GLOBAL RISK INSTITUTE

FINANCIAL INNOVATION SERIES

ARTIFICIAL NEURAL NETWORKS IN FINANCIAL MODELLING

AUTHORS: **Alex LaPlante,**
Former Managing Director, Research, Global Risk Institute

Alexey Rubtsov,
Research Associate, Global Risk Institute

Abstract. *In this paper we discuss Artificial Neural Networks (ANNs), one of the most common and complex tools Machine Learning tools. After a non-technical introduction to ANNs, we examine the risks associated with the application of this algorithm to financial data.*

INTRODUCTION

The last decade has witnessed tremendous growth in the adoption of Machine Learning (ML) tools in the financial services industry, with global investment in Artificial Intelligence start-ups rising from $282 million in 2011 to $2.4 billion in 2015.[1] Concurrently, financial regulators and market participants have been raising concerns regarding the risks that such a widespread and rapid adoption may entail. In this paper we examine one of the most common tools of ML, called Artificial Neural Networks (ANNs), and discuss associated risks of applying these tools to financial modelling.[2]

Results of successful applications of ANNs in finance have been widely reported in academic and non-academic literature. In particular, they have been employed in credit-risk evaluation (Baesens et al., 2003); bank failure prediction (Tam and Kiang, 1992); assessment of the risk of management fraud in financial statements (Green and Choi, 1997); prediction of yield curve dynamics (Kondratyev, 2018), assets pricing (Gu et al., 2018), implied volatility movements (Cao et al, 2019), etc..

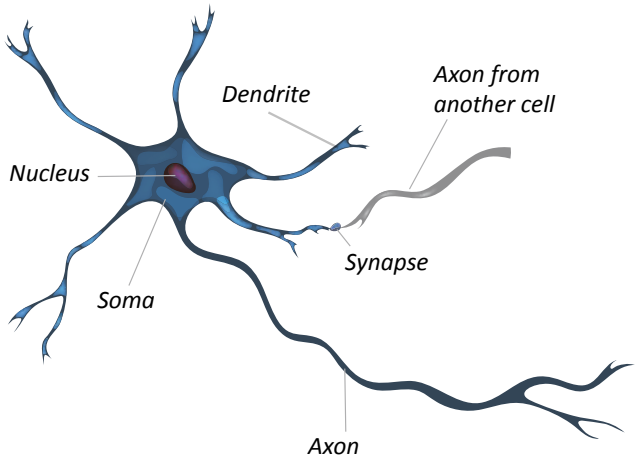
A BASIS IN BIOLOGY

As its name suggests an Artificial Neural Network (ANN) is a mathematical model of the biological neurons that constitute our brain. Before we discuss ANNs it is important to first understand how a single neuron functions. In this section, we discuss a biological neuron and describe a mathematical model of an artificial neuron.

Biological neurons are undoubtedly complex but to put it simply neurons can receive signals from other neurons and if the total signal it receives from all connected neurons is greater than some threshold the neuron excites and transmits an output signal. If the threshold is not met, no signal is transmitted. Schematically, a biological neuron consists of (see also Figure 1):

[1] *See the Financial Stability Board report "Artificial Intelligence and Machine Learning in Financial Services", November 1, 2017.*

[2] *Although there exists many types of ANNs (e.g., recurrent neural networks), we confine our analysis to feedforward multilayer neural networks.*



- **Dendrites** - *receivers of inputs from other neurons;*
- **Soma** - *body of a neuron that also includes the nucleus;*
- **Axon** - *transmitter of the signal generated by the neuron to other neurons;*
- **Synapses** - *connections between axons of other neurons and dendrites. An important role of synapses is that they also determine the strength of the signal received from other neurons.*

Figure 1. Biological neuron

In 1958, Frank Rosenblatt developed a mathematical model of a biological neuron, called a perceptron, which resembles the way biological neurons function in that it:

1. *Receives numeric inputs (signals);*
2. *Multiplies inputs by corresponding (synaptic) weights and sums theses values to produce the total input value;*
3. *Produces an output of 1 if the total input value is greater than 0 (that is, the neuron is activated); otherwise produces an output of 0. The function that transforms a given input value into an output value is called the* ***activation function*** *of the neuron since it determines the level of 'activity' of the neuron (0 - not active, 1 - active).*

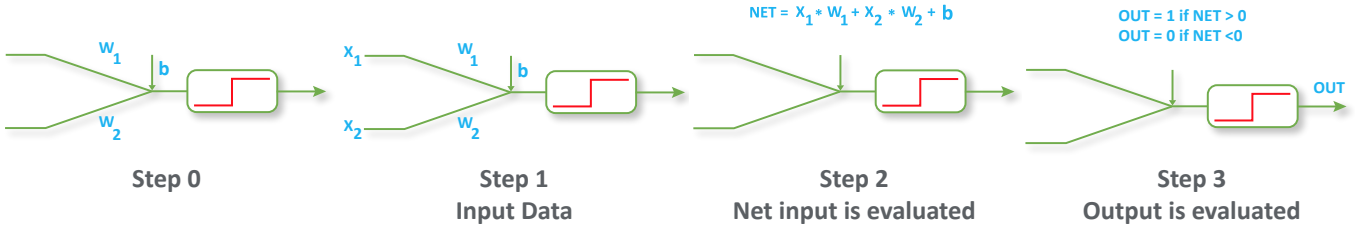
The zero threshold at step (3) is quite restrictive in modelling. To ameliorate this issue, one can add a ***bias*** to the total signal at step (2) allowing the user to decide the neuron's activation value. To avoid changing steps (1)-(3), an artificial input of 1 with the synaptic weight ***b*** (for "bias") is added to the neuron.

$$\text{Output} = \begin{cases} 1, & \textit{if total input} > 10 \\ 0, & \textit{otherwise} \end{cases}$$

To illustrate this idea, let's assume that we want the neuron to activate (output=1) if the total signal is greater than 10 (instead of 0 as before) and to output 0 if the total signal is less than or equal to 10.

This can be achieved by setting the bias to ***b=***-10 in step (2). Before passing the total input to step (3), we add the bias. If the total input is 12, for example, the final value passed to step (3) will be 2. Consequently, the final output at step (3) will be 1. For additional clarity, the animation in Figure 2 provides a visualization of this process. As shown, this simple assumption of an artificial input adds more flexibility to the way a neuron functions.

Figure 2. Artificial neuron



Example 1

Let $x_1=1$, $x_2=2$ be the two inputs, $w_1=-2$, $w_2=-7$ be the (synaptic) weights for each input, and $b=-10$ be the bias. Then the total input (*NET*) received by the neuron is:

$$NET = \underbrace{1}_{input\ 1} \times \underbrace{(-2)}_{weight\ 1} + \underbrace{2}_{input\ 2} \times \underbrace{7}_{weight\ 2} + \underbrace{(-10)}_{bias} = 2$$

As a result, the neuron output is 1 because the total input is greater than 0. We can formally write $OUT=F(2)=1$, where $F(.)$ is an activation function taking on values 0 or 1 (step (3) above).

ARTIFICIAL NEURAL NETWORKS

Similar to biological neurons, Artificial Neurons are comprised of:

- **Inputs:** *a set of numeric values;*
- **Weights:** *values that determine the significance of each input value;*
- **Bias:** *the weight of an artificial input of 1;*
- **Activation function:** *function that accepts the total input and produces the output according to some pre-specified rule.*

Employing the model of a single neuron, we can already solve simple classification problems.

Example 2

Let us assume that we want to build a classifier that is able to distinguish between *"large assets/low equity"* and *"small assets/high equity"* banks. We have data on the asset values and equity capital of four banks (see Table 1).

It is clear from the table that one possible class is *"large assets/low equity"* (banks 1 and 2) while another is *"small assets/high equity"* (banks 3 and 4). It turns out that if we appropriately choose the weights of the neuron, then the neuron will produce 0 only when the banks of the first class are processed by the neuron and will produce 1 for banks 3 and 4. For example, let w_1=-0.5, w_2=0.5, and b=300. Then for bank 1 we have:

Table 1. *Assets and Equity Capital for two banks (synthetic data)*

Bank	Assets	Equity Capital	Class (Desired Output)
Bank 1	1,000	10	0
Bank 2	1,200	20	0
Bank 3	500	200	1
Bank 4	800	300	1

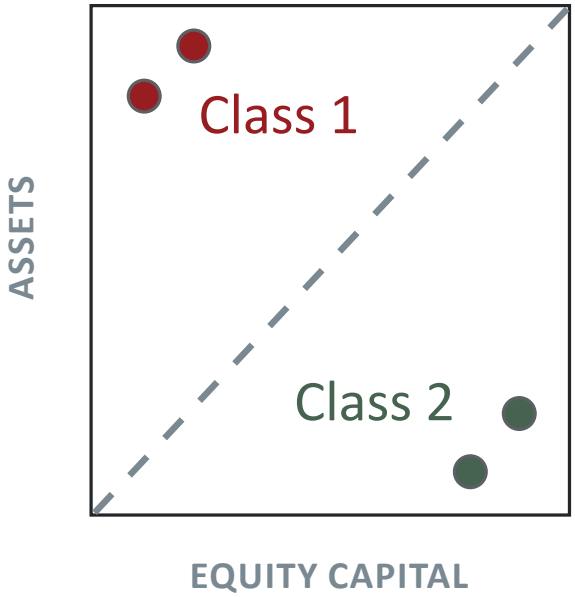
$$NET = \underbrace{1{,}000}_{Assets} \times \underbrace{(-0.5)}_{weight} + \underbrace{10}_{Eq.\ Capital} \times \underbrace{0.5}_{weight} + \underbrace{300}_{bias} = -195$$

This means the neuron's output is 0. Similarly, the output for bank 2 is 0, while the output for the other two banks is 1.

The procedure of finding the appropriate neuron weights to achieve the desired output is referred to as ***training*** and the data used in this process is known as the ***training data*** or ***training set***. In this example, the data in Table 1 was used as the training data. Once trained, the neuron can then be used to classify new banks that were not included in the training set.

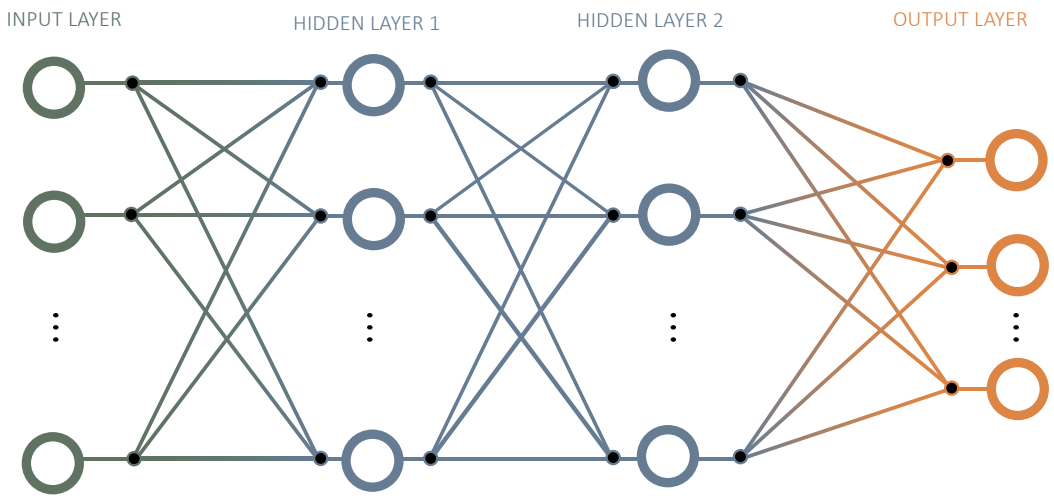
The crucial feature of the data that we exploited in this example is that the classes are linearly separated, that is, the classes can be separated by a line in Equity Capital-Assets space (see Figure 3)

Figure 3. An example of banks data that can be linearly separated



In general, the goal of training is to find the weights and bias of the neuron that will reduce the error, which is defined as the squared difference between desired and actually produced outputs. However, what if the data are not linearly separated? To answer this question, we need to consider many neurons, or a neural network.

An average human brain consists of about 10^{11} neurons. Together, these neurons form a connected network—a neural network—that allows our brains to perform highly sophisticated functions. Similar to biological neural networks, ANNs can become much more powerful when they contain many neurons as opposed to the one-neuron model described in our previous example. In general, ANNs consist of input, hidden, and output layers (see Figure 4). ANNs with many hidden layers (also called deep neural networks) have better generalization capabilities but require large amounts of training data to be able to capture the relationships present in the data.



Determining the optimal network configuration (i.e. the number and type of neuronal layers and the number of neurons in each layer) requires a mix of knowledge, intuition, and experimentation and is very much an iterative process. There are no hard-and-fast rules around the specification of an ANN and the ultimate configuration will be dependent on the problem at hand and the data being used. However, there are several rules of thumb that are often useful in determining a starting point. Table 2 below lists some examples of such rules; one may be more suitable than another depending on the problem specification (number of inputs vs. number of outputs, etc.).

Table 2. *General ANN Configuration*[3]

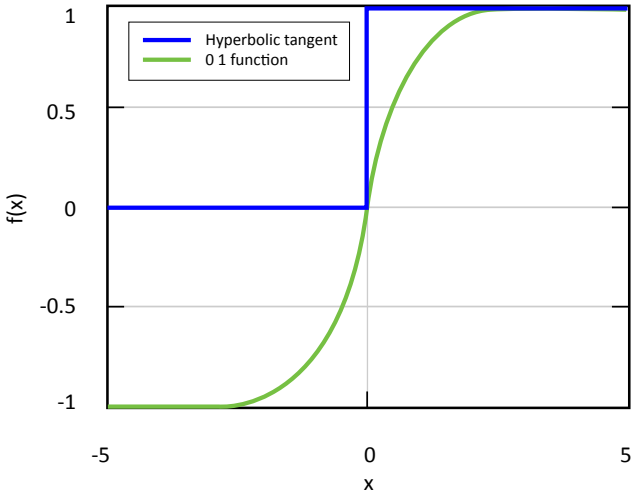
Number of Layers	Number of Nodes
INPUT LAYER	
There is one input layer in every ANN	Determined by the shape of your training data: *# nodes = # features in training set*[4]
HIDDEN LAYER	
There is no general rule of thumb for determining the optimal number of hidden layers, but it is useful to keep in mind the following: 0 Only capable of representing linear separable functions 1 Can approximate any function that contains a continuous mapping from one finite space to another 2 Can represent an arbitrary decision boundary with rational activation functions and approximate any smooth mapping >2 Additional layers can learn complex relationships in data	Possible rules of thumb: 1. $\#\ nodes_{output} \leq \#\ nodes_{hidden\ layer} \leq \#\ nodes_{input}$ 2. $\#\ nodes_{hidden\ layer} < 2 \cdot \#\ nodes_{input}$ 3. $\#\ nodes_{hidden\ layer} = (2/3 \cdot \#\ nodes_{input}) + \#\ nodes_{output}$
OUTPUT LAYER	
There is one output layer in every ANN	Dependent on model configuration: whether your ANN is a regressor (returns a value) or a classifier (returns a class label) • Regressor: output layer has a single node • Classifier: output layer has a single node or output layer has one node per class label

[3] *See Heaton (2008)*
[4] *Add one additional node if using a bias term*

Once an initial competent architecture for the network is determined, the configuration can then be iteratively tuned using ancillary algorithms. One such family of algorithms known as ***pruning*** eliminates unnecessary or redundant nodes within the network.

In addition to changes in configuration, we can solve more complicated problems if we change some of the other assumptions underlying the simple neuron that we described in the previous section. For instance, if we change the activation function to a function that can take on values other than 0 and 1, we can solve a function approximation problem, that is, construct a function that passes through a given set of points. An example of a common alternative activation function specification is the hyperbolic tangent function which takes on all values between-1 and 1 (Figure 5).

Figure 5. Possible activation functions of a neuron



To illustrate the idea of function approximation, let us consider the following example.

Example 3

Assume that we want to use an ANN to approximate the following function:

$$f(x)=3x^4-4x^3-12x^2+5 \qquad (1)$$

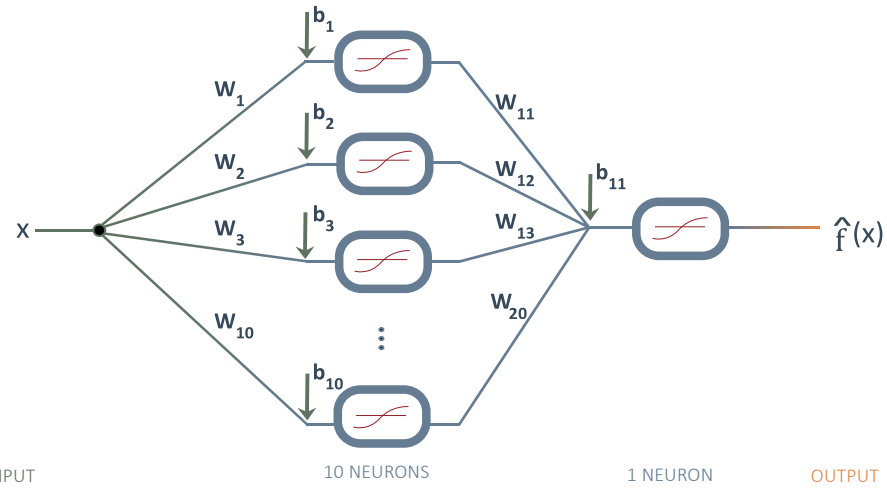
However, we do not know the relationship defined in (1), but instead we know only five observations derived from (1) as shown in Table 3. In other words, our goal is to recover (1) from the available data.

Table 3. *Values of f(x) at five points.*

x	*-2*	*-1*	*0*	*2*	*3*
f(x)	*36*	*0*	*5*	*-27*	*30*

To achieve this, we construct an ANN that consists of 10 neurons in the hidden layer, and 1 neuron in the output layer (see Figure 6).[5]

Figure 6. The ANN used to approximate f(x).



[5] *It has been shown in Cybenko (1989) that only one hidden layer is enough to approximate any continuous function on a given set.*

This ANN is then trained with the available data, or ***training set*** (Table 3). Once trained, we can use this ANN to predict the other function values as depicted by the blue line in Figure 7.

At this point it is very important to recognize that there is obviously an infinite number of ways to smoothly connect the red points. Moreover, the training dataset in Table 3 contains no information about the function's behavior "beyond the scope" of the dataset, that is, there is no information available to an ANN about the behaviour of the function for, say, ***x*** between ***0.5*** and ***1.5 or when x>3*** (Figure 8).

The ANN has relatively good performance in proximity to the training data points but the predictive power of the ANN worsens as we move "beyond the scope" of the training dataset. For example, the training can be considered quite poor for the data points between ***0.5*** and ***1.5***, and even worse when x is greater than 3. A number of approaches have been developed to deal with such problems. One such approach, assuming enough data is available, is to split the data into training, validation, and test sets where training data is used to train the network, validation data is used to determine when the training should stop, and test data is used to assess the quality of the trained network.[6] In this respect it should be emphasized that none of the approaches guarantees that the situation shown in Figure 8 will not occur. Thus, one should perform an extensive study of the behavior of the trained ANN before applying it.

Figure 7. Function approximation: the goal is to approximate the function (blue curve) based on only a given set of points (red points)

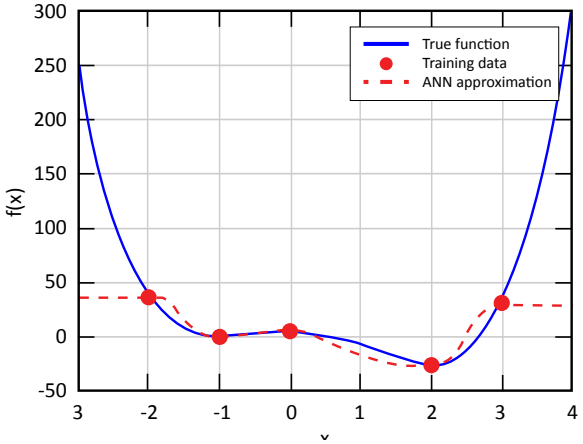
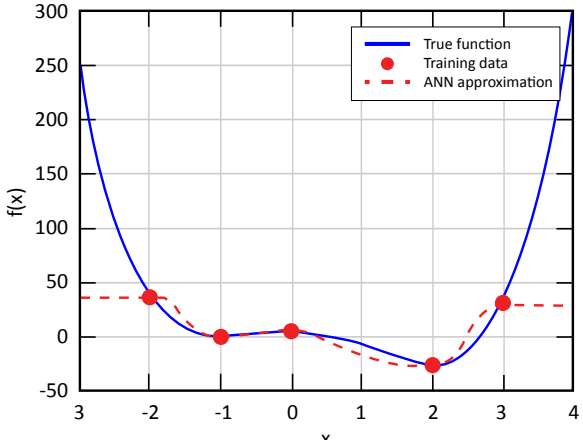


Figure 8. The output of the trained ANN shown in Figure 6 based on the training dataset given in Table 3.



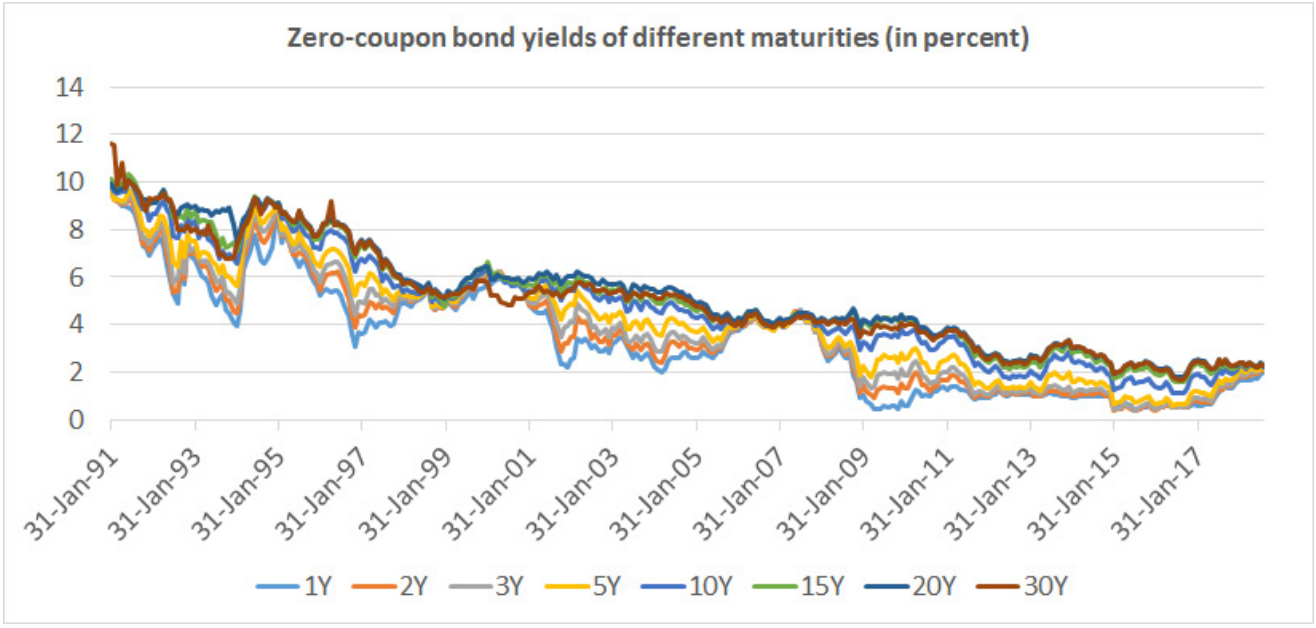
[6] *See also https://www.mathworks.com/help/deeplearning/ug/divide-data-for-optimal-neural-network-training.html*

ANN IN YIELD CURVE MOVE PREDICTION

In this section we apply an ANN to predict the moves of the zero-coupon yield curve. A similar analysis is done in Kondratyev (2018), who applied an ANN to model the dynamics of term structures of forward crude oil prices and interest rate swap rates. We would like to emphasize that the purpose of our analysis in this section is not to solve the problem of curve move prediction, but to highlight the potential risks associated with the use of ANNs. The data used in this example was downloaded from the Bank of Canada website and covers the period from January 2, 1991 until September 5, 2018.[7] The yield curves for 8 different maturities, sampled at the end of each month (monthly frequency), are shown in Figure 9.

As shown in Figure 9, the times series are highly correlated and the smaller the difference in maturities the higher the correlation. Every date in the figure is associated with eight data points, one for each maturity, that together represent the ***zero-coupon yield curve*** at that date. Predicting yield curve dynamics is a well-known challenge in quantitative risk management. In this exercise, we would like to answer the following question:

Figure 9. Zero-coupon bond yields of different maturities



How will the entire yield curve move when only one point on the curve moves by a given amount?

To answer this question, we will use two approaches: ANNs and Principal Component Analysis (PCA). PCA is a typical modelling tool for highly correlated time series and will serve as our benchmark.

[7] *See https://www.bankofcanada.ca/rates/interest-rates/bond-yield-curves/*

Principal Component Analysis (PCA). Since the time series of the yields are highly correlated, it is common practice to apply PCA to reduce the number of time series and work with only 2-3 principal components (PCs).[8] In a nutshell, PCA transforms the original 8 times series in our example into 8 different time series called principal components (PCs). PCs have the property that the first PC explains most of the variation in the original data, the second PC explains most of the ***remaining*** variation not explained by the first PC, and so on. In practice, 3 principal components usually explain more than 90% of the variation in the original time series.

The PCA conducted on the time series shown in Figure 9 indicates that the first 3 PCs explain 97% of the variation. Furthermore, the PCA analysis also indicates that the 5-year yield has the largest variance implying that it has the most influence on the dynamics of the entire curve relative to the yields of other maturities that we consider. Based on this observation, the PCA prediction of the curve move is defined as the transformation of the initial curve suggested by the change in the first principal component that makes the 5-year yield move by a pre-specified amount. For example, assume we want to see how the current curve moves if the 5-year yield increases by 20 basis point. To obtain the curve move, we change the first principal component until the 5-year yield becomes exactly 20 basis points. The yield curve that results from such a change in the first principal component is the PCA prediction of the curve move.

Artificial Neural Networks. The ANN that we will employ for this example has the following specification:

- *Number of inputs: 9*
- *Number of outputs: 8*
- *Number of hidden layers: 2*
- *Number of neurons in each hidden layer: 10*
- *Activation function: hyperbolic tangent*

As previously noted there is no precise rule that allows one to determine the specification of the ANN. In general, the optimal structure of a neural network is determined through experimentation.

The input and output vectors have the following form

$$Input: \left(Y_{T_1}(t),\ldots,Y_{T_8}(t),\Delta Y_{T_k}(t+\Delta t)\right)$$

$$Output: \left(\Delta Y_{T_1}(t+\Delta t),\ldots,\Delta Y_{T_8}(t+\Delta t)\right)$$

where $Y_{Ti}(t)$ is the time t yield on a zero-coupon bond with maturity T_i, $\Delta Y_{Tk}(t+\Delta t)$ is the change in the yield of the zero-coupon bond with maturity T_k that is assumed to occur at time $(t+\Delta t)$. The architecture of the ANN is shown in Figure 10.

It was determined by the PCA that the move in the 5-year yield has the largest impact on the entire curve. Thus, we will use the change in the 5-year yield as the 9th input of the ANN, that is, $\Delta Y_{TK}(t+\Delta t) = \Delta Y_{T5}(t+\Delta t)$.-

[8] *In this paper we work with only 8 time series. However, the Bank of Canada provides 120 time series of yields of different maturities. Thus, reducing the number of time series to only a few principal components significantly simplifies the modelling.*

The Mean-Squared Error (MSE) of predictions for PCA and ANN on the entire dataset are 0.16 and 0.11, respectively. Therefore, based on the MSEs alone the ANN outperformed PCA. However, this result should be taken with a grain of salt. Let us assume that we would like to predict the move of the flat curve when the 5-year yield moves either up or down by 20 basis points. The results are shown in Figure 11.

The PCA prediction is constructed so that it always yields the correct prediction for the 5-year yield in this example. As Figure 11 shows, though, the ANN prediction is not constructed in this way, and its predictions can differ significantly from the correct ones. Panel (a) of the figure shows that the ANN prediction for the 5-year yield is correct when starting from a flat 4% yield curve. Panel (b), however, shows that when starting from a flat 2% yield, instead, the ANN prediction is noticeably incorrect.

To understand this behaviour, we should look at Figure 9 that clearly shows that in January 2007 the yield curve was almost flat at exactly 4%. On the other hand, there is no time period where the yield curve is flat at the 2% level. Thus, it is left to the ANN to decide what type of behavior it should predict for the 2% flat yield curve based on the information provided in the dataset.

We would like to reiterate that the purpose of this example is to point out some of the risks that should be taken into account when ANNs are used as a modelling tool. In this respect there are many possible "fixes" to various problems associated with ANNs and

Figure 10. *ANN to model yield curve moves*

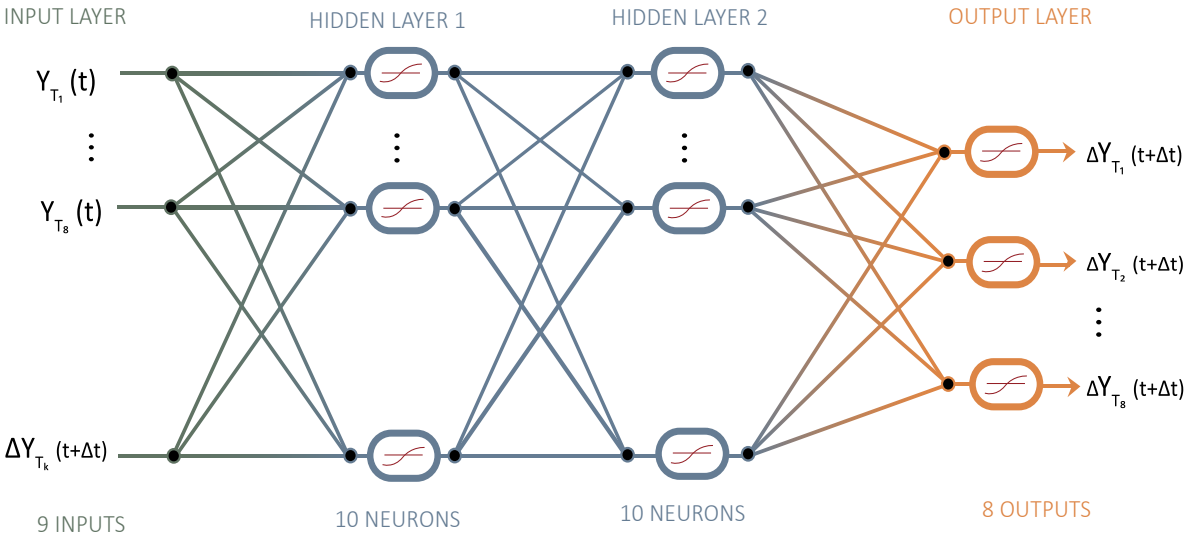
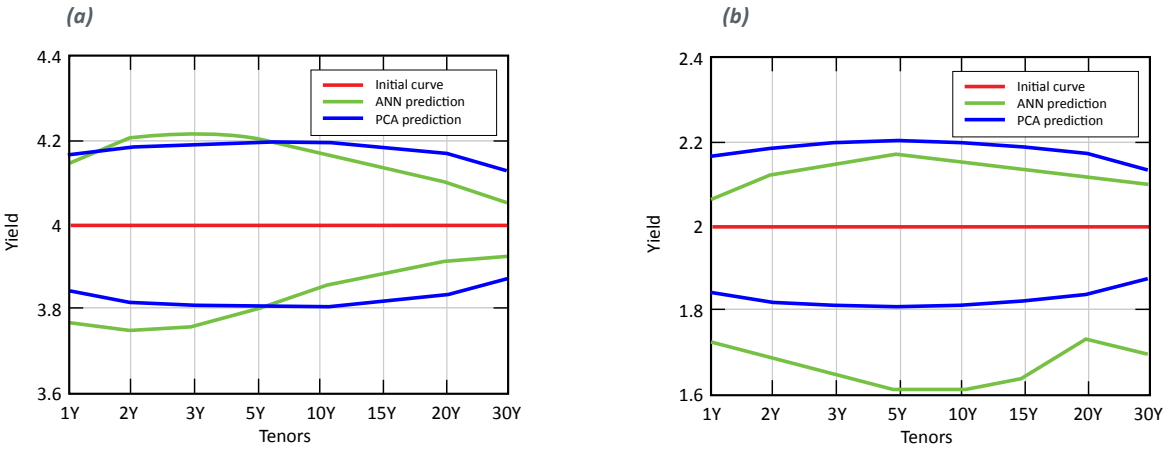


Figure 11. ANN and PCA predictions of the yield curve moves starting from different flat yield curves and assuming that the 5-year yield changes by ±20 basis points. Plot (a) shows the prediction when the original yield curve is at 4% and plot (b) shows the prediction for 2% initial yield curve.



a researcher who applies ANNs should be aware of them.[9] It should be noted, however, that building the ANN for a specific application is more of an art than a science and various "fixes" do not guarantee a good performance of the trained ANN.

RISKS AND CHALLENGES

Before one applies ANNs in financial modelling, he or she should be fully aware of the risks and challenges of these models. ANNs suffer from many of the same issues as other modelling approaches, including overfitting, under fitting, and misspecification, but identifying and correcting for these issues is particularly difficult when dealing with ANNs. More importantly, however, there are several unique risks that can arise when applying ANNs in a financial context; these risks will be introduced in this section.

One of the major challenges in applying ANNs in the finance industry is their lack of interpretability, that is, the inability to parse out how an ANN reaches a certain conclusion.10 As was shown in our yield curve example, it is difficult to explain why, in financial terms, the ANN produced the output shown in Figure 11. Interpretability has been at the forefront of discussions around ensuring fairness and protecting against bias and discrimination, and has led to much regulatory scrutiny around the use of these so-called "black-box" algorithms in production. We are now beginning to see regulators formally address this issue. One such example is the European Union's General Data Protection Regulation (GDPR) which requires a "right to explanation" for users on all decisions made by automated or AI algorithm systems. (GDPR, 2016) In hopes of ameliorating this issue, there has been a recent surge of research focused on explanatory artificial intelligence (XAI) which strives to provide explanations, in some level of detail, around why these algorithms produce the result they do. (Gilpin, 2019) Regardless of these efforts, the lack interpretability still remains problematic and has yet to be adequately addressed.11

Like any other algorithms, ANNs are susceptible to yielding biased results. However, the difficulty in interpreting the results of ANNs can make it particularly hard to track down the sources of these biases, or even detect them at all. Bias can arise through two channels. First, ANNs that are trained on data that are incomplete or not representative can make biased predictions. Alternatively, ANNs may be trained on data that are in fact representative but nevertheless reflect historical sources of bias. In this case, the use of ANNs can perpetuate or even accentuate these biases. For example, recruiters at Amazon designed an algorithm to automatically select promising resumes from a pool of job applicants, but because the training data were dominated by male applicants the algorithm discriminated against females; upon discovering this bias, Amazon halted the program.12 Second, ANN designers can unwittingly introduce their own biases through their programming decisions. In addition to being a potential source of model risk for financial institutions employing ANNs, bias can pose notable legal risks as well. Historically, financial institutions in the United States were not held liable for unintentional discrimination, but Supreme Court decisions in several recent fair lending cases have relied on the "disparate impact" theory of liability, ruling that it is the effect, not the intention, which matters. (Petrasic, 2017) In this context, bias can be both legally and operationally problematic. With the increased use of ANNs comes the risk of herding, which occurs when market many participants engage in similar behavior simultaneously. This risk, of course, is not specific to ANNs exclusively, but their recent popularity may cause investors who use ANN-based algorithms to behave similarly, leading to increased market volatility and, in extreme cases, becoming a source of systemic risk. (FSB, 2017)

The recent success of ANNs in applications like computer vision and the physical sciences has created a strong desire to apply these methods to a wide range of prediction problems, including those in finance. Unfortunately, this has led to a certain amount of misuse and blind application of these techniques for financial modelling. To address this issue, it is crucial that those who are tasked with building these models have both subject matter expertise (in both general and quantitative finance), as well as a deep understanding of ANNs (or whatever alternative methods they are looking to apply). Without this skill set, there are a plethora of risks one faces, including:

9 For example, regularization is often used to deal with the problem of overfitting.

4. *building a model that does not make sense in a financial context;*
5. *applying the methodology to problem to which it is not well suited;*
6. *applying the methodology improperly;*
7. *and being unable to identify when a simpler, traditional method is better (possible criteria: efficiency, complexity, interpretability, error minimization).*

If the financial services industry hopes to reap the benefits that come along with using ANNs, they must be mindful of the limitations and challenges that come along with these methodologies and, most importantly, ensure that the necessary experts are employed to carry out the task.

CONCLUSIONS

Recently, there has been a lot of attention paid to the financial applications of ANNs. Although the use of ANNs in finance seems promising, one should be mindful of the risks that underlie such applications. In particular, special care should be taken in regards to interpretability, the identification of bias, and the hiring those with appropriate expertise.

REFERENCES:

1. Baesens, B., Setiono, R., Mues, C., Vanthienen, J.: Using Neural Network Rule Extraction and Decision Tables for Credit-Risk Evaluation. Management Science 49(3), 312-329 (2003)
2. Cao, J., Chen, J., Hull, J.: A Neural Network Approach to Understanding Implied Volatility Movements. Working Paper (2019)
3. Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems 2, 303-314 (1989)
4. Financial Stability Board: FinTech Credit: Market Structure, Business Models and Financial Stability Implications (2017)
5. General Data Protection Regulation: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. Official Journal of the European Union L119, 1-88, (2016)
6. Gilpin, L., Bau, D., Yuan, B., Bajwa, A., Specter, M., Kagal, L.,: Explaining Explanations: An Overview of Interpretability of Machine Learning. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology (2019)
7. Green, B.P., Choi, J.H.: Assessing the Risk of Management Fraud Through Neural Network Technology. Auditing: A Journal of Practice and Theory 16(1), 14-28 (1997)
8. Heaton, J.: Introduction to Neural Networks with Java. Second edition. Heaton Research (2008)
9. Kondratyev, A.: Curve Dynamics with Artificial Neural Networks. Cutting edge investments: interest rates. Risk.Net, June 2018
10. Linn, S., Tay, N.: Complexity and the Character of Stock Returns: Empirical Evidence and a Model of Asset Prices Based on Complex Investor Learning. Management Science 53(7), 1165-1180 (2007)
11. Petrasic, K., Saul, B.: Algorithms and bias: What lenders need to know. White & Case (2017)
12. Tam, K.Y., Kiang, M.Y.: Managerial Applications of Neural Networks: The Case of Bank Failure Prediction. Management Science 38(7), 926-947 (1992)