ISSN 2601-8683 (Print)
ISSN 2601-8675 (Online)
European Journal of
Formal Sciences and Engineering
July -December 2020
Volume 3, Issue 2

Simpler Machine Learning Using Spreadsheets: Neural Network Predict

Leong Thin-Yin

Singapore University of Social Sciences (SUSS)

Leong Yonghui Jonathan

Singapore University of Social Sciences (SUSS)

Abstract

Machine Learning as a phenomenon has gone viral, with many technologists and software vendors promoting it. However, offered tools remain highly technical and not accessible to those without rigorous training in Computer Science or Business Analytics. It would be more useful if end-users can understand it beyond the sales pitch or blind application, and perhaps, even work from scratch to build simple models without much additional training. With better assimilation and acceptance of this AI methodology as an acquired skill and not just head knowledge, many more may want to invest the intensive effort to learn the required tough mathematics and cryptic programming. Or, after simple trial explorations, be willing to put aside substantial budgets to employ skilled professionals for full-scale business application. With simplicity and accessibility in mind, this paper renders Neural Network, a key machine learning methodology, on the ubiquitous and easily comprehensible spreadsheet without macros or add-ins, employing only elementary operations and if so desired, optionally leveraging on its built-in Solver. We will show that backpropagation can be achieved using the elegant though obscure recursive computation feature, with no need for Solver. We will demonstrate the application of neural network on a familiar problem: early and prior prediction of students' graduation GPA. The paper can be used to form the core content for introducing machine learning to non-technical audiences, particularly those majoring in Business and the Social Sciences.

Keywords: machine learning, neural network, business analytics, spreadsheets, education, higher education, experiential learning

1. Introduction

Machine Learning is garnering lots of attention recently in academia as well as in the public media and businesses. Many technologists and software vendors are promoting

aspects of Machine Learning, particularly Artificial Neural Network (NN) and Deep Learning. Though NN has been around since the 1950s and is thus not entirely new, its resurgence is largely due to the broad-scale, high-speed, and low cost of computation, wireless communications, and the massive availability of live data arising from the Internet and smartphone usage, relative to earlier days of conception. As a result, billions of dollars of investments by vendors and users alike are going into NN development and application.

The original Coursera Machine Learning (ML) course by Ng (2017) has been immensely popular, with 1.7 million enrollments since the course began in 2012 and many more people continue to take it via Coursera (or watch the lecture recordings through YouTube), to learn the basic concepts and techniques in ML (supervised and unsupervised learning, neural network, support vector machines and clustering), underlying statistical techniques (single and multiple linear and logistic regressions) and mathematics (linear algebra, forward propagation and backpropagation, gradient descent optimization and data parallelization), software tools (Octave), practical project ideas (regularization, validation, development strategies, dimensionality reduction, data normalization and visualization), and application (anomaly detection, recommender systems and image recognition).

As the onerous list of related topics above suggests, this is a highly technical course catered for scientists and engineers, though learners are already spared from R or Python programming languages and associated TensorFlow and other libraries. The inherent knowledge, skills, and tools offered by software vendors remain highly technical and not accessible to people without training in Computer Science or Business Analytics. Machine Learning would only become broadly used if the average business executive, non-technical undergraduate or even high-school student could understand it, sans complex mathematics or vague sales pitch. It would be helpful if learners could simply interact with Machine Learning itself, figuratively and from applied fundamentals, starting with building simple models and not just blind application, as is the common practice of imparting programming and subsequently machine learning skills.

The idea of such a teaching technique rests upon pedagogical processes that inculcate the acquiring of skills through a mediated form of practice, rather than simply through verbal instruction of theoretical constructs. The idea of “learning through doing” has been observed in many early civilizations, largely in the form of the master-apprentice relationship, and has in modern times been distilled to what is known as the Experiential Learning Theory (ELT). Kolb (2015) is one of the proponents of this model, which emphasizes the four cyclical stages of Concrete Experience, Reflective Observation, Abstract Conceptualization, and Active Experimentation; the student should repeat the processes ad infineum in the continuous process of learning. The use of such a model in university teaching has been documented by Leong & Ma (2019), in which case studies have shown the deeper levels of understanding from students who have been exposed

to this process of learning, as compared to the common process of imparting of expert knowledge or opinion. The same model also expounds on the relation of academic disciplines as "crafts", and can be deconstructed into the "object" of the craft, the "objective" or goal, and the "skill" required to manipulate the "object" to the "objective". Interestingly, this also turns out to be a qualitative depiction of what NN and ML achieve.

With better assimilation and acceptance of the NN methodology as an acquired skill and not just head knowledge or a black-box, many more may want to spend the intensive effort to learn the required but tough mathematics and cryptic programming, or, after trial application explorations, be willing to put aside substantial budget to employ skilled people to apply them to challenging settings. As the history of AI applications informs us, particularly in medicine, unless end-users can understand, accept and trust it, all abilities deemed "better than human experts", however well-proven, verified and validated, would come to naught.

Therefore, with simplicity and accessibility in mind, this paper attempts to show how NN, a key machine learning methodology, can be rendered on ubiquitous and more comprehensible off-the-shelf spreadsheet applications, directly without macros or add-ins, employing the SUMPRODUCT function for forward-propagation and, if so desired, optionally leveraging the built-in Solver to do gradient descent to optimize the objective function. We will demonstrate the application of NN using spreadsheets on a problem familiar to most institutes of higher learning: early prediction of students' graduation grade point average (GPA) from a basket of factors.

Our paper's key contribution is to show non-technical users more comprehensively what NN is and how to more simply do it, and refresh upon the efforts of others to promote the use of spreadsheets as a tool for learning and applying basic NN concepts. In particular, the paper's novelty is our use of the elegant, though obscure, recursive computation spreadsheet feature and so do away with the need for Solver. Despite the desired exclusion, Solver's use for NN is still thoroughly explained and demonstrated in the paper as it would be handy to those who cannot fully understand or accept recursive computation. The detailed demonstration seeks to allay users' fears of complex mathematics, technical programming, and the like, and approach the building of a NN system using an applied and practical perspective, in particular the ELT approach.

The rest of the paper is organized as follows: Section 2 contrasts our approach against related efforts by other authors, and Section 3 explains the base model for predicting a continuous dependent (or output) variable from multiple continuous independent (or input) variables. Next, Section 4 shows how backpropagation could be done fairly simply, by exploiting the recursive computation explained prior, and finally, Section 5 summarizes the results. The paper ends with a broad discussion on research motivation and future work, to introduce machine learning to non-technical students, particularly those majoring in Business and the Social Sciences.

2. Related Work and Literature Review

This paper is part of a series of efforts over the years by the lead author and collaborators to promote the use of spreadsheets as a viable means for the less mathematically- and technically-inclined, in order to explore realistic business scenarios and resolve them, without use of add-ins or macro programming. Problems addressed to date include multi-server queues (Leong 2007a), and data resampling and Monte-Carlo simulation (Leong 2007b, Leong & Lee 2008). NN spreadsheet models may not be suitable for large-scale application but would suffice for teaching machine learning and for preliminary exploratory analysis with less data, prior to embarking on enterprise-level implementation, by which time would involve project teams with capable data scientists and large amounts of data.

There are multiple practical reasons for using spreadsheets (Leong & Cheong 2008). Briefly, spreadsheets permit people with basic mathematical skills and no programming abilities to perform highly elaborate modeling of business challenges by making algebraic, statistical and probabilistic computations to appear arithmetic. Its interface is simple and highly interactive, plus its ubiquitous presence in offices, led many analysts to consider it as the tool of choice for exploring business opportunities.

With many educators, especially in universities, adopting spreadsheets as their main computing platform to support teaching of business mathematics, statistics, and management science courses, not to mention its common use among students, graduates can now more easily apply spreadsheets for quick data analysis, as well as for understanding approaches to formation of complex solutions. Even people without formal training and only direct experience with spreadsheets can pick up advanced modeling techniques (Leong & Cheong 2008, 2015). Therefore, the primary key advantage of using spreadsheets is that it would allow a wider end-user audience to experience first-hand what machine learning actually is and how applications may be easily constructed and flexibly adapted for use.

There appears to be few academic papers on using spreadsheets to apply machine learning. In fact, Sarkar *et al* (2015) lament the lack of tools accessible to non-experts and offered BrainCel, a visual interface system for performing machine learning in a spreadsheet-like environment to do clustering. BrainCel is user-friendly as a machine learning system but it insulates users from the logical processes. Similarly, Koong, Mcgee & Leong (2010) presented a tutorial software application for imparting NN knowledge to less technically inclined students. These used propriety software and, unlike standard spreadsheets such as Excel, require much training that could go to waste if and when their platforms do not stand the test of time.

Werbos (1988), who pioneered backpropagation in NN and demonstrated its use in a spreadsheet, employed macros to cycle through the iterations so as not to involve avoid having numerous rows to store the weights in each step. Rienzo and Athappilly (2012) shares our enthusiasm in applying spreadsheets to NN but in avoiding macros, could

only work on small problems that required no more than 4 iterations. Semerikov *et. al* (2019) addresses the multi-classification of Iris botanical types using spreadsheets and applied Solver to determine the weights. Kendrick, Mercado, and Amman (2006) in chapter 2 of their textbook applied Solver in their NN spreadsheet to forecast stock prices, calling it a "grey-box" concept.

There are some useful articles by professionals on web forums and blogs that address the use of spreadsheets for machine learning. However, they cover only base level statistical analysis approaches such as Linear Regression and Logistic Regression. For example, Granville (2017) uses complex array functions (i.e., CORREL, COVAR and LINEST). This approach makes the models opaque, as users cannot see how the detailed computations are done. Still others like Roberts (2018) introduced the use of the PyXLL Add-in that embeds Python in Excel to extend the spreadsheet's functionality to take advantage of the Python ecosystem. Smith (2018) adopts a play-learning approach to apply Excel to NN, as a "Skynet" game and Ekman (2019) applies spreadsheet macros to do backpropagation. Finally, more focus should be given to NN applications in business and for this, we refer the reader to the literature review and analysis paper by Wong, Bodnovich, and Selvi (1997).

3. Base Model

The basic idea behind machine learning is to be able to understand how a set of independent variables (X1, X2… Xn) relates to a dependent variable (Y). To illustrate, we may let Y, the variable to predicted, be a student's result in an English test, and X1 may be the number of hours the student studied, X2 the hours slept the night prior to the test, and so on. What machine learning aims to achieve in this scenario would be to predict the value of Y, given the X variables' values of a particular student, before the actual test is conducted, through the assignment of weights to each variable and the corresponding mathematical relationship (both of which are computed through the learning process). The choice of X variables is therefore important for proper forming of the relationships between the chosen X variable and the correlation to the Y variable. In a basket of chosen X variables – e.g., students' weight, height, gender, or results in their past English or Math tests – we may also find that there are different weights to each variable, and relationship between the independent and dependent variables; sometimes unexpected, and in varying degrees.

Now if we have access to a set of data from past years' students, we may then be able to find a function (F) to fit the multiple Xs to Y. The output of F(X1… Xn) using the given data, and after the algorithm has been formed, should give a value as close as possible to the actual Y. In Statistics and NN, we use linear functions as follows:

$$YP = F(X1 \ldots Xn) = W0 + W1 * X1 + W2 * X2 + \ldots + Wn * Xn \quad (1)$$

where YP is the predicted value of Y and weights W0, W1, …, Wn are constants to be determined as part of the algorithm. In the single X variable case, W0 is the Y-intercept

and W1 is the slope. As the variables may be quite different in magnitudes from one another, the data need to be normalized and re-scaled -this is assumed done prior to any computation, and detailed discussion is left for later.

The analysis thus far is called "Linear Regression". NN expands on this idea, using concurrent multiple copies of this function (shown schematically in Figure 1). Furthermore, beyond continuous variables, NN also deals with categorical variables. To avoid complex mathematics, we will explain this in practical terms using spreadsheets.

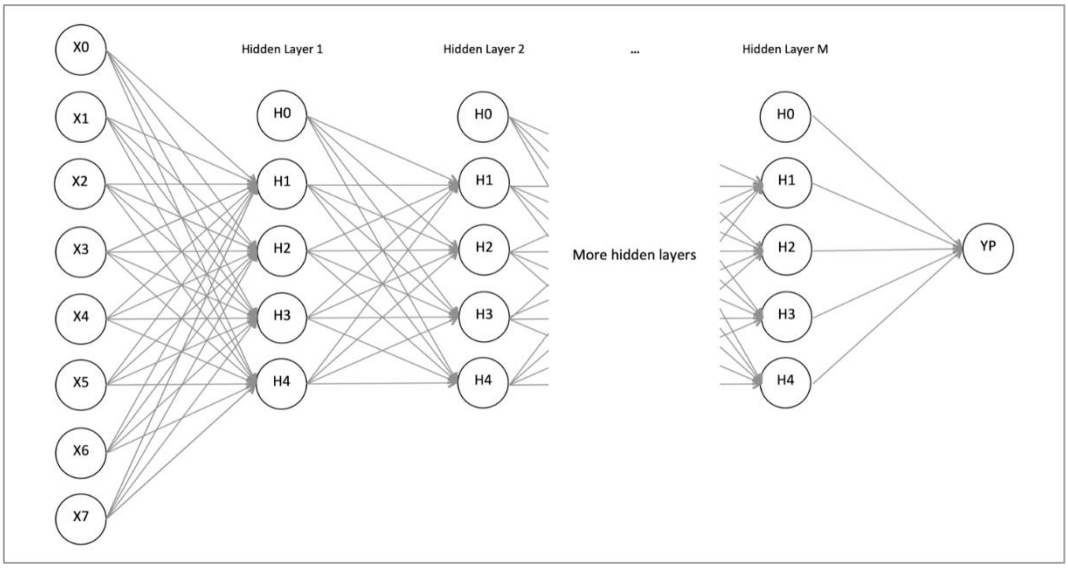

Figure 1: Neural Network Schematic

The final function of Y can be thought of simply as a weighted sum (or function) of the various X variables. With the final function, we compute the YP value for each data point and compare it against its Y value. For goodness of fit, it is easiest to use Mean Absolute Error, the average of absolute values of YP minus Y. The W values, initialized usually with random values, are also tuned to improve the fit. The function with weights being improved is thus learning from the data. In NN, this is termed "supervised learning" since there exist Y values in the given data that can be used to compare against the YP values and thereby improve the function. "Unsupervised learning", in which the outcome (Y) value or variable is not known, will not be discussed further in this paper. (Do note that the Computer Science community may use a different nomenclature from mathematicians; notably that variables are referred to as features and formula (1) above as forward propagation.)

We can use spreadsheets to illustrate the process of determining this function of Y respect to X. To do so, we first examine the NN Predict GPA spreadsheet model (see Figure 2). This model tries to predict students' graduation GPA from the grade points of

seven pre-selected courses (indicated in the table header as C01-C07). These courses were specifically chosen because they are taken early in the students' academic programme and their students' grade point values in these courses singly correlate to the GPA better than other level 1 or 2 courses (typically taken in their first two years of a four-year undergraduate course). The data values involved are all continuous and the variables' minimum and maximum possible values are the same, thus well behaved.

Neural Network Predict 2-Layer (7-4-1)

GPA from Early Courses' Grade Points — Using Solver

Initialise weights with random values

Mean Abs Error			0	1	2	3	4	5	6	7	0	1	2	3	4	
0.13	0.18	W1	0.99	0.52	0.60	-0.11	0.28	0.30	0.23	0.69	0.11	0.27	0.05	-0.01	0.11	W
Train	Test	W2	0.74	-0.01	0.00	0.00	0.00	0.00	0.00	0.00	-0.46	0.01	0.48	-0.15	0.36	Rand
		W3	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00						
ReguTerm		W4	1.18	0.17	0.00	0.00	0.00	0.66	0.04	0.00						
0.00		Rand	0.40	-0.23	0.31	0.36	-0.01	0.49	-0.19	-0.27						

Regu

0.01				C01	C02	C03	C04	C05	C06	C07	Hidden Layer					Error	Out
S/N	YP	Y	X0	X1	X2	X3	X4	X5	X6	X7	Z0	Z1	Z2	Z3	Z4	YP	YP
1	3.57	3.29	1	3.5	3.0	4.0	3.0	5.0	3.5	4.5	1	10.4	0.7	0.4	5.3	0.28	3.57
2	4.13	4.13	1	4.0	5.0	4.0	4.5	4.0	5.0	5.0	1	12.7	0.7	0.4	4.8	0.00	4.13
3	4.35	4.58	1	5.0	5.0	4.0	4.5	5.0	5.0	4.5	1	13.2	0.7	0.4	5.6	-0.23	4.35
4	3.64	3.58	1	4.0	4.0	4.5	5.0	4.0	4.0	3.5	1	10.9	0.7	0.4	4.7	0.06	3.64
5	4.06	4.24	1	4.0	5.0	4.5	4.0	4.5	5.0	4.5	1	12.3	0.7	0.4	5.1	-0.18	4.06

Figure 2: NN Predict GPA (2-layer; using Solver)

Cells E5:L5 contain the weights for computing variable Z1 from constant X0 (value 1) and variables X1 to X7. Z1 (in cell N15) is given by the formula:

=SUMPRODUCT($E$5:$L$5, $E15:$L15)
(2)

Similarly, cells E6:L6 contain the weights for computing Z2. Z2 (in cell O15) is given by:

=SUMPRODUCT($E$6:$L$6, $E15:$L15)
(3)

and so on, each time using the weights in the next row. Therefore, variables Z1 to Z4 are weighted sums of X0 to X7; and with X0 equals 1, its associated weight W0 is the bias. These formulas are copied down from row 12 to other rows to cover other input instances.

Incidentally, output variable Z1's formula above would be the same as that for predicted output YP of a 1-layer NN (i.e., with no hidden layers) and only the weights in E5:L5 are pertinent. Now for the 2-layer NN, variables Z0 to Z4 constitute the hidden layer where

Z0 like X0 equals to 1. The second and last layer gives the predicted output variable YP (in cell S15) is computed from Z0 to Z4 (in cells M15:Q15) as follows:

=SUMPRODUCT(M$5:Q$5, M15:Q15) (4)

We reproduce a copy of this YP in cell C15 at the far left of the spreadsheet model, using formula =S15, to facilitate easy comparison with the actual Y in cell D15. And for the same reason, at the far right, cell R15 (with formula = S15−D15) shows the prediction error.

Instead of having numerous SUMPRODUCT formulas in cells N15:Q49, we can alternatively use one array formula {=MMULT(E15:L49,TRANSPOSE(E5:L8))}. This array formula approach is not recommended because array functions MMULT and TRANSPOSE made the model inflexible: difficult to add more training data or modify the NN's structure, say deleting columns to remove weak variables. Further explanations explanations on the use of the $ sign in cell references, and function SUMPRODUCT, and array functions TRANSPOSE and MMULT are included in the Appendix.

To expand the model with more layers, hidden layers can be inserted and SUMPRODUCT formulas applied similarly, by treating each preceding hidden column's Z's as if they are X's. Having many layers and inclusion of more complex structures give rise to what is known as "Deep Learning" in NN. In all NN, the learning comes from progressively adjusting the weights in response to feedback errors, thereby improving predictability. An easy approach to do this is to use Solver: minimize cell B5 by changing E5:L8 and M5:Q5, where the Mean Absolute Error (MAE) objective or loss function in cell B5 is given by array formula:

{=SUM(ABS(R15:R39))/COUNT(R15:R39)} (5a)

where R15:R39 contains the prediction errors for the training data. That is, errors of the test data in rows 40 to 49 are excluded.

Solver's GRG Nonlinear Solving Method used is the vendor-enhanced version of the Generalized Reduced Gradient (GRG2) code developed by Leon Lasdon and Alan Waren. We have started to apply array formulas (not the same as array functions). Though often unfamiliar to casual spreadsheet users, array formulas are simpler than array functions (see Appendix).

Formula (5a) computes the average of the absolute magnitude of errors between the actual and predicted dependent variable values. We have chosen MAE because it gives equal weightage to errors regardless of error size and is thus easier for end-users to understand. The objective function enhanced by adding a regularization term is as follows:

=SUM(ABS(R15:R39))/COUNT(R15:R39)

+ B12*SUM(ABS(F5:L8),ABS(N5:Q5))/COUNT(F5:L8,N5:Q5) (5b)

where cell B12 holds regularization rate and cells F5:L8 and N5:Q5 contain weight values less the bias terms. This rate should be adjusted to influence magnitudes of the weights and thereby moderate over-fitting and bias issues. Formulas for key cells of the completed model are summarized in Figure 3.

Documentation		
MAE Obj: Train	B5	{=SUM(ABS(R15:R39))/COUNT(R15:R39) +$B$10*SUM(ABS(F5:L8),ABS(N5:Q5))/COUNT(F5:L8,N5:Q5)}
Test	C5	{=SUM(ABS(R40:R49))/COUNT(R40:R49) +$B$10*SUM(ABS(F5:L8),ABS(N5:Q5))/COUNT(F5:L8,N5:Q5)}
MSE Obj: Train	B5	{=SUM((R15:R39)^2)/COUNT(R15:R39) + ...
Test	C5	{=SUM((R40:R49)^2)/COUNT(R40:R49)} + ...
W1, Wgts for Z1	E5:L5	<Input>, initialized with random values
W for Output YP	M5:Q5	<Input>, initialized with random values
Output Predicted Y	C15	=S15
Y	D15	<Input>
Input X0, X1-X7	E15:E49	1, <Input>
Z0	M15:M49	<Input>, 1
Z1	N15	=SUMPRODUCT($E$5:$L$5,$E15:$L15)
Z2	O15	=SUMPRODUCT($E$6:$L$6,$E15:$L15)
Z3	P15	=SUMPRODUCT($E$7:$L$7,$E15:$L15)
Z4	Q15	=SUMPRODUCT($E$8:$L$8,$E15:$L15)
YP Error	R15	=S15-D15
YP	S15	=SUMPRODUCT(M$5:Q$5,M15:Q15)

Figure 3: Formulas of key cells in the NN Predict GPA

4. Results

An illustrative explanation to the model is to treat each Z value as the score given by an expert evaluator, computed by taking a weighted sum of independent variables' values. These expert-specific weights are based on individual evaluator's "subjective judgment". In turn, a weighted sum of the experts' scores can be taken as the score of an expert panel. There can be more than one of such panels, and then in turn, weighted average of panel scores can be taken to give an expert committee's score, and so on. Each time, a hidden layer is added to the NN in an attempt to make it more robust.

Often only a simple NN model with 1 or few hidden layers, with each layer having few variables, is needed. In its simplest, with no hidden layers, the NN model is the same as linear regression. We can put the linear regression model into effect by setting irrelevant X and Z weights (cells E5:L8, M5 and O5:Q5) to 0, and weight of Z1 (cell N5) to 1, after which Solver only needs to change cells E5:L5 to get the local optimal solution for the weights.

In our model, we have split the data into two mutually exclusive sets: the training set and the testing set. As per usual machine learning practice, the training data is used to build the model by calibrating weights, while the testing data (where the weights, obtained from the training data, are held constant) checks how well the model performs. More

clearly, the testing data cannot be involved in tuning the weights. If actual outputs are known, especially soon after prediction rather 4 years later upon student graduation, the weights can be updated continually, as each data point is being introduced, allowing early discovery of prediction errors which may influence future predictions.

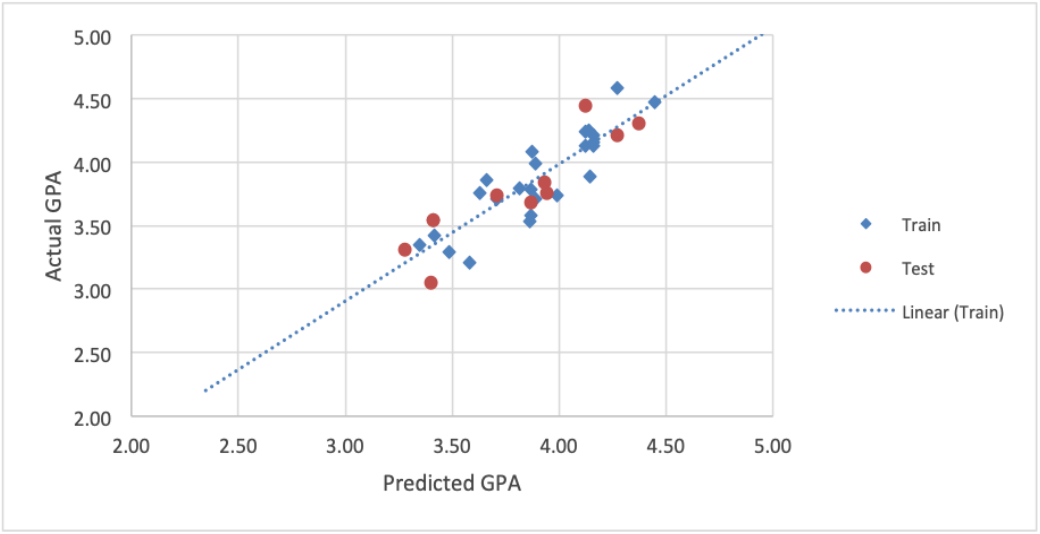

Figure 4: NN Predict GPA Results (using Early Course Grades)

The results (Figures 2 and 4) show that the NN model can predict the GPA with high levels of accuracy, with training and testing MAE of 0.13 and 0.18 respectively for the data set. This means that the weights that have been tuned can fit the model to give an average deviation from actual GPAs by 0.13 grade points. When model is applied to data outside the training set, it is expected to do slightly worse and the average deviation is now 0.18 grade points. (i.e. if the model predicts that a student GPA is 4.0, then on the average it is wrong by ±0.18, or that the actual grades are likely in the 3.82 to 4.18 range.)

In an actual problem, the use of pre-admission variables such as demographics (race and gender), education track (academic junior college or technical polytechnics) and other educational priors (high school English or Mathematics results; admission screening test scores). Some variables are binary (e.g., female = 0, male = 1; junior college = 0, polytechnic = 1) while others are categorical, both ordinal and non-ordinal. Non-ordinals would be converted to many binary variables, one for each category, while ordinals may be treated as rounded continuous variables.

When input data are of different magnitudes, pre-processing is required to normalize and rescale them. In this case, we make all variables to be in [0,1] range as follows:

$$\text{Normalized and rescaled } X = (X - Xlow)/(Xhigh - Xlow) \quad (6a)$$

where Xlow and Xhigh are the lowest and highest possible data values respectively of the variable.

If Xlow and Xhigh are infinitely small or large, we use instead:

$$\text{Normalized and rescaled } X = (X - Xmean)/(6 \times Xstdev) + 0.5 \quad (6b)$$

where Xmean and Xstdev are the estimated mean and standard deviation of the population.

In applying the model, caution should be taken here that, for both formulae (6a) and (6b), the Xlow, Xhigh, Xmean and Xstdev should not be changed once decided. They should apply to all data and not dynamically changed as more data is added to the training set. This may pose limitations if the initial training set is not representative of the general range of values, and is therefore a reminder that the data used in both the training and testing sets should firstly be of a significant volume, and secondly, representative of the actual situation.

By adapting the model in Figure 1, replacing the 7 course grade points with 7 other demographic and prior education variables (with the input data normalized and rescaled) and then further pruned the variables to get better results (Figures 5 and 6). As the results show, the model fit now is not as good and is predictably significantly worse. We also eliminated variables found to be weak for prediction (by deleting their weights and excluding them from the “By changing cells” input field in Solver). Overall, it seems pointless to include any of these 7 new variables to the earlier 7 to give a better model. We may then conclude that these seven newer variables are not as strong indicators of the final predicted value as the earlier seven course grade points – at this point, note that we cannot yet say that they have no influence whatsoever, only that they are considerably weaker than the initial seven course grade points.

Neural Network Predict 2-Layer (7-4-1)

GPA from Demographics & Edu Priors — Using Solver

Initialise weights with random values

Mean Abs Error			0	1	2	3	4	5	6	7	0	1	2	3	4	
0.16	0.32	W1	0.14	-1.10	1.66	0.64	-0.04	-0.38	-0.89	0.69	3.94	0.00	-0.08	0.21	0.08	W
Train	Test	W2	0.02	0.17	-0.01	-1.84	0.58	0.37	0.02	0.00	-0.22	0.19	0.32	-0.06	0.36	Rand
		W3	1.68	-0.52	1.07	1.44	-0.30	0.05	-0.53	0.08						
ReguTerm		W4	2.79	-3.46	1.07	0.00	0.00	0.09	-0.30	1.48						
0.01		Rand	0.41	0.03	-0.09	0.43	-0.28	0.37	0.23	-0.36						

Regu

0.01				Race	Sex	Edu	T1	T2	T3	T4	Hidden Layer					Error	Out
S/N	YP	Y	X0	X1	X2	X3	X4	X5	X6	X7	Z0	Z1	Z2	Z3	Z4	YP	YP
1	3.30	3.29	1	1.0	0.0	1.0	6.0	5.0	8.0	3.8	1	-7.0	3.8	-2.9	3.0	0.01	3.30
2	3.92	4.13	1	1.0	0.0	1.0	3.0	7.0	7.0	5.1	1	-5.8	2.8	-1.2	5.4	-0.21	3.92
3	3.96	4.58	1	1.0	1.0	1.0	4.0	4.0	8.0	4.7	1	-4.2	2.3	-1.2	5.4	-0.62	3.96
4	4.12	3.58	1	0.0	0.0	1.0	3.0	4.0	8.0	4.4	1	-4.9	1.6	-1.4	7.3	0.54	4.12
5	4.47	4.24	1	1.0	1.0	1.0	3.0	6.0	6.0	5.9	1	-2.3	2.4	0.4	7.9	0.23	4.47

Figure 5: NN Predict GPA (using Demographics and Education priors)

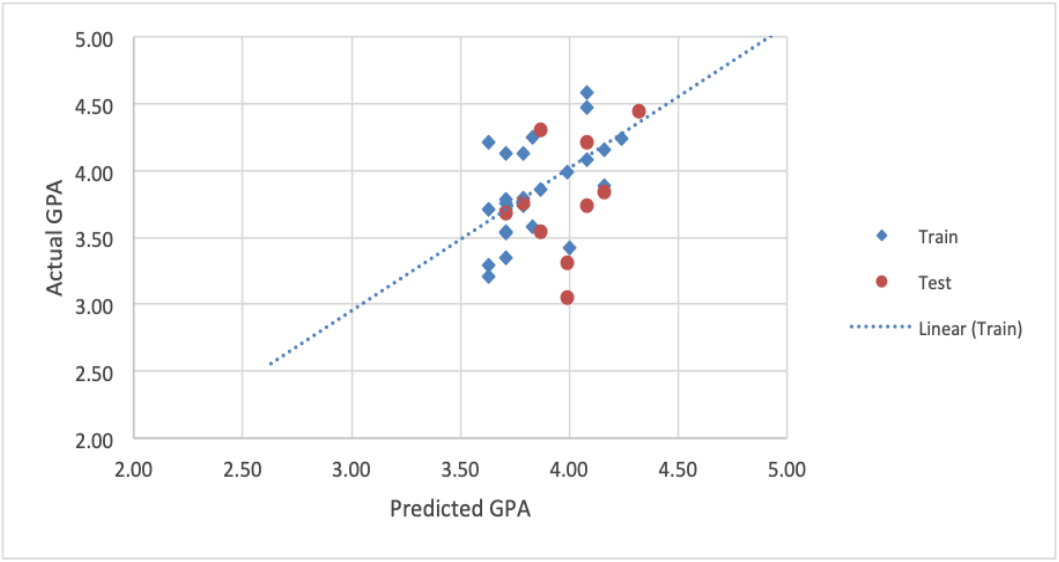

Figure 6: NN Predict GPA Results (with Demographics and Education Priors)

5. Simpler Approach to NN and ML – An Illustrative Explanation

For learners in business and social science majors, weights can be analogous to prices, predicted output(s) to supply, actual (or required) output(s) to demand, and the

objective function as an overall measure of the extent supply exceeds demand. For the purpose of modelling, prices can initially be set to random values, with a small increase in price (analogous to the learning rate), one at a time, to see how the objective function change, and subsequently increase or decrease that price by the small amount to reduce the objective function value.

With many such prices, we can also at each iteration change all of them simultaneously, although preferably after we have understood how each individual price change impacts the objective function. The algorithm ends when the objective function value cannot be reasonably reduced further with price changes. The price changes at each iteration have to be small, especially if simultaneously changed. Otherwise, the algorithm may not converge and the prices may just be cycling up and down. To get better results and faster convergence, we moderate the price changes to be progressively smaller with each iteration, instead of having constant very small price change sizes. With relatively larger price changes at the beginning, the objective function value would generally drift downwards though cycling up and down slightly. This process can of course be automated using Solver.

Machine learning in general employs a technique called backpropagation to optimize the NN model. Backpropagation computes the partial derivatives for each of the weighted variables, which is the gradient associated to the objective function value change with a small change in a weight. This allows us to adjust weights incrementally using gradient descent, which is identical to the optima obtaining process mentioned, but can be considered a myopic optimization heuristic.

Since MAE is not differentiable, the underlying objective function used is the Mean Square Error (MSE), as given in place formula (5a) by array formula:

{=SUM((R15:R39)^2)/COUNT(R15:R39)}
(7)

where cells R15:R39 contain the prediction errors.

The key criticism of MSE is that by squaring it gives too much emphasis to large errors and thus also difficult for non-technical people to interpret. We therefore retain MAE as the key prediction performance measure.

It can be quite challenging to show the partial derivatives' formulas for NN models with hidden layer(s), and therefore this approach would seem tedious even in a spreadsheet and often involved macro programming. We had therefore used Solver to support end-users who may not want to deal with the complexity and algorithms involved. However, we will next show that by using recursive computation, backpropagation is not as difficult as some make it out to be.

Taking the base NN model in Figure 1, one can render it with no hidden layers (Figure 7). The predicted output variable YP (in cell N15) is now a weighted sum of the independent variable values, given by as in earlier explained formula (2):

=SUMPRODUCT(E$5:L$5, E15:L15) (2)

where E5:L5 contains the current weights. Separately, we set up cells E6:L6 to keep the next iteration's weights. The formula for cell E5 is then:

=IF($D$12="No",RAND(),E6) (8)

where Iteration Start indicator in cell D12 initially contains "No" and thus initialize current weights to random values.

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**Neural Network Predict**				1-Layer (7-1)								
2	GPA from Early Courses' Grade Points										Using Iterative Calculation		
3													
4	Mean Abs Error			0	1	2	3	4	5	6	7		
5	0.14	0.16	W	0.87	0.08	0.19	-0.04	0.17	0.12	0.04	0.15		
6	Train	Test	Next W	0.87	0.08	0.19	-0.04	0.17	0.12	0.04	0.15		
7													
8	ReguTerm												
9	0.00												
10													
11	Regu	Learn	Start	Select "Yes" to run									
12	0.01	1.E-03	Yes		C01	C02	C03	C04	C05	C06	C07	Error	Out
14	S/N	YP	Y	X0	X1	X2	X3	X4	X5	X6	X7	YP	YP
15	1	3.50	3.29	1	3.5	3.0	4.0	3.0	5.0	3.5	4.5	0.21	3.50
16	2	4.18	4.13	1	4.0	5.0	4.0	4.5	4.0	5.0	5.0	0.05	4.18
17	3	4.31	4.58	1	5.0	5.0	4.0	4.5	5.0	5.0	4.5	-0.27	4.31
18	4	3.80	3.58	1	4.0	4.0	4.5	5.0	4.0	4.0	3.5	0.22	3.80
19	5	4.06	4.24	1	4.0	5.0	4.5	4.0	4.5	5.0	4.5	-0.18	4.06

Figure 7: NN Predict GPA (1-layer; using Backpropagation)

Cell E6 will then update the current weight in cell E5 using array formula:

{=E5−$C$12*(SUM(E15:E39*$M15:$M39)−$B$12*E5)/COUNT($M15:$M39)}

(9)

where C12 is the learning rate and the SUM(…) term is the weight gradient. Copying cells E5:E6 to the rest of the columns completes the computations for other weights.

We can then activate Excel's iteration calculation feature (in the Calculation Option) and then change Start to "Yes" to set the recursive action between E5:L5 and E6:L6. The recursion stops when number of iterations reaches 100 or difference between successive computed values are not more than 0.001, where these values 100 and 0.001 being the two defaults limits given by Excel. For the GPA prediction scenario examined,

the initial training and testing MAE respective results of 0.17 and 0.23 did not attain the 0.14 and 0.16 results earlier found with Solver. However, these results improved to 0.14 and 0.16 respectively when the "Maximum iterations" is increased to 10,000 and "Maximum change" decreased to 0.00001. This is in line with the idea in Machine Learning that by increasing the number of layers and/or iterations, the computed function then produces a result that is more accurate to that of actuality.

With the 1-layer backpropagation process clarified, we will attempt to expand this to as many layers as desired. Backpropagation, as commonly termed in Machine Learning, is called such because the 1-layer approach had been shown (arising from the Chain Rule in mathematical differentiation) to be applicable to multi-layer models simply by working backwards from the last layer, one layer at a time. Each examined layer's input values are those which are forward-propagated (taking weighted sums) from preceding layers and error values backward-propagated (taking weighted sums, interestingly using the same weights) from the proceeding layer, with the last layer obtaining errors from the difference between predicted and actual output values.

Thus, each of the examined layer's input weights' gradients takes the same form as the SUM term in the earlier formula (9): averaging over all input values instances multiplied by their error term. Mathematical formulae involved can be quite onerous, but if done properly, spreadsheet portrayals are simple. The 2-layer NN backpropagation model using recursive computation is shown in Figure 8. As in the 1-layer model, there are two copies of all weights: current and next iteration. There are additional tables for error terms: one column each for hidden and final layers' output variables.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	**Neural Network Predict**			2-Layer (7-4-1)																		
2	GPA from Early Courses' Grade Points									Using Iterative Calculation										Using Iterative Calculation		
3																						
4	Mean Abs Error			0	1	2	3	4	5	6	7					0	1	2	3	4		
5	0.14	0.17	W1	0.52	0.45	-0.03	0.25	0.57	-0.05	-0.11	0.03					W	0.64	-0.07	-0.40	0.17	0.14	
6	Train	Test	W2	0.30	0.09	-0.19	0.36	-0.26	-0.34	-0.11	-0.23					Next W	0.64	-0.07	-0.40	0.17	0.14272	
7			W3	0.68	0.42	0.50	0.03	0.22	0.18	0.07	0.22											
8			W4	0.00	0.67	0.15	0.83	0.51	0.05	-0.09	0.01											
9			Next: W1	0.52	0.45	-0.03	0.25	0.57	-0.05	-0.11	0.03											
10			W2	0.30	0.09	-0.19	0.36	-0.26	-0.34	-0.11	-0.23											
11	ReguTerm		W3	0.68	0.42	0.50	0.03	0.22	0.18	0.07	0.22											
12	0.00		W4	0.00	0.67	0.15	0.83	0.51	0.05	-0.09	0.01											
13																						
14	Regu	Learn	Start	Select "Yes" to run																		
15	0.01	0.002	Yes		C01	C02	C03	C04	C05	C06	C07	Error				Hidden Layer					Error	Out
17	S/N	YP	Y	X0	X1	X2	X3	X4	X5	X6	X7	Z1	Z2	Z3	Z4	Z0	Z1	Z2	Z3	Z4	YP	YP
18	1	3.49	3.29	1	3.5	3.0	4.0	3.0	5.0	3.5	4.5	0.0	-0.1	0.0	0.0	1	4.2	-2.4	6.5	7.6	0.20	3.49
19	2	4.14	4.13	1	4.0	5.0	4.0	4.5	4.0	5.0	5.0	0.0	0.0	0.0	0.0	1	5.1	-3.1	8.1	8.8	0.01	4.14
20	3	4.35	4.58	1	5.0	5.0	4.0	4.5	5.0	5.0	4.5	0.0	0.1	0.0	0.0	1	5.5	-3.2	8.6	9.5	-0.23	4.35
21	4	3.78	3.58	1	4.0	4.0	4.5	5.0	4.0	4.0	3.5	0.0	-0.1	0.0	0.0	1	5.6	-2.4	7.3	9.4	0.20	3.78
22	5	4.05	4.24	1	4.0	5.0	4.5	4.0	4.5	5.0	4.5	0.0	0.1	0.0	0.0	1	4.9	-2.8	8.0	9.0	-0.19	4.05

Figure 8: NN Predict GPA (2-layer; using Backpropagation)

The formula for YP error (in cell V18), the difference between YP and Y, is given by:

=W18−D18 (10)

The YP errors, which are the differences between YP and Y (in cells V18:V42, Figure 8), are a major component in the partial derivative computations, as shown in the SUM term in formula (9) in the 1-layer NN case. In the 2-layer case, by backpropagation, it appears in the formula for the next weights associated with the hidden layer's variables.

The formula for next weight for Z0 (in cell Q6, Figure 8) is:

{=Q5−$C$15*(SUM(Q$18:Q$42*$V$18:$V$42)−$B$15*Q5)

/COUNT($V$18:$V$42)}
(11)

and it retains the structure of formula (9). This process can be repeated for all hidden variables' associated weights.

The YP error is backpropagated to give the Z1 error (in cell M18, Figure 8) using formula:

=R$5 * $V18 (12)

where R5 is the weight that linked Z1 to YP, and similarly we can compute the errors for Z2 to Z4.

The hidden layer errors Z1 to Z4 (in cells M18:P42, Figure 8) collectively is used in the computation of the weight gradients for the Xs and thus the formula for next weight for X0 (in cell E9, Figure 8) is

{=E5−$C$15*(SUM(E$18:E$42*$M$18:$M$42)−$B$15*E5)

/COUNT($M$18:$M$42)}
(13)

This process can be repeated using for input variables' associated weights.

For completeness, all the key formulas for the 2-layer NN are given in Figure 9 below.

Documentation		
MAE: Train	B5	{=SUM(ABS(V18:V42))/COUNT(V18:V42) +B15*SUM(ABS(F5:L8),ABS(Q5:U5))/COUNT(F5:L8,Q5:U5)}
Test	C5	{=SUM(ABS(V43:V52))/COUNT(V43:V52) +B15*SUM(ABS(F5:L8),ABS(Q5:U5))/COUNT(F5:L8,Q5:U5)}
X0-Z1: W	E5	=IF($D$15="No",RAND(),E9)
Next W1	E9	{=E5-$C$15*(SUM(E$18:E$42*$M$18:$M$42)-$B$15*E5)/COUNT($M$18:$M$42)}
Next W2	E10	{=E6-$C$15*(SUM(E$18:E$42*$N$18:$N$42)-$B$15*E6)/COUNT($N$18:$N$42)}
Next W3	E11	{=E7-$C$15*(SUM(E$18:E$42*$O$18:$O$42)-$B$15*E7)/COUNT($O$18:$O$42)}
Next W4	E12	{=IF($D$15="No",RAND()/10,E12)}
Z0-YP: W	Q5	=IF($D$15="No",RAND(),Q9)
Next W	Q6	{=Q5-$C$15*(SUM(Q$18:Q$42*$V$18:$V$42)-$B$15*Q5)/COUNT($V$18:$V$42)}
Regularization Rate	B15	<Input>, small value usually less than 0.5
Learning Rate	C15	<Input>, small value usually 0.1, 0.01, ..., 1.E-6
Iteration Start	D15	<Input>, Yes or No
Predicted Y, Y	C18, D18	=W18, <Input>
X0, X1-X7	E18:L52	1, <Input>
Z1 Error	M18	=R$5*$V18
Z0	Q18:Q52	<Input>, always 1
Z1	R18	=SUMPRODUCT($E$5:$L$5,$E18:$L18)
Z2	S18	=SUMPRODUCT($E$6:$L$6,$E18:$L18)
Z3	T18	=SUMPRODUCT($E$7:$L$7,$E18:$L18)
Z4	U18	=SUMPRODUCT($E$8:$L$8,$E18:$L18)
YP Error	V18	=W18-D18
Predicted Y	W18	=SUMPRODUCT(Q$5:U$5,Q18:U18)

Figure 9: Formulas of key cells in the NN Predict (2-layer; using Backpropagation)

7. Discussion and Further Research

Conventionally, fixed but possibly differing weights are used to combine attributes and scores into evaluation scores, e.g. in student admissions. It is also plausible if for some reasons the weights used were forgotten and we want to reproduce them using only past record of applicants data and evaluation scores or if, in addition to evaluation scores, admission committee members also apply subjective judgments in granting admission (individually or otherwise). In any of these cases, we can easily apply linear regression and machine learning to determine the forgotten and implied weights.

Where there are sufficient data and with concurrence of academic administration, the admission decision process may even be largely automated by machine learning, and eliminate the need for admission committees to exhaustively review all but only randomly sampled cases, needed to refresh the weights. Of course, exception cases still need the attention of human decision-makers, particularly so if the consideration requires a qualitative measure of judgment, or is of a feature not easily quantified.

Another motivation of our work is to be able to apply the approach to Recognition of Prior Learning (RPL), specifically to help mature students with poor or no high school

education gain admission in undergraduate study programs. With some requirements and in lieu of others, universities can allow adult learners to take a small set of pre-specified courses pre-admission and use their aggregated results (with validation support from NN analytics illustrated in this paper) as supporting evidence for admission or even credit recognition. With these scenarios in mind, this paper had focused on the NN model with one continuous output variable and pre-selected structures.

In the interests of machine learning instruction, and particularly for those who are more technically inclined, these spreadsheets and the related concepts such as recalibration of the weights and the inclusion of additional variables can be used to formulate tutorial exercises or study projects. The key goal in this paper, however, is to find the simplest possible model that performs better than required. In light of that principle, objective function results should be carefully recorded and plotted against parameter values, number of hidden variables and number of hidden layers. This documentation would inform us at which value incremental changes would only bring marginal or no further benefit.

Proposed future work to this study includes a follow-up paper on categorical single and multiple output variables, namely in the domain of binary classification and multi-classification. With the simpler approach toward machine learning and neural networks using spreadsheets, we hope to bring even more techniques in the stable of machine learning to our classroom, including Clustering, and Decision Trees.

References

[1] Ekman K, (2019). Machine Learning in Excel, Code Project.

[2] https://www.codeproject.com/Articles/1273000/Machine-Learning-in-Excel

[3] Granville V, (2017). Advanced Machine Learning with Basic Excel, Blog, Data Science Central. https://www.datasciencecentral.com/profiles/blogs/advanced-machine-learning-with-basic-excel

[4] Kendrick DA, Mercado PR & Amman HM, (2006). *Computational Economics*. Princeton University Press, Princeton

[5] Kolb D, (2015). *Experiential Learning*. Upper Saddle River, New Jersey: Pearson Education

[6] Koong LS, Mcgee K & Leong WLB, (2010). A Tool to Teach Artificial Neural Networks to Non-Technical Students, NUROP Congress

[7] Leong TY & Cheong MLF, (2015). *Business Modeling with Spreadsheets*, (3rd ed.), McGraw-Hill

[8] Leong TY & Cheong MLF, (2008). *Teaching Business Modeling Using Spreadsheets*, INFORMS Transactions on Education 9(1): 20–34

[9] Leong TY & Lee WL, (2008). *Spreadsheet Data Resampling for Monte-Carlo Simulation*, Spreadsheet in Education 3(1):70-78

[10] Leong TY, (2007a). *Simpler Spreadsheet Simulation of Multi-server Queues*, INFORMS Transactions on Education 7(2):172-177

[11] Leong TY, (2007b). *Monte Carlo Spreadsheet Simulation using Resampling*, INFORMS Transactions on Education 7(3):188-200

[12] Leong YJ & Ma NL, (2019). *Using Experiential Learning to Improve Teaching and Learning in Higher Education*. European Journal of Social Science Education and Research 6(1): 123-132

[13] Ng A, (2017). Machine Learning Course, Coursera.

[14] https://www.coursera.org/learn/machine-learning and

[15] https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN

[16] Rienzo TF & Athappilly KK, (2012). *Introducing Artificial Neural Networks through a Spreadsheet Model*. Decision Sciences Journal of Innovative Education 10(4):515–520.

[17] Roberts T, (2018). *Machine Learning in Excel with Python*. DataScience (https://datascienceplus.com/machine-learning-in-excel-with-python/)

[18] Sarkar A, Jamnik M, Blackwell AF & Spott M, (2015). *Interactive Visual Machine Learning in Spreadsheets*, IEEE Symposiumon Visual Languages and Human-Centric Computing, pp159-163

[19] Semerikov SO, Teplytskyi IO, Yechkalo YV, Markova OM, Soloviev VN & Kiv AE, (2019). *Computer Simulation of Neural Networks Using Spreadsheets*: Dr. Anderson, Welcome Back. CEUR Workshop Proceedings (to appear)

[20] Smith D, (2018). Machine Learning with Spreadsheets! Part 1: Gradient Descent and Backprop for Beginners, Medium.

[21] https://medium.com/excel-with-ml/machine-learning-with-spreadsheets-part-1-gradient-descent-f9316676db9b and https://www.excelwithml.com

[22] Werbos PJ, (1989). *Maximizing Long-term Gas Industry Profits in Two Minutes in Lotus using Neural Network Methods*. Transactions on Systems Man and Cybernetics 19(2):315–333

[23] Wong BK, Bodnovich TA & Selvi Y, (1997). *Neural Network Applications in Business: A Review and Analysis of the Literature (1988-95)*, Decision Support Systems 19:301-320

APPENDIX

Brief explanations of selected spreadsheet topics are provided here for easy reference on some salient but unfamiliar features. Further notes on the use of spreadsheet features and functions for modeling can be found in Appendix A of *Leong & Cheong* (2015).

Cell Referencing

To speed up model building, formulas in some cells are often copied over to other cells. A leading cell in a top row of a table (below the header) may be copied over to cells in the same row but other columns right of it to complete of a series of similar cells, with some adjustments made where needed. The whole top row of the table is then copied (or filled) down to complete the table.

When copied to another cell, simple (i.e., relatively referenced; no $ signs) cell references in formulas would be changed by exactly the number of rows and number of columns that separates the source and destination cells. For example, if the destination cell is 3 rows down and 2 columns right of the origin cell, a C7 in the first cell will become E10 in the second cell.

Cell references can be prevented from changing when copied into another cell by using the $ sign. To disallow column change, just prefix $ to the column letter in the cell reference, and to disallow row change, do the same to the row number. You can also do both concurrently.

Staying with the illustrative example, (absolutely referenced) $C$7 in the origin cell would remain as $C$7 in the destination cell; (mixed referenced) $C7 would only have its row number changed in the destination cell as $C10; and (mixed referenced) C$7 would have its column changed in the destination cell formula as E$7.

SUMPRODUCT (array1, array2, [array3, ..., array_n])

This function that multiplies ranges of cells (otherwise, also known as arrays) and returns the sum of their products. These arrays are either columns or rows of values. It multiplies the first value in the first array to the first value in the second array, the second value with the other second value and so on, and then adds all these values to give the output.

Logically, there must be at least two arrays. If there are more arrays, then the same logic applies with all the first values multiplied together, second values multiplied together and so on, and then summed. All arrays must naturally have the same number of values. Not numeric array entries are treated as zeros.

Array Formulas

Array formulas are entered with *SHIFT + CTRL + ENTER*, instead of the usual *ENTER* key and would be displayed with the distinctive { } brackets. Array formulas may or may not involve array functions. Array formulas as so called because they involve computations with arrays and at times even return array outputs.

Many interesting computations can be done using array formulas, giving spreadsheet models that are more compact and easier to comprehend.

Example 1. Find the mean absolute error between the actual and predicted values of a variable, as in cell B5 of Figure 2 (see formula 5a).

Not using array formulation, we need to use an extra cell range to keep the absolute differences and then apply the SUM and COUNT functions as follows:

Set the formula for cell R15 as =ABS(S15−D15) and copy this to cells R16:R49.

Replace the formula in B5 with =SUM(R15:R39)/COUNT(R15:R39).

Alternatively, select cells R15:R49 and (entering using SHIFT + CTRL + ENTER) apply formula {=ABS(S15:S39−D15:D49)}. This array formula returns an array output.

More directly, without any need for cells R15:R39, we can apply array formula {=SUM(ABS(S13:S37-D13:D37))/COUNT(D13:D37)} to cell B5.

Array Functions

Array functions are like normal functions except they return array outputs. As such, array functions are entered as array formulas with *SHIFT + CTRL + ENTER*, instead of the usual *ENTER* key and would be displayed with the distinctive { } brackets.

Example 2. Array function LINEST(knownY's, knownX's, [const], [stats]) returns the statistics that describe the least squares method for fitting a straight line to the given data set comprising knownY's and knownX's. If the optional variable const is TRUE or omitted, the intercept value is computed. Otherwise, the intercept value is assumed to be 0. If optional stats value is FALSE or omitted, then only the coefficients and the intercept of the fitted straight line are computed. If it is TRUE, four additional rows of statistics are computed. To key in the formula, first select a row with number of X's plus 1 number of columns for the FALSE optional stats case. For the other case, expand the selection to 5 rows.

Example 3. Array function TRANSPOSE(array) rotates the array by 90 degrees to make first row of the array as the first column of the output array, second row of the array ass the second column of the output array, and so on. It is important the size of the output array fits the expected output values.

Example 4. Array function MMULT(array1, array2) multiplies the rows of array1 to columns of array2 as in SUMPRODUCT and so on to return a matrix. The number of

columns in array1 must be the same as the number of columns in array2, and all cells in these arrays must contain numbers, no blanks allowed. So instead of applying many SUMPRODUCT functions in each cell of N15:Q49 of Figure 2 and select the cell range and apply formula {=MMULT(E15:L49,TRANSPOSE(E5:L8))}.

Solver

Solver is available as a standard *Add-in* in *Excel*. That is, it only becomes available after you activate it by ticking Solver Add-in option in *File/Options/Add-ins/Excel Add-ins/Go*. Solver allows you to find the maximum or minimum value of an output variable (i.e., the objective function), subject to the constraints you specify. These constraints can include lower and upper limits on individual input values or their combinations (e.g., budget constraints).