Supplementary Material

Ice nucleation time on silicone rubber surfaces having various roughness parameters and wettability: experimental investigation and machine learning-based prediction

S. Keshavarzi[1*], A. Entezari[2], K. Maghsoudi[1], G. Momen[1], R. Jafari[1]

[1]Department of Applied Sciences, University of Québec in Chicoutimi, Chicoutimi, Québec, Canada

[2] The Hong Kong Polytechnic University, Hong Kong

Corresponding author: Samaneh Keshavarzi,

Email address: samaneh.keshavarzi1@uqac.ca

Section 1. Calculation of cooling time

When nucleation conditions were studied, it could be necessary to estimate the time it took for the droplet placed at room temperature to cool to substrate temperature. Due to the temperature dependence of nucleation, it is important to see the delay effect caused by cooling compared to the measured nucleation delay time. Hence, the cooling time of the droplet can be expressed as [1]:

$$\tau[s] = \frac{\rho V C_{pw}}{a_a A} \ln \frac{T_{wi} - T_s}{T_w - T_s} \tag{S1}$$

where ρ, V, c_p, a_a, A, T_{wi}, T_w and T_s are density, Volume, specific heat, heat transfer coefficient, droplet surface area, initial temperature of the liquid, the temperature of the phase transition and surface temperature respectively.

The calculated cooling timescale (τ) of 10 µL water droplet on sample 1 at −10 °C and −20 °C is 29 and 18 sec respectively and for 20 µL water droplet is 59 and 38 sec at −10 °C and −20 °C. Since these calculations were based on some assumptions, in this study based on some literatures, we measured the time from when the droplet was placed on the precooled substrate to the onset of freezing and use this time for analyzing. It is worth mentioning that using CFD simulation to calculate the cooling time could be considered as more accurate approach to assess whether cooling time can affect the measured nucleation time that we will investigate in our next work.

Section 2. DNN model

In DNN model, there are a number of nodes as well as an activation function in each hidden layer. Although different activation functions can be used, it is common to use one activation function for all neurons within a layer.

Each layer's output becomes the input for the next layer, which is calculated by Eq. S2.

$$h^k = \sigma^k(b^k + W^k h^{k-1}) \tag{S2}$$

where k is the layer number, h^k is the output array of the layer k (h^o is the network's input (named x here), $\sigma^k$is the activation function of the layer k, b^k is the array of bias values in layer k , W^k is the matrix of weights of the layer k. The output of the final layer is the prediction of the output variable.

To add nonlinearity to input-output relationships, activation functions are employed. When linear activation functions are applied to all layers, the prediction of DNN will be a linear combination of input variables no matter how many layers are presented. Hornnik et al. [2] demonstrated that any function with any degree of nonlinearity could be approximated by "squashing functions" and sufficient hidden layers. Squashing functions are nonlinear functions that transform input variables to a range, such as [-1, 1]. A rectifier like relu was also proposed and used during the past decade, especially in hidden layers [3]. Aside from being non-linear, this activation function also does not always activate all neurons at once. Some of the most common and well-known activation functions are shown in Table S1. There needs to be nonlinear and differentiable activation functions to allow optimization algorithms to better analyze them [4].

Table S1. Some of the most common activation functions

Activation Function	*Formula*
Sigmoid	$\mathrm{s(x)} = \frac{1}{1 + \mathrm{e}^{-\mathrm{x}}}$
Tanh	$\tanh(\mathrm{x}) = \frac{\mathrm{e}^{\mathrm{x}} - \mathrm{e}^{-\mathrm{x}}}{\mathrm{e}^{\mathrm{x}} + \mathrm{e}^{-\mathrm{x}}}$
Relu	$\mathrm{y(x)} = \max(0, \mathrm{x})$

Leaky-relu	$y(x) = \max(0, x) : 0 < \alpha \ll 1$
Linear	$y(x) = x$

A DNN is trained by minimizing the difference between the actual output variables and the predictions. For this purpose, it is important to define a cost function that represents the dissimilarity referred to above. Table S2 shows examples of some of the most common cost functions used with a DNN trained on a dataset with m samples.

Table S2. Common cost functions used in DNN

Cost function name	*Abbreviation*	*Formula*
Mean of squared error	*MSE*	$J(y, \hat{y}) = \frac{1}{m}\sum_{i}^{m}(\hat{y}_i - y_i)^2$
Mean of absolute error	*MAE*	$J(y, \hat{y}) = \frac{1}{m}\sum_{i}^{m}(\lvert\hat{y}_i - y_i\rvert)$
Mean of absolute percent error	*MAPE*	$J(y, \hat{y}) = \frac{1}{m}\sum_{i}^{m}\left\lvert\frac{\hat{y}_i - y_i}{y_i}\right\rvert$
Mean of squared logarithmic error	*MSLE*	$J(y, \hat{y}) = \frac{1}{m}\sum_{i}^{m}(\log(\hat{y}_i + 1) - \log(y_i + 1))^2$

In the deep neural network, the weights are initialized at the start of training, with the bias vector at a constant value (1 typically). Using randomized weights, the DNN model will make an initial prediction. By optimization algorithms, the weights of DNN are updated in an iterative manner until the cost function is minimized. This optimization algorithm normally is called on the training data set in a DNN. Most optimization algorithms employed in the training of a DNN are variations of batch gradient descent (BGD). Gradient descent minimizes cost through an update of the model parameters in the opposite direction of the gradient of cost function($J(\theta)$) [4,5].This process involves calculating the learning rate (η) which determines the size of the steps needed for the (local) minimum to be reached. Eq. S3 represents the BGD formulas as follows:

$$\theta_{updated} = \theta_{old} - \eta . \nabla_\theta J(\theta; x^m; y^m) \quad \text{(S3)}$$

where θ is a neuron's weight (neuron is the connection between two nodes), m is the samples size, x^m is the set of input and and y^m is the output variables, and $\nabla_\theta J(\theta)$ is the average of the partial derivative of the cost function obtained by Eq.S4:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta} J\left(\theta; x^i; y^i\right) \quad \text{(S4)}$$

where $x^i$and y^icorrespond to sample i's input and output variables. With BGD, all samples within a data set are used to update the parameters. Thus, finding an optimal point with a large dataset can be too slow. Stochastic gradient descent (SGD) [6,7], as shown in Eq. S5, adjusts parameters one by one over the training samples.

$$\theta_{updated} = \theta_{old} - \eta \cdot \frac{\partial}{\partial \theta} J(\theta; x^i; y^i) \quad \text{(S5)}$$

Each of these iterations is called an epoch and it is repeated for each sample in the dataset. Online projects can benefit from SGD because it converges to the minimum more quickly than BGD. To reach the minimum, however, it fluctuates redundantly, and may result in it continuing to update after the minimum is reached since it must process all of the samples in the training dataset. The solution to this problem was the mini-batch gradient descent, where the parameters were updated with n training examples randomly selected from the entire dataset (Eq.S6):

$$\theta_{updated} = \theta_{old} - \eta . \nabla_\theta J(\theta; x^{[i:i+n)}; y^{[i:i+n)}) \quad \text{(S6)}$$

where [i:i+n]is a subset of input (x) and output (y) variables with the size of n. Mini-batch gradient descent is plagued with the problem that the same learning rate (η) is used for updating all parameters, making it difficult to select a learning rate [8]. Dauphin et al. [9] also reported limitations to SGD with saddle points, in which slopes of target function are trending in opposite directions in two dimensions. The problem has been addressed by a number of algorithms, including Nesterov Accelerated Gradient [10], Adagrad [11], Adadelta [12], and Adam [13] algorithms. Among these algorithms, the Adadelta, RMSprop, and Adam methods are conceptually similar [14], as they update the initial learning rate for the different weights (*W_{ij}*) in the optimization process, but it is found that the Adam optimizer performed better than the other two [13]. As shown in Eq. S7 and Eq. S8, Adam algorithm modulates step t's learning rate by the first (m_t) and second moments (v_t) of past gradients.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta) \tag{S7}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2 \tag{S8}$$

The constants β_1 and β_2 are user-defined. Parameters m_t and v_t tend to remain close to zero in subsequent iterations since m_0 and v_0 are usually initialized as zero vectors. This issue can be addressed with bias-corrected formulae proposed by Kingma and Ba (Eq. S9 and Eq. S10) [13]:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \tag{S9}$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t} \tag{S10}$$

According to the Adam algorithm, the parameters will be updated as follows (Eq.S11):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon}\widehat{m_t} \tag{S11}$$

The proposed values for β_1, $\beta_2$and $\in$ are 0.9, and 0.999 and 10^{-8}, respectively. Data analysts do not usually change these parameters since they are inherent to the algorithm. In this study, we have adopted them in the optimizer algorithm, and it is to be noted that the pursuit of a global minimum is not always the right approach, as this may be associated with the issue of overfitting [15]. Fortunately, overfitting did not happen in our developed models.

Section 3. Cross correlation matrix

To investigate correlation coefficients between variables, a cross correlation matrix was used. The results presented in Table.S3 showed a strong correlation between the height parameters (S_a, S_q, S_z and S_t). Although the ice nucleation can be considered as a complex system with issues of nonlinearity, the correlation coefficients presents the linear correlation between variables. So, in order to understand which parameters play more significant role in ice nucleation time, feature importance analyze performed using machine learning algorithms.

Table S3. Roughness parameter correlation matrix

	T(°C)	*V(μl)*	*CA (°)*	*CAH(°)*	*Sa (μm)*	*Sq (μm)*	*Sz (μm)*	*St (μm)*	*Ssk*	*Time (sec)*
T (°C)	1									
V(μl)	-0.091	1								
CA (°)	0.040	-0.021	1							
CAH (°)	-0.042	0.026	-0.931	1						
Sa (μm)	0.046	-0.035	0.871	-0.982	1					
Sq (μm)	0.046	-0.037	0.818	-0.961	0.995	1				
Sz (μm)	0.052	-0.044	0.885	-0.942	0.972	0.961	1			
St (μm)	0.052	-0.044	0.898	-0.931	0.957	0.941	0.998	1		
Ssk	-0.014	0.041	-0.219	0.341	-0.430	-0.440	-0.441	-0.445	1	
Time (sec)	0.740	-0.315	0.477	-0.511	0.542	0.542	0.575	0.573	-0.224	1

References

[1] M. BT, U. MA, Z. AV, Freezing of spherical droplets in the environment of cold air, Int. J. Petrochemical Sci. Eng. (2018). https://doi.org/10.15406/ipcse.2018.03.00073.

[2] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks. 6 (1993) 861–867. https://doi.org/10.1016/S0893-6080(05)80131-5.

[3] M.J. Brown, L.A. Hutchinson, M.J. Rainbow, K.J. Deluzio, A.R. De Asha, A comparison of self-selected walking speeds and walking speed variability when data are collected during repeated discrete trials and during continuous walking, J. Appl. Biomech. 33 (2017) 384–387. https://doi.org/10.1123/jab.2016-0355.

[4] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation Functions: Comparison of trends in Practice and Research for Deep Learning, (2018) 1–20. http://arxiv.org/abs/1811.03378.

[5] M.A. Cauchy, Méthode générale pour la résolution des systèmes d'équations simultanées,

C. R. Hebd. Seances Acad. Sci. 25 (1847) 536–538. https://cs.uwaterloo.ca/~y328yu/classics/cauchy-en.pdf.

[6] M. Statistics, A Stochastic Approximation Method Author (s): Herbert Robbins and Sutton Monro Source : The Annals of Mathematical Statistics , Vol . 22 , No . 3 (Sep ., 1951), pp . 400-407 A STOCHASTIC APPROXIMATION METHOD ' x , then the method is irrespective of th, 22 (2014) 400–407.

[7] M. Statistics, Author (s): J . Wolfowitz Source : The Annals of Mathematical Statistics , Vol . 23 , No . 1 (Mar ., 1952), pp . 1-13 Published by : Institute of Mathematical Statistics Stable URL : http://www.jstor.org/stable/2236396, Ann. Math. Stat. 23 (1952) 1–13.

[8] S. Ruder, An overview of gradient descent optimization algorithms, (2016) 1–14. http://arxiv.org/abs/1609.04747.

[9] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Adv. Neural Inf. Process. Syst. 4 (2014) 2933–2941.

[10] Y.E. Nesterov, A Method of Solving a Convex Programming Problem with Convergence Rate $\mathcal{O}(1/k^2)$, Sov. Math. Dokl. (1983).

[11] J.C. Duchi, P.L. Bartlett, M.J. Wainwright, Randomized smoothing for (parallel) stochastic optimization, Proc. IEEE Conf. Decis. Control. 12 (2012) 5442–5444. https://doi.org/10.1109/CDC.2012.6426698.

[12] M.D. Zeiler, ADADELTA: An Adaptive Learning Rate Method, (2012). http://arxiv.org/abs/1212.5701.

[13] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc. (2015) 1–15.

[14] V. Asghari, Y.F. Leung, S.-C. Hsu, Deep neural network based framework for complex correlations in engineering metrics, Adv. Eng. Informatics. 44 (2020) 101058. https://doi.org/10.1016/j.aei.2020.101058.

[15] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, Y. LeCun, The loss surfaces of multilayer networks, J. Mach. Learn. Res. 38 (2015) 192–204.