ASEE'S VIRTUAL CONFERENCE
At Home with Engineering Education JUNE 22 - 26, 2020 #ASEEVC
Paper ID #28486

Teach Machine Learning with Excel

Prof. Yumin Zhang, Southeast Missouri State University

Yumin Zhang is a professor in the Department of Engineering and Technology, Southeast Missouri State University. His research interests include semiconductor devices, electronic circuits, neural networks, and engineering education.

Teach Machine Learning with Excel

Yumin Zhang

Department of Engineering and Technology
Southeast Missouri State University
Cape Girardeau, MO 63701

Abstract

In the fourth industrial revolution, data becomes as important as energy and materials. The major tool to deal with the huge amount of data is machine learning, which can analyze audio and visual information effectively. The fundamentals of machine learning were introduced to Engineering Physics students in a one-credit introductory course. Since many concepts in machine learning are related to physics and engineering, the students have no trouble understanding the principles. In order to avoid the challenges of programming, Excel was used to demonstrate the training processes.

Introduction

In the last six decades, there has been an exponential growth of data, and now the quantity is far beyond the human intelligence to process it. In addition, the traditional computer programs cannot process the data effectively either. Therefore, neural network based artificial intelligence is needed. This approach emulates the learning processes of the human brain, so it is also called machine learning (ML).

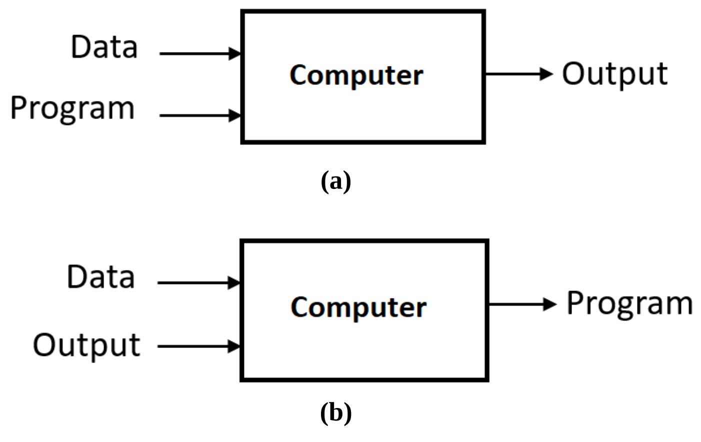


Fig. 1 Schemes of (a) traditional programing vs. (b) machine learning.

The traditional way of data processing is algorithm based, with the computer being a static device to execute the program, which is shown in Fig. 1(a). In this approach, the program is customized to the specific data to be processed. The optimization of this process relies on humans to update the program. On the other hand, machine learning has a different scheme, which is shown in Fig. 1(b). First, humans need to design the configuration of the neural network and set up a number of parameters. Second, in a supervised training process, the system is fed with both the input data and correct output, and the system will be optimized by comparing the calculated output with the provided correct ones. With enough training processes, the system can process new data with high confidence [1].

The first generation of a neural network is called a *perceptron* [2], which is shown in Fig. 2. A perceptron has three layers of artificial neurons: an input layer, a hidden layer, and an output layer. In a densely connected perceptron, each neuron in the hidden layer is connected to all the neurons in the input and output layers. The data from the input layer are summed together with different weights at the neurons in the hidden layer. This happens between the hidden layer and the output layer as well. These weights are the key parameters to be optimized in the training processes. At the beginning, these weights can be assigned with random numbers, but there are advanced approaches in initialization [3].

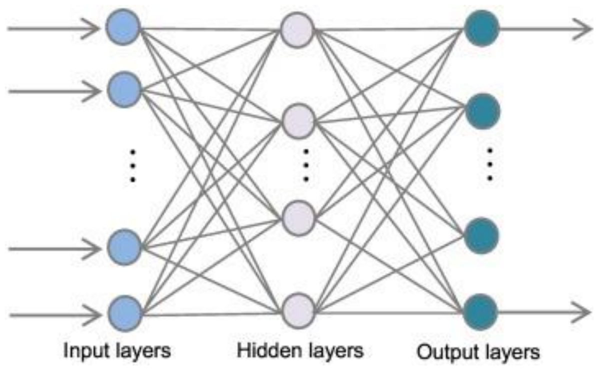


Fig. 2 Perceptron—neural network with a single hidden layer.

With a single hidden layer, the capability of a perceptron is rather limited. The natural evolution is to add more hidden layers, which is also reflected in biological evolution. For example, the cortex of turtles has three layers of neurons, but the human brain has six layers in the neocortex. The architecture with many hidden layers is called a *deep* neural network, and its operation is correspondingly called deep learning [5-6].

One of the major applications of ML is on image recognition, where the data is in the two-dimensional (2D) format. If a traditional neural network is used, the 2D matrix needs to be flattened into a one-dimensional (1D) vector. In the 2D format, there are strong correlations among the neighboring pixels, but this important information becomes intricate in the 1D format. Therefore, a new approach was developed, which is called a convolutional neural network (CNN) [7-8]. In a conventional CNN, the hidden layers are also 2D matrices, which are called filters.

The architecture of a CNN follows the occipital lobe of the human brain, where the visual information from the retina is processed in several steps. In the primary visual cortex region, only the primitive patterns are detected, such as vertical and horizontal edges of an object. The results from the primary visual cortex are fed into the secondary region, and so on. There has been tremendous progress in image recognition with CNNs, which exceed human performance in many areas now, such as cancer cell detection from CT images [9-10].

Besides image recognition, neural networks can also translate languages or analyze audio information, which is the function of the temporal lobe of the human brain. Unlike the information in an image, audio signals carry time series information, and thus memory is intimately involved. On one hand, an audio signal can be recorded in 1D vector, and there is no 2D correlation. On the other hand, there is a temporal correlation within the signal. For example, a word in a sentence often refers to another word in an earlier sentence. Therefore, the neural networks need a special configuration to process audio information, and the popular type is called a recurrent neural network (RNN) [11].

The most exciting applications of ML belong to the type of neural network with reinforcement learning capability, which includes game playing and automatic driving [12-13]. The counterpart with similar functions in the human brain is the frontal lobe, where a reward pathway plays an important role in learning, as well as addiction. The major difference between humans and animals lies at the frontal lobe, so the neural network with this type of architecture can substitute humans for many jobs, and we will be accompanied by robots with learning capabilities in the near future.

In Spring 2019, ML was introduced in a one credit hour course, and students met once a week in learning the related concepts and methods. The content of this course covered the perceptron model and the deep neural network in detail, and the more advanced topics were also introduced, such as the CNN, RNN, and reinforcement learning.

In order to lower the threshold of programming, Excel was used in training the simple neural networks in this course. Since the math formulae were very simple and the training process was very short, Excel is capable of handling these procedures. In addition, the data of every layer of the neural network was able to be presented on the same page, and the algorithm could be revised very easily. It proved to be a good platform to introduce ML to students without extensive programming background.

Structure and Procedure

There are several popular open source packages for ML, and the most popular ones are TensorFlow and PyTorch, which were developed by Google and Facebook, respectively. However, it takes some time to learn how to use them, and thus they are not good options for this introductory course. On the other hand, if students do not practice training neural networks, their knowledge on ML would be rather shallow. Therefore, Excel was selected as the tool for simple ML projects that can demonstrate the training procedures.

A simple example is temperature conversion, although it is very easy in conventional calculations: $y = w \cdot x + b$, where x and y are the temperatures in Celsius and Fahrenheit, and the two constants are w = 1.8 and b = 32. With the approach of ML, a training data set with temperature pairs in Celsius and Fahrenheit (x, y) are provided, and the weights (w, b) in the neural network will be optimized. A more general neural network is shown in Fig. 3, in this special case only one input is needed.

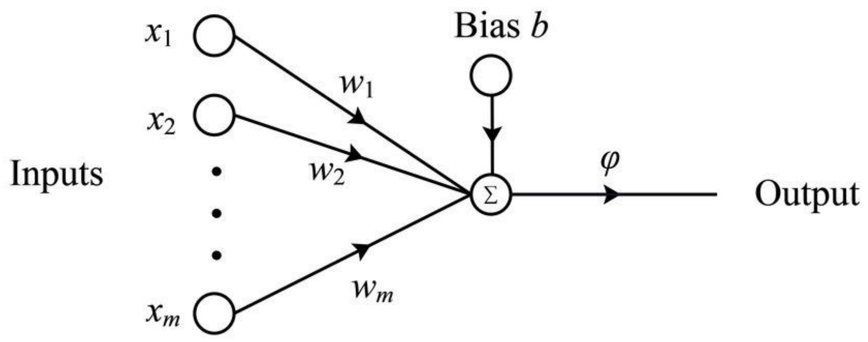


Fig. 3 A perceptron with a single output.

Assume that the temperature in Celsius is the input, and then the temperature in Fahrenheit is the output. In Fig. 3 there is another input of the constant 1, and the corresponding weight is called the *bias*. In this simple neural network, there are only two weights to be optimized, and the output can be expressed in the following formula:

$$\hat{y} = f(w \cdot x + b) \tag{1}$$

In this equation, *x* stands for the input temperature in Celsius and *w* is its weight, *b* is the bias. In general, a nonlinear function *f* is needed to generate the output. However, in this linear problem, the equation can be simplified:

$$\hat{y} = w \cdot x + b \tag{2}$$

Since the corresponding temperature in Fahrenheit—the true value of the output *y*—is provided in the training data sets, the error or loss function can be calculated:

$$E = \frac{1}{2}(y - \hat{y})^2 \tag{3}$$

In this equation, *x* and *y* are constants, *w* and *b* are variables. This function can be illustrated as a two-dimensional parabola, which is shown in Fig. 4.

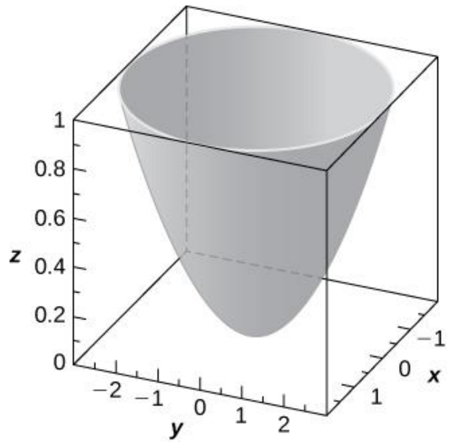


Fig. 4 Loss function of neural network.

If the two weights (*w*, *b*) are optimized, the solution is at the bottom of this parabolic surface. However, any unoptimized set of weights correspond to a point above the bottom point, and the gradient vector at this point is the direct of steepest climb. Therefore, the inverse of the gradient vector is the steepest descent path towards the bottom. From this intuition, the weights can be updated by following the direction of the steepest descent:

$$\begin{aligned} \Delta w &= -\eta \frac{\partial E}{\partial w} = \eta (y - \hat{y}) x \\ \Delta b &= -\eta \frac{\partial E}{\partial b} = \eta (y - \hat{y}) \end{aligned} \tag{4}$$

Here the learning rate parameter η is introduced, which is a positive number less than one. If the parabolic surface is very steep, and then the gradient is quite large, in this case the updated weights are likely to pass the bottom point and jump into the opposite side of the parabolic surface. In the worst-case scenario, this approach can become unstable and the error gets larger and larger. With a small learning rate, this situation can be stabilized, and the parameters move towards the bottom progressively. However, the learning process could become very slow if the learning rate is too low.

Now the learning process can start, and the weights can be updated in an iterative way:

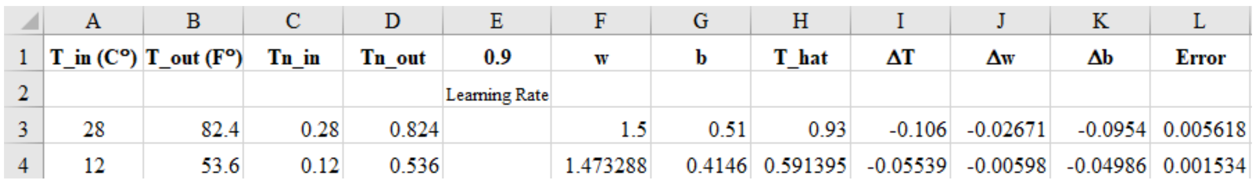
$$w_{i+1} = w_i + \Delta w_i, \qquad b_{i+1} = b_i + \Delta b_i \tag{5}$$

If there is more than one input, this approach can be applied in the same way. However, it is hard to illustrate a parabola in higher dimensions.

Results with Excel

First, we need to generate the training data. In this example, one hundred random temperatures in Celsius between 0 °C and 100 °C are the input (column A in Table I), and the corresponding temperatures in Fahrenheit are the output (column B in Table I). In order to train the neural network effectively, the input and output data should be normalized, so they are divided by 100 (columns C and D in Table I). In this normalized scheme, the bias should also be scaled in the same way. In other words, the optimized values of *w* and *b* are 1.8 and 0.32, respectively.

Table I Data and Variables in Neural Network



	A	B	C	D	E	F	G	H	I	J	K	L
1	T_in (C°)	T_out (F°)	Tn_in	Tn_out	0.9	w	b	T_hat	ΔT	Δw	Δb	Error
2					Learning Rate							
3	28	82.4	0.28	0.824		1.5	0.51	0.93	-0.106	-0.02671	-0.0954	0.005618
4	12	53.6	0.12	0.536		1.473288	0.4146	0.591395	-0.05539	-0.00598	-0.04986	0.001534

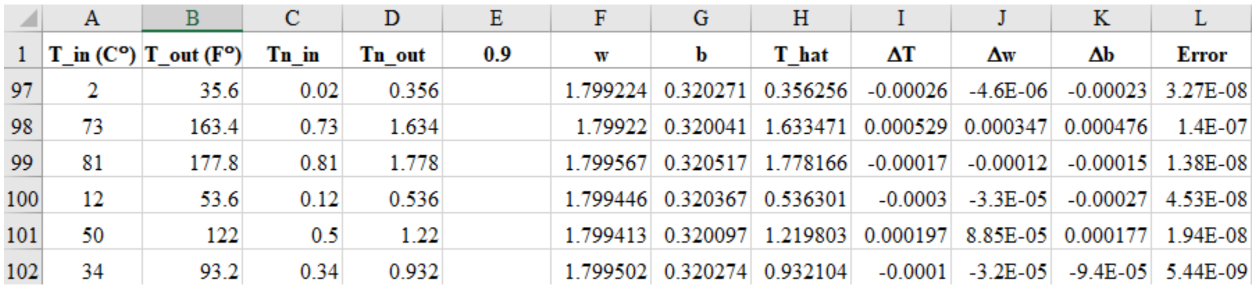
Table I shows the data and variables used in the training of this neural network, F3 and G3 are randomly selected initial values, and the relationship between the variables are shown in Table II. Since this neural network is very simple and stable, a very high learning rate of 0.9 was used. In this way the training process can be quicker. When this number is used in the expression of Δw and Δb, it is in the format of a constant (E1) in Excel. If the conventional (E1) is used in the expression instead, it would be automatically changed to (E2) in the calculations in row 5.

Table II Relationship between Variables

Cell	Variable	Meaning	Expression
E1	η	Learning rate	0.9
F4	w	Updated weight	F4 = F3 + J3
G4	b	Updated bias	G4 = G3 + K3
H4	T_hat	Output from the neural network	H4 = C4*F4 + G4
I4	ΔT	Difference between true value and output	I4 = D4 - H4
J4	Δw	Change in weight	J4 = (E1)*I4*C4
K4	Δb	Change in bias	K4 = (E1)*I4
L4	Loss	Quadratic error	L4 = (I4*I4)/2

With the first two rows (#3 and #4) calculated as shown in Table I, one can select the cells in the fourth row from F4 to L4, and then move the mouse cursor to the lower right corner. When the icon of the mouse cursor is changed to a cross, one can pull it all the way down to the end of the data. In this way, the training process is done, and Table III shows the results of the last six rows.

Table III Result of Training



	A	B	C	D	E	F	G	H	I	J	K	L
1	T_in (C°)	T_out (F°)	Tn_in	Tn_out	0.9	w	b	T_hat	ΔT	Δw	Δb	Error
97	2	35.6	0.02	0.356		1.799224	0.320271	0.356256	-0.00026	-4.6E-06	-0.00023	3.27E-08
98	73	163.4	0.73	1.634		1.79922	0.320041	1.633471	0.000529	0.000347	0.000476	1.4E-07
99	81	177.8	0.81	1.778		1.799567	0.320517	1.778166	-0.00017	-0.00012	-0.00015	1.38E-08
100	12	53.6	0.12	0.536		1.799446	0.320367	0.536301	-0.0003	-3.3E-05	-0.00027	4.53E-08
101	50	122	0.5	1.22		1.799413	0.320097	1.219803	0.000197	8.85E-05	0.000177	1.94E-08
102	34	93.2	0.34	0.932		1.799502	0.320274	0.932104	-0.0001	-3.2E-05	-9.4E-05	5.44E-09

The true value of the weight is w = 1.8, and the result generated from this neural network in column F is 1.799502. The true value of the bias is b = 0.32, and the result obtained in column G is 0.320274. In addition, the loss or error in column L becomes very low at the end of the training process (E = 5.44E-09), which indicates that it is very close to the bottom of the parabolic surface of the loss function. Therefore, the neural network is trained very well with 100 training data pairs, and it can be used to predict the temperatures in Fahrenheit from the input temperatures in Celsius.

Results with TensorFlow

This simple neural network can also be trained with TensorFlow 2.0, and the core section of the code is shown in Fig. 5. Instead of 100 data pairs, only 20 of them are used. In general, a larger data set makes the training process more effective. Fortunately, these training data can be used repeatedly in TensorFlow, and each iteration is called an *epoch*. In addition, the learning rate is set at 0.1. Compared with TensorFlow 1.x, the 2.0 version is much easier to debug. In addition, the adoption of Keras [14] makes the code very concise.

```
temp_c = np.array([0.28, 0.12, 0.03, 0.61, 0.94, 0.4, 0.09, 0.52, 0.36, 0.56, 0.09,
 0.6, 0.39, 0.29, 0.75, 0.01, 0.99, 0.64, 0.23, 0.53], dtype=float)
temp_f = np.array([0.824, 0.536, 0.374, 1.418, 2.012, 1.04, 0.482, 1.256, 0.968, 1.
328, 0.482, 1.4, 1.022, 0.842, 1.67, 0.338, 2.102, 1.472, 0.734, 1.274], dtype=floa
t)
layer_1 = tf.keras.layers.Dense(units=1, input_shape=[1])
model = tf.keras.Sequential([layer_1])
model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.Adam(0.1))
history = model.fit(temp_c, temp_f, epochs=150, verbose=False)

print("The weights are: {}".format(layer_1.get_weights()))
```

Fig. 5 Core section of the code in TensorFlow 2.0.

The loss magnitude in the training process with 150 epochs is shown in Fig. 6. The steep drop of the curve at the beginning indicates that the training process is very fast, and the final value falling to zero means a successful training. After the training process completed, the weights of this neural network are w = 1.8002 and b = 0.32003, which are very close to the true values.

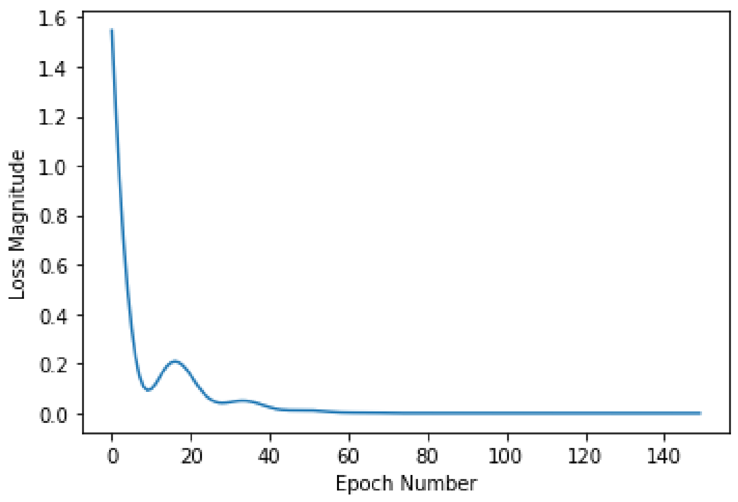


Fig. 6 Loss magnitude in the training process with 150 epochs.

Compared with Excel calculation, TensorFlow is much more powerful. Although the code is very simple, beginners need some time to learn and understand the meanings of the statements in the code. Therefore, for an introductory course to Engineering Physics students, Excel is a much better tool to start with.

Conclusion

In the wave of fourth industrial revolution, machine learning is an essential tool to analyze the huge amount of data. Therefore, engineering students should be exposed to the knowledge of machine learning, and Excel can be used as the simulator at the beginning. However, for advanced applications, TensorFlow or PyTorch should be engaged.

Acknowledgement

This work is supported in part by a GRFC grant received from Southeast Missouri State University. The author would like to thank Kyle Stedman, a student who took this introductory course, for editing the manuscript.

Reference

[1] Ethem Alpaydin, *Introduction to Machine Learning*, MIT Press (2014). ISBN-13: 978-0262028189.

[2] Marvin Minsky, Seymour A. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press (2017). ISBN-13: 978-0262534772.

[3] M. Emre Celebia, Hassan A. Kingravib, Patricio A. Velab, “A comparative study of efficient initialization methods for the k-means clustering algorithm”, *Expert Systems with Applications*, vol. 40 (1), pp. 200 - 210, 2013.

[4] Siddharth Krishna Kumar, “On weight initialization in deep neural networks”, *arXiv:1704.08863*, 2017.

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016. ISBN-13: 978-0262035613

[6] Charu C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer (2018). ISBN-13: 978-3319944623.

[7] Ragav Venkatesan, Baoxin Li, *Convolutional Neural Networks in Visual Computing*, Routledge (2017). ISBN-13: 978-1138747951.

[8] Iffat Zafar, Giounona Tzanidou, Richard Burton, Nimesh Patel, Leonardo Araujo, *Hands-On Convolutional Neural Networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python*, Packt Publishing (2018). ISBN-13: 978-1789130331.

[9] Anant Madabhushi, George Lee, "Image analysis and machine learning in digital pathology: Challenges and opportunities", *Medical Image Analysis*, vol. 33, pp. 170 - 175, 2016.

[10] Wei Shen, Mu Zhou, Feng Yang, Dongdong Yu, Di Dong, Caiyun Yang, Yali Zang, Jie Tian, "Multi-crop Convolutional Neural Networks for lung nodule malignancy suspiciousness classification", *Pattern Recognition*, vol. 61, pp. 663 - 673, 2017.

[11] Simeon Kostadinov, *Recurrent Neural Networks with Python Quick Start Guide: Sequential learning and language modeling with TensorFlow*, Packt Publishing (2018). ISBN-13: 978-1789132335.

[12] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., A Bradford Book (2018). ISBN-13: 978-0262039246.

[13] Maxim Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*, Packt Publishing (2018). ISBN-13: 978-1788834247.

[14] François Chollet, *Deep Learning with Python*, 2nd ed., Manning Publications (2017). ISBN-13: 978-1617294433.