Big Data and Information Analytics

BDIA, 7: 1–12.
DOI: 10.3934/bdia.2022001
Received: 20 June 2022
Revised: 29 July 2022
Accepted: 09 September 2022
Published: 26 September 2022

http://www.aimspress.com/journal/BDIA

Research article

Modeling path-dependent state transitions by a recurrent neural network

Bill Huajian Yang*

Garland Tech Inc., 54 Courtsfield Crescent, Etobicoke, ON M9A 4S9, Canada

* **Correspondence:** Email: h_y02@yahoo.ca.

Abstract: Rating transition models are widely used for credit risk evaluation. It is not uncommon that a time-homogeneous Markov rating migration model will deteriorate quickly after projecting repeatedly for a few periods. This is because the time-homogeneous Markov condition is generally not satisfied. For a credit portfolio, the rating transition is usually path-dependent. In this paper, we propose a recurrent neural network (RNN) model for modeling path-dependent rating migration. An RNN is a type of artificial neural network where connections between nodes form a directed graph along a temporal sequence. There are neurons for input and output at each time period. The model is informed by the past behaviors for a loan along the path. Information learned from previous periods propagates to future periods. The experiments show that this RNN model is robust.

Keywords: path-dependent; rating transition; recurrent neural network; deep learning; Markov property; time homogeneity

1. Introduction

Rating transition models are widely used in the financial industry for credit risk evaluations, including stress testing and IFRS 9 expected credit loss evaluation [1–4], under the assumption that a rating transition is a time-homogeneous Markov process, depending only on the current rating and covariates. However, it is not uncommon that a Markov model deteriorates quickly after projecting for a few periods. This is because the Markov condition is generally not satisfied. A rating transition for a credit portfolio is generally path dependent. A test for this assumption is required [5,6] for the use of these Markov models.

There are various methods for path-dependent credit risk modeling [7], including regime-switching models [8] and the conditional methods [9,10]. The latter is comparative to a cohort analysis.

In this paper, we propose a recurrent neural network (RNN) model for modeling path-dependent rating transition. An RNN is a type of artificial neural network where connections between nodes form a directed graph along a temporal sequence. There are neurons for input and output at each time-period. The RNN is informed by past behaviours along the path. Information learned from previous periods propagates to future periods [11–17].

The network structure for the proposed RNN model is described in Section 2 by using (2.1)–(2.4). This RNN model was implemented in Python. The experiments show that this RNN model is robust, compared to Markov transition models. Applications of this RNN model include the following, wherever path dependence is relevant:

(a) Path-dependent asset evaluation or credit risk evaluation
(b) Decisioning for account management
(c) Forecasting loss for stress testing, expected credit loss for IFRS 9 projects
(d) Estimating conditional probability of default for survival analysis

The paper is organized as follows. In Section 2, we setup the proposed RNN model. In Section 3, we calculate the partial derivatives for the network cost function. In Section 4, we present the experimental results for this proposed RNN model, as benchmarked with the time-homogeneous and time-inhomogeneous rating transition models.

2. Recurrent neural network models for multiperiod state transition

In this section, we describe, the proposed RNN model, as given in (2.1)–(2.4), for modeling path-dependent rating transition. Given an observation horizon with T periods:

$$0 = t_0 < t_1 < t_2 < \cdots < t_T;$$

our goal is to estimate at each $i \geq 0$ the probability of transitioning to a rating at time t_{i+1} given the rating and covariates at time t_i.

Traditional rating transition models assume the Markov condition, i.e., the transition probability depends only on the current rating and covariates. The type of Markov transition models includes the following:

(a) Time-homogeneous Markov rating transition, as represented by one single transition model for all periods

(b) Time-inhomogeneous Markov rating transition, as represented by one transition model for each period

It is not uncommon that a time-homogeneous Markov model will deteriorate quickly after projecting only for a few periods. This is because the Markov condition is generally not satisfied. A rating transition for a credit portfolio is generally path dependent.

An RNN is an artificial neural network where connections between nodes form a directed graph along a temporal sequence. The chart below depicts the structure of an RNN. At the i^{th} time period of the temporal sequence, the input neurons, the hidden neurons and the output neurons are respectively labeled $x^{(i)}$, $h^{(i)}$ and $y^{(i)}$:

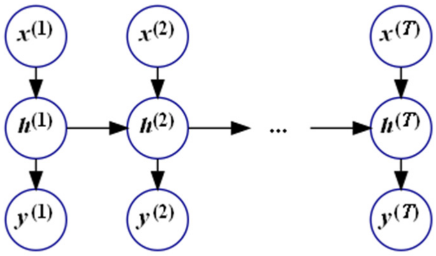

Figure 1. An RNN for rating transition.

An RNN shares the advantages of common neural networks; particularly, information learned at a point is propagated back and forward to all periods. It is path dependent.

Let $\{R_i\}_{i=1}^{n}$ denote the n ratings for a credit portfolio. For a loan portfolio, we reserve R_{n-1} as the withdrawal rating and R_n as the default ratings. Both the default and withdrawal ratings are assumed to be absorbed states, which means that a loan rated by a default or withdrawal rating will be excluded from the sample for future subsequent observations. Rating labels are observable at the beginning and the end of a period.

Let $r_j^{(i)}$ denote the indicator with a value of 1 if the rating for a loan at the end of the i^{th} period is R_j and 0 otherwise. Let n_p denote the number of non-absorbed ratings. An input at the i^{th} period is denoted as

$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}),$$

where the first $(m - n_p)$ input components $(x_1^{(i)}, x_2^{(i)}, \dots, x_{m-n_p}^{(i)})$ denote the covariates observable at the beginning of the i^{th}period, and the remaining n_p components are the rating indicators for non-absorbed ratings observed at the end of the $(i-1)^{th}$ period:

$$x_j^{(i)} = r_j^{(i-1)}, \qquad m - n_p + 1 \le j \le m.$$

That is, the non-absorbed rating observed at the end of the $(i-1)^{th}$ period is used as the input for the next period. The output at the i^{th} period is denoted as

$$y^{(i)} = (y_1^{(i)}, y_2^{(i)}, \dots, y_n^{(i)})$$

where

$$y_j^{(i)} = r_j^{(i)}, 1 \le j \le n.$$

The structure for this RNN is described as shown in (2.1)–(2.4) below. Initially, for the first period, we have

(a) Input: $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \dots, x_m^{(1)})$;

(b) Output: $y^{(1)}$=$(y_1^{(1)}, y_2^{(1)}, \dots, y_n^{(1)})$, i.e., a unit vector where all components are zero except for one , which has a value of 1, and it is a random realization generated by the multinomial probability $p_1 = (p_{11,} p_{12}, \dots, p_{1n})$, where

$$p_{1j} = \frac{\exp\left(v_j^{(1)}\right)}{\exp\left(v_1^{(1)}\right)+\exp\left(v_2^{(1)}\right)+\cdots+\exp\left(v_n^{(1)}\right)}, \tag{2.1}$$

and

$$v_j^{(1)} = a_{j1}^{(1)} x_1^{(1)} + a_{j2}^{(1)} x_2^{(1)} + \cdots + a_{jm}^{(1)} x_m^{(1)}. \tag{2.2}$$

Vector $(v_1^{(1)}, v_2^{(1)}, \dots, v_n^{(1)})$ in (2.1) and (2.2) represents the information learned during the 1st period, which is stored in hidden neurons $h^{(1)} = (h_1^{(1)}, h_2^{(1)}, \dots, h_n^{(1)})$ in the 1st period.

In general, given the vector $(v_1^{(i-1)}, v_2^{(i-1)}, \dots, v_n^{(i-1)})$ for the $(i-1)^{th}$ period ($i \geq 2$), we have the following at the i^{th} period:

(c) Input: $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)})$;

(d) Output: $y^{(i)} = (y_1^{(i)}, y_2^{(i)}, \dots, y_n^{(i)})$, i.e., a unit vector, which is a random realization generated by multinomial probability $p_i = (p_{i1,} p_{i2}, \dots, p_{in})$, where

$$p_{ij} = \frac{\exp\left(v_j^{(i)}\right)}{\exp\left(v_1^{(i)}\right) + \exp\left(v_2^{(i)}\right) + \dots + \exp\left(v_n^{(i)}\right)}. \tag{2.3}$$

and

$$v_j^{(i)} = a_{j1}^{(i)} x_1^{(i)} + a_{j2}^{(i)} x_2^{(i)} + \cdots + a_{jm}^{(i)} x_m^{(i)} + b_{j1}^{(i)} v_1^{(i-1)} + b_{j2}^{(i)} v_2^{(i-1)} + \cdots + b_{jn}^{(i)} v_n^{(i-1)}. \tag{2.4}$$

As observed, $v_j^{(i)}$ consists of two parts, one from the current input, and the other from history, i.e., $(v_1^{(i-1)}, v_2^{(i-1)}, \dots, v_n^{(i-1)})$, corresponding to the information learned up to the $(i-1)^{th}$ period. Similarly, the vector $(v_1^{(i)}, v_2^{(i)}, \dots, v_n^{(i)})$ represents the information learned so far up to i^{th} period, which is stored in hidden neurons $h^{(i)} = (h_1^{(i)}, h_2^{(i)}, \dots, h_n^{(i)})$.

This RNN model works, at the i^{th} stage, in the way as described below:

1) Collects the input $x^{(i)}$, and information $h^{(i-1)}$ learned up to the end of $(i-1)$
2) Learns from $x^{(i)}$ and $h^{(i-1)}$and stores the learned information in hidden neurons $h^{(i)}$
3) Derives the multinomial probability $(p_{i1}, p_{i2}, \dots, p_{in})$, where p_{ij} is the probability that the event will transition to j^{th} rating.
4) Output: $y^{(i)}$ is a random multinomial realization, given the multinomial probability distribution $(p_{i1}, p_{i2}, \dots, p_{in})$

Remark 2.1. The formulation of (2.4) does not come with a bias (i.e., intercept). An intercept can be inserted by adding a covariate with a constant value of 1, whenever necessary.

3. Training the RNN rating transition model

Let $y^{(k)} = (y_1^{(k)}, y_2^{(k)}, \dots, y_n^{(k)})$ be the observed outcome at the k^{th} period. The cost function at the k^{th} period is denoted by L_k, which is given as (for one single data point):

$$L_k = -\sum_{j=1}^{n} y_j^{(k)} \log(p_{kj}). \tag{3.1}$$

This is the negative log-likelihood for observing the multinomial outcome $(y_1^{(k)}, y_2^{(k)}, \dots, y_n^{(k)})$. The total cost function to be minimized for this recurrent neural network is

$$L = L_1 + L_2 + \cdots + L_T. \tag{3.2}$$

summing over entire training sample.

3.1. Partial derivatives of L with respect to network weights

Training a neural network involves a series of gradient descent searches. Evaluation of partial derivatives is essential. In this sub-section, we calculate the partial derivatives for the network cost function with respect to network weights.

3.1.1. Partial derivatives of L_k with respect to $v_i^{(k-r)}$

Let $d_i^{(k,r)}$ denote the partial derivative of L_k with respect to $v_i^{(k-r)}$, $0 \le r \le k-1$. By (2.1) and (2.3), we have the partial derivative $\frac{\partial p_{ki}}{\partial v_j^{(k)}}$ as:

$$\frac{\partial p_{ki}}{\partial v_j^{(k)}} = \begin{cases} p_{ki}(1-p_{ki}), & i=j, \\ -p_{ki}p_{kj}, & i \ne j. \end{cases} \tag{3.3}$$

Hence, by (3.1) and (3.3), we have the following for $1 \le i \le n$:

$$d_i^{(k,0)} = \partial L_k / \partial v_i^{(k)} = -\sum_{j=1}^{n} y_j^{(k)}\, \partial[\log(p_{kj})]/\partial v_i^{(k)}$$

$$= -y_i^{(k)}(1-p_{ki}) + \sum_{j\ne i}^{n} y_j^{(k)} p_{ki} = -\left(y_i^{(k)} - p_{ki}\right), \tag{3.4}$$

where $\sum_{j=1}^{n} y_j^{(k)} = 1$ is used.

Given $d_j^{(k,0)}, 1 \le j \le n,$ we can calculate $d_i^{(k,1)}$ from top-down for $k > 1$ and $1 \le i \le n$ by using (2.4):

$$d_i^{(k,1)} = \frac{\partial L_k}{\partial v_i^{(k-1)}} = \sum_{j=1}^{n} \left(\frac{\partial L_k}{\partial v_j^{(k)}}\right)\left(\frac{\partial v_j^{(k)}}{\partial v_i^{(k-1)}}\right) = \sum_{j=1}^{n} b_{ji}^{(k)} d_j^{(k,0)}.$$

Inductively, we have the following for $k > r$ and $0 \le i \le n$:

$$d_i^{(k,r)} = \frac{\partial L_k}{\partial v_i^{(k-r)}}$$

$$= \sum_{j=1}^{n} \left(\frac{\partial L_k}{\partial v_j^{(k-r+1)}}\right)\left(\frac{\partial v_j^{(k-r+1)}}{\partial v_i^{(k-r)}}\right)$$

$$= \sum_{j=1}^{n} b_{ji}^{(k-r+1)} d_j^{(k,r-1)}. \tag{3.5}$$

3.1.2. Partial derivatives of $L = \sum_{k=1}^{T} L_k$ with respect to $v_i^{(r)}$

We will use the following fact:

$$\frac{\partial \boldsymbol{L_k}}{\partial v_j^{(i)}} = 0 \text{ if } k < i. \tag{3.6}$$

Let D_i^{T-r} denote the partial derivative of L with respect to $v_i^{(T-r)}$. Given $\{d_i^{(k,0)} | \; 1 \le i \le n, 1 \le k \le T\}$, we can calculate $\{D_i^{T-r} | \; 1 \le i \le n, 0 \le r < T\}$ top-down. Initially, at the top period, we have the following by (3.6):

$$D_i^T = \frac{\partial \boldsymbol{L}}{\boldsymbol{\partial} v_i^{(T)}} = \frac{\partial \boldsymbol{L_T}}{\boldsymbol{\partial} v_i^{(T)}} = d_i^{(T,0)}.$$

Next, backward from the top period, we have $D_i^{T-1}$for $T > 1$ and $1 \le i \le n$ as follows:

$$D_i^{T-1} = \frac{\partial L}{\partial v_i^{(T-1)}} = \frac{\partial (L_T + L_{T-1})}{\partial v_i^{(T-1)}}$$
$$= \frac{\partial L_T}{\partial v_i^{(T-1)}} + \frac{\partial L_{T-1}}{\partial v_i^{(T-1)}} = \sum_{j=1}^{n} \frac{\partial L_T}{\partial v_j^{(T)}} \frac{\partial v_j^{(T)}}{\partial v_i^{(T-1)}} + d_i^{(T-1,0)}$$
$$= \sum_{j=1}^{n} b_{ji}^{(T)} \frac{\partial L}{\partial v_j^{(T)}} \quad + d_i^{(T-1,0)}$$
$$= \sum_{j=1}^{n} \quad b_{ji}^{(T)} D_j^T + d_i^{(T-1,0)},$$

where (3.6) is used for 2nd and 5^{th}equality signs. Inductively, we have D_i^{T-r} for $T > r$ and $1 \le i \le n$ from top-down as follows:

$$D_i^{T-r} = \frac{\partial L}{\partial v_i^{(T-r)}} = \frac{\partial (L_T + L_{T-1} + \cdots + L_{T-r+1})}{\partial v_i^{(T-r)}} + \frac{\partial L_{T-r}}{\partial v_i^{(T-r)}}, \tag{3.7}$$

$$= \sum_{j=1}^{n} \frac{\partial (L_T + L_{T-1} + \cdots + L_{T-r+1})}{\partial v_j^{(T-r+1)}} \frac{\partial v_j^{(T-r+1)}}{\partial v_i^{(T-r)}} + d_i^{(T-r,0)}$$
$$= \sum_{j=1}^{n} \frac{\partial L}{\partial v_j^{(T-r+1)}} \frac{\partial v_j^{(T-r+1)}}{\partial v_i^{(T-r)}} + d_i^{(T-r,0)}$$
$$= \sum_{j=1}^{n} \quad b_{ji}^{(T-r+1)} D_j^{(T-r+1)} + d_i^{(T-r,0)},$$

where (3.6) is used for 2nd and 4^{th} equality signs.

3.1.3. Partial derivatives of $L = \sum_{k=1}^{T} L_k$ with respect to $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$

Given $\{D_i^r\}$, i.e., the partial derivatives of the cost function L with respect to $v_i^{(r)}$; we can now find the partial derivatives of L with respect network weights $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$ as defined in (2.2) and (2.4).

Let δ_{ij}^r and σ_{ij}^r denote respectively the partial derivatives of L with respect to $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$. By (2.4), at the top time period $r = T$, we have

$$\delta_{ij}^T = \frac{\partial L}{\partial a_{ij}^{(T)}} = \frac{\partial L}{\partial v_i^{(T)}} \frac{\partial v_i^{(T)}}{\partial a_{ij}^{(T)}} = x_j^{(T)} D_i^T,$$

$$\sigma_{ij}^T = \frac{\partial L}{\partial b_{ij}^{(T)}} = \frac{\partial L}{\partial v_i^{(T)}} \frac{\partial v_i^{(T)}}{\partial b_{ij}^{(T)}} = v_j^{(T-1)} D_i^T.$$

If general, we have

$$\delta_{ij}^{T-r} = \ x_j^{(T-r)} D_i^{T-r}\,, \tag{3.8}$$

$$\sigma_{ij}^{T-r} = \ v_j^{(T-r-1)} D_i^{T-r}\,. \tag{3.9}$$

3.2. Initialization of network weights

A good initialization of the network weights $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$ speeds up the convergence for the network training. In this sub-section, we propose an algorithm for initializing the network weights.

Let $w_j^{(r)}$ denote the vector of weights in (2.4) for $v_j^{(r)}$ at the r^{th} time period, i.e.,

$$w_j^{(r)} = \left(a_{j1}^{(r)}, a_{j2}^{(r)}, \ldots, a_{jm}^{(r)}, b_{j1}^{(r)}, b_{j2}^{(r)}, \ldots,\ b_{jn}^{(r)}\right)^{transpose} \tag{3.10A}$$

for $r > 1$, and for $r = 1$,

$$w_j^{(1)} = \left(a_{j1}^{(1)}, a_{j2}^{(1)}, \ldots, a_{jm}^{(1)}\right)^{transpose}. \tag{3.10B}$$

Then the weight matrix for the network at the r^{th} time-period is given by

$$w^{(r)} = (w_1^{(r)}, w_2^{(r)}, \ldots, w_n^{(r)}), 1 \leq r \leq T. \tag{3.11}$$

Algorithm 3.1 (Initialization). Initialize network weights $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$ as follows, step-by-step, starting from the first time period:

(a) Find $w_j^{(1)}, 1 \leq j \leq n,$ by running a linear (or logistic if more sensitivity is required for some $y_j^{(1)}{}'s$) regression against the binary target $y_j^{(1)}$ with $x^{(1)}$ as the explanatory variable. Derive $v_j^{(1)}$ by (2.2).

(b) Given $x^{(2)} = (x_1^{(2)}, x_2^{(2)}, \ldots,\ x_m^{(2)})$ and $v^{(1)} = (v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)})$, find $w_j^{(2)}, 1 \leq j \leq n$ by running a linear (or logistic if more sensitivity is required for some $y_j^{(2)}{}'s$) regression against $y_j^{(2)}$ with the components of $x^{(2)}$ and $v^{(1)}$ as explanatory variables. Derive $v_j^{(2)}$ by (2.4).

(c) Repeat (b) to obtain the initial weights for $w_j^{(r)}$ at the r^{th} time period for $1 \leq j \leq n$ and $1 \leq r \leq T$.

3.3. Training the recurrent neural network

Given initial weights, network training involves a series of gradient descent searches, as described in the next algorithm. Let $w^{(r)}$ be the weight matrix as in (3.11) for the network at the r^{th} time period, i.e.,

$$w^{(r)} = (w_1^{(r)}, w_2^{(r)}, \ldots, w_n^{(r)}), 1 \leq r \leq T.$$

Algorithm 3.2 (Network training). Update network weights $w^{(r)}, 1 \leq r \leq T$, step-by-step, as described below

(a) Forward scoring: Randomly select a small batch of examples (1–10 loan accounts, for example) from the time series of the training sample and calculate p_{rj} by (2.3) using the current weights for $1 \leq r \leq T$ and $1 \leq j \leq n$.

(b) Select a time period r, from 1 to T in sequence. At the r^{th} time period, find the partial derivatives of L with respect to $a_{ij}^{(r)}$ and $b_{ij}^{(r)}$ by (3.8) and (3.9); then, calculate $\Delta w_j^{(r)}$ as follows

$$\Delta w_j^{(r)} = avg\left(\frac{\partial L}{\partial a_{j1}^{(r)}}, \frac{\partial L}{\partial a_{j2}^{(r)}}, \ldots, \frac{\partial L}{\partial a_{jm}^{(r)}}, \frac{\partial L}{\partial b_{j1}^{(r)}}, \frac{\partial L}{\partial b_{j2}^{(r)}}, \ldots, \frac{\partial L}{\partial b_{jn}^{(r)}}\right)^{Transpose}, 1 \leq j \leq T,$$

which is the average of the partial derivatives of L over the batch; hence, obtain the following weight matrix:

$$\Delta w^{(r)} = \left(\Delta w_1^{(r)}, \Delta w_2^{(r)}, \ldots, \Delta w_n^{(r)}\right).$$

(c) Select a learning rate η from a grid of values such that the update of $w^{(r)}$ is

$$w^{(r)} \Leftarrow w^{(r)} + \eta(\Delta w^{(r)}), \tag{3.12}$$

which gives rise to the biggest decrease for the cost function described by (3.2) over the entire training sample. Execute the update for $w^{(r)}$ by (3.12).

(d) Steps (a)–(c) are repeated until no material improvement is possible.

With the partial derivatives being evaluated over only one small batch of examples, these partial derivatives are called the mini-batch stochastic gradient for the cost function. This gradient can go off in a direction far from the batch gradient (i.e., the gradient over the entire training sample). Nevertheless, this noisiness is what we need for non-convex optimization [18,19] to escape from saddle points or local minima (Theorem 6 in [19]). The disadvantage is that more iterations are required to reach a good solution.

Remark 3.3. For Step (c) in Algorithm 3.2, there are better approaches for selecting a value for the learning rate η, rather than exhausting all possible values in the grid. For example, let η_i be the i^{th} value in the grid from 1 downward, and assume that currently η_i is the best learning rate so far, and it leads to a decrease for the cost function; stop the search for the learning rate and use η_i as the best learning rate, if η_{i+1} does not lead to a bigger decrease for the cost function than η_i.

4. Experimental results

In this section, we present the experimental results for the proposed RNN model, as benchmarked with two other Markov rating transition models.

The data we used constituted a synthetic sample, simulating a commercial loan portfolio with seven ratings $\{R_i\}_{i=1}^{7}$ over seven quarters (periods). At the end of each quarter, accounts are rated by one of seven ratings, with ratings R_6 and R_7 being, respectively, the withdrawal and default ratings. Both the default and withdrawal ratings are absorbed ratings that were excluded from later quarters for observation. For simplicity, we included only three covariates, which simulated the following drivers for a loan:

(a) Debt service coverage ratio

(b) Debt to tangible net worth ratio

(c) Current ratio

The sample contained 10,000 accounts. It was split 50:50 into training and validation. We focued on the following three models:

1) Model 1—The proposed RNN rating transition model;

2) Model 2—Time-inhomogeneous Markov transition model, with one separate transition model for each period;

3) Model 3—Time-homogeneous Markov transition model, with one single transition model for all periods.

All three model use the same covariates.

Let y_j denote a binary variable for a loan with a value of 1 if the loan has the rating R_j at the quarter end, and 0 otherwise. Let $(p_{1,} p_2, \ldots, p_7)$ be the multinomial probabilities for a loan estimated by a rating transition model at the beginning of a quarter, with p_j being the probability of transitioning to R_j at the quarter end.

Tables 1 and 2 below show the Gini coefficients, over the training and validation samples respectively, for each of the above three models for ranking each of these seven ratings individually. For example, in Table 1, for the RNN transition model over the training sample, it had a Gini of 0.84 for the ranking rating R_1. This Gini was calculated by using p_1 to predict $y_1$over the entire training sample. The results shown in these two tables demonstrate strong performance for the RNN transition model over the other two models.

Table 1. Gini by rating on training.

Model	Rating							
	1	2	3	4	5	6	7	Avg
1	0.84	0.69	0.62	0.48	0.54	0.50	0.68	0.62
2	0.73	0.45	0.39	0.24	0.37	0.37	0.55	0.44
3	0.53	0.33	0.19	0.15	0.20	0.32	0.53	0.32

Table 2. Gini by rating on vaildation.

Model	Rating							
	1	2	3	4	5	6	7	Avg
1	0.82	0.68	0.60	0.46	0.45	0.34	0.66	0.57
2	0.74	0.45	0.39	0.24	0.28	0.36	0.54	0.43
3	0.52	0.34	0.18	0.12	0.28	0.32	0.51	0.33

In the remainder of this section, we focus on the robustness of a model in terms of predicting the default event, and the quality of using p_7 to predict y_7, the default indicator. Tables 3 and 4 below show the Gini coefficients period by period, over the training and validation samples respectively, for ranking the default indicator over each of seven periods. Again, the RNN transition model significantly outperformed the other two benchmark models across all periods.

Table 3. Gini by period for default rating on training.

Model	Period							
	1	2	3	4	5	6	7	Avg
1	0.66	0.63	0.61	0.62	0.57	0.67	0.51	0.61
2	0.66	0.07	0.43	0.37	0.33	0.21	0.16	0.32
3	0.66	0.31	0.22	0.27	0.33	0.21	0.16	0.31

Table 4. Gini by period for default rating on validation.

Model	Period							
	1	2	3	4	5	6	7	Avg
1	0.64	0.62	0.58	0.55	0.46	0.55	0.53	0.56
2	0.64	0.05	0.35	0.28	0.25	0.04	0.20	0.26
3	0.64	0.27	0.11	0.18	0.25	0.04	0.20	0.24

The following six tables show the actual and predicted default rates for each model by decile over the training and validation samples. For example, Table 5 shows the actual and predicted default rates over the training sample for the RNN rating transition model. These values in the table were calculated by first sorting p_7 ascendingly, and then dividing the sample into 10 buckets, each of which was about 10%. The averages of the actual and predicted default rates over each bucket were taken.

Table 5. RNN on training (Gini-68%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	3.62%	4.90%	4.97%	5.33%	21.16%	23.79%	36.43%	42.47%	61.72%	81.19%
Pred	3.20%	4.33%	4.59%	5.63%	19.68%	23.24%	35.62%	44.20%	62.25%	82.85%

Table 6. RNN on validation (Gini-66%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	4.53%	4.89%	4.53%	8.49%	24.73%	24.03%	34.46%	45.04%	61.44%	81.09%
Pred	3.17%	4.31%	4.57%	6.45%	20.65%	23.49%	36.65%	45.87%	63.57%	84.49%

Table 7. One migration matrix per period on training (Gini-55%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	11.29%	12.50%	4.55%	7.03%	23.01%	26.99%	38.99%	32.67%	50.14%	78.42%
Pred	1.46%	5.04%	5.84%	6.19%	8.20%	15.54%	24.29%	49.62%	73.66%	95.73%

Table 8. One migration matrix per period on validation (Gini-54%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	11.87%	14.89%	5.25%	8.42%	22.65%	26.40%	39.86%	34.39%	51.80%	77.71%
Pred	1.52%	5.16%	5.97%	6.36%	8.58%	16.22%	24.82%	51.71%	76.60%	96.29%

Table 9. One single migration matrix on training (Gini-53%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	19.11%	4.90%	12.93%	4.97%	19.03%	29.05%	27.70%	39.28%	50.07%	78.57%
Pred	2.52%	5.36%	6.20%	6.71%	9.82%	21.95%	25.37%	35.67%	75.51%	96.46%

Table 10. One single migration matrix on validation (Gini-51%).

Decile	0	1	2	3	4	5	6	7	8	9
Actual	21.30%	5.90%	14.46%	4.53%	19.41%	29.06%	29.50%	39.64%	52.09%	77.35%
Pred	2.52%	5.49%	6.35%	6.87%	10.39%	22.60%	25.81%	37.94%	78.29%	96.97%

These results demonstrate a significant improvement for the RNN model over the other two models, either on training or validation.

5. Conclusions

A rating transition for a credit portfolio is generally path dependent. A Markov rating transition model, either homogeneous or inhomogeneous, usually does not perform well after projecting for a few periods. The RNN model proposed in this paper provides a solution for modeling state transitions under non-Markov settings. This RNN is informed by the information history along the path. The experiments show that this proposed RNN model significantly outperforms Markov models where path dependence is relevant.

Acknowledgements

The author thanks Biao Wu for many valuable discussions in the last 3 years in deep machine learning, Python financial engineering, as well as his insights and comments. Thanks also go to Felix Kan for many valuable comments, to Zunwei Du, Kaijie Cui, and Glenn Fei for many valuable conversations.

Conflict of interest

The views expressed in this article are not necessarily those of the banks the author works with. Please direct any comments to the author Bill Huajian Yang at: h_y02@yahoo.ca.

References

1. Yang BH, Du Z, (2016) Rating transition probability models and CCAR stress testing. *J Risk Model Validation* 10: 1–19. https://doi.org/10.21314/JRMV.2016.155
2. Yang BH, (2017) Forward ordinal models for point-in-time probability of default term structure. *J Risk Model Validation* 11: 1–18. https://doi.org/10.21314/JRMV.2017.181

3. Dos Reis G, Pfeuffer M, Smith G, (2020) Capturing model risk and rating momentum in the estimation of probabilities of default and credit rating migrations. *Quant Finance* 20: 1069–1083. https://doi.org/10.1080/14697688.2020.1726439
4. Miu P, Ozdemir B, (2009) Stress testing probability of default and rating migration rate with respect to Basel II requirements. *J Risk Model Validation* 3: 3–38. https://doi.org/10.21314/JRMV.2009.048
5. Kiefer NM, Larson CE, (2004) *Testing Simple Markov Structures for Credit Rating Transitions*. Comptroller of the Currency.
6. Kiefer NM, Larson CE, (2007) A simulation estimator for testing the time homogeneity of credit rating transitions. *J Empirical Finance* 14: 818–835. https://doi.org/10.1016/j.jempfin.2006.08.001
7. Juhasz P, Vidovics-Dancs A, Szaz J, (2017) Measuring path dependency. *UTMS J Econ* 8: 29–37.
8. Russo E, (2020) A discrete-time approach to evaluate path-dependent derivatives in a regime-switching risk model. *Risks* 8: 9. https://doi.org/10.3390/risks8010009
9. Yang BH, (2017) Point-in-time PD term structure models for multi-period scenario loss projection. *J Risk Model Validation* 11: 73–94. https://doi.org/10.21314/JRMV.2017.164
10. Zhu S, Lomibao D, (2005) A conditional valuation approach for path-dependent instruments. *SSRN Electron J*. https://doi.org/10.2139/ssrn.806704
11. Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J, (2009) A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans Pattern Anal Mach Intell* 31: 855–868. https://doi.org/10.1109/TPAMI.2008.137
12. Tealab A, (2018) Time series forecasting using artificial neural networks methodologies: A systemati review. *Future Comput Inf J* 3: 334–340. https://doi.org/10.1016/j.fcij.2018.10.003
13. Hyötyniemi H, (1997) *Proceedings of STeP'96*, (eds. Jarmo Alander, Timo Honkela and Matti Jakobsson), Publications of the Finnish Artificial Intelligence Society, 13–24.
14. Elman JL, (1990) Finding structure in time. *Cognitive Sci* 14: 179–211. https://doi.org/10.1016/0364-0213(90)90002-E
15. Schmidhuber J, (2015) Deep learning in neural networks: An overview. *Neural Networks* 61: 85–117. https://doi.org/10.1016/j.neunet.2014.09.003
16. Abiodun OI, Jantan A, Omolara AE, Dada KV, Mohamed NA, Arshad H (2018), State-of-the-art in artificial neural network applications: A survey, *Heliyon* 4: e00938. https://doi.org/10.1016/j.heliyon.2018.e00938
17. Dupond S (2019), A thorough review on the current advance of neural network structures. *Annu Rev Control* 14: 200–230.
18. Bottou L, (2010) Large-scale machine learning with stochastic gradient descent, In *Proceedings of COMPSTAT'2010* Physica-Verlag HD, 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16
19. Ge R, Huang F, Jin C, Yuan Y, (2015) Escaping from saddle points-online stochastic gradient for tensor decomposition, In *Conference on Learning Theory*, PMLR, 1–46.