Modeling Path-Dependent State Transition by a Recurrent Neural Network

Yang, Bill Huajian

18 August 2022

Modeling Path-Dependent State Transition by a Recurrent Neural Network

BILL HUAJIAN YANG [1]

Abstract

Rating transition models are widely used for credit risk evaluation. It is not uncommon that a time-homogeneous Markov rating migration model deteriorates quickly after projecting repeatedly for a few periods. This is because the time-homogeneous Markov condition is generally not satisfied. For a credit portfolio, 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 networks 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 behaviours for a loan along the path. Information learned from previous periods propagates to future periods. Experiments show 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 financial industry for credit risk evaluation, including stress testing and IFRS 9 expected credit loss evaluation ([11], [12], [16], [17]), under the assumption that rating transition is a time-homogenous Markov process, depending only on current rating and covariates. However, it is not uncommon that a Markov model deteriorates quickly after projecting for a few periods. This is because Markov condition is generally not satisfied. Rating transition for a credit portfolio is generally path dependent. A test for this assumption is required ([9], [10]) for a use of these Markov models.

There are various methods for path-dependent credit risk modeling ([8]), including regime-switching models ([13]) and the conditional methods ([18], [19]). 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 networks 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 ([1], [3], [4], [6], [7], [14], [15]).

The network structure for the proposed RNN model is described in section 2 by (2.1)- (2.4). This RNN model is implemented in Python. Experiments show 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 set up the proposed RNN model. In section 3, we calculate the partial derivatives for the network cost function. In section 4, we present the experiment

[1] Please direct all your comments to Bill Huajian Yang at h_y02@yahoo.ca

results for this proposed RNN model, 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, as in (2.1) - (2.4), the proposed recurrent neural network (RNN) model 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 transiting 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., transition probability depends only on current rating and covariates. The type of Markov transition models includes:

(a) Time-homogeneous Markov rating transition, represented by one single transition model for all periods
(b) Time-inhomogeneous Markov rating transition, represented by one transition model for each period

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

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

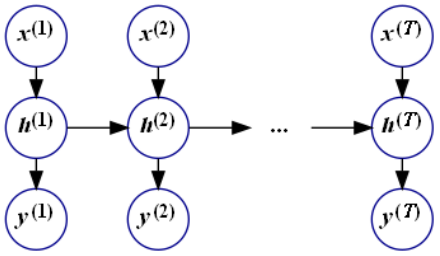


Figure 1. An RNN for rating transition

An RNN shares the advantages of common neural networks that 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 withdraw rating and R_n the default ratings. Both default and withdraw ratings are assumed to be absorbed states, which means, a loan rated by a default or withdraw 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 value 1 if the rating for a loan at the end of i^{th} period is R_j and 0 otherwise. Let n_p denote the number of non-absorbed ratings. An input at 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 $(i-1)^{th}$ period:

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

That is, non-absorbed rating observed at the end of $(i-1)^{th}$ period is used as the input for the next period. The output at 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 in (2.1) - (2.4) below. Initially, at the first period, we have:

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

(b) Output: $y^{(1)}$=$(y_1^{(1)}, y_2^{(1)}, \dots, y_n^{(1)})$, a unit vector (all components are zero except for one with value 1), which 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 at 1st period, which is stored at hidden neurons $h^{(1)} = (h_1^{(1)}, h_2^{(1)}, \dots, h_n^{(1)})$ in the 1st period.

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

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

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

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

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

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

1) Collects input $x^{(i)}$, and information $h^{(i-1)}$ learned up to the end of $(i-1)^{th}$
2) Learns from $x^{(i)}$ and $h^{(i-1)}$and stores the learned information at hidden neurons $h^{(i)}$
3) Derive multinomial probability $(p_{i1}, p_{i2}, \dots, p_{in})$, where p_{ij} is the probability for the event transiting 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. 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 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 k^{th} period. The cost function at k^{th} period is denoted by L_k, which is given as (for one single data point):

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

This is the negative log-likelihood for observing 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:

$$L = L_1 + L_2 + \cdots + L_T, \quad (3.2)$$

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.

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:

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

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

$$\begin{aligned} d_i^{(k,0)} &= \partial L_k / \partial v_i^{(k)} = -\sum_{j=1}^n y_j^{(k)} \, \partial[\log(p_{kj})] / \partial v_i^{(k)} \\ &= -y_i^{(k)}(1-p_{ki}) + \sum_{j \neq i}^n y_j^{(k)} p_{ki} = -\left(y_i^{(k)} - p_{ki}\right), \end{aligned} \tag{3.4}$$

where equation $\sum_{j=1}^n y_j^{(k)} = 1$ 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):

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

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

$$\begin{aligned} d_i^{(k,r)} &= \frac{\partial L_k}{\partial v_i^{(k-r)}} \\ &= \sum_{j=1}^n \left(\frac{\partial L_k}{\partial v_j^{(k-r+1)}}\right)\left(\frac{\partial v_j^{(k-r+1)}}{\partial v_i^{(k-r)}}\right) \\ &= \sum_{j=1}^n b_{ji}^{(k-r+1)} d_j^{(k,r-1)}. \end{aligned} \tag{3.5}$$

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

We will use the fact:

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

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 by (3.6):

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

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

$$D_i^{T-1} = \frac{\partial L}{\partial v_i^{(T-1)}} = \frac{\partial (L_T + L_{T-1})}{\partial v_i^{(T-1)}}$$

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

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

$$= \sum_{j=1}^{n} b_{ji}^{(T)} D_j^T + d_i^{(T-1,0)},$$

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:

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

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

$$= \sum_{j=1}^{n} \frac{\partial L}{\partial v_j^{(T-r+1)}} \frac{\partial v_j^{(T-r+1)}}{\partial v_i^{(T-r)}} + d_i^{(T-r,0)}$$

$$= \sum_{j=1}^{n} b_{ji}^{(T-r+1)} D_j^{(T-r+1)} + d_i^{(T-r,0)},$$

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

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 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 subsection, 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 r^{th} time-period, i.e.,

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

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

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

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

$$w^{(r)} = (w_1^{(r)}, w_2^{(r)}, \dots, w_n^{(r)}), 1 \le r \le T. \quad (3.11)$$

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

(b) Given $x^{(2)} = (x_1^{(2)}, x_2^{(2)}, \dots, x_m^{(2)})$, and $v^{(1)} = (v_1^{(1)}, v_2^{(1)}, \dots, v_n^{(1)})$, find $w_j^{(2)}, 1 \le j \le n,$ by running a linear (or a logistic if more sensitivity is required for some $y_j^{(2)\prime}s$) regression against $y_j^{(2)}$ with 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 r^{th} time-period for $1 \le j \le n$ and $1 \le r \le T$.

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 r^{th} time-period, i.e.:

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

Algorithm 3.2 (Network training). Update network weights $w^{(r)}, 1 \le r \le T,$ 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 training sample, calculate p_{rj} by (2.3) using the current weights for $1 \le r \le T$ and $1 \le j \le n$.

(b) Select a time-period r, from 1 to T in sequence. At 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:

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

the average of the partial derivatives of L over the batch, hence, obtain the 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 $w^{(r)}$ by:

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

gives rise to the biggest decrease for cost function (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 ([2], [5]) to escape from saddle points or local minima (Theorem 6 in [5]). 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 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, 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. **Experiment results**

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

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

(a) Debt service coverage ratio
(b) Debt to tangible net worth ratio
(c) Current ratio

The sample contains 10,000 accounts. It splits by 50:50 into training and validation. We focus 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 value 1 if the loan has the rating R_j at the quarter end and is 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 transiting 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 7 ratings individually. For example, in Table 1, for the RNN transition model over the training sample, it has a Gini of 0.84 for ranking rating R_1. This Gini is calculated by using p_1 to predicted $y_1$over the entire training sample. Results shown in these two tables demonstrate a strong performance for the RNN transition model over the other two models.

Table 1. Gini by rating on training

	Rating							
Model	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 validation

	Rating							
Model	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 remaining of this section, we focus on the robustness of a model in predicting the default event, 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 default indicator over each of 7 periods. Again, the RNN transition model significantly outperforms other two benchmark model across all periods.

Table 3. Gini by period for default rating on training

	Period							
Model	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

	Period							
Model	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 a table are calculated by first sorting p_7 ascendingly, then dividing the sample into 10 buckets, each is about 10%. Averages of the actual and predicted default rates over each bucket are taken.

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

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

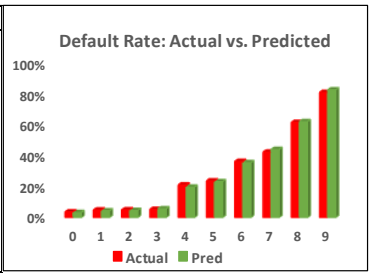


Table 6. RNN on validation (Gini - 66%)

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

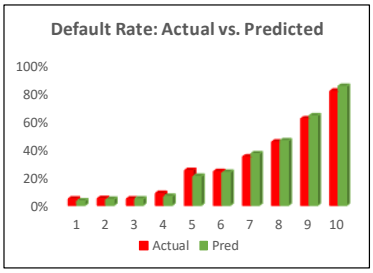


Table 7. One migration matrix per period on training (Gini-55%)

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

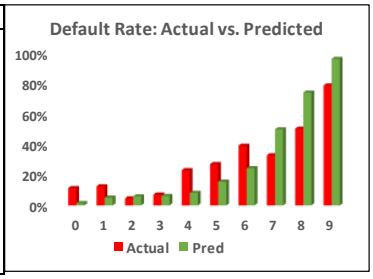


Table 8. One migration matrix per period on validation (Gini-54%)

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

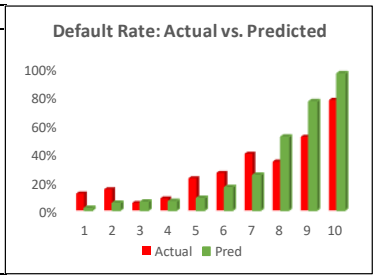


Table 9. One single migration matrix on training (Gini-53%)

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

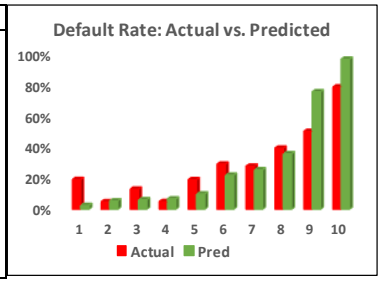
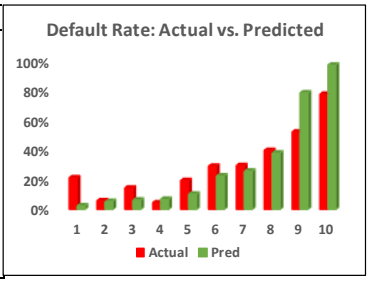


Table 10. One single migration matrix on validation (Gini-51%)

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



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

Conclusion. 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 transition under non-Markov settings. This RNN is informed by the information history along the path. Experiments show this proposed RNN model significantly outperform Markov models where path-dependence is relevant.

Acknowledgements: The author thanks Biao Wu for many valuable discussions in the past 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.

References

[1] Abiodun, Oludare Isaac; Jantan, Aman; Omolara, Abiodun Esther; Dada, Kemi Victoria; Mohamed, Nachaat Abdelatif; Arshad, Humaira. (2018). State-of-the-art in artificial neural network applications: A survey. Heliyon. 4 (11): e00938. doi:10.1016/j.heliyon.2018.e00938.

[2] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[3] Dupond, Samuel. (2019). A thorough review on the current advance of neural network structures. Annual Reviews in Control. 14: 200–230.

[4] Elman, Jeffrey L. (1990). Finding Structure in Time. Cognitive Science. 14 (2): 179–211. doi:10.1016/0364-0213(90)90002-E.

[5] Ge, R., Huang, F., Jin, C., & Yuan, Y. (2015, June). Escaping From Saddle Points-Online Stochastic Gradient for Tensor Decomposition. In COLT (pp. 797-842).

[6] Graves, Alex; Liwicki, Marcus; Fernandez, Santiago; Bertolami, Roman; Bunke, Horst; Schmidhuber, Jürgen. (2009). A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. doi:10.1109/tpami.2008.137.

[7] Hyötyniemi, Heikki. (1996). Turing machines are recurrent neural networks. Proceedings of STeP '96/Publications of the Finnish Artificial Intelligence Society: 13–724.

[8] Juhasz, Peter; Varadi, Kata; Vidovics-Dancs, Agnes; and Szaz, Janos. (2017). Measuring Path Dependency. Special issue, UTMS Journal of Economics 8 (1): 29–37.

[9] Kiefer, Nicholas M.; Larson, C. Erik. (2014). Testing Simple Markov Structures for Credit Rating Transitions. OCC Economics Working Paper 2004-3

[10] Kiefer, Nicholas M.; Larson, C. Erik. (2007). A simulation estimator for testing the time homogeneity of credit rating transitions. Journal of Empirical Finance, Elsevier, vol. 14(5), pages 818-835

[11] Miu, P.; Ozdemir, B. (2009). Stress testing probability of default and rating migration rate with respect to Basel II requirements. Journal of Risk Model Validation, Vol. 3 (4), 3-38

[12] Reis, G. dos; Pfeuffer, M.; Smith, G. (2020). Capturing model risk and rating momentum in the estimation of probabilities of default and credit rating migrations, Quantitative Finance, 20:7, 1069-1083, DOI: 10.1080/14697688.2020.1726439

[13] Russo, Emilio. (2020). Discrete-Time Approach to Evaluate Path-Dependent Derivatives in a Regime-Switching Risk Model, Risks 2020, 8(1), 9; doi:10.3390/risks8010009

[14] Schmidhuber, Jürgen. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks. 61: 85–117.

[15] Tealab, Ahmed. (2018). Time series forecasting using artificial neural networks methodologies: A systematic review. Future Computing and Informatics Journal. 3 (2): 334–340. doi:10.1016/j.fcij.2018.10.003. ISSN 2314-7288.

[16] Yang, B. H.; Zunwei Du. (2016). Rating Transition Probability Models and CCAR Stress Testing. Journal of Risk Model Validation 10 (3), 1-19

[17] Yang, B. H. (2017). Forward ordinal models for point-in-time probability of default term structure. Journal of Risk Model Validation, Vol 11 (3), 1-18

[18] Yang, B. H. (2017). Point-in-time PD term structure models for multi-period scenario loss projection. Journal of Risk Model Validation, Vol 11 (1), 73-94

[19] Zhu, Steven; Lomibao, Dante. (2005). A Conditional Valuation Approach for Path-Dependent Instruments. August 2005SSRN Electronic Journal. DOI: 10.2139/ssrn.806704