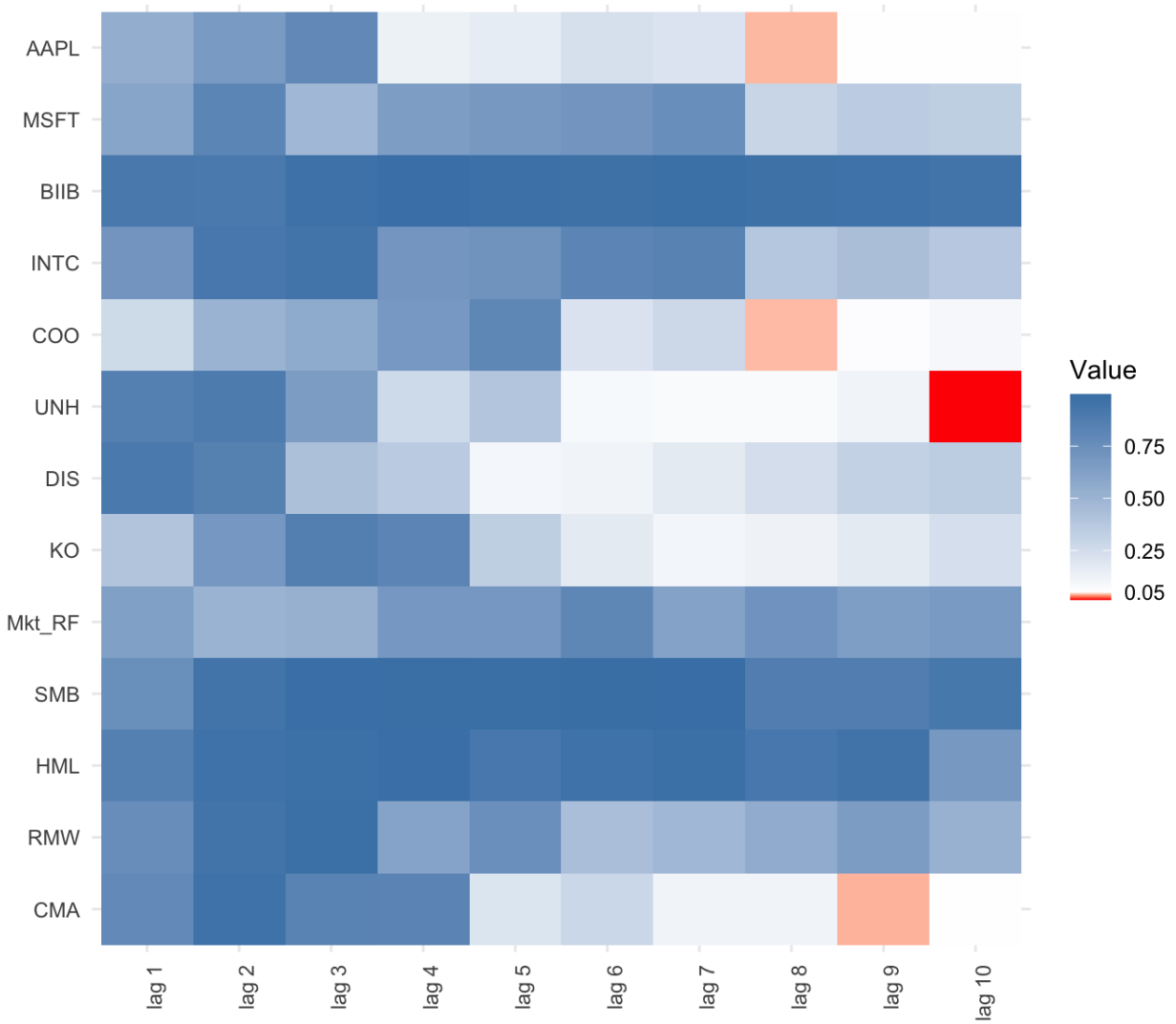


Figure 7.6 Heatmap of the p-values of the Ljung-Box test statistic (2.13) for different lags.

We then transform the standardized residuals to pseudo copula data by computing

$$u_t^{(a)} = F\left(r_t^{(a)}; \hat{\nu}^{(a)}\right) \qquad \textit{and} \qquad u_t^{(f),\mathsf{FF}} = F\left(r_t^{(f),\mathsf{FF}}; \hat{\nu}^{(f),\mathsf{FF}}\right), \tag{7.3}$$

where $F(\cdot, \hat{\nu})$ is the cumulative distribution function of an univariate Student's t distribution with mean 0, scale 1 and degrees of freedom given by $\hat{\nu}^{(a)}$ and $\hat{\nu}^{(f),\mathsf{FF}}$, $a = 1, \ldots, 8$, $f = 1, \ldots, 5$ and $t = 1, \ldots, T$.

Figure 7.7 Marginal normalized contours for assets AAPL, MSFT, BIIB, INTC, COO, UNH, DIS, KO and the five Fama and French factors.

In our study, we pick the pseudo copula data $u_t^{(1)}$ (AAPL) and utilize the Fama and French pseudo copula data $u_t^{(f),\mathsf{FF}}$, $f = 1, \ldots, 5$ in order to predict the value at risk VaR_α at level $\alpha \in \{0.01, 0.05, 0.1\}$ described in Section 2.2.2.

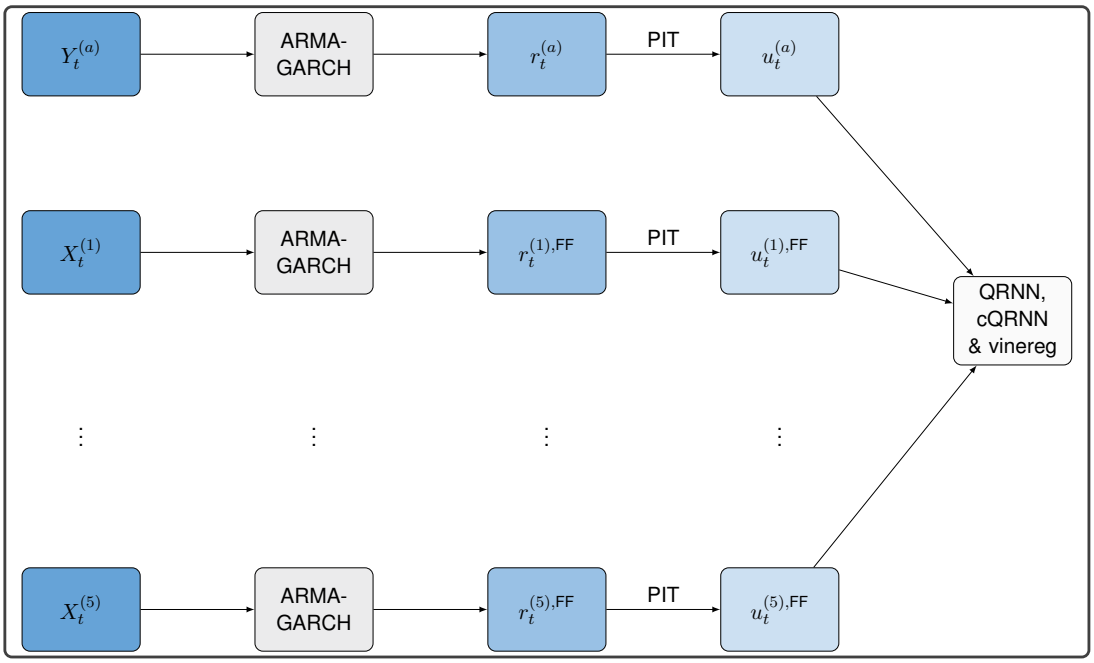


Figure 7.8 Modeling of log-return time series $Y_t^{(a)}$ for $t = 1, \ldots, T$ and $a = 1, \ldots, 8$

8 Supervised learning setup for Apple Inc. stock

In the following, we will train the following models in order to predict the VaR$_\alpha$, $\alpha \in \{0.01, 0.05, 0.1\}$ for the Apple Inc. stock (AAPL) $Y_t^{(1)}$:

- **QRNN**: Quantile regression neural network $\Phi_{(5,200,200,3)}^{(\text{ReLU, sigmoid}),\rho_\alpha}$ with input dimension 5 (the five Fama and French factors) and two hidden layers. Each of the layers contains 200 neurons, and we apply the ReLU function to the first hidden layer, while for the second layer, we utilize the sigmoid function. (See Section 5)
- **cQRNN**: Cumulative quantile regression model, where each **quasi quantile neural network** consists of one hidden layer with 200 neurons equipped with the sigmoid activation function. Overall, we accumulate three neural networks, each with one hidden layer with 200 neurons and the sigmoid function as the activation function. (See Section 5.1)
- **D-vinereg**: D-vine quantile regression. (See Section 4.2)

As seen in previous sections, we split the data in a training set D_{train} and a test set D_{test} in order to fit and validate the models. Since we are in a timeseries scenario, the training and test set is given as

$$D_{train} = \{(Y_t^{(1)}, \mathbf{X}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = 1, \dots, T_{train}\} \tag{8.1}$$

and

$$D_{test} = \{(Y_t^{(1)}, \mathbf{X}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = T_{train} + 1, \dots, T_{test}\}, \tag{8.2}$$

where $Y_t^{(1)}$ is the log-return of the Apple Inc. stock and $\mathbf{X}_t := (X_t^{(1)}, \dots, X_t^{(5)})$ where $X_t^{(f)}$, $f = 1, \dots, 5$ are the five Fama and French factor time series. Note that the index shift in the predicting variables $\mathbf{X}_{t-1}$ is attributed to the unavailability of the Fama and French factors for the current day. For the training and testing period, we choose the following time spans:

- **Financial crisis**: The models are fitted to the training data dated prior to the 2008 financial crisis. Specifically, our training dataset spans from January 4, 2007, to March 14, 2008, while the testing phase covers the period from March 17, 2008, to January 5, 2009. In total, we have 300 observations in the training set $n_{\text{train}} = 300$ and 200 observations in the test set $n_{\text{test}} = 200$.
- **Corona crisis**: We study the performance of the models using pre-coronavirus crisis data up to 2020. Our training dataset spans from July 18, 2016, to July 11, 2019, while the testing phase covers the period from July 12, 2019, to November 27, 2020. The training set consists of $n_{train} = 750$ observations, while our testing size is $n_{test} = 350$.
- **Post financial crisis**: The models are trained using both pre-crisis and crisis data. The training period spans from January 4, 2007, to May 22, 2009, while the testing phase extends from May 26, 2009, to March 9, 2010. We have a training size of $n_{train} = 400$ and testing size of $n_{test} = 200$.
- **Post corona crisis**: Following a similar approach to the pre-crisis period, our models are trained using data containing both the pre-crisis and crisis periods. The training period spans from January 9, 2019, to January 10, 2022, while the testing phase extends from January 11, 2022, to January 9, 2023. Our training data set consists of $n_{train} = 650$ observations, while the test set utilizes $n_{test} = 250$ observations.

In our thesis we make a simplifying assumption, namely, we fit the *ARMA-GARCH* to the set $D := D_{train} \cup D_{test}$ which results in pseudo copula data $u_t^{(1)}$ and $u_t^{(f),\mathsf{FF}}$ for $t = 1, \dots, T_{test}$ and $f = 1, \dots, 5$. In real-life examples, one has to fit the *ARMA-GARCH* model **solely** to the training data set and predict the conditional mean and volatility in order to transform the log returns of the test set to pseudo copula data. In practice, this is done via back-testing with a rolling window. For a further description and application, look at Sommer, Bax, and Czado (2023). In our work, however, fitting the models QRNN, cQRNN and D-vinereg is very time consuming for larger training and test sets. We will proceed as follows, where the first two steps are visualized in Figure 7.8, setting $a = 1$:

1. Fit an *ARMA*$(1,1)$-*GARCH*$(1,1)$ to the set $D = D_{train} \cup D_{test}$ and calculate the pseudo copula data $u_t^{(1)}$ and $u_t^{(f),\mathsf{FF}}$ for $t = 1, \dots, T_{train}, \dots, T_{test}$.

2. For levels $\alpha \in \{0.01, 0.05, 0.1\}$, we fit the models **QRNN**, **cQRNN** and **D-vinereg** to the transformed training data
$$D_{train}^u := \left\{ (u_t^{(1)}, \mathbf{u}_{t-1}^{\mathsf{FF}}) \in \mathbb{R} \times \mathbb{R}^5 | t = 1, \dots, T_{train} \right\},$$
where $\mathbf{u}_t^{\mathsf{FF}} = \left(u^{(1),\mathsf{FF}}, \dots, u_t^{(5),\mathsf{FF}} \right)$ for $t = 1, \dots, T_{train}$.

3. Next, we predict the conditional quantiles at levels $\alpha \in \{0.01, 0.05, 0.1\}$ at the u-scale by applying the models to copula data of the Fama and French factors of the test set
$$D_{test}^u := \left\{ (u_t^{(1)}, \mathbf{u}_{t-1}^{\mathsf{FF}}) \in \mathbb{R} \times \mathbb{R}^5 | t = T_{train} + 1, \dots, T_{test} \right\},$$
where $\mathbf{u}_t^{\mathsf{FF}} = \left(u_t^{(1),\mathsf{FF}}, \dots, u_t^{(5),\mathsf{FF}} \right)$ for $t = T_{train} + 1, \dots, T_{test}$. We write the estimate of the conditional quantile at level $\alpha\{0.01, 0.05, 0.1\}$ as $\hat{q}_{\alpha,t}^u$, where u symbolizes the u-scale.

4. Further, we transform the quantiles back to the original scale in order to receive an estimate of the VaR_α at level $\alpha \in \{0.01, 0.05, 0.1\}$ for each model **QRNN**, **cQRNN** and **D-vinereg**:
$$\widehat{\mathsf{VaR}}_{\alpha,t} = F^{-1}(\hat{q}_{\alpha,t}^u, \hat{\nu}^{(1)}) \cdot \hat{\sigma}_t^{(1)} + \hat{\mu}_t^{(1)}, \tag{8.3}$$
for $t = T_{train} + 1, \dots, T_{test}$.

5. Lastly, we calculate the **average number of exceedances in the testing period**
$$\widehat{\mathsf{AE}}_\alpha := \frac{1}{|D_{test}|} \sum_{t=T_{train}+1}^{T_{test}} \mathbb{1} \left\{ \widehat{\mathsf{VaR}}_{\alpha,t} < Y_t^{(1)} \right\}, \tag{8.4}$$
for all levels $\alpha \in \{0.01, 0.05, 0.1\}$, where $|\cdot|$ is the cardinality function and $Y_t^{(1)}$ is the log-return time series of the Apple Inc. stock.

This process is then repeated for the scenario where the remaining stock prices are utilized as predictive variables. Additionally, we consider a scenario where we combine the remaining stock prices with the five Fama and French factors in order to predict the conditional value at risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$.

8.1 Results for Fama and French setting

We begin by presenting the model performances for the scenario where we use the five Fama and French factors as the predictive variables. The training and test set is given by:
$$D_{train}^u := \left\{ (u_t^{(1)}, \mathbf{u}_{t-1}^{\mathsf{FF}}) \in \mathbb{R} \times \mathbb{R}^5 | t = 1, \dots, T_{train} \right\}, \tag{8.5}$$
$$D_{test}^u := \left\{ (u_t^{(1)}, \mathbf{u}_{t-1}^{\mathsf{FF}}) \in \mathbb{R} \times \mathbb{R}^5 | t = T_{train} + 1, \dots, T_{test} \right\}, \tag{8.6}$$

where $\mathbf{u}_t^{\mathsf{FF}} = \left(u_t^{(1),\mathsf{FF}}, \ldots, u_t^{(5),\mathsf{FF}}\right)$ for $t = 1, \ldots, T_{train}, \ldots, T_{test}$. As indicated in Table 8.1, the estimated conditional Value at Risk $\widehat{\mathsf{VaR}}_\alpha$ for the **financial- and corona crisis** is surpassed more frequently than expected for all levels $\alpha \in \{0.01, 0.05, 0.1\}$. This can be attributed to the absence of crisis periods in the training data. We observe that the **D-vinereg** and **cQRNN** models are a more conservative and better prediction of the conditional Value at Risk for both crises scenarios, since the average number of exceedances is closest to the levels $\alpha \in \{0.01, 0.05, 0.1\}$.

	Financial crisis			Corona crisis		
	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$
D-vinereg	**0.0049**	0.0788	**0.1478**	**0.0287**	0.0688	0.1261
cQRNN	0.0148	**0.0739**	0.1626	0.0315	**0.0630**	**0.1203**
QRNN	0.0148	0.0887	0.1921	0.0401	0.0688	0.1347
	Post financial crisis			Post corona crisis		
	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$
D-vinereg	**0.0050**	0.0201	0.0704	**0.0080**	0.0600	0.1800
cQRNN	**0.0050**	0.0302	0.0704	**0.0080**	0.0400	**0.1080**
QRNN	0.0000	**0.0402**	**0.0804**	**0.0080**	**0.0520**	0.1720

Table 8.1 $\widehat{\mathsf{AE}}_\alpha$ results for Financial crisis, Corona crisis, Post financial crisis, and Post corona crisis

In the post-crisis scenarios, the conditional value at risk estimations demonstrate greater accuracy, often resulting in risk overestimation since the models were trained with training data consisting of crisis observations. In this context, risk overestimation means that the average occurrences of conditional value at risk exceedances $\widehat{\mathsf{AE}}_\alpha$ are below the specified levels $\alpha \in \{0.01, 0.05, 0.1\}$. Notably, in the post-financial crisis scenario, the **QRNN** model's estimation of the $\widehat{\mathsf{VaR}}_{0.01,t}$ does not get exceeded by the log-return time series $Y_t^{(1)}$ during the testing period $t = T_{train} + 1, \ldots, T_{test}$. Additionally, across all models in the post-financial crisis scenario, the average occurrences of conditional value at risk exceedances are below the specified levels $\alpha \in \{0.01, 0.05, 0.1\}$. Similar trends are observed in the post-corona crisis scenario. However, all models **QRNN** and **cQRNN** and **D-vinereg** appear to overestimate the risk at levels $\alpha = 0.01$. For the levels $\alpha \in \{0.05, 0.1\}$ the models give a more accurate estimation of $\widehat{\mathsf{VaR}}_{\alpha,t}$ for the related levels. For the crisis scenarios, the models **D-vinereg** and **cQRNN** outperform the **QRNN** model in terms estimating the conditional $\widehat{\mathsf{VaR}}_{\alpha,t}$ for all $\alpha \in \{0.01, 0.05, 0.1\}$. A visualization of the model estimates of $\widehat{\mathsf{VaR}}_{\alpha,t}$ for $\alpha \in \{0.01, 0.05, 0.1\}$ can be seen in Figure 8.3 (post financial crisis), Figure 8.4 (post corona crisis), Figure 8.1 (financial crisis) and Figure 8.2 (corona crisis). Surprisingly, the **QRNN** model results exhibit **no quantile crossings** for all scenarios and levels $\alpha \in \{0.01, 0.05, 0.1\}$.

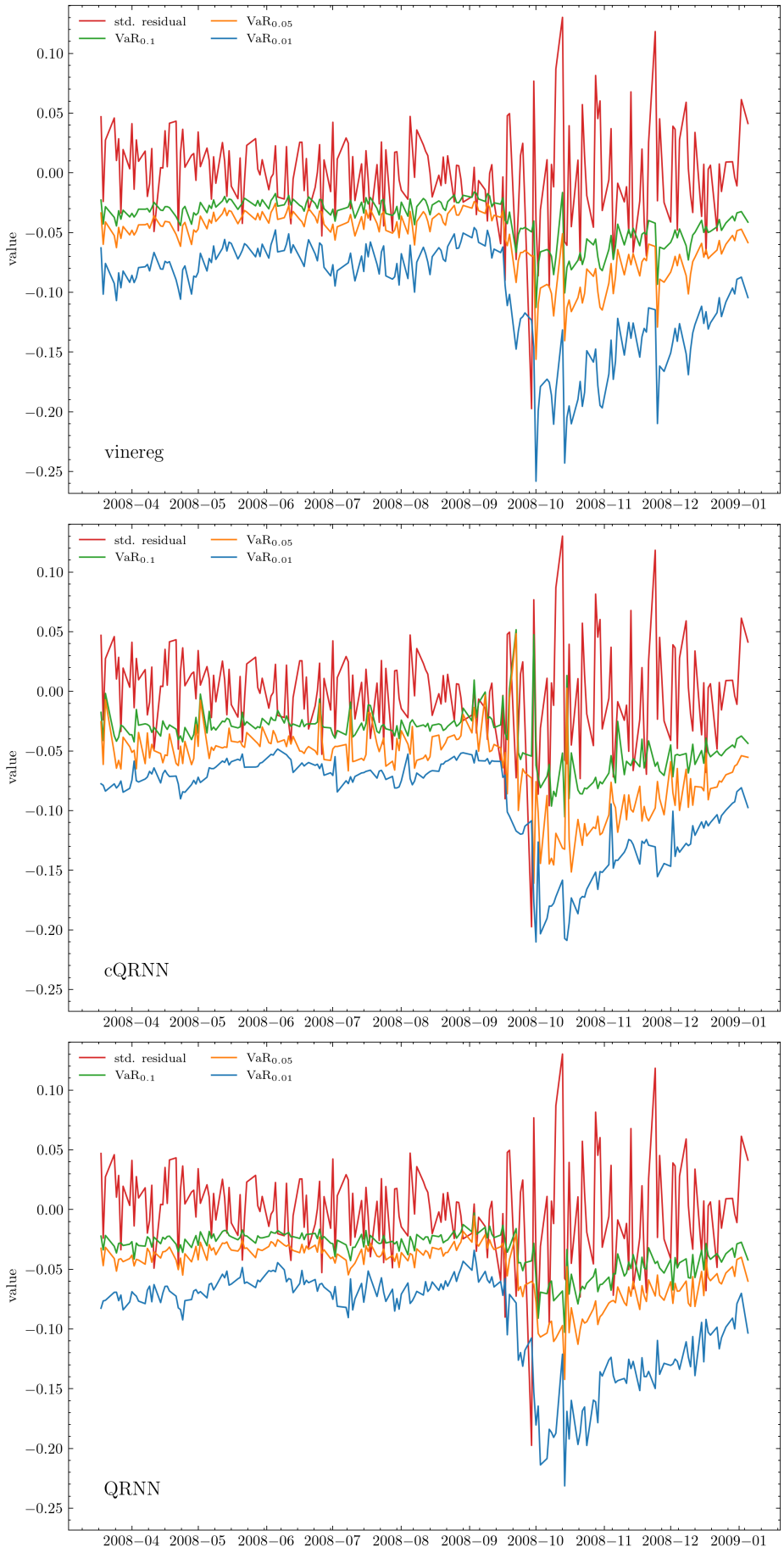


Figure 8.1 vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **financial crisis** scenario.

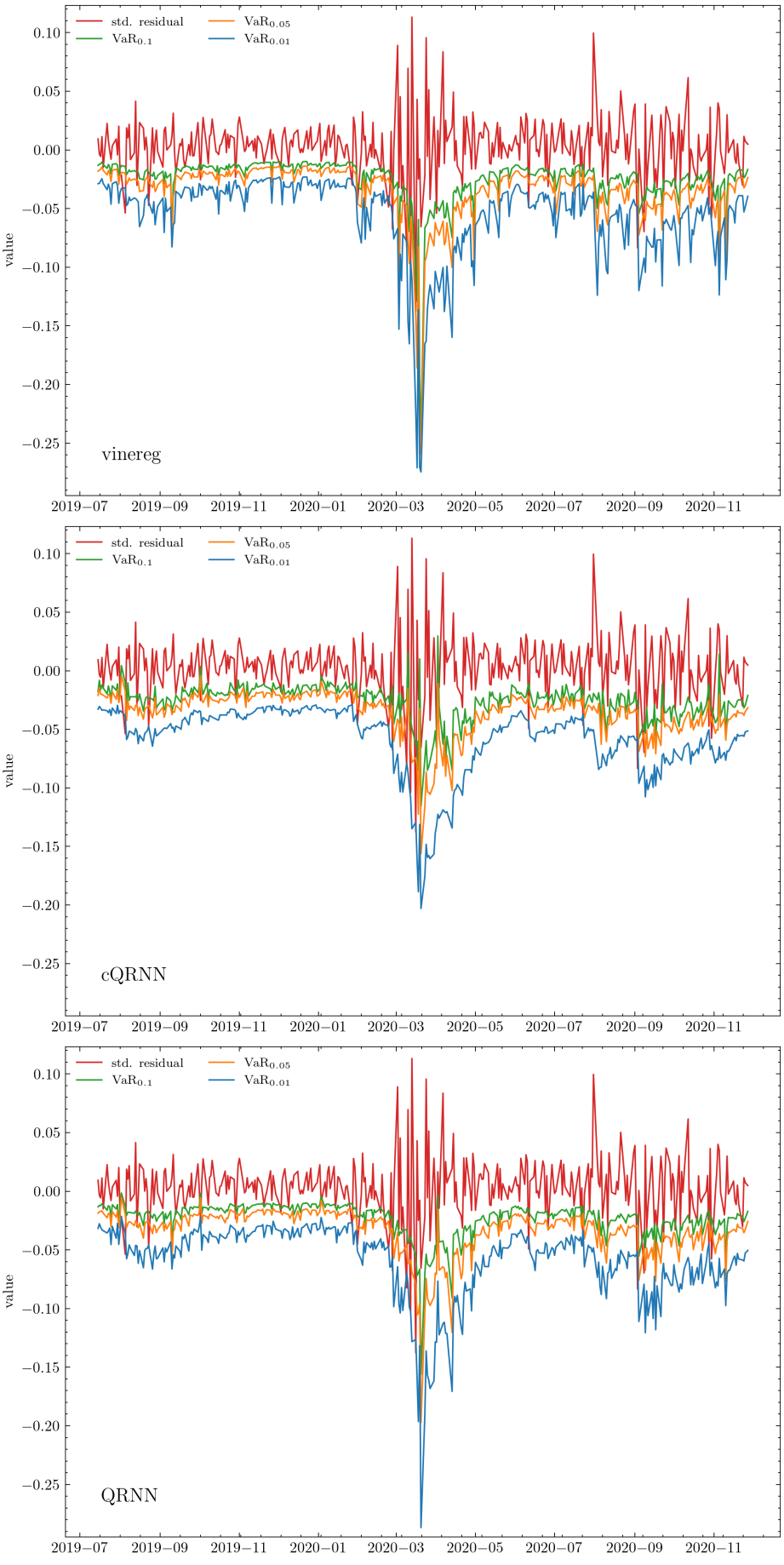


Figure 8.2 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **corona crisis** scenario.

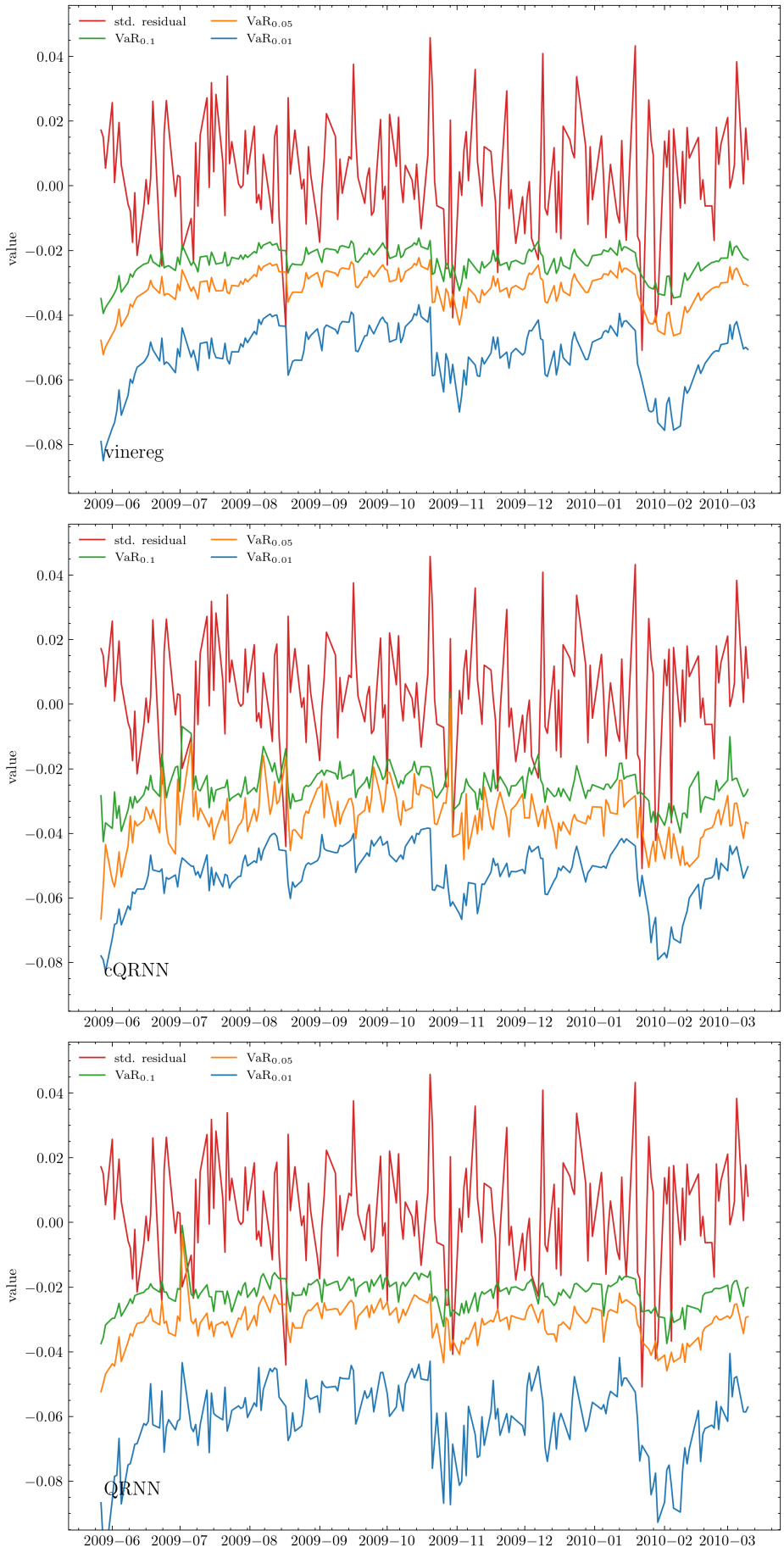


Figure 8.3 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post financial crisis** scenario.

Figure 8.4 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post corona crisis** scenario.

8.2 Stock prices as predictive variables

Next, we utilize the transformed copula data time series related to the seven stocks as the predicting variables in order to estimate the conditional value at risk. Following this approach, the training and test data set are given as :

$$D^u_{train} := \left\{(u_t^{(1)}, \mathbf{u}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = 1, \ldots, T_{train}\right\}, \tag{8.7}$$

$$D^u_{test} := \left\{(u_t^{(1)}, \mathbf{u}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = T_{train}, \ldots, T_{test}\right\}, \tag{8.8}$$

where $\mathbf{u}_{t-1} := \left(u_{t-1}^{(2)}, \ldots, u_{t-1}^{(8)}\right)$ for $t = 1, \ldots, T_{test}$. Note that we again take the shifted time series of the transformed log returns u_t^1, $a = 2, \ldots, 8$ since the returns are only known retrospectively.

	Financial crisis			Corona crisis		
	$\widehat{\text{AE}}_{0.01}$	$\widehat{\text{AE}}_{0.05}$	$\widehat{\text{AE}}_{0.1}$	$\widehat{\text{AE}}_{0.01}$	$\widehat{\text{AE}}_{0.05}$	$\widehat{\text{AE}}_{0.1}$
D-vinereg	**0.0099**	**0.0591**	0.1576	**0.0201**	0.0688	0.1089
cQRNN	0.0197	0.0640	**0.1527**	0.0229	**0.0573**	0.0888
QRNN	0.0148	0.1232	0.2365	**0.0201**	0.0659	**0.1003**
	Post financial crisis			Post corona crisis		
	$\widehat{\text{AE}}_{0.01}$	$\widehat{\text{AE}}_{0.05}$	$\widehat{\text{AE}}_{0.1}$	$\widehat{\text{AE}}_{0.01}$	$\widehat{\text{AE}}_{0.05}$	$\widehat{\text{AE}}_{0.1}$
D-vinereg	**0.0101**	0.0251	0.0653	0.0160	0.0760	0.1840
cQRNN	0.0000	0.0352	0.0704	0.0040	**0.0560**	**0.1800**
QRNN	0.0050	**0.0402**	**0.0804**	**0.0080**	0.0600	**0.1800**

Table 8.2 $\widehat{\text{AE}}_\alpha$ results for Financial crisis, Corona crisis, Post financial crisis, and Post corona crisis, where post stock prices of the remaining seven stocks are used as predictive variables.

We observe a performance improvement of all models during the **financial- and corona crisis** scenarios compared to the Fama and French five-factor model. However, in the **post crisis scenarios**, models that use stock prices as predictive variables perform worse on average with respect to the average number of exceedances $\widehat{\text{AE}}_\alpha$ compared to models that use the Fama and French factors. Similar to the results in the Fama and French setting, the **cQRNN** model's estimation of $\widehat{\text{VaR}}_{0.01,t}$ is not exceeded by the log-return time series $Y_t^{(1)}$ during the testing period $t = T_{train}+1, \ldots, T_{test}$ in the post-financial crisis scenario. Based on our results for $\widehat{\text{AE}}_\alpha$ with $\alpha \in \{0.01, 0.05, 0.1\}$ and using stocks as the predicting variables, we recommend using the **D-vinereg** model for crisis scenarios and the **QRNN** approach for post-crisis scenarios. A visualization of the model estimates of $\widehat{\text{VaR}}_{\alpha,t}$ for $\alpha \in \{0.01, 0.05, 0.1\}$ can be seen in Figure 8.5 (financial crisis), Figure 8.6 (corona crisis), Figure 8.7 (post financial crisis) and Figure 8.8 (post corona crisis). We add that the **QRNN** model results show **zero quantile crossings** for all scenarios and levels $\alpha \in \{0.01, 0.05, 0.1\}$.

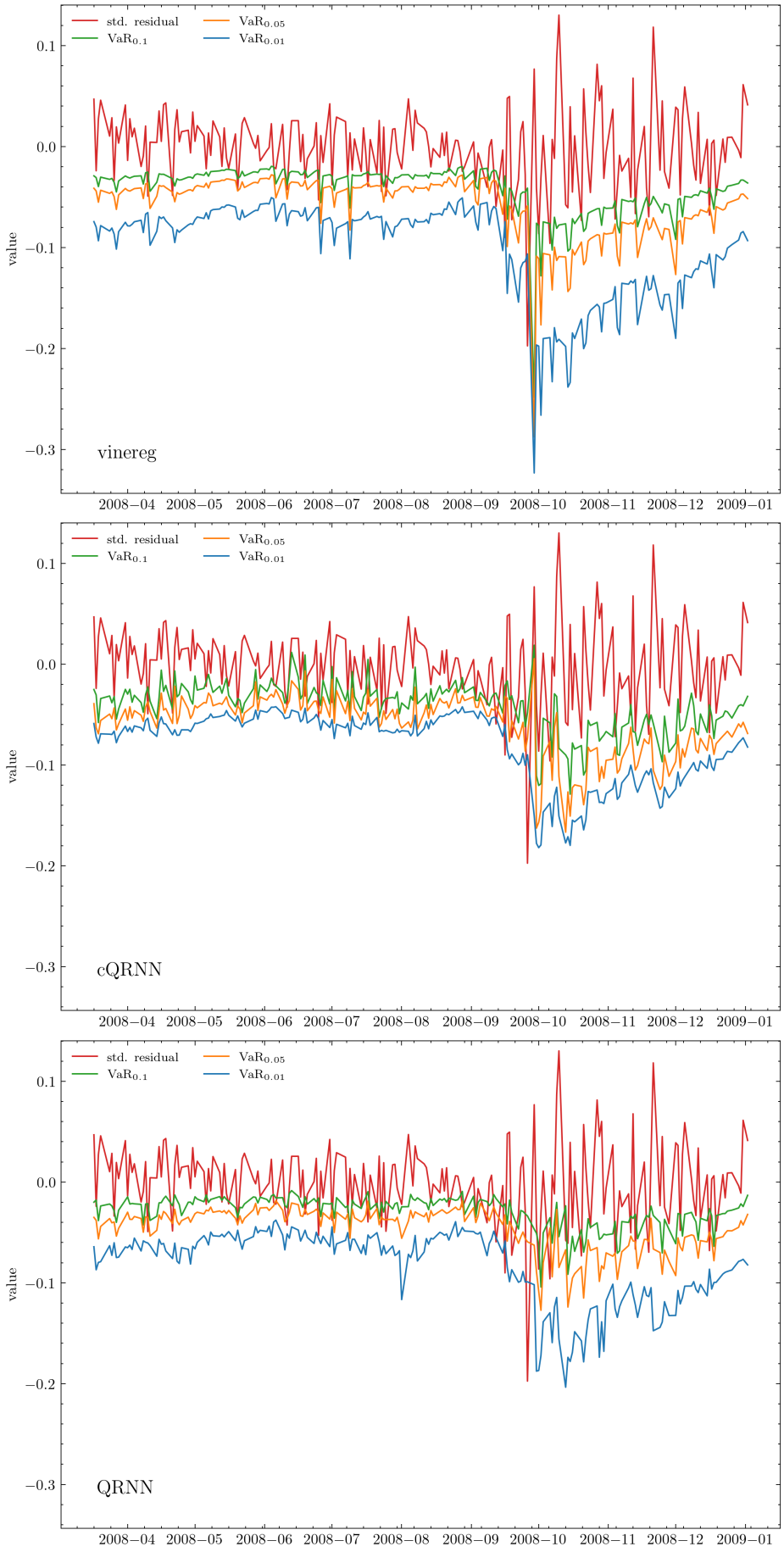


Figure 8.5 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **financial crisis** scenario with historical stock prices as predictors.

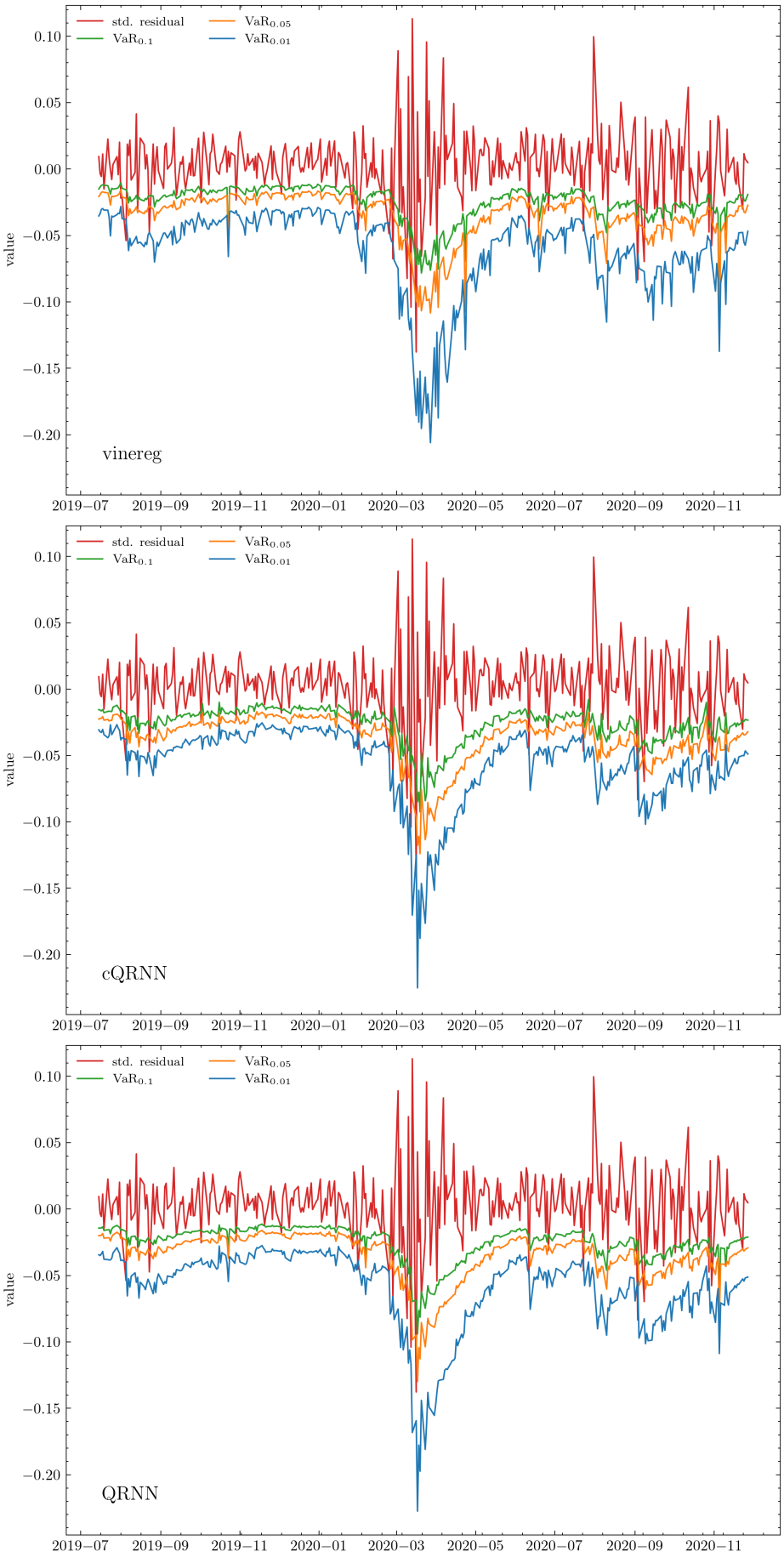


Figure 8.6 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **corona crisis** scenario with historical stock prices as predictors.

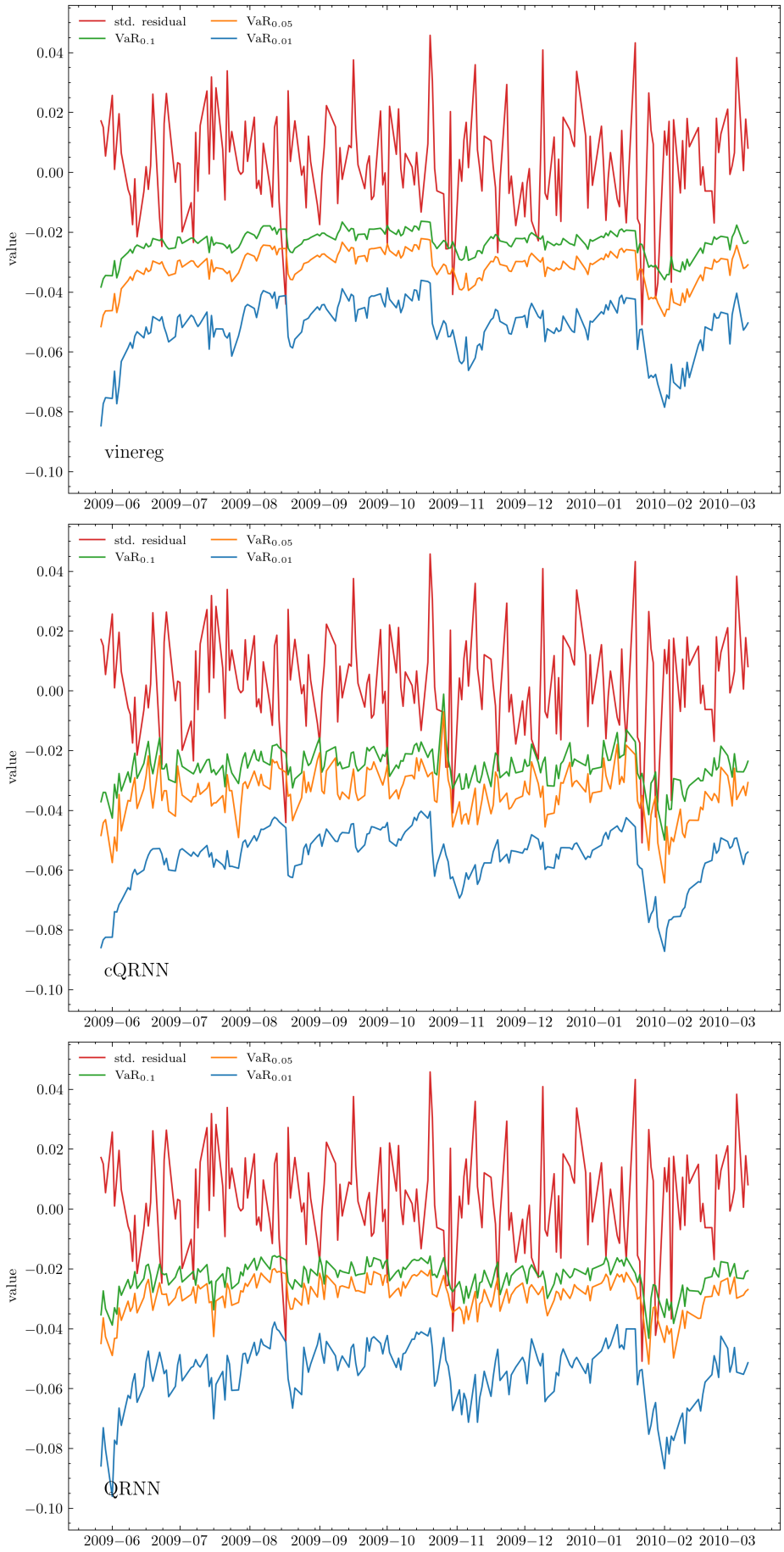


Figure 8.7 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post financial crisis** scenario with historical stock prices as predictors.

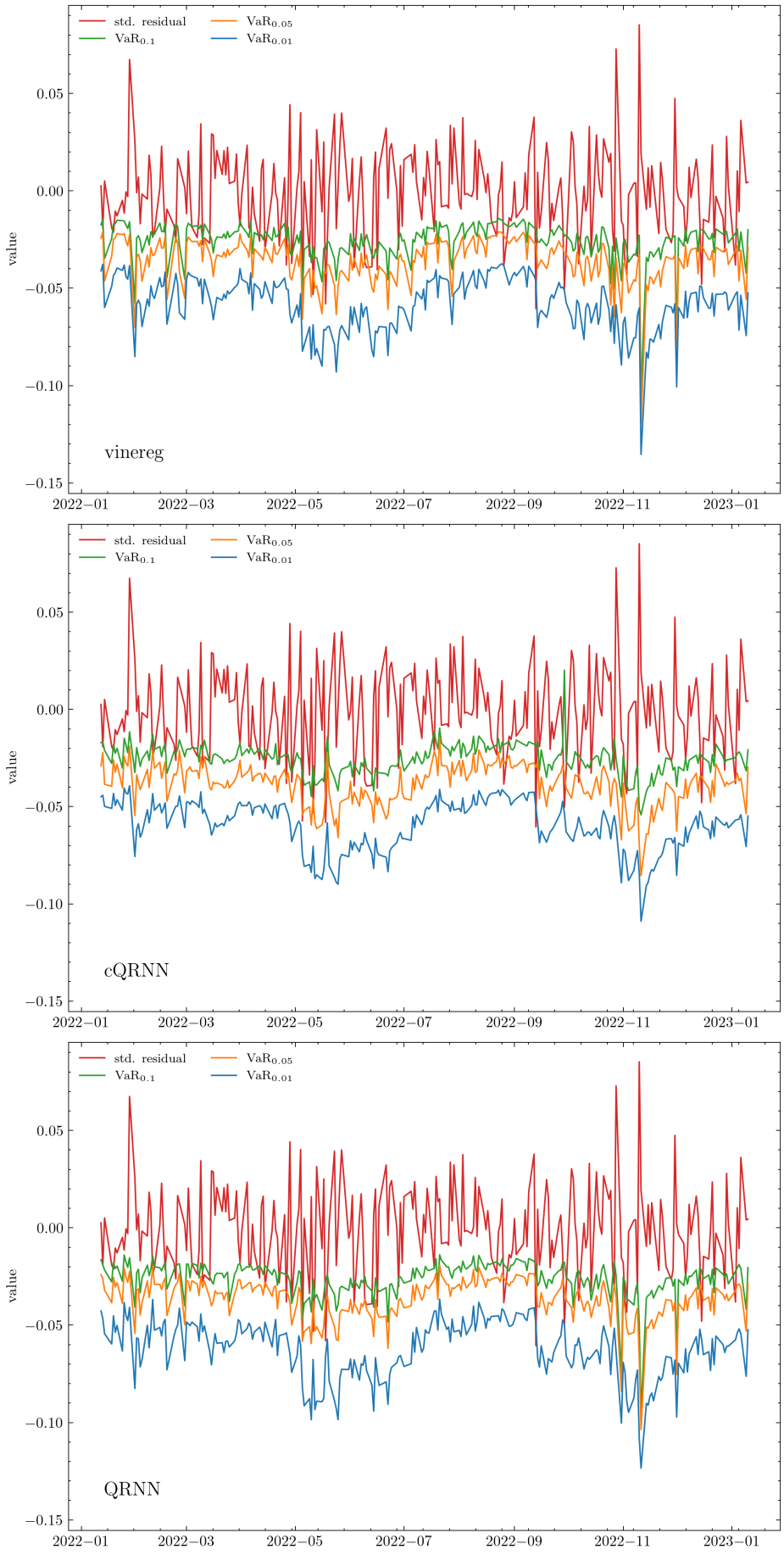


Figure 8.8 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post corona crisis** scenario with historical stock prices as predictors.

8.3 Stock prices and Fama and French factors as predictive variables

Furthermore, to improve our estimates of the conditional value at risk estimates across all scenarios and levels, we incorporate the transformed copula data time series related to the stocks into the predicting variables, resulting in the following training and test set:

$$D^u_{train} := \left\{(u^{(1)}_t, \mathbf{u}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = 1, \ldots, T_{train}\right\}, \tag{8.9}$$

$$D^u_{test} := \left\{(u^{(1)}_t, \mathbf{u}_{t-1}) \in \mathbb{R} \times \mathbb{R}^5 | t = T_{train}, \ldots, T_{test}\right\}, \tag{8.10}$$

where $\mathbf{u}_{t-1} := \left(u^{(2)}_{t-1}, \ldots, u^{(8)}_{t-1}, u^{(1),\mathsf{FF}}_{t-1}, \ldots, u^{(5),\mathsf{FF}}_{t-1}\right)$ for $t = 1, \ldots, T_{test}$. Note that we also take the shifted time series of the transformed log returns u^a_t, $a = 2, \ldots, 8$ since the returns are only known retrospectively.

	Financial crisis scenario			Corona crisis scenario		
	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$
D-vinereg	0.0246	0.0690	**0.1478**	**0.0287**	0.0716	**0.1232**
cQRNN	0.0296	**0.0443**	**0.1478**	**0.0287**	0.0774	0.1547
QRNN	**0.0197**	0.1379	0.1773	**0.0287**	**0.0659**	0.1289
	Post financial crisis scenario			Post corona crisis		
	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$	$\widehat{\mathsf{AE}}_{0.01}$	$\widehat{\mathsf{AE}}_{0.05}$	$\widehat{\mathsf{AE}}_{0.1}$
D-vinereg	**0.0050**	0.0352	0.0754	**0.0080**	0.0720	0.1920
cQRNN	0.0000	**0.0553**	0.0754	0.0040	**0.0680**	**0.1480**
QRNN	**0.0050**	0.0402	**0.0804**	0.0160	0.0760	0.1960

Table 8.3 $\widehat{\mathsf{AE}}_\alpha$ results for Financial crisis scenario, Corona crisis scenario, Post financial crisis scenario, and Post corona crisis

Overall, including the transformed stock prices of the assets as predicting variables does not enhance the models performance with regard to the average number of exceedances $\widehat{\mathsf{AE}}_\alpha$ for all levels $\alpha \in \{0.01, 0.05, 0.1\}$ (see Table 8.3). Again the results are visualized and can be seen in Figure 8.9 (financial crisis), Figure 8.10 (corona crisis), Figure 8.11 (post financial crisis) and Figure 8.12 (post corona crisis). We note that the **QRNN** model results show **zero quantile crossings** for all scenarios and levels $\alpha \in \{0.01, 0.05, 0.1\}$.

8.3.1 Real data example results

In summary, using the five Fama and French factors as predictive variables for the models **D-vinereg**, **QRNN**, and **cQRNN** is more effective for post-crisis scenarios (see Table 8.1). For crisis scenarios, utilizing stock prices as predictive variables is a better approach for estimating the conditional Value at Risk (see Table 8.2). This is because stock prices are highly correlated during turbulent times, as Tabash et al. (2024) notes, with stocks often driven by broad events like financial or corona crises. It is important to consider which stock prices are used as predictive variables, as some sectors may not be equally impacted by crisis events. Combining the five Fama and French factors with past stock prices does not improve the model performance with respect to the average number of exceedances (see Table 8.3).

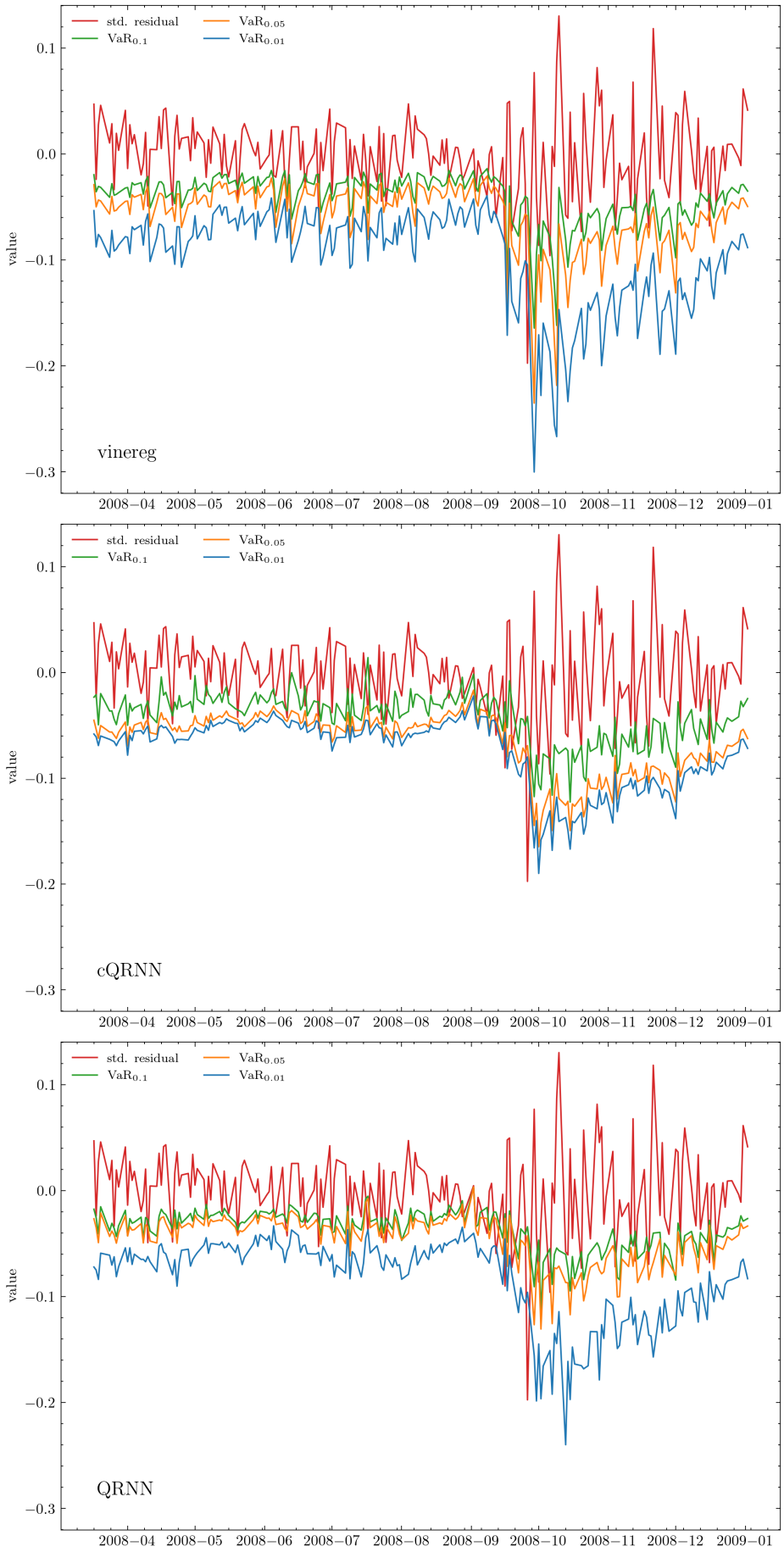


Figure 8.9 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **financial crisis** scenario with including historical stock prices as predictors.

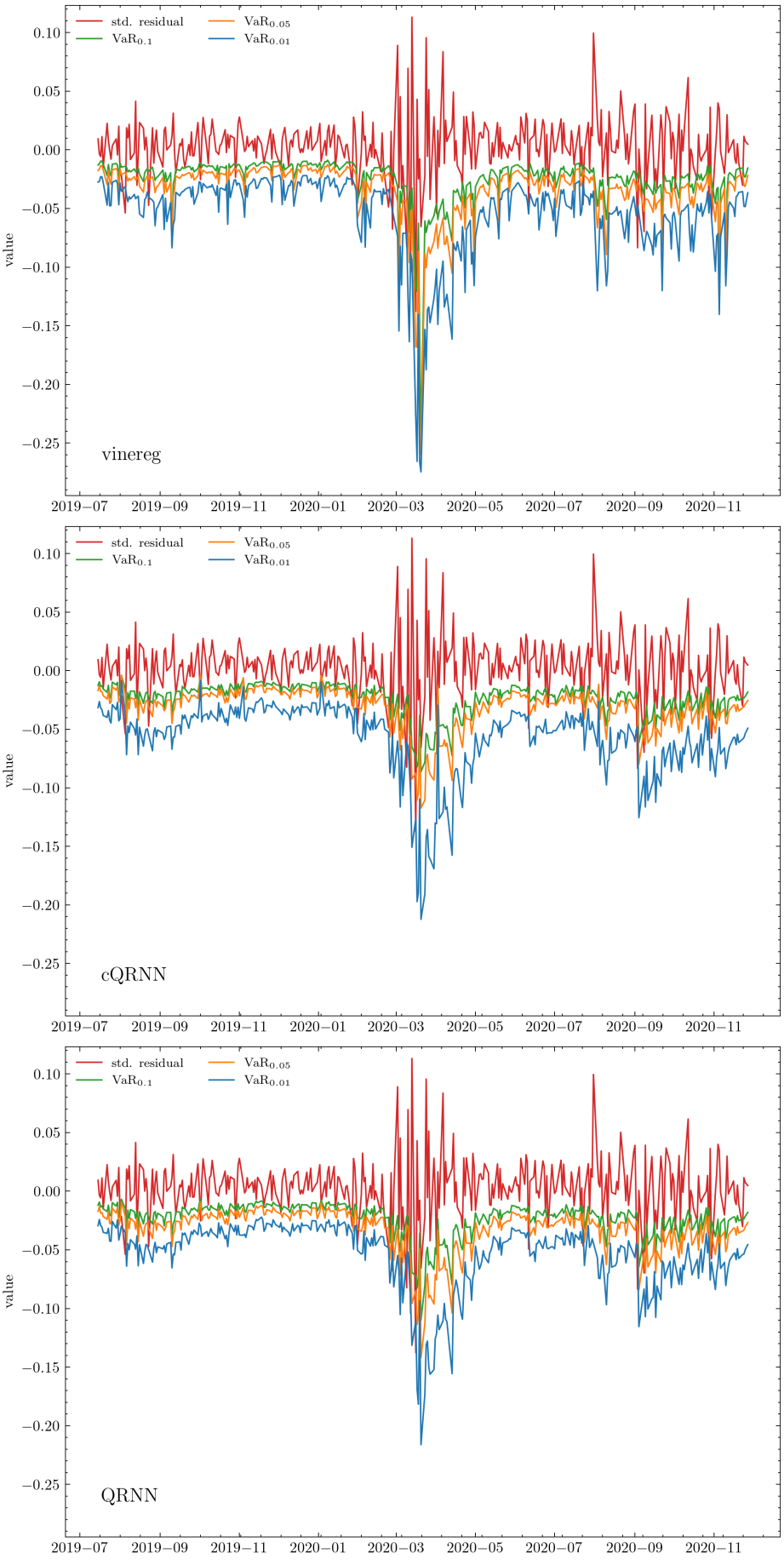


Figure 8.10 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **corona crisis** scenario with including historical stock prices as predictors.

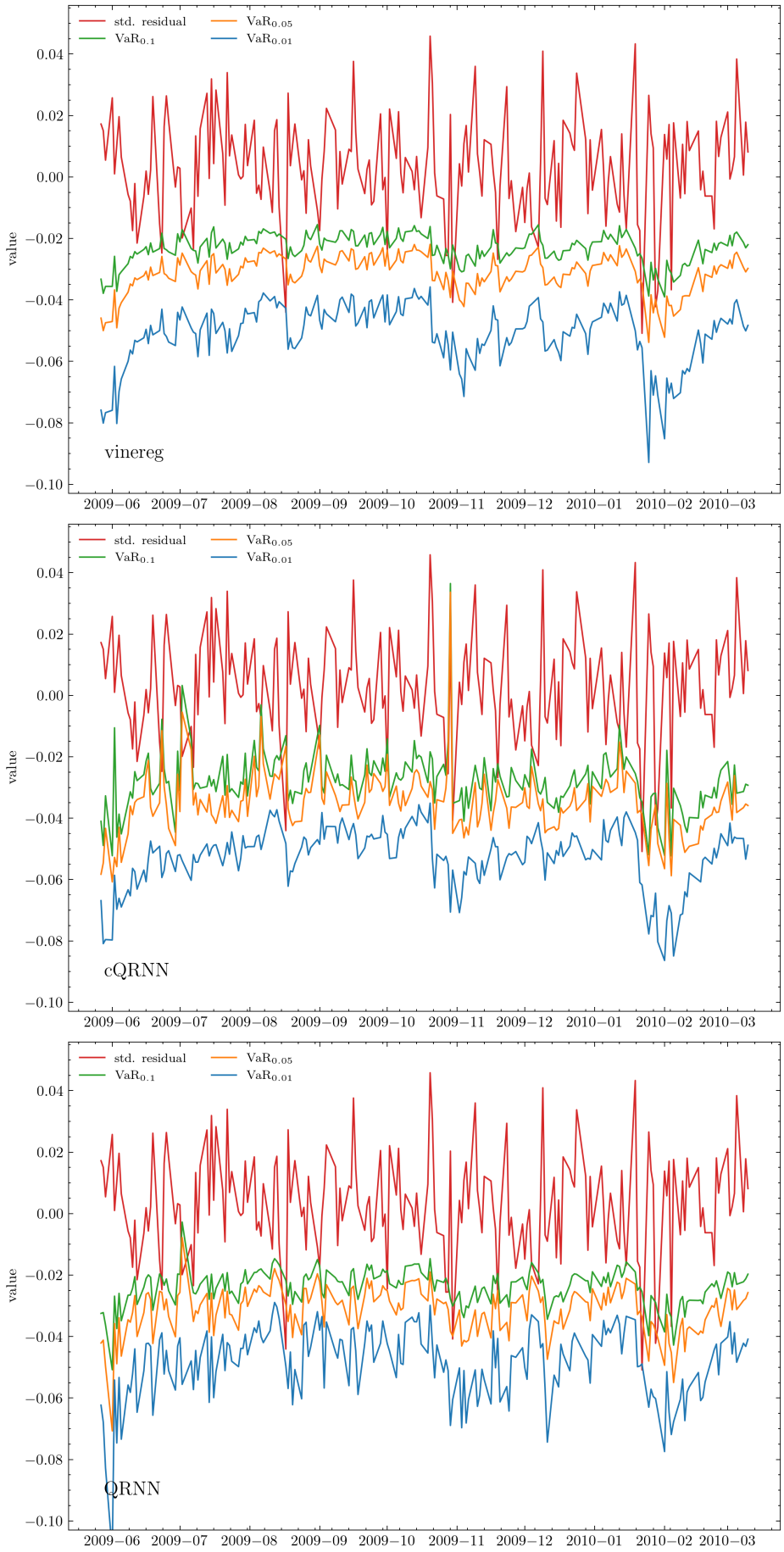


Figure 8.11 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post financial crisis** scenario with including historical stock prices as predictors.

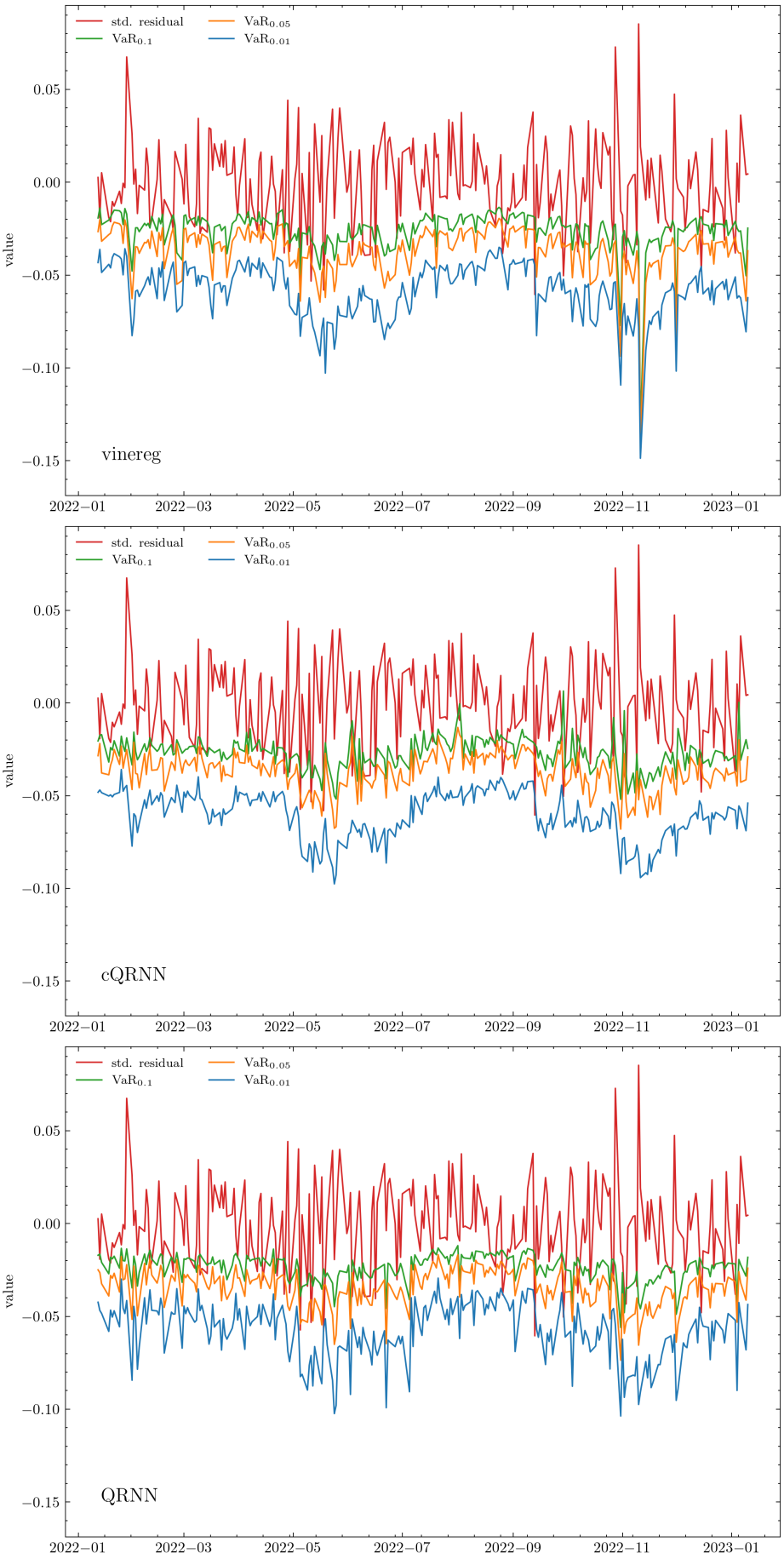


Figure 8.12 D-vinereg, cQRNN and QRNN model estimates of the conditional Value at Risk at levels $\alpha \in \{0.01, 0.05, 0.1\}$ for the **post corona crisis** scenario with including historical stock prices as predictors.

9 Conclusion and outlook

We studied and introduced new approaches to estimate the conditional quantile of a response variable given a set of predictive variables. The **cQRNN** shows good performance in both the simulation and real world example while it excludes the existence of quantile crossing.

Moreover, we modified the approach of Tagasovska, Ozdemir, and Brando (2023) regarding the uncertainty analysis of neural networks in a regression setting. Note that this approach is faster due to the nature of the forward algorithm of the sparse D-vine quantile regression. The fitting of the prediction interval, however, is only possible for classical feed forward neural network architectures as we have introduced them in Definition (3.2.1). The developed algorithm can be applied to models such as the **QRNN** (see Section 5) and **redQRNN** (see Section 5.2) to obtain prediction intervals for the estimated quantiles. For the **cQRNN** model, which is an accumulation of **quasi-quantile neural networks** (see Section 5.1), the algorithm for obtaining prediction intervals is not applicable due to the different structure of the network compared to standard quantile regression neural networks. Further work could be done to adapt the approach proposed by Tagasovska, Ozdemir, and Brando (2023) such that the algorithm can be applied for cumulative neural networks, and thereby extending its applicability to the **cQRNN** model. This could be achieved by fitting a prediction interval with retraining (see Section 5.3 for the procedure) to each **quasi-quantile regression neural network** within the cQRNN. The resulting prediction intervals can then be added up to obtain a simultaneous prediction interval, similar to the simultaneous prediction interval (5.22).

In the simulation study (Section 6), it became evident that in scenarios featuring a large number of input variables, the neural network models **QRNN**, **redQRNN**, and **cQRNN** significantly outperform both **D-vine quantile regression** and **sparse D-vine quantile regression** in terms of computational time. Remarkably, despite the shorter computational times, these neural network models exhibit comparable or even better accuracy compared to the quantile regression models based on D-vine copulas. We observed that the **QRNN** and **redQRNN** approaches exhibit quantile crossings in approximately 10%-15% of the estimated conditional quantiles in the test set. Therefore, we prefer the **cQRNN** model which eliminates the phenomenon of quantile crossings per definition. The short computing time of the quantile neural network approaches gives rise to the idea of utilizing a version of the **QRNN** or **cQRNN** model as a benchmark model for the forward algorithm of the **D-vine quantile regression** and therefore truncate the resulting D-vine after a certain accuracy threshold is reached. This is only an idea and needs further investigation and theory.

Another intriguing idea of combining vine copula theory with neural network theory involves utilizing the fitted copula as a synthetic data generator.The basic idea of using vine copulas as a generator of synthetic data for classification was already studied in the master's thesis of Griesbauer (2022), as well as in the paper by Tagasovska, Ackerer, and Vatter (2019), where the generating property of copulas was used in the context of autoencoders. In our setting, the fitted D-vine copula could be used to generate synthetic data in a real world problem in order to improve the accuracy of the various quantile regression neural network models **QRNN**, **cQRNN** and **redQRNN**.

Following the example we presented in Section 4.3, one can further investigate the approximation of D-vine copulas using neural networks. Approaches proposed by Sun, Cuesta-Infante, and Veeramachaneni (2018) and Zeng and Wang (2022) have shown promising initial results. However, the resulting structures of both approaches do not necessarily guarantee a vine structure with the properties studied in Section 4.2.

Glossary

$\mathcal{K}^{a,L}_{(d,k_1,\ldots,k_H,s)}$

Set of neural networks with input dimension d, output dimension s and H hidden layers where k_h is the number of neurons in the h-th hidden layer, $h = 1, \ldots, H$. The neural networks are equiped with an activation function $a : \mathbb{R} \to \mathbb{R}$ for each hidden layer and a loss function $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. . 12, 80

$\mathcal{K}_{(d,k_1,\ldots,k_H,s)}$

Abbreviation of $\mathcal{K}^{a,L}_{(d,k_1,\ldots,k_H,s)}$. If it is not stated otherwise we use the sigmoid function as the activation function and the L^2 loss as the loss function. . 12, 80

$\Phi^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)}$

Neural network with paramtervector $\boldsymbol{\theta}$, input dimension d, output dimension s and H hidden layers where k_h is the number of neurons in the h-th hidden layer, $h = 1, \ldots, H$. The neural network is equiped with an activation function $a : \mathbb{R} \to \mathbb{R}$ for each hidden layer and a loss function $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. 12, 80

Φ

Abbreviation of $\Phi^{\boldsymbol{\theta},a,L}_{(d,k_1,\ldots,k_H,s)}$. If it is not stated otherwise we use the sigmoid function as the activation function and the L^2-loss as the loss function and omit the parameters a, L. To furhter shorten the notation we often omit the parameter vector $\boldsymbol{\theta}$, and the vector $(d, k_1, \ldots, k_H, s)$. We indicate the dimensions of the neural net by $\Phi \in \mathcal{K}^{a,L}_{(d,k_1,\ldots,k_H,s)}$. 12, 80

$\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}$

Quantile regression neural network with parameter vector $\boldsymbol{\theta}$, estimating quantiles at the levels specified by $\boldsymbol{\alpha} \in (0,1)^K$, $K \in \mathbb{N}$. The network has input dimension d, output dimension $|\boldsymbol{\alpha}| = K$, where $|\cdot|$ is the cardinality function. The neural network is equiped with an activation function $a : \mathbb{R} \to \mathbb{R}$ and posesses H hidden layers where k_h is the number of neurons in the h-th hidden layer, $h = 1, \ldots, H$. The loss function is the pinball loss $\rho_{\boldsymbol{\alpha}}$. 80

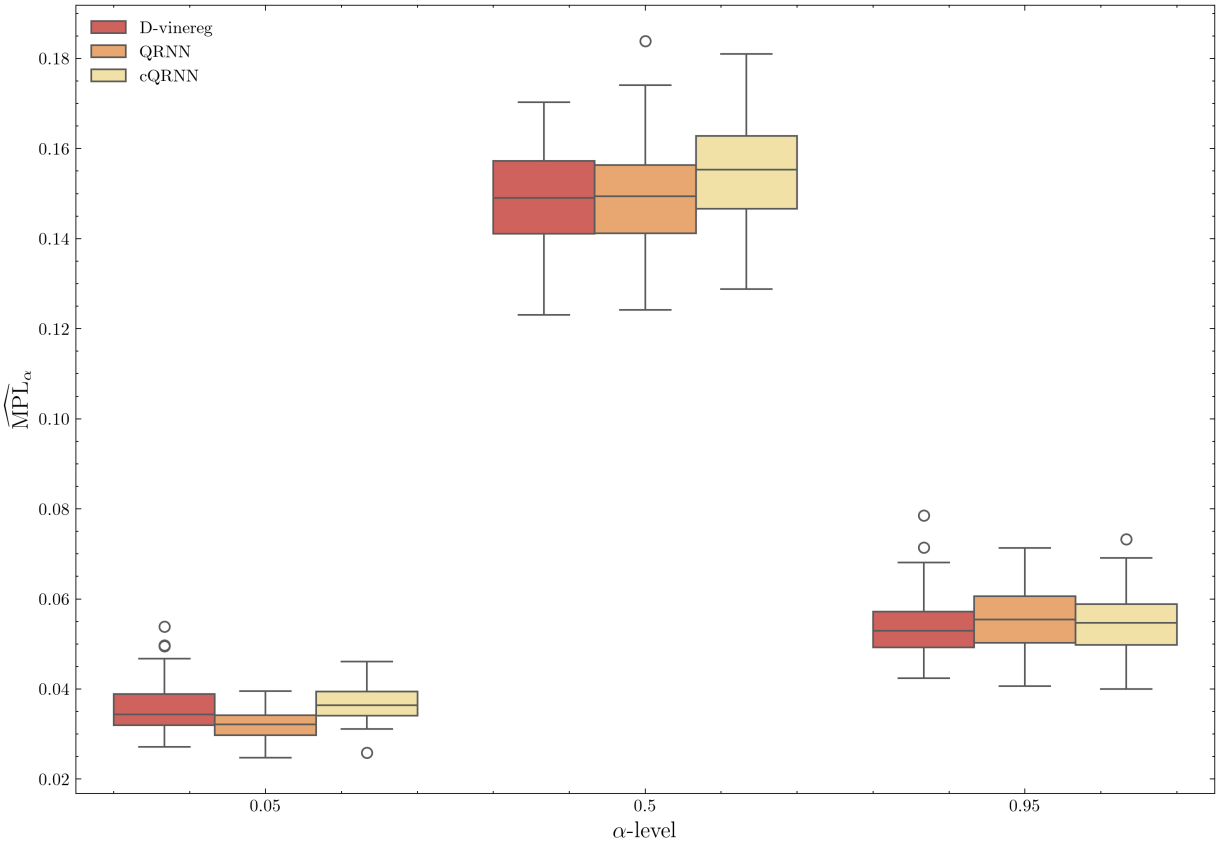
$\Phi^{\rho_{\boldsymbol{\alpha}}}$

Abbreviation of $\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}}}_{(d,k_1,\ldots,k_H,K)}$.To simplify the notation, we exclude the parameter vector $\boldsymbol{\theta}$ and vector $(d, k_1, \ldots, k_H, K)$. We also assume that the activation function is the sigmoid function, unless stated otherwise, resulting in the shortened notation $\Phi_{\boldsymbol{\alpha}}$. The dimensions of the quantile regression neural network are indicated with the set, $\Phi^{\boldsymbol{\alpha}} \in \mathcal{K}^{a,\rho_\alpha}_{(d,k_1,\ldots,k_H,K)}$. 80

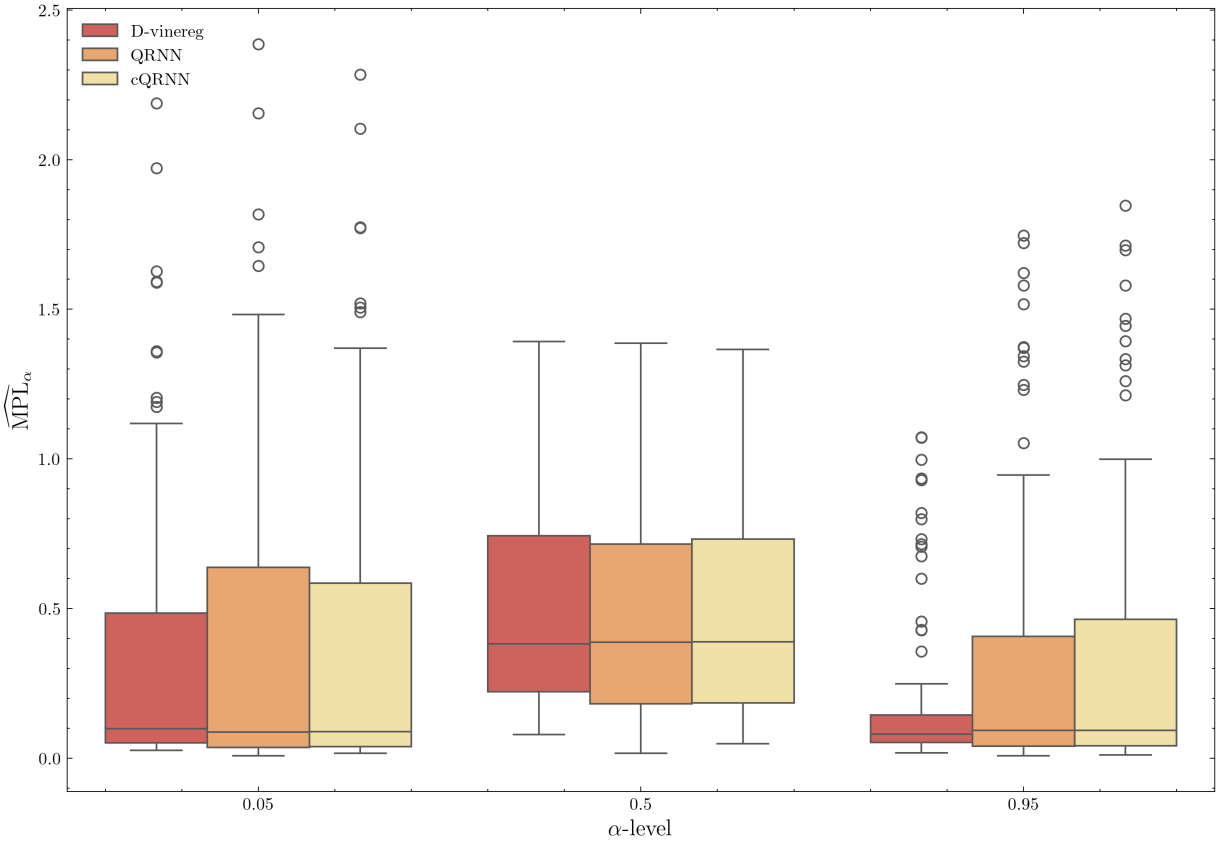
$\Phi^{\boldsymbol{\theta},a,\rho_{\boldsymbol{\alpha}},\text{red}}_{(k,k_1,\ldots,k_L,K)}$

Quantile regression neural network with reduced input dimension $1 \leq k \leq d$. The reduction is done by the forward selection algorithm which constructs a D-vine using the conditional log liklihood maximising covariate sequence and possibly obmits covariates. We use the short notation $\Phi^{\rho_{\boldsymbol{\alpha}},\text{red}}$. 80

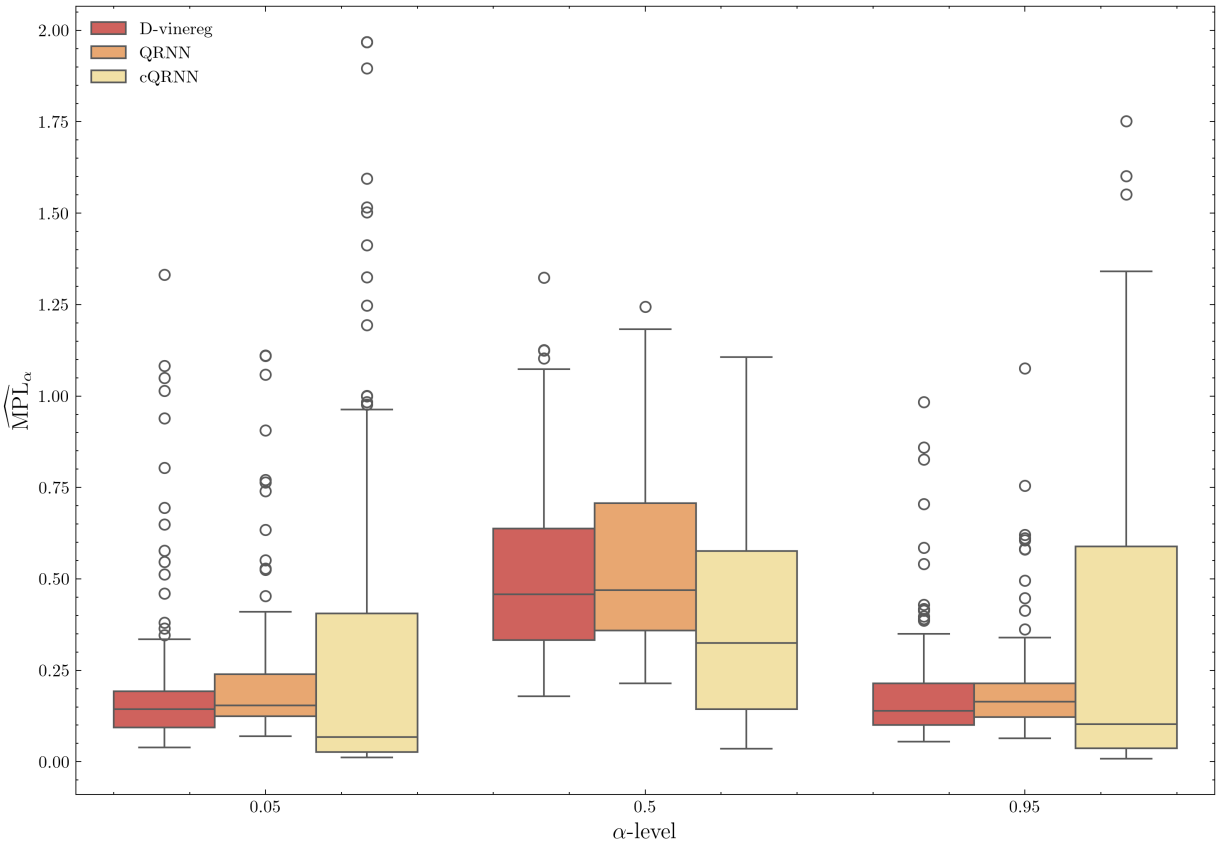
A Appendix



(a) Scenario D3

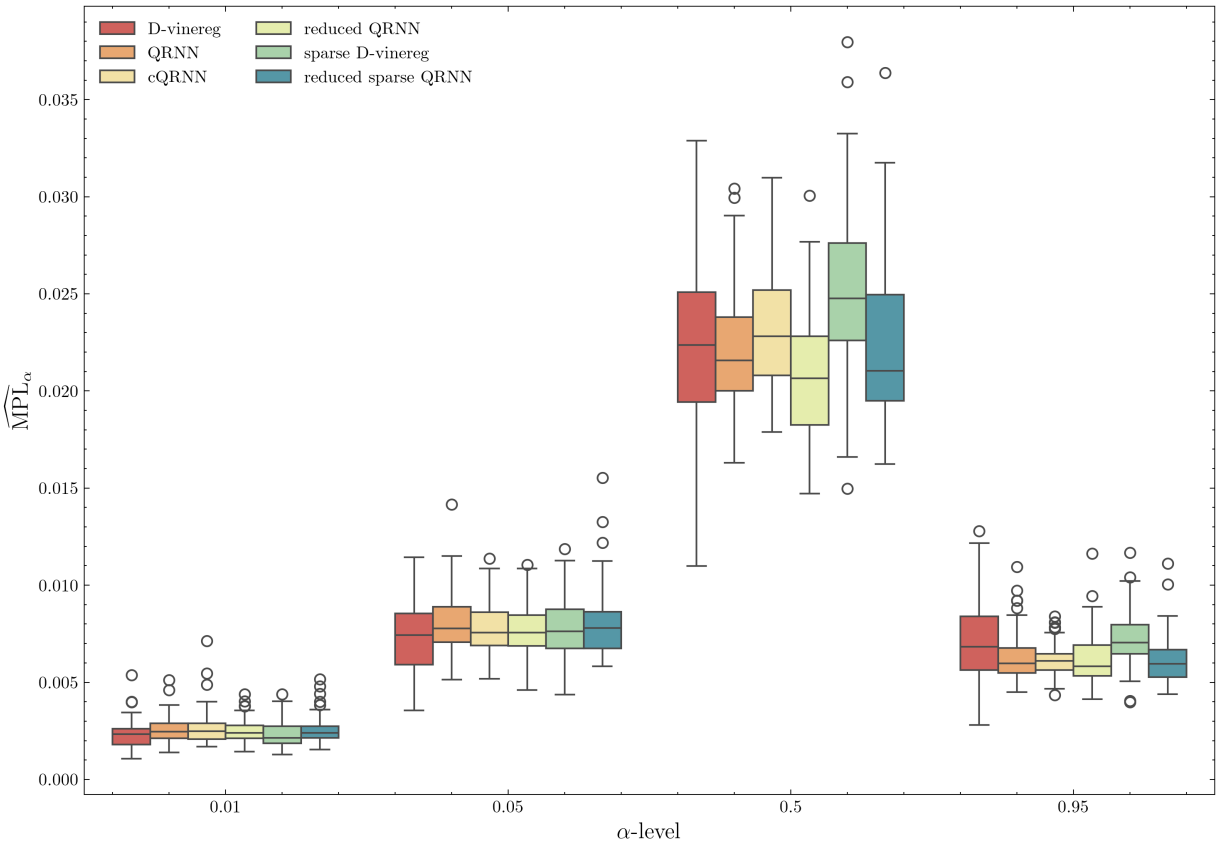


(b) Scenario NH4

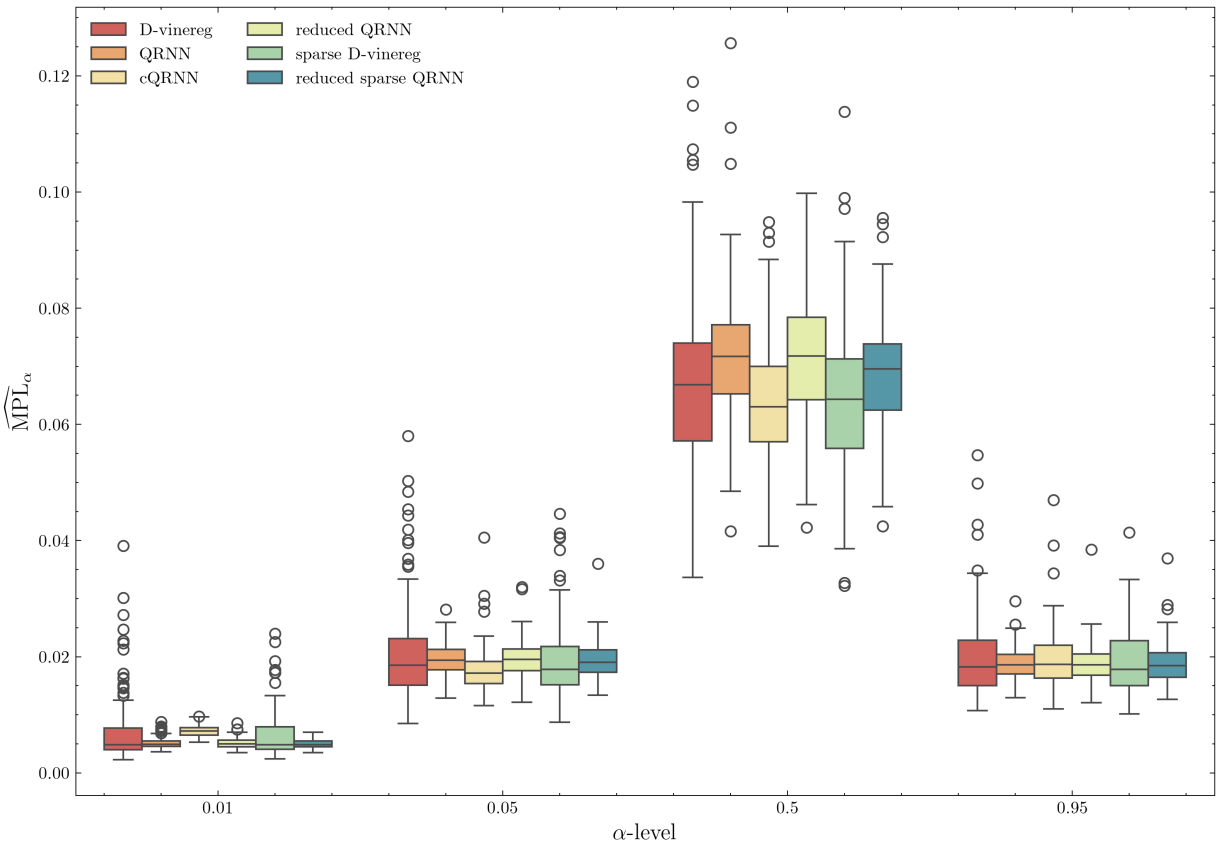


(c) Scenario LH4

Figure A.1 Boxplots of $\widehat{\text{MPL}}_{m,\alpha}$, $\alpha \in \{0.05, 0.5, 0.95\}$ for scenario D3, NH4 and LH4 with small sample training data size $n_{train} = 400$ and $m \in \{$D-vinereg, QRNN, cQRNN $\}$.



(a) Scenario D6



(b) Scenario G20

Figure A.2 Boxplots of $\widehat{\text{MPL}}_\alpha$, $\alpha \in \{0.01, 0.05, 0.5, 0.95\}$ for scenarios D6 and G20 with small dimensional training data $n_{train} = 400$.

Bibliography

Baldi, Pierre, and Peter Sadowski. 2014. The dropout learning algorithm. *Artificial Intelligence* 210:78–122. ISSN: 0004-3702. https://doi.org/https://doi.org/10.1016/j.artint.2014.02.004. https://www.sciencedirect.com/science/article/pii/S0004370214000216.

Bedford, T., and R. M. Cooke. 2002. Vines: a new graphical model for dependent random variables. *Annals of Statistics* 30:1031–1068.

Bernard, Carole, and Claudia Czado. 2015. Conditional quantiles and tail dependence [in English]. Publisher Copyright: © 2015 Elsevier Inc. *Journal of Multivariate Analysis* 138 (June): 104–126. ISSN: 0047-259X. https://doi.org/10.1016/j.jmva.2015.01.011.

Cannon, Alex J. 2011. Quantile regression neural networks: implementation in r and application to precipitation downscaling. *Computers Geosciences* 37 (9): 1277–1284. ISSN: 0098-3004. https://doi.org/https://doi.org/10.1016/j.cageo.2010.07.005. https://www.sciencedirect.com/science/article/pii/S009830041000292X.

Cont, Rama. 2001. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance* 1:223–236.

Czado, C. 2010. Pair-copula constructions of multivariate copulas. In *Copula theory and its applications,* 93–109. Springer.

———. 2019. *Analyzing dependent data with vine copulas: A practical guide with r.* Springer Cham.

Demuth, Howard B., Mark H. Beale, Orlando De Jesús, and Martin T. Hagan. 2014. *Neural network design.* Martin Hagan.

Dissmann, J., E. C. Brechmann, C. Czado, and D. Kurowicka. 2013. Selecting and estimating regular vine copulae and application to financial returns. *Computational Statistics Data Analysis* 59:52–69.

Fama, E., and K. French. 1993. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics* 33:3–56.

———. 2014. A five-factor asset pricing model. *Fama-Miller Working Paper* (September). https://ssrn.com/abstract=2287202.

Funahashi, Ken-ichi. 1989. On the approximate realization of continuous mappings by neural networks. *Neural Networks* 2 (3): 183–192.

Griesbauer, Elisabeth. 2022. Vine copula based synthetic data generation for classification [in en]. Master's thesis, Technische Universität München, July.

Grønneberg, Tilmann, and Adrian E. Raftery. 2007. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association* 102 (477): 359–378.

Gühring, Ingo, Mones Raslan, and Gitta Kutyniok. 2020. *Expressivity of deep neural networks.* arXiv: 2007.04759 [cs.LG].

Hanin, Boris. 2019. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics* 7 (10).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2016. *The elements of statistical learning.* 2nd. Berlin: Springer.

Hinton, G.E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*

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

Joe, Harry. 1997. *Multivariate models and multivariate dependence concepts.* 1st ed. CRC Press.

Killiches, M., D. Kraus, and C. Czado. 2016. Model distances for vine copulas in high dimensions. ArXiv preprint arXiv:1510.03671.

Koenker, Roger, and Gilbert Bassett Jr. 1978. Regression quantiles. *Econometrica: journal of the Econometric Society,* 33–50.

Kraus, Daniel, and Claudia Czado. 2017. D-vine copula based quantile regression. *Computational Statistics Data Analysis* 110:1–18. ISSN: 0167-9473. https://doi.org/https://doi.org/10.1016/j.csda.2016.12.009. https://www.sciencedirect.com/science/article/pii/S0167947316303073.

Makariou, D, P Barrieu, and G Tzougas. 2021. A finite mixture modelling perspective for combining experts' opinions with an application to quantile-based risk measures. *Risks* 9 (6): 115. https://doi.org/10.3390/risks9060115.

Mhaskar, Hrushikesh, and Tomaso Poggio. 2016. *Deep vs. shallow networks : an approximation theory perspective.* arXiv: 1608.03287 [cs.LG].

Nagler, T., and T. Vatter. 2022. *Kde1d: univariate kernel density estimation.* R package version 1.0.4.

Netrapalli, Praneeth. 2019. Stochastic gradient descent and its variants in machine learning. *Journal of the Indian Institute of Science* 99 (2): 201–213.

Ng, Andrew Y. 2004. *Feature selection, l1 vs. l2 regularization, and rotational invariance.* ICML, Banff, AB, Canada, July.

Parzen, E. 1962. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics* 33:1065–1076.

Prechelt, L. 2012. Early stopping — but when? In *Neural networks: tricks of the trade,* edited by G. Montavon, G.B. Orr, and K.R. Müller, vol. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-35289-8_5.

R Core Team. 2021. *R: a language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Robbins, Herbert, and Sutton Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics* 22 (3): 400–407. https://doi.org/10.1214/aoms/1177729586.

Rosenblatt, M. 1952. Remarks on multivariate transformation. *Annals of Mathematical Statistics* 23:1052–1057.

Rüschendorf, L. 1981. Stochastically ordered distributions and monotonicity of the oc-function of sequential probability ratio tests. *Statistics: A Journal of Theoretical and Applied Statistics* 12 (3): 327–338.

Sahin, O., and C. Czado. 2022. High-dimensional sparse vine copula regression with application to genomic prediction. *arXiv preprint arXiv:2208.12383.*

Shumway, Robert H, and David S Stoffer. 2017. *Time series analysis and its applications: with r examples.* Springer.

Sommer, E., K. Bax, and C. Czado. 2023. Vine copula based portfolio level conditional risk measure forecasting. *Econometrics and Statistics.*

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15 (1): 1929–1958.

Sun, Yi, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2018. *Learning vine copula models for synthetic data generation.* arXiv: 1812.01226 [cs.LG].

Tabash, Mosab I., Neenu Chalissery, T. Mohamed Nishad, and Mujeeb Saif Mohsen Al-Absy. 2024. Market shocks and stock volatility: evidence from emerging and developed markets. *International Journal of Financial Studies* 12 (1): 2. https://doi.org/https://doi.org/10.3390/ijfs12010002.

Tagasovska, Natasa, Damien Ackerer, and Thibault Vatter. 2019. Copulas as high-dimensional generative models: vine copula autoencoders. In *Advances in neural information processing systems,* edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/15e122e839dfdaa7ce969536f94aecf6-Paper.pdf.

Tagasovska, Nataša, Firat Ozdemir, and Axel Brando. 2023. *Retrospective uncertainties for deep models using vine copulas.* arXiv: 2302.12606 [cs.LG].

Tepegjozova, Marija. 2019. D- and c-vine quantile regression for large data sets [in en]. Master's thesis, Technische Universität München, November. https://mediatum.ub.tum.de/node?id=1536070.

Tepegjozova, Marija, Jing Zhou, Gerda Claeskens, and Claudia Czado. 2022. Nonparametric c- and d-vine-based quantile regression. *Dependence Modeling* 10, no. 1 (January): 1–21. ISSN: 2300-2298. https://doi.org/10.1515/demo-2022-0100. http://dx.doi.org/10.1515/demo-2022-0100.

Tsay, Ruey. 2010. *Analysis of financial time series.* CourseSmart. Wiley.

Turinici, Gabriel. 2021. *The convergence of the stochastic gradient descent (sgd) : a self-contained proof* [in en]. Technical report. https://doi.org/10.5281/ZENODO.4638695. https://zenodo.org/record/4638695.

Van Rossum, Guido, and Fred L. Drake. 2009. *Python 3 reference manual.* Scotts Valley, CA: CreateSpace. ISBN: 1441412697.

Wolf, Michael M. 2023. *Mathematical foundations of supervised learning.* Graue Literatur. https://mediatum.ub.tum.de/node?id=1723378.

Xu, Qifa, Kai Deng, Cuixia Jiang, Fang Sun, and Xue Huang. 2017. Composite quantile regression neural network with applications. *Expert Systems with Applications* 76:129–139. ISSN: 0957-4174. https://doi.org/https://doi.org/10.1016/j.eswa.2017.01.054. https://www.sciencedirect.com/science/article/pii/S0957417417300726.

Zeng, Zhi, and Ting Wang. 2022. *Neural copula: a unified framework for estimating generic high-dimensional copula functions.* arXiv: 2205.15031 [cs.LG].

Zhang, Wenjie, Hao Quan, and Dipti Srinivasan. 2019. An improved quantile regression neural network for probabilistic load forecasting. *IEEE Transactions on Smart Grid* 10 (4): 4425–4434. https://doi.org/10.1109/TSG.2018.2859749.