The Market Generator

Alexei Kondratyev[†] Christian Schwarz[‡]

January 27, 2020

Abstract

We propose to use a special type of generative neural networks − a Restricted Boltzmann Machine (RBM) − to build a powerful generator of synthetic market data that can replicate the probability distribution of the original market data. An RBM constructed with stochastic binary activation units in both the hidden and the visible layers (Bernoulli RBM) can learn complex dependence structures while avoiding overfitting. In this paper we consider an efficient data transformation and sampling approach that allows us to operate Bernoulli RBM on real-valued data sets and control the degree of autocorrelation and non-stationarity in the generated time series.

Keywords: Restricted Boltzmann Machine, non-parametric sampling, generation of market scenarios, complex dependence structures

JEL classification: C63, G17

[†]Standard Chartered Bank, London EC2V 5DD, UK. Email: alexei.kondratyev@sc.com
[‡]Standard Chartered Bank, London EC2V 5DD, UK. Email: christian.schwarz@sc.com

1 Introduction

An efficient generation of realistic market scenarios, i.e., sampling from the joint distribution of risk factors, is one of the most important and challenging problems in quantitative finance. Historically, this problem was solved through sampling from some easy to calibrate parametric models, such as the multivariate Normal distribution of risk factor log-returns (equities) or a Gaussian copula combining the multivariate Normal dependence structure with the heavy-tailed univariate marginal distributions of individual risk factors (credit). However, there are well known issues with this approach that often outweigh the benefits provided by simplicity and transparency.

A parametric model is often a poor approximation of reality. In order to be useful, a parametric model has to be relatively simple: one should be able to describe the key features of the risk factor distribution with a handful of parameters achieving the best possible fit to either the empirical distribution derived from the historical data or the prices of traded instruments observed in the market at the time of model calibration. Making the parametric model too complex would lead to overfitting and poor generalisation.

It is even more difficult to model a realistic dependence structure. A typical parametric approach used in most Monte Carlo risk engines starts with modelling the dynamics of various risk factors independently, and then imposes a dependence structure by correlating the corresponding stochastic drivers. The stochastic drivers are, almost invariably, Brownian motions, and the linear correlations between them are supposed to be sufficient to construct the joint distribution of risk factors.

An alternative approach is to use non-parametric modelling where the joint and marginal distributions of risk factors are learned directly from the available data sets. This approach is realised, for example, in the Generative Adversarial Network (GAN) framework where the distribution learned from the data set by the generative neural network is tested by the discriminative neural network trying to judge whether samples are coming from the true distribution (data) or from the reconstructed distribution (generated samples) [1]. A fully trained GAN should be able to generate samples which are indistinguishable from the ones coming from the training data set. An application of feed-forward neural networks to the problem of learning natural risk factor configurations and their transformations was studied in [2].

Although GANs have achieved many outstanding results in sampling from complex distributions [3, 4, 5], in this paper we propose to build a market scenario generator based on a different type of generative neural network − the Restricted Boltzmann Machine (RBM). RBMs were among the first generative neural networks successfully applied to a wide range of use cases [6, 7, 8] and have many special properties one hopes to find in a market scenario generator.

The paper is organised as follows. In Section 2 we introduce the Bernoulli RBM, a type of RBM best suited for the learning and sampling from the joint distribution of risk factors. In Section 3 we illustrate the performance of the Bernoulli RBM on a specially constructed mixture of two Normal distributions. In Section 4 we apply Bernoulli RBM to sampling from the joint distribution of the market risk factors. The section also discusses how Bernoulli RBM can be used to control the degree of autocorrelation and non-stationarity in the generated time series. Finally, we conclude and outline directions of further research in Section 5.

2 Restricted Boltzmann Machine

An RBM is a two-layer neural network with stochastic activation units. "Restricted" in RBM indicates a constraint imposed on the network architecture: there are no connections between the units within the same layer. The RBM layer which is exposed to the training data set is called the visible layer. Inputs from the visible layer flow through the network

(forward pass) to the hidden layer, where they are aggregated and added to the hidden layer biases. The hidden layer sigmoid activation function converts aggregated inputs into probabilities. Each hidden unit then "fires" randomly − its output is a Bernoulli random variable: "1" is generated with probability p, which is equal to the sigmoid activation function value, and "0" is generated with probability $1 - p$.

The outputs from the hidden layer then flow back (backward pass) to the visible layer, where they are aggregated and added to the visible layer biases. Similar to the hidden layer, the visible layer sigmoid activation function translates aggregated inputs into probabilities. However, the visible layer does not necessarily convert probabilities into random variables − some of the RBM implementations output visible layer probabilities rather than random variables.

The network learns the joint distribution of the configurations of visible and hidden activation units by trying to reconstruct the inputs from the training data set (visible unit values) by finding an optimal set of the network weights and biases. Note that the same network weights are used for the forward and backward passes. We provide below a sketch of the training algorithm based on the stochastic gradient ascent approach that maximises the log likelihood of the training data.

The most obvious application of the RBMs is as autoencoders since the hidden layer can learn the low-dimensional probabilistic representation of the data set. RBMs can also be used for feature extraction and as one of the layers in a combined feed-forward neural network. Several RBMs can be stacked to form a deep belief network. Although the real strength of RBMs lies in them being generative neural networks trained in an unsupervised way, layer-wise trained stacked RBMs can be seen as a pre-trained feed-forward neural network that can be fine-tuned as a classifier with gradient descent and backpropagation if the training samples are labelled.

There are several compelling reasons to work with Bernoulli RBM, where both hidden and visible layers convert probabilities obtained with sigmoid activation functions into Bernoulli random variables:

- In a Bernoulli RBM, every unit communicates at most one bit of information. This is especially important for the hidden units since this feature implements the information bottleneck structure that acts as a strong regularizer [9].
- Operations on binary numbers are more efficient from the speed, memory and energy consumption perspective [10].
- Bernoulli RBMs naturally lend themselves to being efficiently trained on quantum annealers [11, 12].

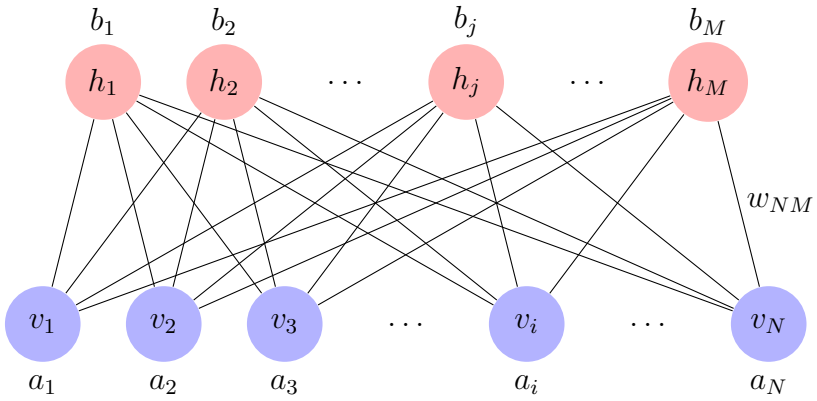

Figure 1: Schematic representation of Bernoulli RBM with the visible layer units (blue) and hidden layer units (red) forming a bipartite graph. For illustrative purposes only.

Let us have a look at a Bernoulli RBM with N stochastic binary visible units and M stochastic binary hidden units as shown in Figure 1. Let $\mathbf{v} = (v_1, \ldots, v_N)$ and $\mathbf{h} = (h_1, \ldots, h_M)$ denote, respectively, configurations of the visible and hidden units. A joint configuration of the visible and hidden units has an energy

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{M} b_j h_j - \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} v_i h_j$$

that defines the joint probability distribution $p(\mathbf{v}, \mathbf{h})$ of the configuration $(\mathbf{v}, \mathbf{h})$. Here, a_i, b_j and w_{ij} are, respectively, the biases for the visible and hidden layers and the network weights. The joint probability of a given pair of a visible and a hidden vector is given by the Boltzmann distribution:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})} ,$$

where the partition function Z is given by summing over all possible pairs of visible and hidden vectors:

$$Z = \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} .$$

The probabilities of the visible (hidden) states are given by summing over all possible hidden (visible) vectors:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} , \qquad p(\mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{v}} e^{-E(\mathbf{v}, \mathbf{h})} .$$

The network is trained through the updates of weights and biases which increase the log probability of a training vector. The sensitivities of the log probability of the training vector with respect to the weights and biases are given by the following expressions:

$$\frac{\partial p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} ,$$

$$\frac{\partial p(\mathbf{v})}{\partial a_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} ,$$

$$\frac{\partial p(\mathbf{v})}{\partial b_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{model} ,$$

where $\langle \ldots \rangle$ denote expectations under the distribution specified by the subscript. The weights and biases updates are then given by the expressions:

$$\Delta w_{ij} = \eta \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) ,$$

$$\Delta a_i = \eta \left(\langle v_i \rangle_{data} - \langle v_i \rangle_{model} \right) ,$$

$$\Delta b_j = \eta \left(\langle h_j \rangle_{data} - \langle h_j \rangle_{model} \right) ,$$

where η is the learning rate. Because there are no direct connections between the hidden units and no direct connections between the visible units, we can obtain an unbiased estimate of $\langle v_i h_j \rangle_{data}$. With $\sigma(x)$ denoting the logistic sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} ,$$

the binary state h_j of the hidden unit j is set to 1 with probability

$$p(h_j = 1 | \mathbf{v}) = \sigma \left(b_j + \sum_{i=1}^{N} v_i w_{ij} \right) ,$$

given the training vector $\mathbf{v}$. The product $v_i h_j$ is then an unbiased sample [9]. Similarly, the binary state v_i of the visible unit i is set to 1 with probability

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_{j=1}^{M} h_j w_{ij}\right) ,$$

given the hidden vector $\mathbf{h}$.

Getting an unbiased sample of $\langle v_i h_j \rangle_{model}$ requires performing alternating sampling from the model Boltzmann distribution for a long time (needed to achieve the thermal equilibrium), starting from some randomly initialised state. In Section 4 we show that in some cases the system thermalises only after $\sim 10^3$ alternating Boltzmann samplings (forward and backward passes between the visible and the hidden layers). However, there are two alternatives: i) sampling from the Boltzmann distribution using quantum annealing [11] – the quantum part of the hybrid quantum-classical RBM training protocol, and ii) the k-step contrastive divergence method [9, 13] which approximates $\langle v_i h_j \rangle_{model}$ with another, easier to calculate expectation. The latter can be formulated as the following training algorithm:

Algorithm 1 k-step contrastive divergence

Result: Weights and biases updates
Input: training minibatch S
Input: model parameters $a_i, b_j, w_{ij},\ i = 1, \ldots, N,\ j = 1, \ldots, M$ (before update)
Initialization: $\forall i, j : \Delta w_{ij} = \Delta a_i = \Delta b_j = 0$
for $\mathbf{v} \in S$ **do**
  $\mathbf{v}^{(0)} \leftarrow \mathbf{v}$
  for $t = 0, \ldots, k-1$ **do**
    for $j = 1, \ldots, M$ **do**
      sample Bernoulli random variable $h_j^{(t)} \sim p(h_j|\mathbf{v}^{(t)})$
    end
    for $i = 1, \ldots, N$ **do**
      sample Bernoulli random variable $v_i^{(t+1)} \sim p(v_i|\mathbf{h}^{(t)})$
    end
  end
  for $i = 1, \ldots, N,\ j = 1, \ldots, M$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \left(p(h_j = 1|\mathbf{v}^{(0)}) v_i^{(0)} - p(h_j = 1|\mathbf{v}^{(k)}) v_i^{(k)}\right)$
  end
  for $i = 1, \ldots, N$ **do**
    $\Delta a_i \leftarrow \Delta a_i + \eta \left(v_i^{(0)} - v_i^{(k)}\right)$
  end
  for $j = 1, \ldots, M$ **do**
    $\Delta b_j \leftarrow \Delta b_j + \eta \left(p(h_j = 1|\mathbf{v}^{(0)}) - p(h_j = 1|\mathbf{v}^{(k)})\right)$
  end
end

3 Learning the empirical distribution

Sampling from the trained RBM consists of three steps: i) generation of a random input, ii) performing K forward and backward passes through the network (alternating Boltzmann sampling), iii) taking the generated visible unit values after the last backward pass as

an output. An RBM trained with the bottleneck hidden layer structure works as an autoencoder that tries to reconstruct the input. In order to generate an independent sample it is necessary to achieve the state of thermal equilibrium where practically all information about the input is lost. Therefore, the *thermalisation* parameter K should be large enough to ensure the independence of the generated samples.

Bernoulli RBM operates on stochastic binary variables. Therefore, we need to design a method of converting real-valued samples into binary features that can be taken by the network as an input for training and a method of converting the generated binary network output (sampling) into real-valued variables. Such a method can be realised as a two-step routine:

1. Conversion of real-valued samples into the corresponding integer samples.
2. Conversion of integer samples into the corresponding binary features.

Given the generated binary output, the same routine can be used in a reverse mode to produce real-valued samples:

1. Conversion of the generated binary network output into integer samples.
2. Conversion of integer samples into the corresponding real-valued samples.

The following algorithms describe transformations of real-valued samples into 16-digit binary RBM features and then back into real-valued samples:

Algorithm 2 Real-valued to integer to binary transformation (training phase)

Result: Conversion of real-valued data set into binary features
Input: $X^{(n)}_{real}(l)$ – a real-valued data set: $l = 1, \ldots, N_{samples}$, $n = 1, \ldots, N_{variables}$
for $n = 1, \ldots, N_{variables}$ **do**
  $X^{(n)}_{min} \leftarrow \min_{\{l\}}(X^{(n)}_{real}) - \epsilon^{(n)}_{min}, \;\; \epsilon^{(n)}_{min} \geq 0$
  $X^{(n)}_{max} \leftarrow \max_{\{l\}}(X^{(n)}_{real}) + \epsilon^{(n)}_{max}, \;\; \epsilon^{(n)}_{max} \geq 0$
  for $l = 1, \ldots, N_{samples}$ **do**
    $X^{(n)}_{integer}(l) \leftarrow \text{int}\left(65535 \times \left(X^{(n)}_{real}(l) - X^{(n)}_{min}\right) / \left(X^{(n)}_{max} - X^{(n)}_{min}\right)\right)$
    $X^{(n)}_{binary}(l) \leftarrow \text{binarize}\left(X^{(n)}_{integer}(l)\right)$
  end
end
Each data sample is represented by a 16-digit binary number ($2^{16} - 1 = 65535$) with every digit becoming a separate feature. The total number of features is $16 \times N_{variables}$.

Algorithm 3 Binary to integer to real-valued transformation (sampling phase)

Result: Conversion of the generated 16-digit binary sample into real-valued sample
Input: $\hat{X}^{(n)}[m]$ – generated 16-digit binary sample: $m = 0, \ldots, 15$, $n = 1, \ldots, N_{variables}$
for $n = 1, \ldots, N_{variables}$ **do**
  $\hat{X}^{(n)}_{integer} \leftarrow 0$
  for $m = 0, \ldots, 15$ **do**
    $\hat{X}^{(n)}_{integer} \leftarrow \hat{X}^{(n)}_{integer} + 2^m \times \hat{X}^{(n)}[15 - m]$
  end
  $\hat{X}^{(n)}_{real} \leftarrow X^{(n)}_{min} + \hat{X}^{(n)}_{integer} \times \left(X^{(n)}_{max} - X^{(n)}_{min}\right) / \; 65535$
end

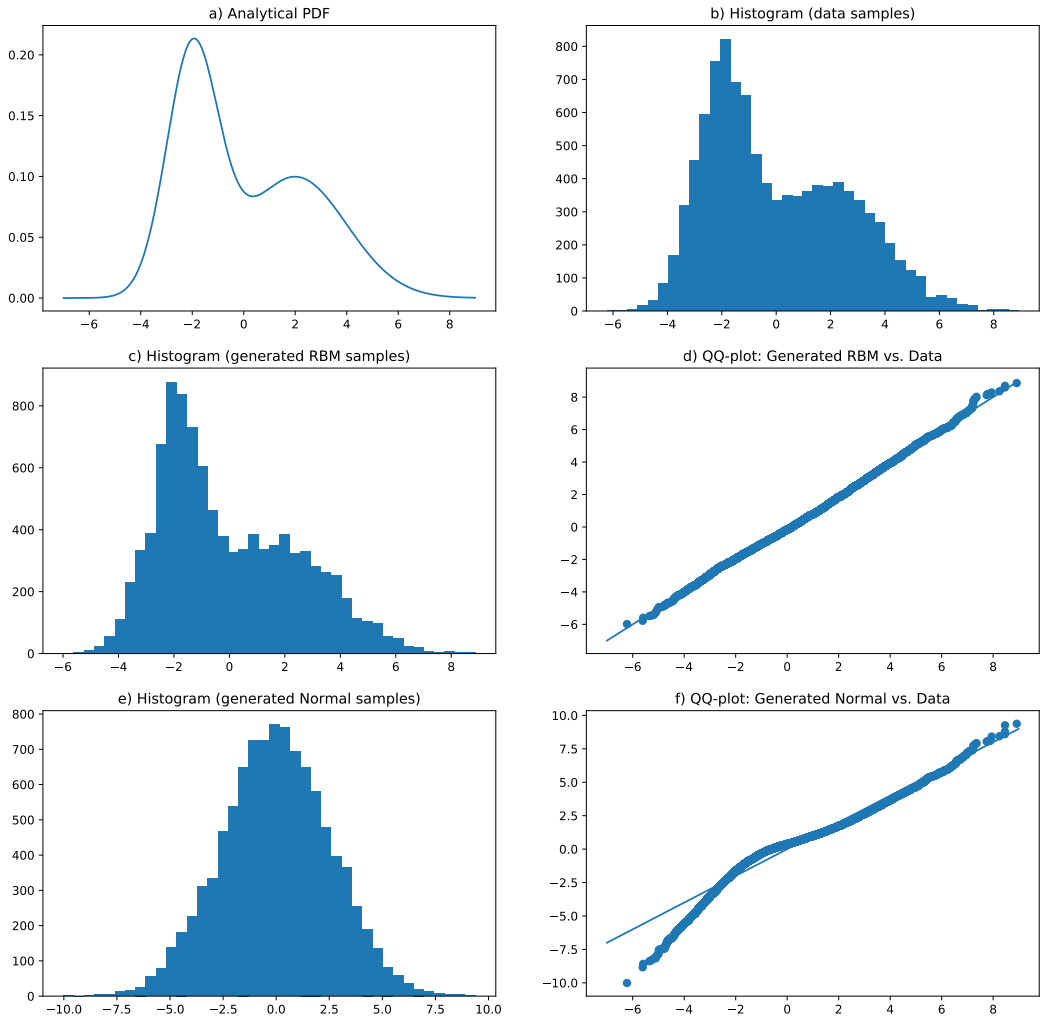

Figure 2: Learning the empirical distribution. a) The probability density function of the mixture of two equally weighted Normal distributions ($N(-2,1)$ and $N(2,2)$). b) The training data set (10,000 samples) was created by sampling from the mixture distribution. c) Sampling from the trained Bernoulli RBM closely matches the data set PDF. d) QQ-plot of the Bernoulli RBM-generated samples vs. the data set samples. e) Sampling from the Normal distribution that matches the first two moments of the data set distribution. f) QQ-plot of the Normal distribution-generated samples vs. the data set samples.

Before training Bernoulli RBM on a multivariate distribution of the market risk factors, we would like to try it first on a univariate distribution constructed as a mixture of two Normal distributions with different means and variances ($N(-2,1)$ and $N(2,2)$). The resulting probability density function (PDF) is shown in Figure 2 a). We constructed an empirical distribution by sampling from the mixture Normal distribution. Figure 2 b) displays the histogram of the empirical distribution based on 10,000 samples. These samples form the training data set for the RBM.

The RBM was trained using Algorithm 1 with the 1-step contrastive divergence routine. Real-valued samples were converted into binary samples as per Algorithm 2. Thus, the number of visible units was set equal to 16. The number of hidden units was set equal to 12. Following the standard RBM training practice, the size of a minibatch was set equal to 10. The samples were generated with the thermalisation parameter K set equal to 10^3 and were converted from binary to real-valued representation as per Algorithm 3.

The quality of the RBM-generated samples can be judged by looking at Figure 2 c) and Figure 2 d) showing the histogram and the QQ-plot against the empirical distribution. Both charts indicate near perfect reconstruction of the empirical distribution by the RBM-generated samples. At the same time an attempt to approximate the data set with a fitted Normal distribution (by matching the first two moments) clearly fails as shown by Figure 2 e) and Figure 2 f).

4 The market generator

4.1 Sampling the market

In this section we use actual historical market data to test our central proposal: to use a Bernoulli RBM trained on the historical joint distribution of the market risk factors in order to simulate future market scenarios that would maintain close resemblance to and replicate specific patterns learned from the empirical distribution. These market scenarios should not only match the mean, the variance and the correlation − they should also replicate the complete dependence structure, including the joint tail distribution.

The RBM was trained on the data set consisting of 5,070 daily log-returns of EURUSD, GBPUSD, USDJPY and USDCAD spot FX rates (1999-2019). Each log-return from the training data set was converted into a 16-digit binary number. Every digit of the binary number was treated as a separate feature (16 features per currency pair; 64 features in total) as described in Algorithm 2, with $\epsilon_{min}^{(n)} = \epsilon_{max}^{(n)} = 0$, $n = 1, \ldots, 4$. The number of hidden units was set equal to 30. Thus, the network was trained as a strongly regularised autoencoder. The generated log-returns (in the binary format) were converted back into integer numbers and then into real-valued samples as per Algorithm 3. We used minibatches of size 10, which, in our experience, is an optimal minibatch size when using stochastic gradient ascent. Larger minibatch sizes give more stable gradient estimates but slow down the learning process.

The network successfully learned the joint distribution of binary features and was able to reproduce the correlation structure (Table 1), annualised volatilities (Table 2) and tail behaviour as shown in Table 3 and in the QQ-plots (Figure 3). The QQ-plots also demonstrate that the traditional approach based on the multivariate Normal distribution of log-returns (calibrated to match the first two moments) fails to adequately capture the heavy tails of the empirical distribution.

Table 1 compares three different correlation metrics. Pearson's correlation measures linear correlation between the log-returns of two different currency pairs (bivariate linear correlation). Kendall's and Spearman's are rank correlations: Kendall's correlation measures the ordinal association between the log-returns and Spearman's correlation measures linear correlation between the rank values of log-returns.

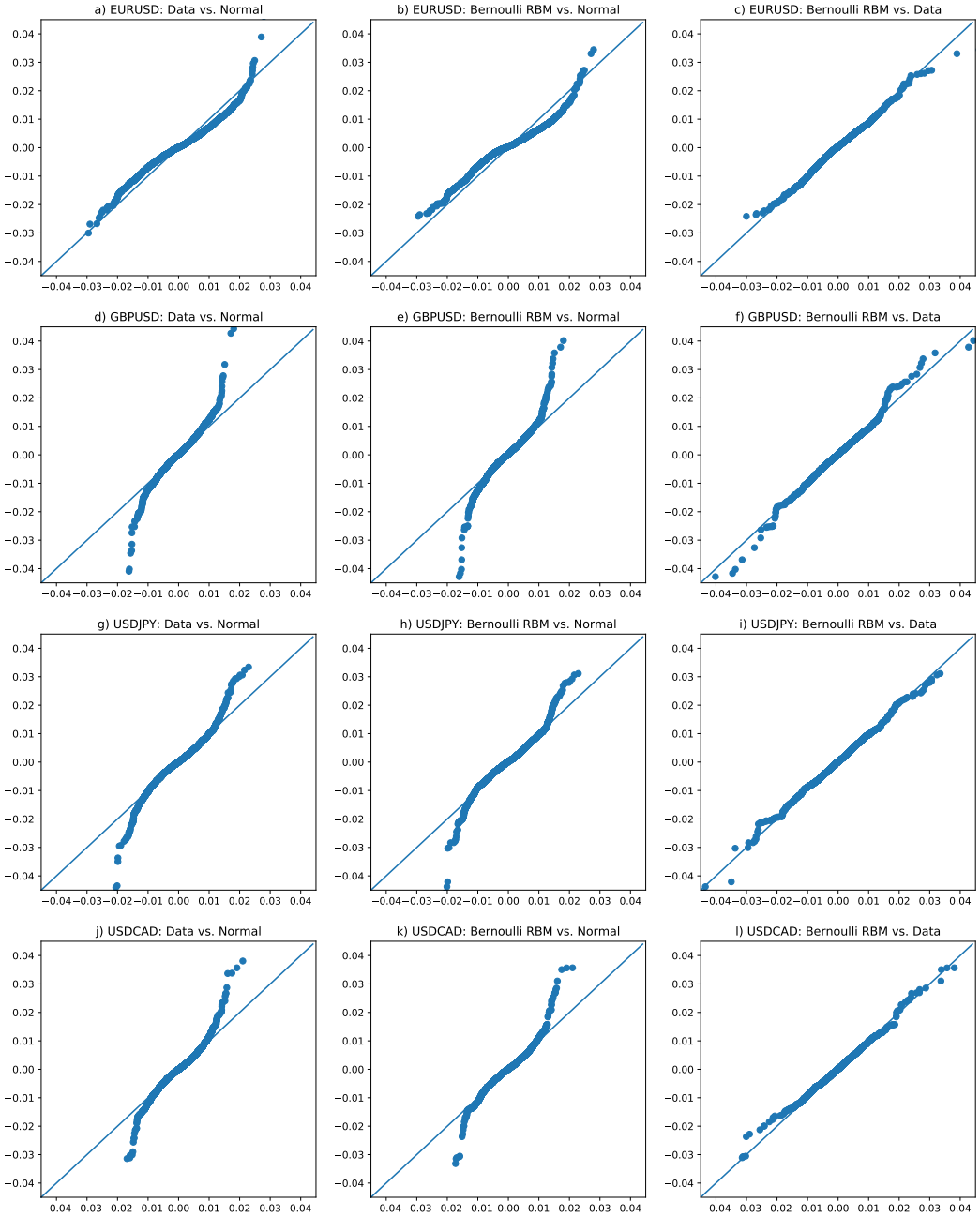

Figure 3: QQ-plots of the generated and empirical log-returns (5,070 samples). a)-c) EURUSD. d)-f) GBPUSD. g)-i) USDJPY. j)-l) USDCAD. Bernoulli RBM learns the heavy tailed empirical distribution.

Currency pairs	data set			RBM-generated samples		
	Pearson	Spearman	Kendall	Pearson	Spearman	Kendall
EURUSD/GBPUSD	63.7%	62.5%	45.3%	55.7% (±4.4%)	59.1% (±2.2%)	42.7% (±1.8%)
EURUSD/USDJPY	−28.2%	−31.2%	−21.8%	−26.0% (±2.5%)	−28.7% (±1.4%)	−20.0% (±1.0%)
EURUSD/USDCAD	−44.3%	−38.6%	−26.8%	−40.1% (±3.8%)	−37.1% (±1.8%)	−25.7% (±1.3%)
GBPUSD/USDJPY	−13.5%	−21.2%	−14.7%	−15.5% (±5.4%)	−21.2% (±0.9%)	−14.6% (±0.7%)
GBPUSD/USDCAD	−43.1%	−35.8%	−24.8%	−38.5% (±3.2%)	−34.3% (±2.4%)	−23.7% (±1.7%)
USDJPY/USDCAD	3.0%	7.4%	5.1%	5.6% (±2.5%)	8.5% (±1.5%)	5.8% (±1.0%)

Table 1: Reconstruction of correlations with Bernoulli RBM in the format: average (±1 standard deviation); 10 runs with 5,000 samples per run.

Currency pair	Historical volatility	RBM-generated samples
EURUSD	9.7%	9.8% (±0.7%)
GBPUSD	9.4%	9.5% (±0.7%)
USDJPY	10.2%	10.4% (±0.4%)
USDCAD	8.9%	8.6% (±0.8%)

Table 2: Reconstruction of annualised volatilities with Bernoulli RBM in the format: average (±1 standard deviation); 10 runs with 5,000 samples per run.

Currency pair	data set		RBM-generated samples	
	1st percentile	99th percentile	1st percentile	99th percentile
EURUSD	−1.58%	1.53%	−1.65% (±0.05%)	1.61% (±0.05%)
GBPUSD	−1.48%	1.34%	−1.55% (±0.04%)	1.53% (±0.10%)
USDJPY	−1.73%	1.66%	−1.84% (±0.07%)	1.78% (±0.07%)
USDCAD	−1.42%	1.50%	−1.47% (±0.04%)	1.55% (±0.06%)

Table 3: Reconstruction of tail behaviour with Bernoulli RBM in the format: average (±1 standard deviation); 10 runs with 5,000 samples per run.

Table 1 shows that rank correlations are reconstructed with higher precision than linear correlations, though the reconstructed Pearson's correlation coefficients are also quite close to their empirical values.

Moreover, the Bernoulli RBM was able to learn important, more general features of the dependency structure of the historic log-returns beyond linear/rank correlations. Specifically, Figure 4 shows the empirical lower/upper tail dependence functions of the real historical data, the synthetic data produced by the Bernoulli RBM and sampled data from the parametric approach. It clearly shows that the empirical lower/upper tail dependence functions generated through the RBM approach are very close to the ones of the historical real data and that the RBM technique once again outperforms the parametric model (minor exceptions occur in some of very far tails of the empirical distributions, e.g. in the left tail of the EURUSD/GBPUSD currency pair, however, there are counter examples, e.g in the right tail of the same currency pair or in the tails of USDJPY/USDCAD).

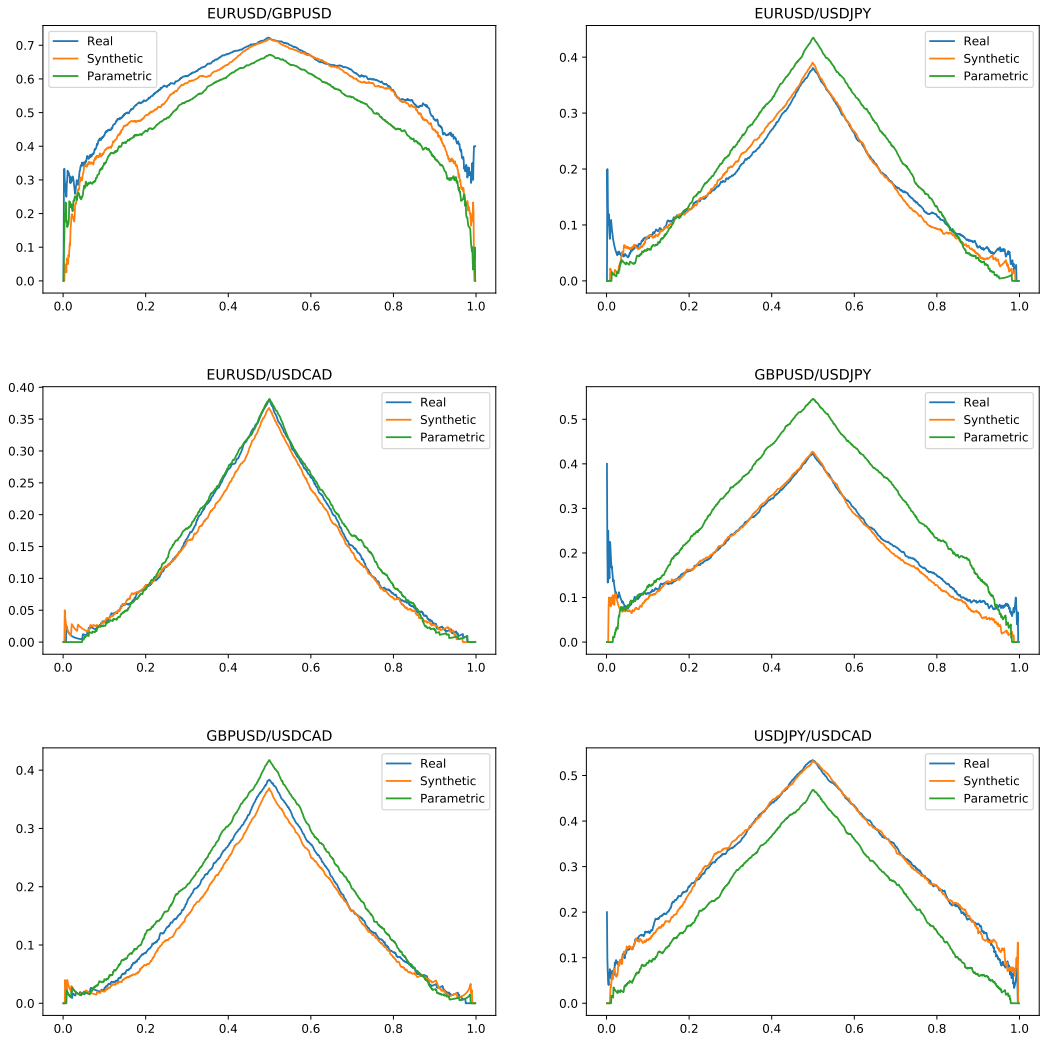

Figure 4: Empirical lower/upper tail concentration functions for all currency pairs. The trained Bernoulli RBM seems to capture the dependence structure of the real data better than the parametric approach.

4.2 Controlling autocorrelation

As was mentioned above, sampling from the trained Bernoulli RBM consists of three steps. The first step is generation of a random input: each visible unit is initialised with a randomly generated binary variable. The second step is performing a large number of forward and backward passes between the visible and the hidden layers, such that an initial random vector is transformed into a sample from the distribution learned by the RBM. After the final backward pass, the values of the visible units are taken as the output. The sampling cycle then starts again with the generation of a new random input.

It is important to perform sufficiently large number of forward and backward passes between the visible and the hidden layers (the alternating Boltzmann sampling). The fact that Bernoulli RBM was trained as an autoencoder means that it is likely to reconstruct

a randomly generated input with relatively high precision after the first cycle. We need to run many cycles of alternating Boltzmann sampling before the sample loses most (or all) information about the initial state. In other words, we need to perform alternating Boltzmann sampling until the system reaches *thermal equilibrium*. The number of cycles needed to reach the state of thermal equilibrium, the thermalisation parameter K, is problem dependent and is a function of network architecture and network parameters (weights and biases).

Interestingly, we can use the fact that the system takes some time to thermalise in order to generate stochastic processes with the desired degree of autocorrelation. If the network was trained to generate the log-returns of risk factors, then the autocorrelated process can be simulated using the following algorithm:

Algorithm 4 Generation of the autocorrelated log-returns

Result: Returns the new generated log-return taking the previous one as an input
Input: S_{t-1} − the latest simulated log-return for a given risk factor
Definition: sample(input) − performs a single round of alternating Boltzmann sampling
Initialisation: $X \leftarrow S_{t-1}$
for $k = 1, \ldots, K$ **do**
  $X \leftarrow \text{sample}(X)$
end
$S_t \leftarrow X$
Output: S_t

Figure 5 shows the rate of autocorrelation between the two consecutively generated EURUSD daily log-returns as a function of K. If a mild autocorrelation of 3-4% is desirable, the value of K can be set at about 10^3. If the autocorrelation should be eliminated completely, the value of K should be set $\geq 10^4$.

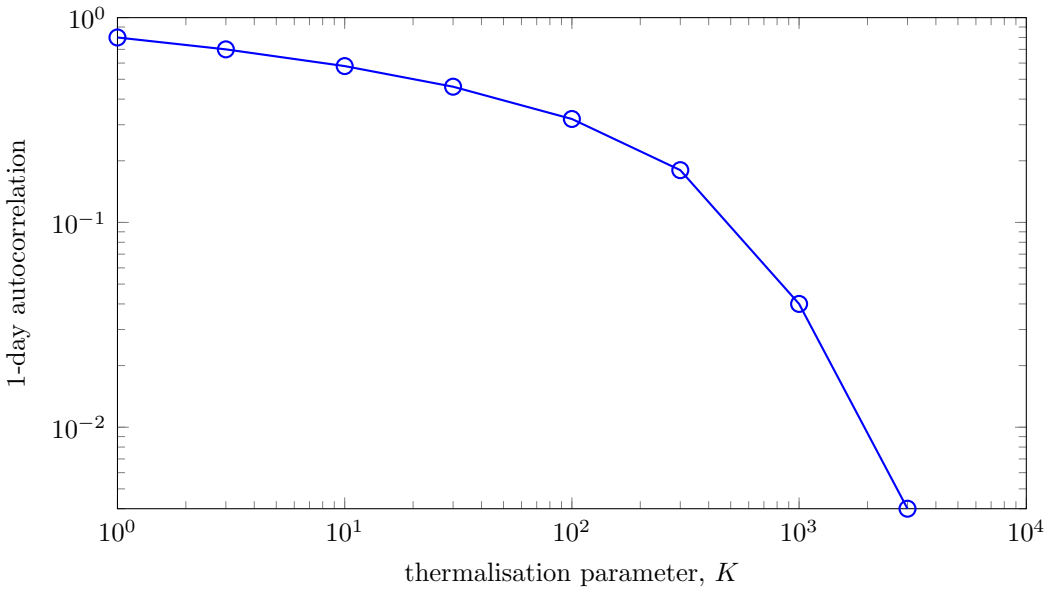

Figure 5: 1-day autocorrelation of the generated EURUSD daily log-returns as a function of the thermalisation parameter (number of forward and backward passes in the sample generation routine). The chart shows an average 1-day autocorrelation over 50,000 generated samples.

4.3 Non-stationarity and conditional sampling

It is a well known fact that historical time series of most financial data are non-stationary to various degrees. Often the non-stationarity manifests itself as clearly observable periods of high/low volatility. Another example is increased correlation between the risk factors during the time of market distress.

An empirical distribution based on historical observations of the risk factor returns (collected over a long period of time) would invariably include samples from different market regimes. A generative model trained to replicate such empirical distribution would faithfully produce samples that are virtually indistinguishable from the historical ones, but combining them into a time series of the corresponding risk factors would hardly show any non-stationarity. In other words, unconditional sampling from the learned distribution results in stationary time series of the simulated risk factors.

Fortunately, the proposed RBM approach is flexible enough to accommodate and reproduce the non-stationarity observed in the historical time series. Figure 6 shows historical time series of the 3-month historical volatility for the currency pairs from our data set. We can see with the naked eye the periods of high (low) volatility defined here as 3-month historical volatility being larger (smaller) than the corresponding long-term median value. This gives us an idea of how to generate conditional samples from different volatility regimes.

We introduce binary volatility indicators for each currency pair in our data set. Every sample in the data set is stamped with a volatility indicator value: '1' if the running 3-month historical volatility of the given risk factor is larger than its long-term median value and '0' otherwise. Therefore, the number of visible units increases by the number of risk factors − there is a new visible unit associated with each risk factor in the data set and its value indicates whether at the time of observation the risk factor's 3-month historical volatility was above or below its long-term median value.

The RBM is trained in the usual way, but now we have an ability to generate samples conditional on the volatility regime. One possible approach is to first sample unconditionally and then go through all simulated scenarios selecting only those that have the desired configuration of the volatility indicators. Although this is the simplest way of generating samples conditional on a particular volatility regime, it is not the most efficient way of generating samples. In the case when we condition on a rare configuration of the volatility indicators, the vast majority of the simulated scenarios cannot be used. Since every generated sample requires thousands of forward and backward passes through the network during the thermalisation phase, wasting most of the generated samples can be prohibitively expensive from a simulation time perspective.

An advanced approach is to use the fundamental properties of the RBM neural network. When generating samples we need to keep the values of the volatility indicators fixed at the desired configuration. By constantly resetting the values of the volatility indicators to their target configuration during the thermalisation phase, the network stimulates activation units in the hidden layer during each forward pass in such a way that, after the backward pass (the reconstruction pass), the visible layer activation units are likely to produce scenarios that are consistent with the fixed values of the volatility indicators.

Table 4 shows the resulting volatilities of the generated samples of two different regimes. The low volatility regime is the one where the values of all volatility indicators were set to '0' during the sampling process. The high volatility regime corresponds to the configuration of all volatility indicators set to '1'. Thus, we have an ability to switch between the low volatility and the high volatility regimes by randomly flipping the values of all (or some) volatility indicators with any desired frequency. In this way we can generate non-stationary time series that preserve the dependence structure and all other properties of the learned distribution. Table 4 also shows historical volatilities conditional on the same volatility regimes. We see that the RBM-generated samples reproduce historical conditional volatilities with reasonably high precision, especially in the case of the

low volatility environment.

Currency pair	Low volatility environment		High volatility environment	
	Historical volatility	RBM-generated samples	Historical volatility	RBM-generated samples
EURUSD	6.8%	6.7% (±0.2%)	12.2%	10.8% (±0.5%)
GBPUSD	6.9%	6.8% (±0.3%)	13.1%	13.7% (±0.7%)
USDJPY	8.0%	8.9% (±0.3%)	12.8%	12.7% (±0.6%)
USDCAD	6.2%	6.4% (±0.2%)	13.0%	11.6% (±0.7%)

Table 4: Reconstruction of conditional annualised volatilities with a Bernoulli RBM in the format: average (±1 standard deviation); 10 runs with 5,000 samples per run.

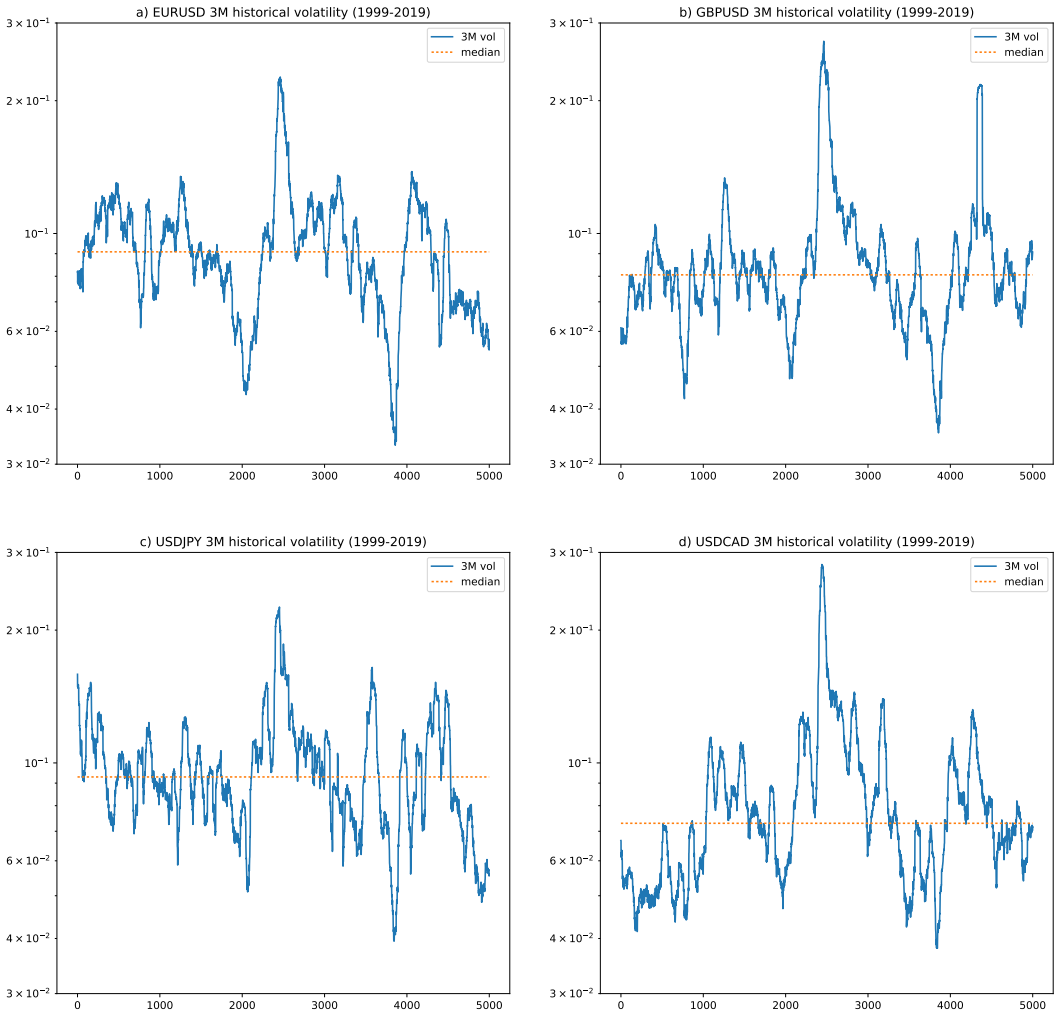

Figure 6: Time series of 3-month historical volatilities. a) EURUSD. b) GBPUSD. c) USDJPY. d) USDCAD.

5 Conclusions

Bernoulli RBMs are a powerful type of generative neural network capable of learning complex multivariate distributions from a limited number of samples. Overfitting, which would be an issue with most other neural network architectures in this context, is avoided due to the strong regularization imposed by the bottleneck structure with stochastic binary units.

The training and sampling procedures outlined in Algorithms 1-3 are straightforward to implement and easy to run due to a small number of hyperparameters. The hyperparameters can be optimised with a simple grid search method. By varying the thermalisation parameter, it is possible to either control the degree of autocorrelation in the generated time series or switch it off altogether depending on the use case.

The proposed application of a Bernoulli RBM for the generation of possible market scenarios is justified by the network's ability to reproduce the key statistics of the empirical joint distribution of the risk factors, such as mean, variance, correlation and tail behaviour. Moreover, Bernoulli RBMs are able to reproduce specific patterns in the dependence structure of the risk factor dynamics they simulate.

Bernoulli RBM-generated scenarios are usually of much higher quality than the ones produced by the standard parametric models one can find in a typical Monte Carlo risk engine. The non-parametric way of learning from an empirical distribution also makes Bernoulli RBMs strong competitors to conventional VaR engines.

Arguably, the VaR-style historical simulation technique is the one most likely to be disrupted by the proposed approach. Historical simulation is, essentially, a method of direct sampling from the empirical distribution with the limited number of samples. In the likely case of sampling a scenario for the hundreds of risk factors from a pool of just 250 historical scenarios, it is unrealistic to expect the VaR to be an accurate measure of risk at a high percentile confidence level. In contrast, an RBM trained on the data set of 250 samples can generate any number of scenarios from the learned distribution.

This leads us to a more general use case of building a synthetic data generator able to learn the multivariate distributions from small training data sets in a completely non-parametric way.

Although the contrastive divergence training method described in Algorithm 1 works reasonably well in practice, there is an even more promising hybrid quantum-classical training algorithm that utilises the deep connection between the Bernoulli RBM energy function and quadratic unconstrained binary optimisation (QUBO) problems solvable on quantum annealers. As quantum computing hardware matures and becomes easily available via cloud access, the potential of a hybrid quantum-classical risk engine architecture should not be underestimated. We would strongly advocate this topic for further research.

Disclaimer and Declaration of Interest

The authors report no conflict of interest. The authors alone are responsible for the content and writing of the paper. The opinions expressed are those of the authors and do not necessarily reflect the views and policies of their respective institutions. All figures are based on our own calculations.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio. Generative Adversarial Nets. *arXiv preprint arXiv:1406.2661*, 2014.

[2] Alexei Kondratyev. Curve Dynamics with Artificial Neural Networks. *Risk*, Volume 31, Issue 6, June, 2018.

[3] Alec Radford, Luke Metz and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[4] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele and Honglak Lee. Generative adversarial text to image synthesis. *In Proceedings of The 33rd International Conference on Machine Learning*, Volume 3, 2016.

[5] Stephanie L. Hyland, Cristóbal Esteban and Gunnar Rätsch. Real-valued (Medical) time series generation with recurrent conditional GANs. *arXiv preprint arXiv:1706.02633*, 2017.

[6] Ruslan Salakhutdinov, Andriy Mnih and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

[7] Asja Fischer and Christian Igel. An Introduction to Restricted Boltzmann Machines. *In: Alvarez L., Mejail M., Gomez L., Jacobo J. (eds) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. CIARP 2012. Lecture Notes in Computer Science*, Volume 7441, Springer, Berlin, Heidelberg, 2012.

[8] Ruhi Sarikaya, Geoffrey Hinton and Anoop Deoras. Application of Deep Belief Networks for Natural Language Understanding *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Volume 22, Issue 4, April, 2014.

[9] Geoffrey Hinton. A Practical Guide to Training Restricted Boltzmann Machines. Department of Computer Science, University of Toronto, UTML TR 2010-003, 2010.

[10] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv and Yoshua Bengio. Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[11] Steven H. Adachi and Maxwell P. Henderson. Application of Quantum Annealing to Training of Deep Neural Networks. *arXiv preprint arXiv:1510.06356*, 2015.

[12] Jeremy Liu, Federico M. Spedalieri, Ke-Thia Yao, Thomas E. Potok, Catherine Schuman, Steven Young, Robert Patton, Garrett S. Rose and Gangotree Chamka. Adiabatic Quantum Computation Applied to Deep Learning Networks. *Entropy*, Volume 20, Issue 5, 2018.

[13] Asja Fischer and Christian Igel. Training Restricted Boltzmann Machines: An Introduction. *Pattern Recognition*, Volume 47, 2014.