Complex Networks and the Applications in Deep Learning

by

Jiahao Li, Jarvis
(1730026042)

Xiangying Wei, Shawn
(1730026116)

A Final Year Project Thesis (COMP1001; 3 Credits)
submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Science (Honors)
in
Computer Science and Technology

at
BNU-HKBU
UNITED INTERNATIONAL COLLEGE

December, 2020

DECLARATION

We commit that all the works have been done in this Project are our independent effort. We also certify that we have never submitted the idea and product of this Project for academic or employment credits.

Jiahao Li, Jarvis	Xiangying Wei, Shawn
(1730026042)	(1730026116)

Date: 12/14/2020

BNU-HKBU

United International College

Computer Science and Technology Program

We hereby recommend that the Project submitted by Jarvis and Shawn entitled "Complex Networks and the Applications in Deep Learning" be accepted in partial fulfillment of the requirements for the degree of Bachelor (Honors) of Science in Computer Science and Technology Program.

Jiahao Li, Jarvis	Xiangying Wei, Shawn
(1730026042)	(1730026116)
Date: 12/14/2020	**Date: 12/14/2020**

ACKNOWLEDGEMENT

We would like to express our great gratitude towards our supervisor, Dr. Zhiyuan Li, Goliath who had given us invaluable advice to this project.

Complex Networks and the Applications in Deep Learning

by

Jiahao Li, Jarvis
(1730026042)

Xiangying Wei, Shawn
(1730026116)

A Final Year Project Thesis (COMP1001; 3 Credits)

Submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science (Honors)

in

Computer Science and Technology

at

BNU-HKBU

UNITED INTERNATIONAL COLLEGE

December, 2020

Abstract

Deep Learning [12] is a quite exciting academic researching aspect currently, which is also one of the main parts of modern Artificial Intelligent Technologies. One of the most significant tools in Deep Learning is the Neural Network [20]. Plenty of the current AI researchers focus on designing well-performed Neural Networks. Such as the Deep Residual Neural Network [6], the VGG [7], the Alex-Net [2], or some other densely connected neural networks [4]. However, all of these well-performed neural networks are designed by human beings. It means all these kinds of neural networks have the fixed data-flow patterns and trivial topologies. Whereas for our actual brains' neural networks, the topologies are extremely non-trivial and the data-flow patterns are random. Therefore, some current AI researchers start to apply the complex graphs with non-trivial topologies to form the neural networks and study the differences between the human-designed neural network and the complex-graphs-based neural network, and they also found that the performance of the complex-graphs-based neural network is not worse than the human-designed neural network. For example, the Facebook AI Research (FAIR) team's works [9]. According to this circumstance, based on the FAIR's works, we proposed our project to try to figure out the relationship between the topology and the performance of the neural networks. We applied the randomly wired complex graphs [1][3][11] into the deep learning, and we focused on the degree distribution at present.

Contents

1.Introduction

1.1.Deep Learning and Neural Networks

Nowadays, what we call deep learning is mainly talking about neural networks. There are plenty of well-performed neural networks. Such as ResNet [6], VGG [7], Alex-Net [2], and so on. The targets of all these human-designed neural networks are catching up or even substituting the human beings in some special aspects. For example, the Image Classification problem, Object Detection problem, and so on.

In terms of the current neural networks, they are the data mapping from the input training data to the input training labels.

Definition 1. The neural network is a mapping: $\mathbb{M} \rightarrow \mathbb{L}$

where $\mathbb{M}$ represents the input training data and $\mathbb{L}$ represents the input training labels. However, the neural network is not generator the $\mathbb{L}$ directly, it will try its best to generate the predicted $\mathbb{L}^p$ which is more like the $\mathbb{L}$.

Figure 1 demonstrates the normal topology of a three layers simple neural network.

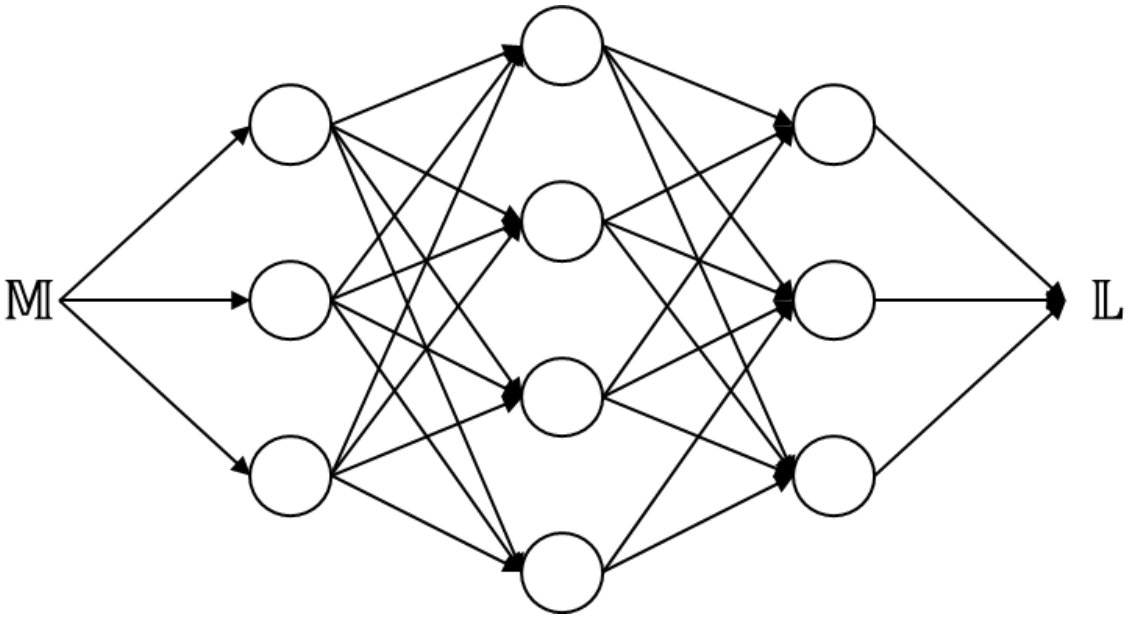


Figure 1. The topology of a three-layers densely connected simple neural network

In Figure 1, *lines* represent the weights and the bias, which are used to transmit the data along with the neural network. They are single-directional either, which means the data $\mathbb{M}$ can only be transmitted along with the weight to $\mathbb{L}$ but not transmitted back. *White nodes* represent the

computation units, which have some built-in computations to combine the training data and the weights. *Layers* are stacking by the nodes vertically; each layer would receive the data from the previous layer and transmit the result to the next layer. *The Input layer* receives the input training data $\mathbb{M}$ and *the output layer* generates the predicted label $\mathbb{L}^p$. And all the other layers between input and output layers are called *hidden layers*. Due to the graphs like Figure 1 illustrate the data-flow patterns of the neural networks, they are also called computation graphs. Meanwhile, they are the most used graphs to represent the topologies of the neural networks either. Due to the weights and bias are single directed, the computation graph is called a directed acyclic graph (DAG) either. For all the neural networks, they contain two main computations which are forward propagation and backward propagation [20]. *The forward propagation* is a process to map the input training data $\mathbb{M}$ to the predicted label $\mathbb{L}^p$. *The backward propagation* is a process to optimize the weights and bias via the derivatives. Due to all the weights and bias are initialized randomly at the beginning, it is necessary to apply the backward propagation to get the best weights and bias, in order to make $\mathbb{L}^p$ more like the $\mathbb{L}$.

Almost all the computations in forward propagation are inside the computation units. The normal pipeline of the computation units is illustrated in Figure 2.

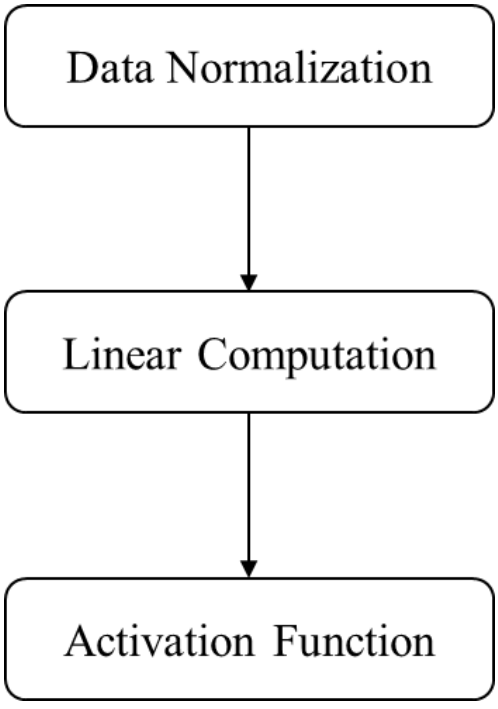


Figure 2. The normal computation pipeline of the computation unit

In Figure 2, there are three main computation parts which are Data normalization, Linear Computation, and the Activation Function [22]. The order of these three computations is random, whereas, generally, the data normalization and linear computation would be applied before the activation function. *Data normalization* is a process to centered the distribution of the data and made all the data satisfy the same distribution. There are two ways to do it, the first one is called input normalization which is used to normalize the input training data $\mathbb{M}$ and normally applied in the input layer.

Input normalization is defined as

$$\tilde{X} = \frac{X - \mu}{\sigma} \tag{1}$$

where X is the input data, μ is the mean value of X, σ is the standard deviation of X and $\tilde{X}$ is the normalized input data.

The other one is called batch normalization [21] which is normally applied in the hidden layer and also is the one always before the activation function [22].

Batch normalization is defined as

$$\overline{X} = \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2}$$

$$\tilde{X} = \gamma \overline{X} + \beta \tag{3}$$

where X is the original data, μ is the mean value of X, σ^2 is the variance of X, $\overline{X}$ is the temporary result of the normalization, and $\tilde{X}$ is the final result of the batch normalization. Furthermore, ϵ avoids the zero denominators, γ and β are used to tune the distribution of the original data.

Linear Computation is a process to combined normalized data and weights. There are also two ways to do linear computation either. One is called hypothesis function, the other one is called convolutional function.

Hypothesis function is defined as

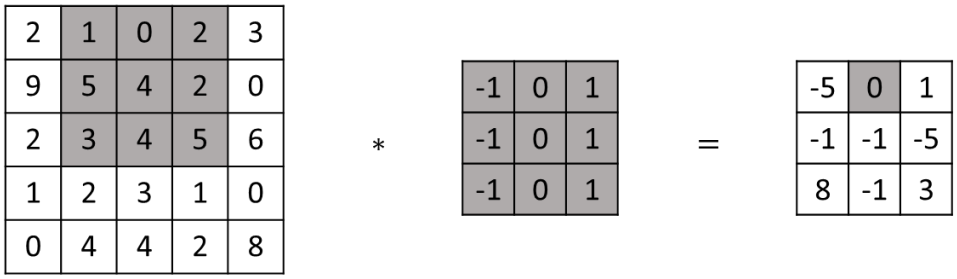
$$Z = WX + B \tag{4}$$

Convolutional function is defined as:

$$Z = W * X + B \tag{5}$$

where W is the weight, X is input data, B is the bias and Z is the result of the linear computation. The difference between these two linear computations is the method of applying the weight.

For the hypothesis function, it directly uses the weight to do the matrix multiplication with the input data. Whereas, in convolutional function, the weight is looked at as the kernel, which is used to scan the data from the up to down and left to right. When the kernel scans the data, it will be used to do the element multiplication with the scanned area firstly. Then for all elements' results, do the addition to get the final result of the convolutional function of this scanned area. There is an illustration of the convolutional function in Figure 3.



$$(-1) \times 1 + 0 \times 0 + 1 \times 2 + (-1) \times 5 + 0 \times 4 + 1 \times 2 + (-1) \times 3 + 0 \times 4 + 1 \times 5 = 0$$

Figure 3. The illustration of the convolutional production

Activation Function is a process to transformed linear form data into the non-linear form. In the deep learning aspect, activation functions [22] are significant for the performance of the neural networks. The reason is that it can transform the data form. In **Figure 1**, it is obvious that a neural network contains plenty of the computation units. If all the computation units only contain the linear computation, then all the computation units just equivalent to one computation unit. Because all the data are the linear form. The illustration is demonstrated in **Figure 4**.

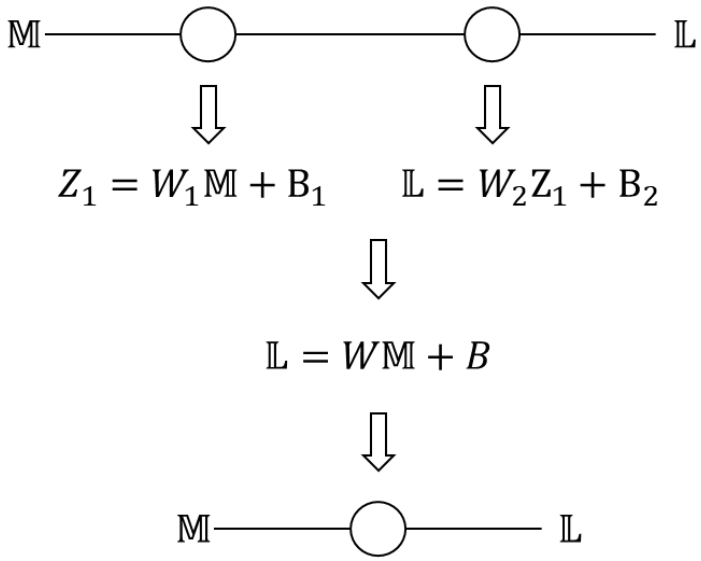


Figure 4. The reason why activation function is necessary where $W = W_1W_2$ and $B = B_1 + B_2$

Therefore, if there is no activation function in the computation units, the topologies of the neural networks are meaningless.

There are three most used activation functions [22] which are sigmoid function, tanh function and ReLu function.

The sigmoid function is defined as

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

The tanh function is defined as

$$g(z) = \frac{(e^{z} - e^{-z})}{(e^{z} + e^{-z})} \tag{7}$$

The ReLu function is defined as

$$g(z) = max(0, z) \tag{8}$$

where z in all these three functions represents the result of the linear computation.

Figure 5 shows the graphs of these three functions.

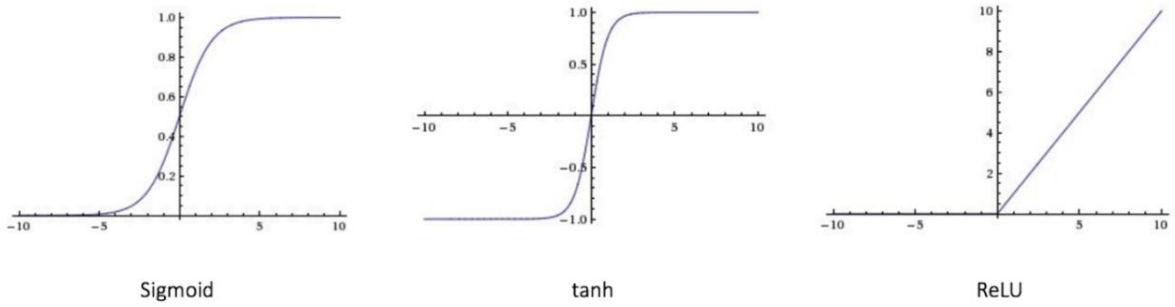


Figure 5. The image of the Sigmoid Function, Tanh Function, and $ReLu$ Function

There is a special activation function that only applies to the output layer's computation units when there are more than one computation unit in the output layer. It is called $softmax$ activation function.

The $Softmax$ function is defined as

$$g(z) = \frac{e_j^z}{\sum_{i=1}^{k} e_i^z} \tag{9}$$

where z is the result of the linear computation, k is the number of the computation nodes in the output layer, e_j^z and e_i^z are the exponential result of the j^{th} and i^{th} computation unit.

There are also some other computations in forward propagation but not in the computation units. The first one is the loss function. *The Loss function* is used to estimate the error between the input training label and the result of the output layer. The most used loss function is named Cross-Entropy loss [24]. There are two different Cross-Entropy losses, one is suitable for the binary classification problem and another one is suitable for the multi-class classification problem.

Binary Cross-Entropy Loss is defined as

$$Loss = -ylog(a) - (1-y)log(1-a) \tag{10}$$

where y is the input training label and a is the result of the output layer.

Multiple Cross-Entropy Loss is defined as

$$Loss = -\sum_{i=1}^{k} y_i log(a_i) \tag{11}$$

where k is the number of the different classes, a_i is the result of the i^{th} class in the output layer and y_i is the i^{th} value of the input training label.

For each training element in input training data, it will have a loss. The final error of the whole neural network is the addition of all the losses. The addition result is called the Cost function.

The Cost function is a function of the weight (W) and bias (b), it is used to estimate the total error of the neural network.

Binary Cross-Entropy Cost is defined as

$$Cost(W,b) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log\left(a^{(i)}\right) + (1-y^{(i)})log(1-a^{(i)}) \tag{12}$$

where m is the total number of the input training data, $y^{(i)}$ is the i^{th} input training label and $a^{(i)}$ is the result of the output layer for the i^{th} input training data.

Multiple Cross-Entropy Cost is defined as

$$Cost(W,b) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)}\left(a_j^{(i)}\right) \tag{13}$$

where m is the total number of the input training data, k is the total number of the classes, $y_j^{(i)}$ is the j^{th} input training label for the i^{th} input training data and $a_j^{(i)}$ is the result of the j^{th} computation units for the i^{th} input training data.

The next one is the regularization. It is used to reduce the overfitting problem [23]. *The overfitting* is a modeling error that happens when the model is too closely fit the input training data. On the contrary, *the underfitting* is a modeling error that happens when the model is not fitting the input training data. Sometimes, they are called *high variance* and *high bias*. Figure 6 shows the graphs of these two abnormal situations.

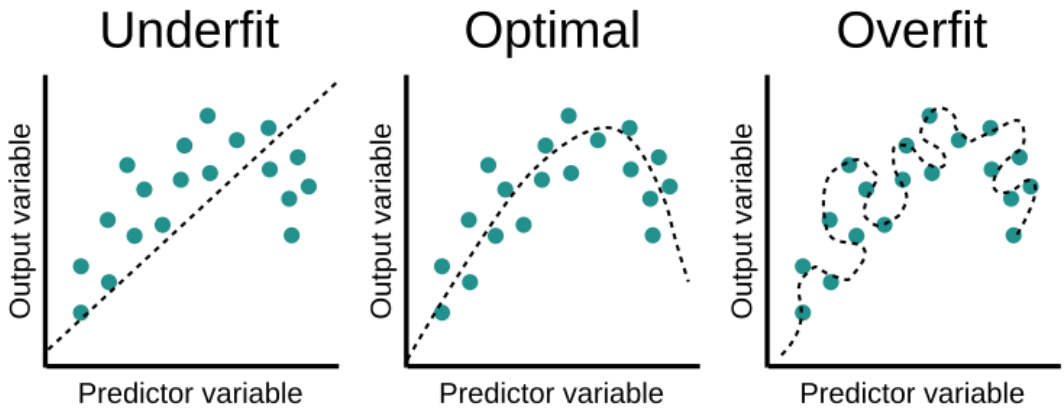


Figure 6. The overfitting (high variance) and underfitting (high bias) where the green points are the input training data and the dotted line is the fitting line

There are two most used methods to do the regularization which are L-2 Regularization and Dropout Regularization.

L-2 Regularization is defined as

$$\frac{\lambda}{2m}||W||_2^2 \tag{14}$$

where λ is the regularization coefficient, m is the total number of the input training data and $||W||_2^2$ is the L-2 norm of the weights.

Normally, the L-2 Regularization is added into the Cost function, when the λ increases, the overfitting problem is reduced. However, if λ is too large, then the model tends to the underfitting.

Dropout Regularization is the process of randomly dropped out some computation units in some layers. It has a dropout probability p ($0 < p < 1$). And for each computation unit, they will get a random probability r ($0 < r < 1$). If r is larger than p, then this computation unit will be dropped out. When p decreases, the overfitting problem is reduced. However, if p is too small, then the model tends to the underfitting.

In terms of the reason why these two regularization methods can reduce overfitting, for L-2 regularization, it gives the weights a penalty that shrinks the searching area of the weights to make

the process of finding the best weights becomes easy. For the dropout regularization, it directly makes the topology of the neural network becomes simple.

The next significant part of the computations of the neural network is backward propagation. It consists of the derivatives computation and the gradient descent [13]. *Derivatives computation* is a process to computed the derivatives of the weights (W) and bias (b) via the cost function. *Gradient descent* is a process to optimize the weights (W) and bias (b) by using the derivatives of them.

The normal gradient descent is defined as

$$x \coloneqq x - \alpha \frac{\partial f}{\partial x} \tag{15}$$

where x can be weights or bias, $\frac{\partial f}{\partial x}$ is the derivatives of x of cost function f and α is the learning rate to control the speed of the gradient descent.

Whereas, the normal gradient descent method is not efficient to optimize the weights and bias in deep learning. Therefore, some AI researchers come up with another two more efficient gradient descent methods, which are momentum gradient descent [15] and Adam gradient descent [14].

The momentum gradient descent is defined as

$$v_x^i = \beta v_x^{i-1} + (1-\beta)\frac{\partial f}{\partial x} \tag{16}$$

$$x \coloneqq x - \alpha v_x \tag{17}$$

where x can be weights or bias, $\frac{\partial f}{\partial x}$ is the derivative of x in cost function f, β is a coefficient to transform the derivative of x to be the gradient v_x^i, v_x^i is the gradient of the x, α is learning rate and i is the number of times of the gradient descent.

The Adam gradient descent is defined as

$$v_x^i = \beta_1 v_x^{i-1} + (1-\beta_1)\frac{\partial f}{\partial x} \tag{18}$$

$$s_x^i = \beta_2 s_x^{i-1} + (1-\beta_2)\frac{\partial f}{\partial^2 x} \tag{19}$$

$$\left(v_x^i\right)^{correct} = \frac{v_x^i}{1-\beta_1^i} \tag{20}$$

$$\left(s_x^i\right)^{correct} = \frac{s_x^i}{1-\beta_2^t} \tag{21}$$

$$x := x - \alpha \frac{\left(v_x^i\right)^{correct}}{\sqrt{\left(s_x^i\right)^{correct}} + \epsilon} \tag{22}$$

where x can be weights or bias, $\frac{\partial f}{\partial x}$ is the derivative of x in cost function f, β_1 is a coefficient to transform the derivative of x to be the gradient v_x^i, v_x^i is the gradient of the x, β_2 is a coefficient to transform the derivative of x to be the gradient s_x^i, s_x^i is another gradient of the x, $\left(v_x^i\right)^{correct}$ and $\left(s_x^i\right)^{correct}$ are the gradients after bias correction, α is learning rate, i is the number of times of the gradient descent and ϵ avoids the zero denominators.

In deep learning, there are also two different neural networks, except the neural networks like Figure 1. One is called Convolutional Neural Network (CNN) whose linear computation is the convolutional function. CNN is efficient for image processing. Therefore, it is normally used in image classification or object detection problems. Another one is called Recurrent Neural Network (RNN) [10]. RNN is used to process the sequence data like the natural language and the audio. Its computation units will receive two inputs, which are not the same as Figure 2. One is the current input training element (A word if the training data is a sentence), another one is the previous training elements computation result. For each input element, the RNN unit would use the same weights and bias to do the computation. Normally, the RNN contains only one computation unit and one layer. The basic computation units and topology of RNN are demonstrated in Figure 7.

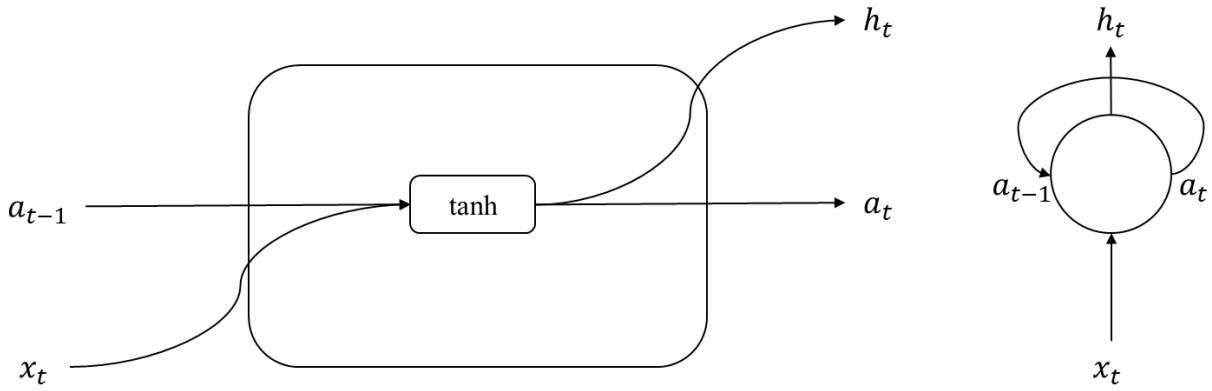


Figure 7. The computation unit and topology of RNN where a_{t-1} is the result of the previous training element, x_t is the current training element, a_t and h_t are the current results and h_t is used to compute the cost function

1.2.Artificial NN vs Human NN vs Complex-Graph-Based NN

Compared with the actual brains' neural networks, the topology of manually designed neural networks (like Figure 1) is extremely simple. Meanwhile, in our human beings' neural network, a neuron, which is simulated by a computation unit in the artificial neural network, can be activated randomly to transmit the data rather than determinately. To offer an artificial neural network non-trivial topologies and randomness, some AI researchers tried to apply the random complex graphs to generate the neural network.

In terms of Figure 1, it is obvious that the topology of a neural network is a Directed Acyclic Graph (DAG). Therefore, the scientists try to convert the random complex graphs into the DAGs and use them to generate the neural networks. In this case, we can further analyze the attributes of the complex graphs and study the relationship between the topology and the performance of neural networks. The attributes include the clustering coefficient, average path length, degree distribution, and so on. *The clustering coefficient* is used to estimate that for each vertex inside the complex graph, is it tending to cluster together. *The average path length* is used to estimate that the average number of steps along the shortest paths. *The degree distribution* is a probability distribution used to describe the connection circumstance of each vertex.

Clustering coefficient is defined as

$$C_i = \frac{3 \times number\ of triangles\ that\ include\ i}{number\ of\ all\ triplets} \tag{23}$$

where c_i is the clustering coefficient of vertex i.

The Average path length is defined as

$$l = \frac{1}{n(n-1)} \sum_{i \neq j} d(v_i, v_j) \tag{24}$$

where l is the average path length, n is the total number of the vertices and $d(v_i, v_j)$ is the shortest path length between v_i and v_j. If v_i cannot reach v_j, then $d(v_i, v_j) = 0$.

The Degree distribution is defined as

$$p(k) = \frac{n_k}{n} \tag{25}$$

where k is the number of the degree, n_k is the number of the vertices whose degrees are k and n is the total number of the vertices in the graph.

In terms of the current study, the Facebook AI Research (FAIR) team [9] used 3 classic random graphs (Erdős-Rényi graph (ER) [3], Barabási-Albert graph (BA) [1] and Watts-Strogatz graph (WS) [11]) to generate the neural networks and trained neural networks with 1000-classes ImageNet dataset. Compared with some human-designed neural networks (ResNet [6], VGG [7], Alex-Net [2], etc.), they found that the complex-graph-based neural networks are not worse than the artificial neural network, they claimed usually better. The results of the FAIR team are demonstrated in Figure 8.

network	top-1 acc.	top-5 acc.	FLOPs (M)	params (M)
MobileNet [16]	70.6	89.5	569	4.2
MobileNet v2 [41]	74.7	-	585	6.9
ShuffleNet [56]	73.7	91.5	524	5.4
ShuffleNet v2 [31]	**74.9**	92.2	591	7.4
NASNet-A [58]	74.0	91.6	564	5.3
NASNet-B [58]	72.8	91.3	488	5.3
NASNet-C [58]	72.5	91.0	558	4.9
Amoeba-A [35]	74.5	92.0	555	5.1
Amoeba-B [35]	74.0	91.5	555	5.3
Amoeba-C [35]	**75.7**	**92.4**	570	6.4
PNAS [27]	74.2	91.9	588	5.1
DARTS [28]	73.1	91.0	595	4.9
RandWire-WS	**74.7**$_{\pm 0.25}$	**92.2**$_{\pm 0.15}$	583$_{\pm 6.2}$	5.6$_{\pm 0.1}$

Figure 8. The result of WS complex-graph-based neural network in FAIR's works [9] comparing with other artificial neural networks in 1000-classes image classification problem

In summary, for our project, we study the relationship between the topology and the performance of the neural network based on the works of Facebook AI Research (FAIR). We apply the three random complex graphs (Erdős-Rényi graph (ER) [3], Barabási-Albert graph (BA) [1], and Watts-Strogatz graph (WS) [11]) either and do the following works.

(1) Generates random complex graphs.

(2) Converts the random graphs into DAGs

(3) Defines the computation pipeline of each vertex in DAGs and also the data-flow patterns

(4) Forms the neural network.

(5) Repeats the FAIR's experiments to find the best neural network.

(6) Uses the best neural network to study the relationship between the degree distribution and the performance of the neural network.

2.Related Work

Due to our project based on the 3 classic random graphs algorithms and FAIR's works [9], we firstly introduce some basic ideas of them in this section.

2.1.Random Graphs

2.1.1.Erdős-Rényi Graph (ER)

Paul Erdős and Alfréd Rényi proposed the first random graph model in 1960 [3] which is called Erdős- Rényi random graph (ER graph).

The ER graph is defined as $G(n,p)$, where n is the total number of vertices in the ER graph and p is the wire-able probability between two different vertices. The outline of generating the ER graph is described in Figure 9 and the details of the algorithm are clarified in section 3.

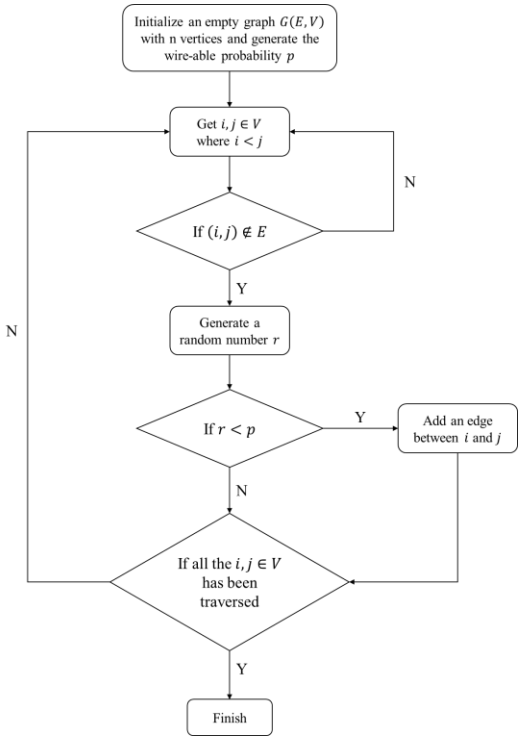


Figure 9. The outline of generating the ER graph

Thus, the number of the edges (e) in an ER graph is $0 \leq e \leq C_n^2$.

The degree distribution of the ER graph is defined as

$$p(k) = C_{n-1}^{k} p^k (1-p)^{n-1-k} \tag{26}$$

where C_{n-1}^{k} is the number of possible combinations for the vertex which connected with k other vertices, p^k is the probability that C_{n-1}^{k} occurs, and $(1-p)^{n-1-k}$ is the probability that C_{n-1}^{k} doesn't occur.

Meanwhile, the degree distribution of the ER graph is close to the Poisson distribution ($p(k) = \frac{(np)^k}{k!}e^{-np}$) when $n \to \infty$ and np is constant.

2.1.2.Barabási-Albert Graph (BA)

The Barabási-Albert model [1] generates random graphs that are scale-free. The scale-free means that the degree distribution of the graph follows the power-law distribution. Also, the graph is sel-similar. For example, in the BA graph, no matter how many vertices it has, most of the vertices would always tend to connect with the vertices whose degree is large.

The BA graph is defined as $G(n, e)$, where n is the total number of the vertices in the BA graph and e is the total number of the wire-able edges for each new vertex ($e \ll n$). The outline of generating the BA graph is described in Figure 10 and the details of the algorithm are clarified in section 3.

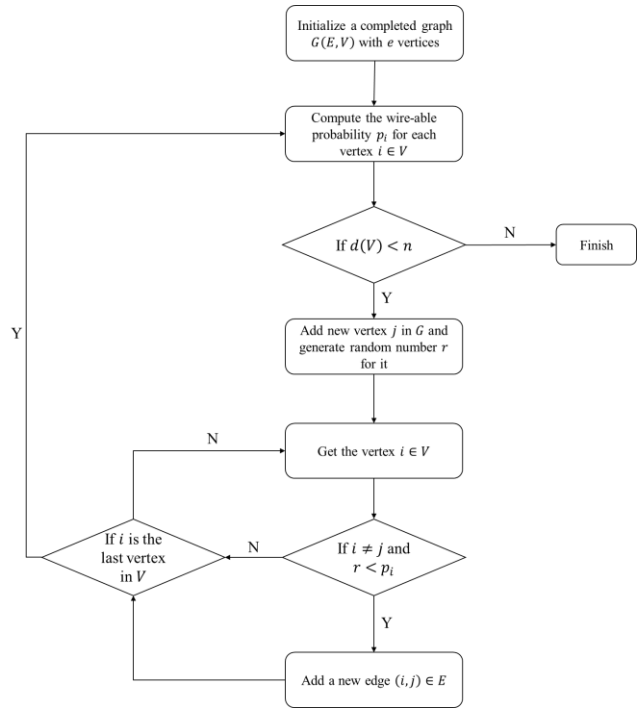


Figure 10. The outline of generating the BA graph where $d(V)$ is the current total number of the vertices in the BA graph

Wire-able probability of vertex i in BA graph is defined as

$$p_i = \frac{k_i}{\Sigma k_j} \tag{27}$$

where k_i is the degree of vertex i, and Σk_j is the total degree for all existing vertices.

The degree distribution of the BA graph is defined as

$$p(k) \sim k^{-3} \tag{28}$$

2.1.3.Watts-Strogatz Graph (WS)

The Watts-Strogatz model (WS) [11] generates random graphs with the 'Small-World' attributes, which means most vertices are not neighbors of one another, but the neighbors of any given vertices are likely to be neighbors of each other and most vertices can be reached from every other vertex by a small number of steps.

The WS graph is defined as $G(k, n)$, where k is the number of the nearest neighbors with which each vertex would connect at the beginning, and n is the total number of the vertices for the WS graph ($k \ll n$). The outline of generating the WS graph is described in Figure 11 and the details of the algorithm are clarified in section 3.

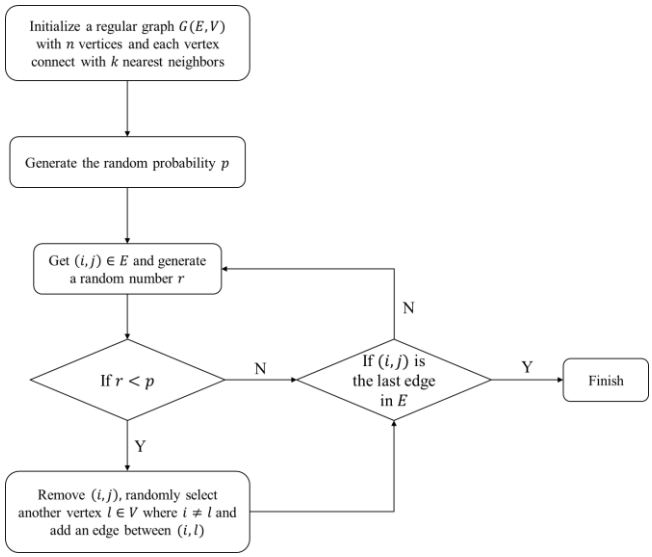


Figure 11. The outline of generating the WS model

For the WS graph when the wire-able probability p equal 0, actually the degree distribution $p(k)$ is the Delta Function as k.

Meanwhile, when the wire-able probability p larger than 0, the degree distribution of the WS graph will have two situations which are $p(c_i^1)$ and $p(c_i^2)$.

The $p(c_i^1)$ is defined as

$$p(c_i^1) = c_{\frac{k}{2}}^{n}(1-p)^n p^{\frac{k}{2}-n} \tag{29}$$

where c_i^1 represents the number of non-rewired edges of vertex i.

The $p(c_i^2)$ is defined as

$$p(c_i^2) = c_{pN\frac{k}{2}}^{d-n-\frac{k}{2}} p^{d-n-\frac{k}{2}} (1-p)^{pN\frac{k}{2}-\left(d-n-\frac{k}{2}\right)} \tag{30}$$

where c_i^2 actually represents that how many new neighbors that vertex i will obtain after rewiring and N is the total number of the vertices of the WS graph.

The degree distribution of the WS graph is defined as

$$p(d) = \sum_{n=0}^{f(d,k)} p(c_i^1) p(c_i^2) = \sum_{n=0}^{f(d,k)} c_{\frac{k}{2}}^{n} (1-p)^n p^{\frac{k}{2}-n} \frac{\left(\frac{pk}{2}\right)^{d-n-\frac{k}{2}}}{\left(d-n-\frac{k}{2}\right)!} e^{-\frac{pk}{2}} \tag{31}$$

where $f(d,k) = \min\left(d-\frac{k}{2}, \frac{k}{2}\right)$ and $d \geq \frac{k}{2}$.

2.2.Randomly Wired Neural Network (RWNN)

2.2.1.Training Pipeline

Regarding the trained pipeline of the RWNN in the Facebook AI Research team [9], they applied three classic random graphs (ER [3], BA [1], WS [11]) to be the computation graphs of the neural networks. Afterward, an index was assigned to each vertex in the random graphs. For the ER model, the assignment is random; for the BA model, the vertex which is added into the model earlier will get a smaller index; and for the WS model, the index is assigned counter-clockwise. Then they formed DAGs by directly using the small index vertex point to the large index vertex. Next, stacked DAGs together to form the neural networks, and each DAG is a DAG layer. Finally, they used ReLu-Conv-BN pipeline in each computation unit in the DAGs, where Conv is the convolutional function and BN is the batch normalization.

2.2.2.RWNN Model

The Facebook AI Research team generated two Randomly Wired Neural Network Models. One is the small regime model with only three DAG layers, the other one is the regular regime model with four DAG layers. For each DAG layer, it contains 32 vertices, except the first DAG layer in the regular regime model which only has a half. Each DAG layer will twice the number of the channels of the input image and shrink the image size to half. The models' descriptions show in Table 1.

Stage	Small Regime	Regular Regime
Conv1	$3 \times 3\ conv, C/2$	
Conv2	$3 \times 3\ conv, C$	$DAG\ Layer$ $N = 16, C$

Conv3	$DAG\ Layer$ $N = 32, C$	$DAG\ Layer$ $N = 32, 2C$
Conv4	$DAG\ Layer$ $N = 32, 2C$	$DAG\ Layer$ $N = 32, 4C$
Conv5	$DAG\ Layer$ $N = 32, 4C$	$DAG\ Layer$ $N = 32, 8C$
Classifier	$1 \times 1\ conv, 1280 - d$ $global\ average\ pool, 1000 - d\ fc, softmax$	

Table 1. The architecture of the RWNN Model where N is the total number of the vertices of the DAG layer and initial value of C is 78 for small regime model and 154 or 109 for regular regime model

2.2.3.Training Result

The Facebook AI Research team has trained the small regime and regular regime model with the ImageNet dataset on different random graphs. The ER model with wire-able probability equal 0.2 gets the best performance, whose top-1 accuracy is 73.4%; the BA model with the number of wire-able edges equal 5 gets the best performance, whose top-1 accuracy is 73.2%; the WS model with the number of nearest neighbor equal 4 and wire-able probability equal 0.75 gets the best performance, whose top-1 accuracy is 73.8%. The total results of each RWNN model are demonstrated in Figure 12.

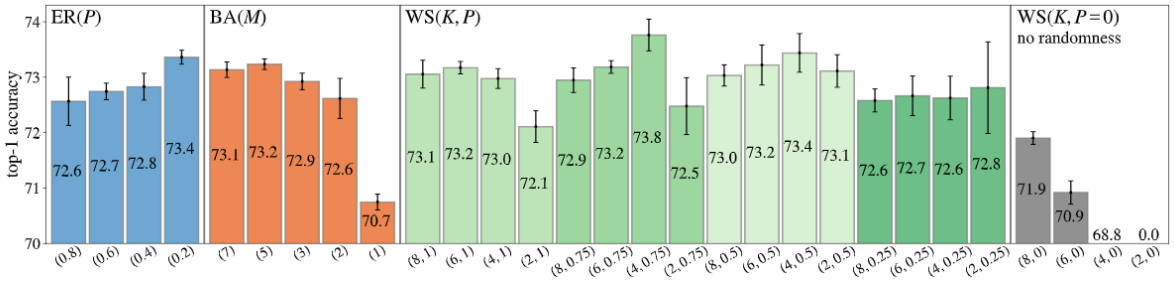


Figure 12. The top-1 accuracy of all RWNN models in FAIR's works [9]

Afterward, the Facebook AI research team also compared the best RWNN model which is the WS(4, 0.75) with some human-designed neural networks such as ResNet [6], ShuffleNet [4], and so on. They found that the performance of the RWNN model is similar to the well-performed human-designed neural networks, they claimed usually better. The results are demonstrated in Figure 8.

3.Methodology

We now introduce the methodologies which are applied in our project. We follow the methods proposed by the FAIR team, rebuild the training pipeline, and do more experiments on more random graphs.

3.1.Random Graph Generator

In terms of the random graph generator, we apply the algorithms of the three classic random graphs which are the ER graph [3], the BA graph [1], and the WS graph [11]. Afterward, we save the random graphs as edge information and adjacent matrices (There are some illustrations of the generated graphs in the appendix). *The edge information* is a list that contains all the edge descriptions like (i, j), where i and j are a pair of vertices. *The adjacent matrix* is a matrix that describes the connection situation of the graph, where the number of the columns and rows represent each vertex in a graph. If there is an edge between (i, j), then the elements (i, j) and (j, i) will be 1, otherwise would be 0.

ER Graph. We generate an empty graph with n vertices and generate a wire-able probability p at first. For each pair of vertices, we generate a random number r. If $r < p$, then we add an edge between them. Through the whole generating processes, we avoid the self-loop and repetitive edges. Algorithm 1 shows the pseudo-code for the ER graph generator.

ER(p) Generator Pseudo-Code

Initialize an empty graph with n vertices and no edges.
Generate the wire-able probability p.
For each pair of the vertices (i, j):
    Generate the random number $r = random(0, 1)$
    If $r < p$ and $adjMatrix(i, j) = 0$ and $adjMatrix(j, i) = 0$:
        Add an edge between $(i,\ j)$
        Set $adjMatrix(i, j) = adjMatrix(j, i) = 1$

Algorithm 1 The pseudo-code of the ER model

BA Graph. We generate a completed graph with e vertices ($e \ll n$) at first, where n is the total number of vertices. Meanwhile, compute the wire-able probability p_i for all the existed vertices i, according to the current degree distribution. Afterward, add the new vertex with a random number r. If $r < p$, then add the edge between the new vertex and vertex i till there are e edges connected to the new vertex or already traverse all the existed vertices. Algorithm 2 shows the pseudo-code for the BA graph generator.

BA(e) Generator Pseudo-Code

Initialize a completed graph with n vertices.
Compute all the p_i with current degree distribution.
For remaining $n - e$ vertices:
    Add new vertex j into the graph.
    Generate the random number $r = random(0, 1)$.
    For all existed vertices i:
        If $r < p_i$ and the number of the edges for vertex j is smaller than e:
            Add an edge between i and j
            Set $adjMatrix(i, j) = adjMatrix(j, i) = 1$
    Refresh all the wire-able probability p_i and compute p_j.

Algorithm 2 The pseudo-code of the BA model

WS Diagram. We generate a regular graph with each vertex connecting with its k nearest neighbors at first. Meanwhile, generate a wire-able probability p. Then generate the random number r for each edge. Afterward, rewire the edge with other vertices by following the clockwise when $r < p$. Algorithm 3 shows the pseudo-code for the WS graph generator.

WS(k, p) Generator Pseudo-Code

Initialize a regular graph with each vertex connecting to its k nearest neighbors.
Generate the wire-able probability p.
For each edge (i, j):
    Generate the random number $r = random(0,1)$.
    If $r < p$:
        Delete the edge (i, j).
        Set $adjMatrix(i, j) = adjMatrix(j, i) = 0$.
        Randomly pick up a vertex l inside the WS model except i and j.
        While $adjMatrix(i, l) = adjMatrix(l, i) = 1$:
            Randomly pick up a vertex l again.
        Add an edge between (i, l).
        Set $adjMatrix(i, l) = adjMatrix(l, j) = 1$

Algorithm 3 The pseudo-code of the WS model

3.2.Neural Network Generator

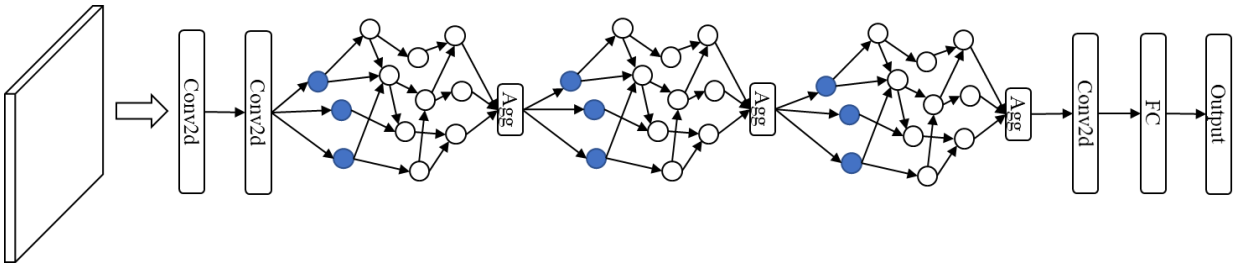


Figure 13. The architecture of the neural network for image recognition

In Figure 13, the first cube is the input image, the Agg means the aggregation function, the input image firstly is feed into two individual convolutional functions, FC is the full-connected layer and before the FC layer there is another individual convolutional function to change the data size matching the size of the FC layer

For the computation pipeline of each vertex in the DAG layer, we apply three computations which are demonstrated in Figure 14.

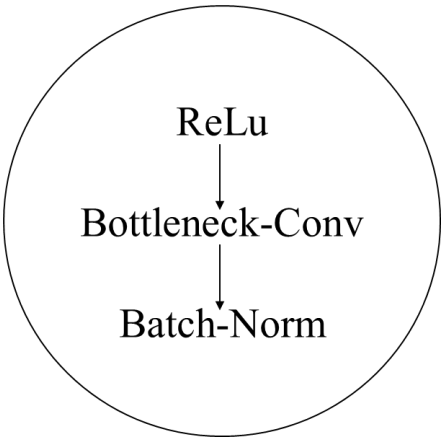


Figure 14. The computation pipeline of the vertex in DAG

The computation pipeline consists of three parts which are one ReLu activation function, one bottleneck-conv, and one batch normalization. The bottleneck-conv is a combination of two convolutional functions which are called depth-wise convolution and point-wise convolution. It is used to shrink the space cost of the convolutional function. *The depth-wise convolution* is only used to change the size of the image. *The point-wise convolution* is only used to change the channels of the image.

Regarding the topology of the whole neural network, we stack three DAG layers with the same number of vertices and same graph types (ER, BA, WS, or some other graphs) together. The topology of the neural network is illustrated in Figure 13. There are a total of six convolutional layers, three individually and three DAGs. Between two DAG layers, there is an aggregation function to aggregate all the output data from the vertices which only have indegree in DAGs, and each DAG layer shrinks the size of the input image to a half and enlarges the number of channels of the input image twice. The model description is demonstrated in Table 2.

Stage	Pipeline	Output Size	Output Channels Amount
$Conv - BN - ReLu$	$kernel \rightarrow 3 \times 3$ $stride \rightarrow 2$ $padding \rightarrow 1$	$I/2$	$C/2$
$Conv - BN$	$kernel \rightarrow 3 \times 3$ $stride \rightarrow 2$ $padding \rightarrow 1$	$I/4$	C
DAG	$\#\ of\ Vertices \rightarrow N$	$I/8$	C
DAG	$\#\ of\ Vertices \rightarrow N$	$I/16$	$2C$
DAG	$\#\ of\ Vertices \rightarrow N$	$I/32$	$4C$
$ReLu - Conv - BN$	$kernel \rightarrow 1 \times 1$ $stride \rightarrow 1$	$I/32$	1280
FC	$avgPooling \rightarrow (1,1)$ $FC \rightarrow (1280, \#\ of\ classes)$ $Softmax$	$\#\ of\ Classes$	1

Table 2. The architecture of the neural network

In Table 2, C is the size of the original channel and I is the size of the original image. For DAG layers, all the vertices are applied bottleneck convolution, which consists of one depth-wise convolution with 3×3 kernel and one point-wise convolution with 1×1 kernel. Only the input vertices are used to change the original image (Blue vertices in **Figure 13**), whose stride for depth-wise conv is 2 and 1 for point-wise.

In terms of the data-flow pattern, for each DAG, we apply a queue to control it. When the input data coming, the input vertices which only have outdegree (Blue vertices in Figure 13) will be put into the queue firstly. Meanwhile, we generate a list whose indices match the indices of the vertices in the DAG, which is used to store the input data for all the vertices. Once the vertex has been put into the queue, it will use the corresponding data in the list to do the computation. As long as the vertex finishes the computation, it will be popped out from the queue and the result of it will be saved into the corresponding position of the list, according to the edge data of the DAG. Afterward, if the number of the data of some indices in the list is equal to the amount of the indegree of the corresponding vertices, these vertices will be put into the queue. Then, repeat the above processes till the queue becomes empty again. Eventually, the output of the DAG layer will be sent to the aggregation function. In the aggregation function, there is an aggregation weight which is used to aggregate the data in the channel-wise. The aggregation weight will be activated by the tanh

function at first. Then multiply with the coming data. Meanwhile, it will also be optimized like the kernel of the convolutional function.

3.3.Datasets

In terms of the FAIR team's work, they applied a 1000-classes ImageNet dataset and trained with 100 epochs, 8192 batch size, and 256 RTX 2080ti 24GB GPUs. However, our computation resources are not suitable for this huge training. We have tried to train a 1000-classes ImageNet dataset with only 32 batch size and 100 epochs. Whereas, it almost consumed us a week to do the training for just one model. Therefore, we have to use another small dataset to continue our experiments, which is named Canadian Institute For Advanced Research 10 (CIFAR10).

CIFAR10 dataset contains 60000 32×32 color images in 10 different classes, which are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6000 images for each class. Comparing with the ImageNet dataset which contains 1 million 224×224 color images of 1000 classes, CIFAR10 is a smaller and more suitable dataset for us to continue our experiments.

4.Experiments and Results

Due to the limitation of the computation resources, we select the small dataset named CIFAR10 with a 10-classes classification task to train our models. Meanwhile, for each random graph, we use 2 different random seeds to generate the graphs and use top-1 accuracy to estimate the performance of the neural networks. Besides, we apply the stochastic gradient descent, label smoothing regularization [18], cosine warm restart learning rate decay [19] to train with 32 batch size, 100 epochs, 0.01 learning rate, and only one RTX 2080ti 24GB GPU. *The stochastic gradient descent* is the 1-batch size mini-batch Adam gradient descent (Formula 18). *The label smoothing regularization* is a process to smooth input training label, to reduce the overfitting problem of the model (Figure 6). *The cosine warm restart learning rate decay* is a process to decay the learning rate during the training, according to the cosine function. 'Restart' means when the learning rate equals 0, it will be set back to the original value and decay again.

4.1.FAIR Experiments Repetition

In terms of the first stage experiment, we try to use our neural network generator to repeat the experiments of the Facebook AI Research team. To check the performance of our neural network generator with different types of random graphs.

ER Neural Network Generator. For the ER model, we train it with 32 vertices. Meanwhile, we pick up 4 different wire-able probabilities from 0.2 to 0.8 with an interval equal to 0.2. The result is demonstrated in Figure 15.

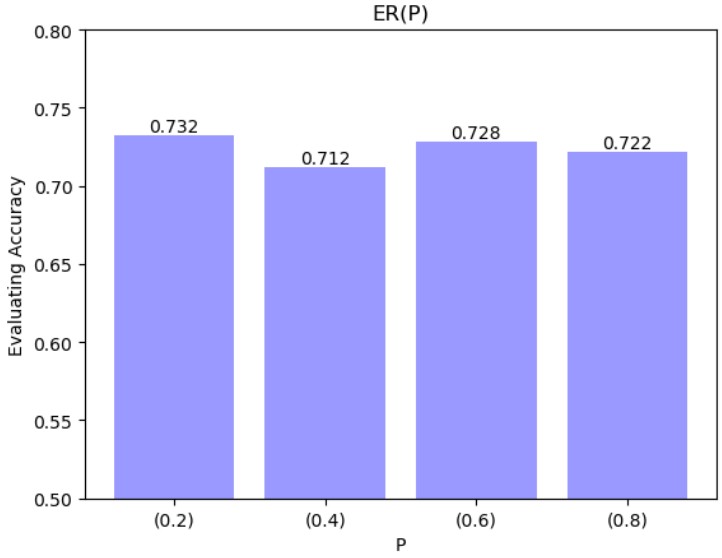


Figure 15. The performance of the ER model

Via the result of the experiments of the ER model, we get the best-performed ER model whose wire-able probability is 0.2 and top-1 accuracy is equal to 73.2%. Compared with the experiments of the FAIR team, our result is only 0.2% lower than their ER(0.2) model.

BA Neural Network Generator. For the BA model, we train it with 32 vertices either. Meanwhile, we pick up 5 different wire-able edges from 1 to 7. The result is demonstrated in Figure 16.

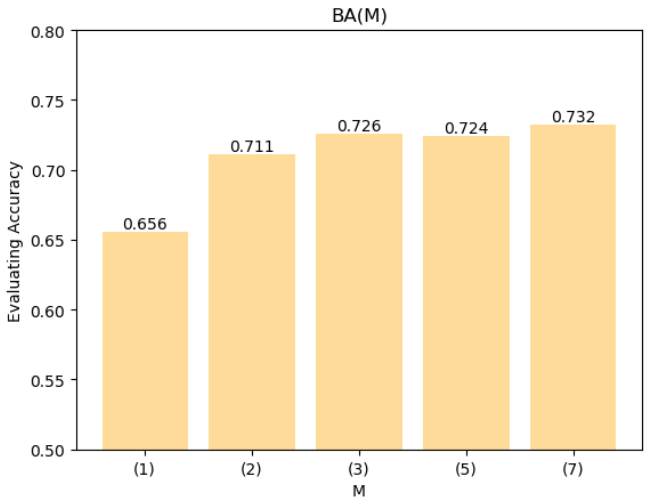


Figure 16. The performance of the BA model

Through the result of the experiments of the BA model, we get the best-performed BA model whose wire-able edge is 7 and top-1 accuracy is equal to 73.2% either. Compared with the works of the FAIR team, our result is 0.1% higher than their BA(7) model and equal to their best BA model which is BA(5).

WS Neural Network Generator. For the WS model, we trained it with 32 vertices. Due to the WS model has two parameters, therefore, we pick up 5 different values for wire-able probabilities and 4 different values for nearest neighbors. Afterward, we train 20 models. The result is demonstrated in Figure 17.

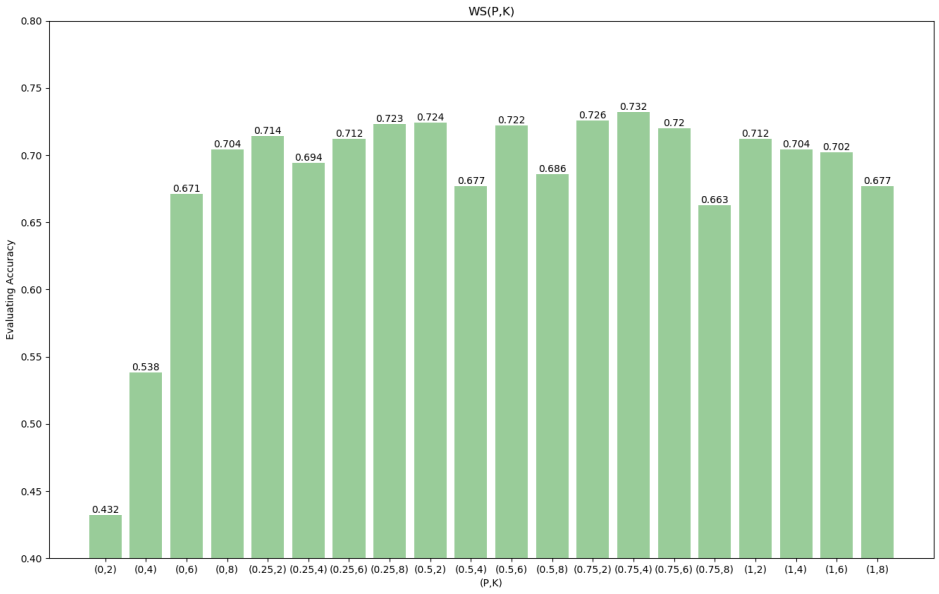


Figure 17. The performance of the WS model

Eventually, we find that the best WS model is WS(4, 0.75) whose top-1 accuracy is 73.2%. Compared with the result of the FAIR, their best model of the WS graph is slightly higher than ours, which is 73.8%.

Therefore, compared to our models with the FAIR's experiments, our models have a similar performance with them, which means that our models can be applied for the follow-up experiments.

4.2.Stage-Two Experiments with ER Model

Although we get similar results with the FAIR's paper [9], our results show that all the best models in different types of the random graphs have similar performance. Therefore, according to our target which is that we want to find how the degree distribution affects the performance of the

neural network. We select the ER(0.2) model to do the follow-up experiments because the degree distribution of the ER graph will be similar to the Poisson distribution when $n \to \infty$ and np is constant. Thus, we come up with the sub-target which is that whether the degree distribution of the topology of the neural network is more similar to the Poisson distribution, the performance is better.

In the beginning, we train the models with the number of vertices in the DAG layer between 4 and 256. And sample the number of vertices as the two of the power of two. The result is demonstrated in Figure 18.

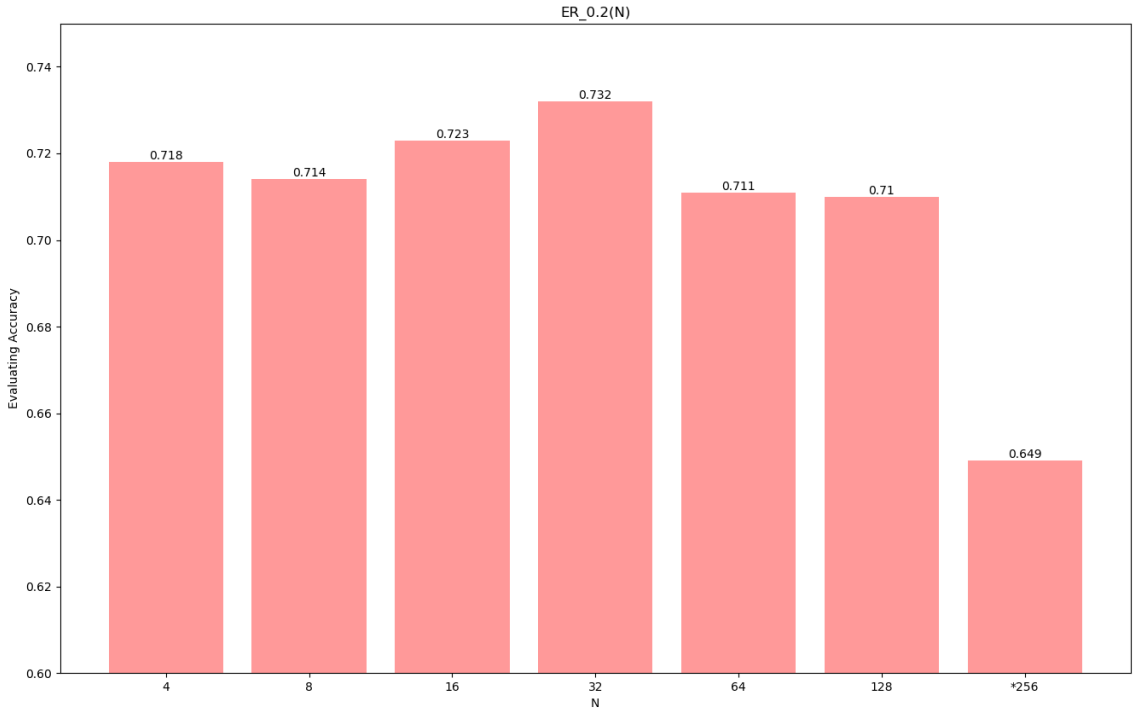


Figure 18. The performance of ER(0.2) model when the number of the vertices is the two of the power of two from 4 to 256, the '*' in 256 vertices means that this model has not finished training, due to the limitation of the computation resources

Through the training result in Figure 18, we still find that the ER(0.2) model with 32 vertices has the best performance. Afterward, we compare the degree distribution of the ER(0.2) model with 32 vertices with the Poisson distribution. The result is demonstrated in Figure 19.

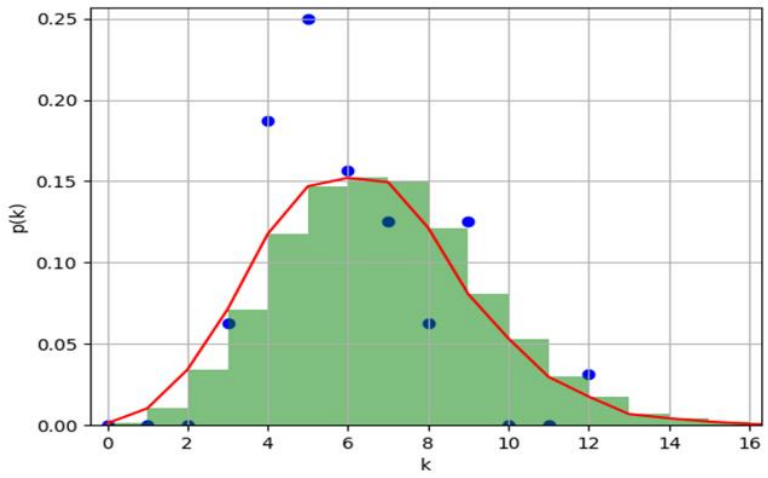


Figure 19. The comparing of the degree distribution of ER(0.2) model with 32 vertices and Poisson distribution where the blue points are the degree distribution of ER(0.2) model with 32 and the red line is the Poisson distribution

In Figure 19, it is obvious that 32 is not enough vertices to make the degree distribution of the ER(0.2) model similar to the Poisson distribution. Therefore, we shrink the sampling interval of the number of vertices. For the follow-up experiments, we sample all the possible numbers of the vertices between 4 and 74 and redo the training. The result is demonstrated in Figure 20.

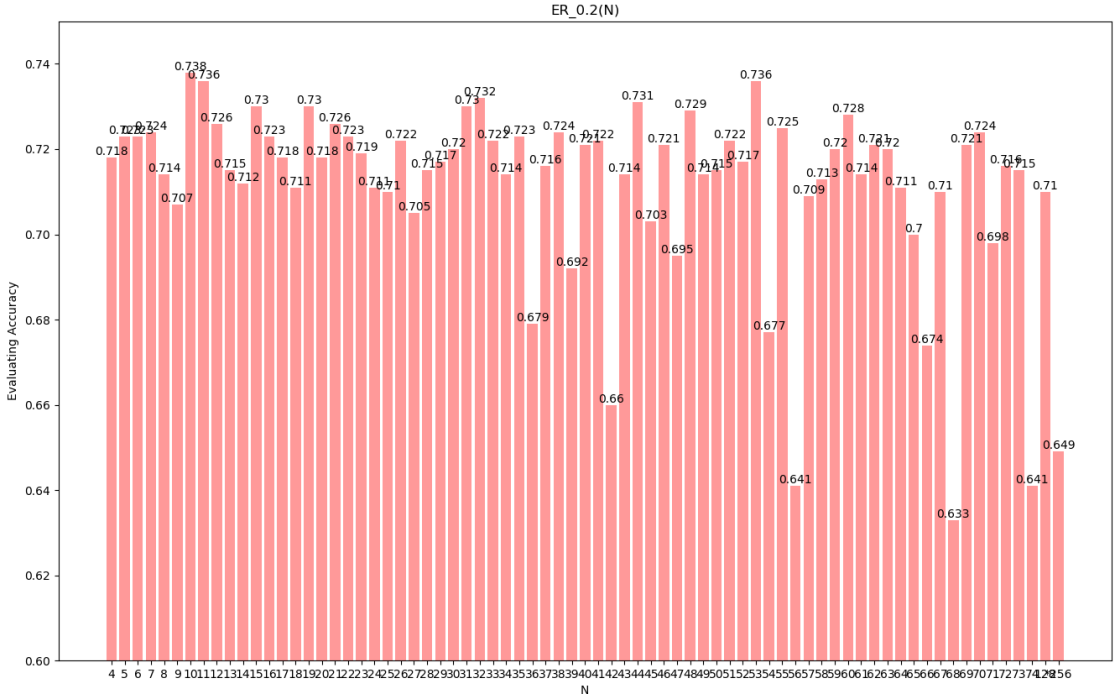


Figure 20. The performance of sampling all the possible number of the vertices from 4 to 74 for the ER(0.2) model

Through the result in Figure 20, we find that when the number of the vertices larger than 32, some models that obtained better performance than the number of the vertices equal 32. For example, when the number of the vertices equal 53, the top-1 accuracy of the ER(0.2) model equal 73.6%, which is 0.4% higher than the top-1 accuracy of the ER(0.2) model with 32 vertices. And only 0.2% lower than the best model of experiments in FAIR. Whereas, compared performance of the models, the models whose number of the vertices smaller than 32 is more robust than the number of the vertices larger than 32.

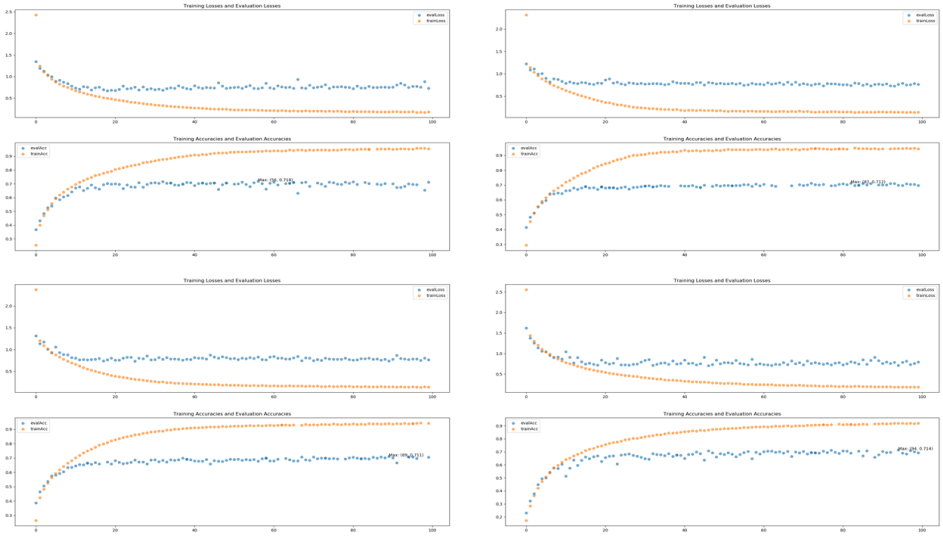


Figure 21. The loss and accuracy for the ER(0.2) model where the left-up corner is 4 vertices, left-bottom corner is 24 vertices, right-up corner is 14 vertices and right-bottom corner is 34 vertices.

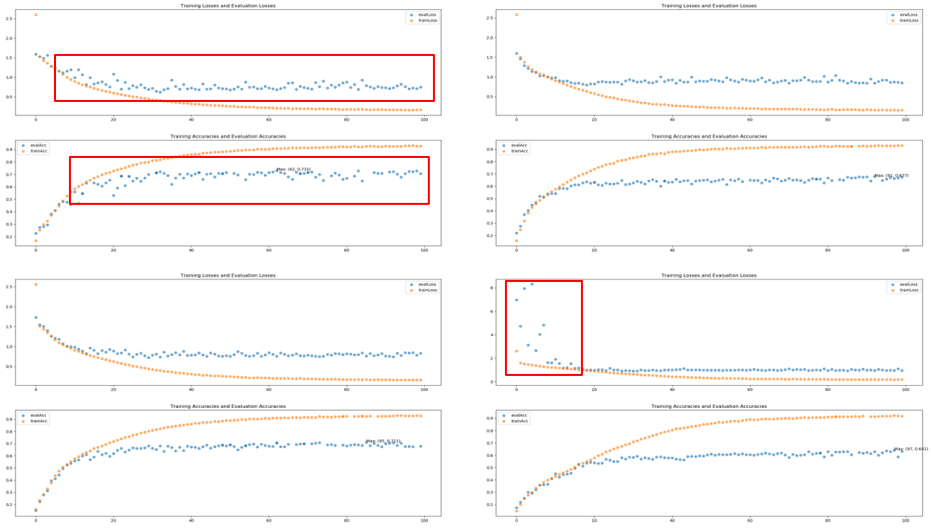


Figure 22. The loss and accuracy for the ER(0.2) model where the left-up corner is 44 vertices, left-bottom corner is 64 vertices, right-up corner is 54 vertices and right-bottom corner is 74 vertices.

In Figure 21 and Figure 22, it is obvious that when the number of the vertices larger than the 32, the loss and accuracy in the red rectangles are fluctuated, which means the models are overfitting (high variance). It represents that when the number of the vertices larger than the 32, the performance of the model will become unstable (More training results will be shown in the appendix).

4.3.Space Cost and Time Cost Estimators

Due to we don't have the powerful computation resources, we must track the space and time cost of each model, and find the trained limits of our computation resources. To let us know what kinds of experiments are suitable for both of our computation resources and our targets.

In terms of the space cost, we compute the space complexity of our model and use the big-oh complexity to be the estimator to fit the space cost tendency. The notations' descriptions of the space complexity are shown in Table 3.

Param Name	Notation
$Image\ Size$	I
$\#\ of\ input\ vertices\ for\ DAG\ i$	k_i
$\#\ of\ the\ total\ vertices\ for\ DAG$	N
$Class\ Size$	CL
$Batch\ Size$	BS

Table 3. The notation description for space complexity

We separate the space complexity into two portions, which are the space cost of the output data and the space cost of parameters of the models. According to the models' descriptions, all the space costs of outputs of the convolutional functions and DAGs are demonstrated in Table 4.

Output	
$Original\ Image$	$3I^2$
$Conv1$	$\frac{39}{4}I^2$
$BN1$	$\frac{39}{4}I^2$
$ReLu$	$\frac{39}{4}I^2$
$Conv2$	$\frac{78}{16}I^2$
$BN2$	$\frac{78}{16}I^2$
$DAG1$	$\frac{39}{8}I^2N + \frac{117}{32}I^2k_1 + \frac{78}{64}I^2$

$DAG2$	$\frac{39}{16}I^2N + \frac{39}{128}I^2k_2 + \frac{39}{64}I^2$
$DAG3$	$\frac{39}{32}I^2N + \frac{39}{256}I^2k_3 + \frac{39}{128}I^2$
$ReLu$	$\frac{39}{128}I^2$
$Conv3$	$\frac{5}{4}I^2$
$BN3$	$\frac{5}{4}I^2$
$AvgPooling$	1280
FC	CL
$Softmax$	CL

Table 4. The space consumption of outputs with unit byte

The space cost of the outputs is defined as

$$S_{Output} = \frac{751}{8}BSI^2 + \frac{273}{16}BSI^2N + \frac{117}{16}BSI^2k_1 + \frac{39}{64}BSI^2k_2 + \frac{39}{128}BSI^2k_3 + 4BSCL + 2560BS(32)$$

Based on the models' descriptions, all the space cost of parameters of the convolutional functions and DAGs are demonstrated in Table 5.

Param	
$Conv1$	1053
$Conv2$	27378
$DAG1$	$6786N$
$DAG2$	$25740N - 12870k_2$
$DAG3$	$100152N - 50076k_3$
$Conv3$	399360
FC	$1280C$

Table 5. The space consumption of parameters with unit byte

The space cost of the parameters is defined as

$$S_{Param} = 1283373 + 398034N - 38610k_2 - 150228k_3 + 3840CL \quad (33)$$

The total space cost is $\frac{273}{16}\mathrm{BSI^2N} + 117\mathrm{BSI^2k_1} + \frac{39}{64}\mathrm{BSI^2k_2} + \frac{39}{128}\mathrm{BSI^2k_3} + \frac{751}{8}\mathrm{BSI^2} + 4\mathrm{BSCL} + 2560\mathrm{BS} + 398034\mathrm{N} - 38610\mathrm{k_2} - 150228\mathrm{k_3} + 3840\mathrm{CL} + 1283373.$

For the big-oh complexity, it only contains the highest exponential item in the equation. Therefore, after removing all the low exponential items, the formula is $\frac{273}{16}BSI^2N + \frac{117}{16}BSI^2k_1 + \frac{39}{64}BSI^2k_2 + \frac{39}{128}BSI^2k_3$. Afterward, according to the property of the DAG, the k_1, k_2 and k_3 always smaller than N, therefore, eliminate the other 3 power items except BSI^2N.

The big-oh space complexity is defined as

$$O(BSI^2N) \tag{34}$$

According to Formula 34, it is obvious that the image size of the input image has the most effect on the space cost. The batch size and the number of the vertices in the DAG layer also influence the space cost.

Regarding the time cost, we also apply the big-oh complexity to estimate the time-cost tendency. The notations' descriptions of the time complexity are shown in Table 6.

Param Name	Notation
$Image\ Size$	I
$Conv\ Kernel\ Size$	f
$Output\ Image\ Size$	H
$Fill\ Pixel\ Size$	p
$Step\ Length$	s
$Batch\ Size$	BS
$Channel\ Size$	C
$\#\ of\ vertices\ in\ DAG$	N
$\#\ of\ edges\ in\ DAG$	E
$Access$	C_i
$Exit\ Channel$	C_o
$\#\ of\ input\ vertices\ in\ DAG$	K
$Degree\ of\ Vertex$	d
$In\ Degree\ of\ Vertex$	d_i
$Out\ Degree\ of\ Vertex$	d_o

Table 6. The notation description for time complexity

We separate the time cost into two parts either, one is the time-cost of the forward propagation and another one is the time-cost for the backward propagation. According to the computations of the convolutional function, all the time costs of the forward propagation are demonstrated in Table 7.

Forward Propagation	
$Conv1$	$BS * N * (2cf^2 - 1) * \frac{c}{2} * \frac{I^2}{4}$
$BN1$	$\frac{c}{2} * \frac{I^2}{4}$
$Conv2$	$BS * N * (2cf^2 - 1) * C * \frac{I^2}{16}$
$BN2$	$\frac{c}{2} * \frac{I^2}{16}$
$DAG1$	$BS * \frac{I^2}{64} * \{K_1[c^2 + c(f^2 - 1) - 1] + \left(\frac{N}{2} - K_1\right)[2c^2 + c(2f^2 - 1) - 1]\}$

$DAG2$	$BS * \frac{I^2}{256} * \{K_2'[4c^2 + 2c(f^2 - 1) - 1] + (N - K_2')[8c^2 + 2c(2f^2 - 1) - 1]\}$
$DAG3$	$BS * \frac{I^2}{1024} * \{K_3'[16c^2 + 4c(f^2 - 1) - 1] + (N - K_3')[32c^2 + 4c(2f^2 - 1) - 1]\}$
$Conv3$	$(8c - 1) * 1280$
$BN3$	$1280 * \frac{I^2}{64 * 64}$

Table 7. The time consumption of forward propagation

The time cost of the forward propagation is defined as

$$T_{sn} = BS * \frac{I}{2^n}^2 * \left\{K_{n-2}\left[2^{n-3^2}c^2 + 2^{n-3}c(f^2 - 1) - 1\right] + (N - K_{n-2})\left[2^{n-3^3}c^2 + 2^{n-3}c(2f^2 - 1) - 1\right]\right\} \quad (35)$$

where $(\mathrm{n} \geq 3)$.

In terms of the backward propagation, we set the training time requisition of the operation on each vertex v as f and divide backward propagation into two parts, one is computing the weight and neuron error, another one is optimizing the weight. For each vertex, the computation time is $f_1(v) = 2 \times d_o - 1$. Then, the time cost f_2 of the weight value is modified to be a degree dependent first-order relation. According to the degree distributions of the ER [3], BA [1], and WS [11], we can know that time cost in backward propagation is $T = f_1 + f_2$. Afterward, we combine the backward propagation with forward propagation, we can derivate the time cost as Formula 36. The big-oh time complexity is defined as

$$O(BSI^2N^3) \quad (36)$$

According to Formula 36, it is obvious that the number of the vertices of the DAG has a great effect on the time cost, and the image size has a huge influence on the time cost either, but slighter than the number of the vertices.

Meanwhile, among space and time complexities, when the number of vertices increasing, the upper bound of the time cost will increase in an exponential way like Figure 23. For example, when we training the model with 256 vertices, it cost us almost 2 days to train. Whereas, the model still has not finished training during these 2 days. However, for the model with only 4 vertices, it only takes us about 4 hours to do the training.

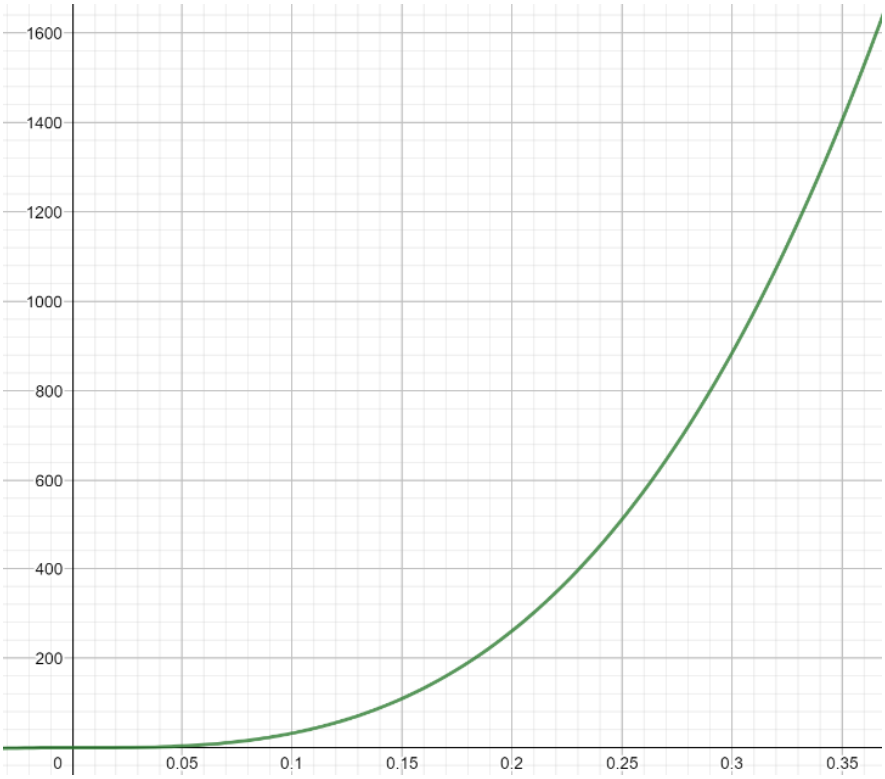


Figure 23. The graph of time complexity when both the image size and batch size equal 32

6.Conclusion

Through all the experiments we have done, we find that our models have a similar performance with the FAIR team, which means the complex-graph-based neural networks are better-performed indeed by comparing with some artificial neural networks. Meanwhile, we also find that, when the number of vertices increasing, the topology of the neural network will become too random to predict. The performance of the neural network sometimes will become unstable and perform high variance either. This circumstance points out that, for the graphs whose degree distributions are the binomial distribution, there will be a critical number of the vertices, which will make the performance of the models get the best with high robust (In our experiments, it is the 32 vertices). Rather than when the degree distribution more similar to the Poisson distribution, the performance of the neural network becomes better. Because the Poisson distribution requires a huge number of vertices, however, the larger number of the vertices, the more unstable model will get.

7.Future Work

7.1.Lambert W Function

When we do the stage two experiments, we try to find the exact number of the vertices which can make the binomial distribution equal to the Poisson distribution. Therefore, we form the equation like Formula 37.

The equation between the binomial distribution and the Poisson distribution is defined as

$$p(k) = C_{n-1}^{k} p^{k} (1-p)^{n-1-k} = \frac{(np)^{k}}{k!} e^{-np} \tag{37}$$

Afterward, we simplify Formula 37 and got Formula 38.

The simplified equation of Formula 38 is defined as

$$e^{an-b} = \frac{n^{k}}{(n-1)(n-2)\dots(n-k)} \tag{38}$$

where $a = [ln(1-p) + p]\ and\ b = (k+1)ln\,(1-p)$.

Compared Formula 37 with the Lambert W Function.

The Lambert W Function is defined as

$$we^{w} = z \tag{39}$$

$$e^{-cx} = a_0 \frac{\prod_{i=1}^{\infty}(x - r_i)}{\prod_{i=1}^{\infty}(x - s_i)} \tag{40}$$

where r_i and s_i are distinct real constants and x is a function of the eigenenergy.

These two equations are similar if we take $an - b$ as $-cx$ and n as x. Therefore, in the future, we can try to use the lambert w function to solve Formula 38 and get the exact number of the vertices which can satisfy Formula 37. Afterward, we can use this number of vertices to do the training and check the exact relationship between the Poisson distribution and performance of the neural network.

7.2.Relational Graph

The relational graph [5] is a new method to apply the complex graphs into deep learning which is proposed by the FAIR team. These kinds of graphs remove the directions between the vertices in DAGs and make each pair of vertices in the complex graphs can exchange information, rather than the single direction in DAG. In this case, the relational graphs remove some constraints in DAG, which let almost all kinds of graph attributes can be used to estimate the relationship between

topology and performance of the neural networks. Therefore, in the future, we can change our neural network generator to support transforming the complex graphs as relational graphs.

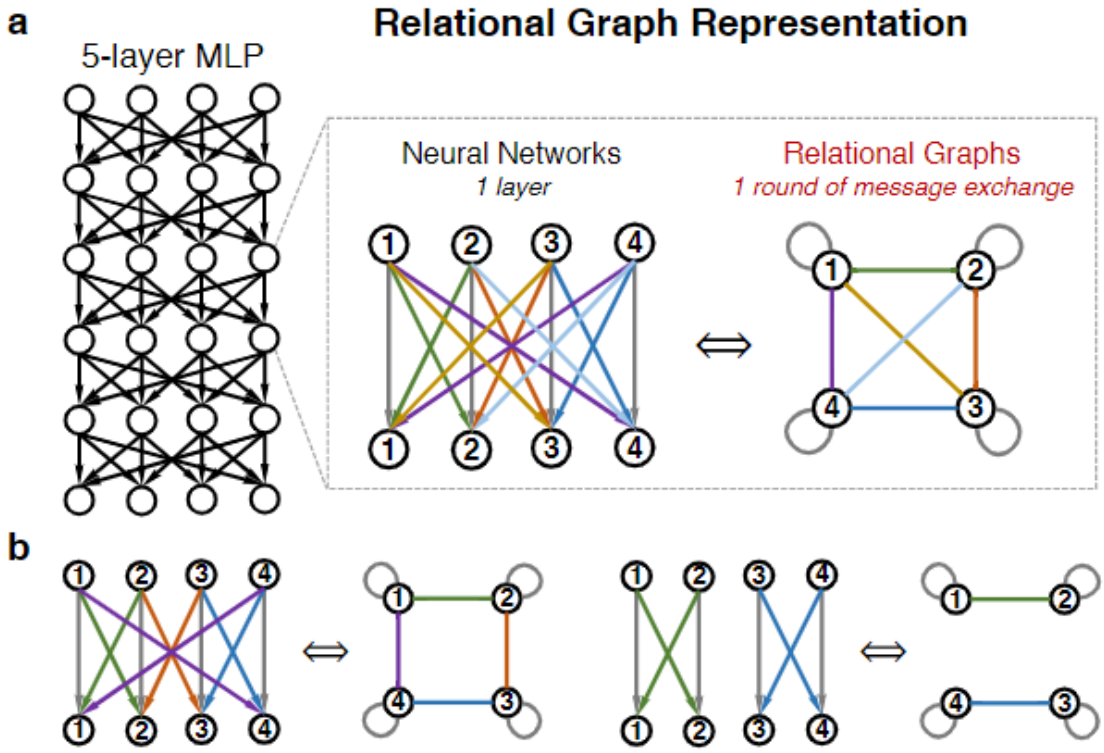


Figure 24. The relational graph representation in Facebook AI Research's new paper Graph Structure of Neural Network [5]

7.3.Recurrent Neural Network

Because all of our works right now are just for image classification with Convolutional Neural Network. In the future, we are excited to change our neural network generator to support the Recurrent Neural Network units and try to find the relationship between the topology and performance of the Recurrent Neural Network [10].

7.4.Real-World Networks

Due to our actual brains' neural networks are the real-world networks. Therefore, we think about whether there are some common topologies or special attributes for the real-world networks that will make the performance of the neural network become better. Therefore, in the future, we will collect some topologies from the real-world, like Twitter, Instagram, and Weibo, to check whether the real-world networks are good as the human-designed neural networks.

Reference

[1] Albert, R., & Barabási, A. L. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, *74*(1), 47.

[2] Krizhevsky, A., Sutskever, I., & Geoffrey, G. E. (2010). ImageNet Classification with Deep Convolutional. *Neural Networks*.

[3] Erdős, P., & Rényi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, *5*(1), 17-60.

[4] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

[5] You, J., Leskovec, J., He, K., & Xie, S. (2020, November). Graph structure of neural networks. In *International Conference on Machine Learning* (pp. 10881-10891). PMLR.

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[7] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[8] Mercier, L. (2016). Dynamic Erdős-Rényi random graph with forbidden degree.

[9] Xie, S., Kirillov, A., Girshick, R., & He, K. (2019). Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1284-1293).

[10] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), 2673-2681.

[11] Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, *393*(6684), 440-442.

[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[13] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[14] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[15] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, *12*(1), 145-151.

[16] Cortes, C., Mohri, M., & Rostamizadeh, A. (2012). L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*.

[17] Wager, S., Wang, S., & Liang, P. S. (2013). Dropout training as adaptive regularization. *Advances in neural information processing systems*, *26*, 351-359.

[18] Hou, J., Zeng, H., Cai, L., Zhu, J., Chen, J., & Ma, K. K. (2019). Multi-label learning with multi-label smoothing regularization for vehicle re-identification. *Neurocomputing*, *345*, 15-22.

[19] Gotmare, A., Keskar, N. S., Xiong, C., & Socher, R. (2018). A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*.

[20] Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 2018). San Francisco, CA: Determination press.

[21] Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in neural information processing systems* (pp. 2483-2493).

[22] Sharma, S. (2017). Activation functions in neural networks. *Towards Data Science*, *6*.

[23] Hawkins, D. M. (2004). The problem of overfitting. *Journal of chemical information and computer sciences*, *44*(1), 1-12.

[24] Busoniu, L., Ernst, D., De Schutter, B., & Babuska, R. (2010). Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *41*(1), 196-209.

Appendix

ER Graph Image. The images below are the illustrations of some of the ER graphs for the first DAG layer, we used for our neural network generator.

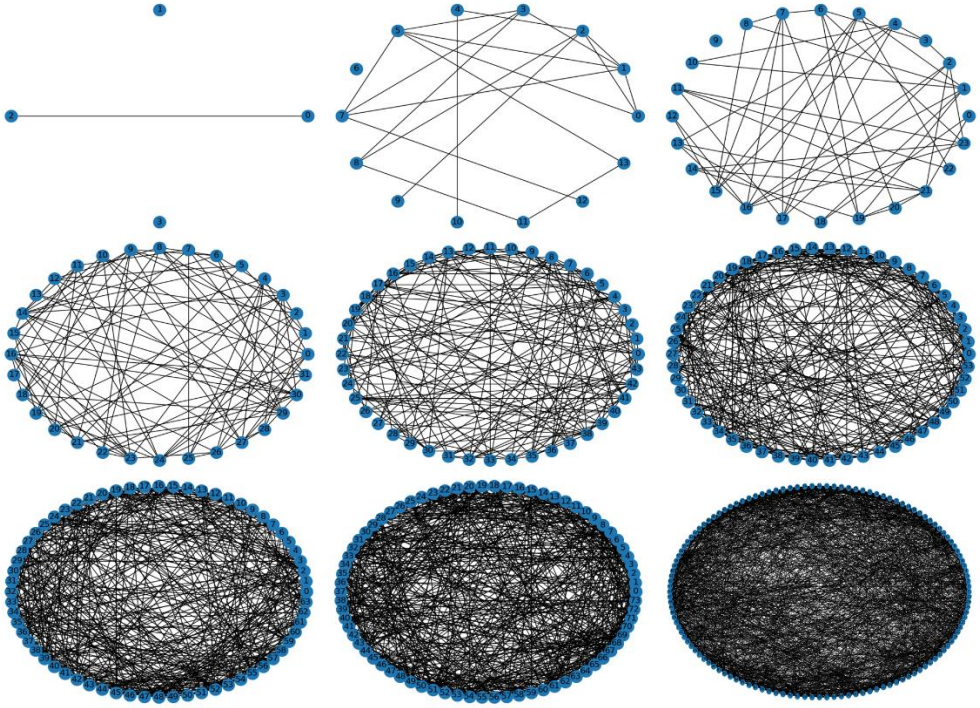


Figure 25. The image of the ER(0.2) model by applying in the first DAG layer from left to right and top to bottom the number of the vertices are 4, 14, 24, 32, 44, 54, 64, 74 and 128

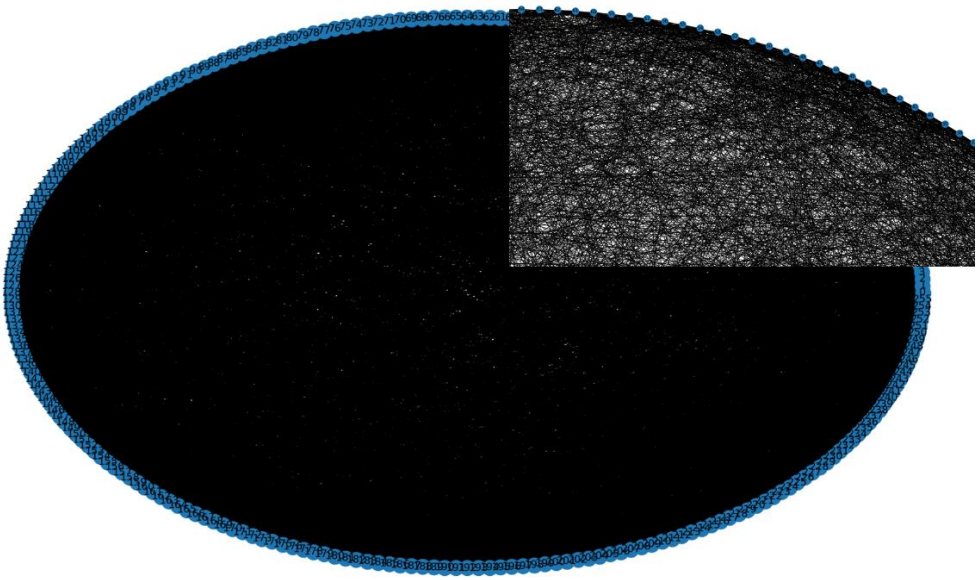


Figure 26. The image of the ER(0.2) model by applying in the first DAG layer with 256 vertices

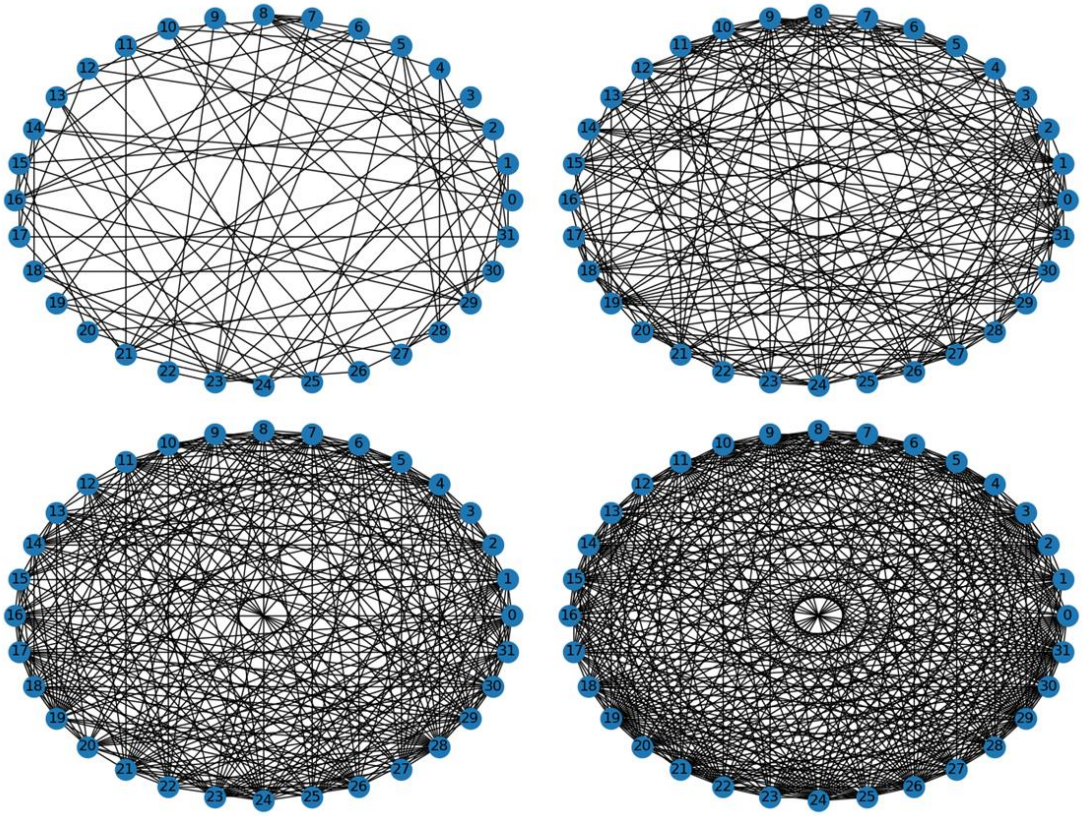


Figure 27. The image of the ER model by applying in the first DAG layer with 32 vertices from left to right and top to bottom the value of the wire-able probabilities are 0.2, 0.4, 0.6 and 0.8

BA Graph Image. The images below are the illustrations of some of the BA graphs for the first DAG layer, we used for our neural network generator.

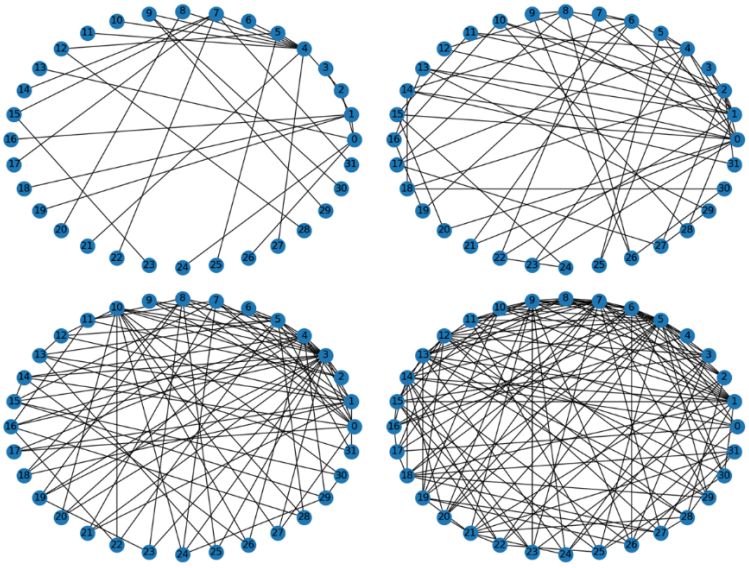


Figure 28. The image of the BA model by applying in the first DAG layer with 32 vertices from left to right and top to bottom the value of the wire-able edges are 1, 2, 3 and 5

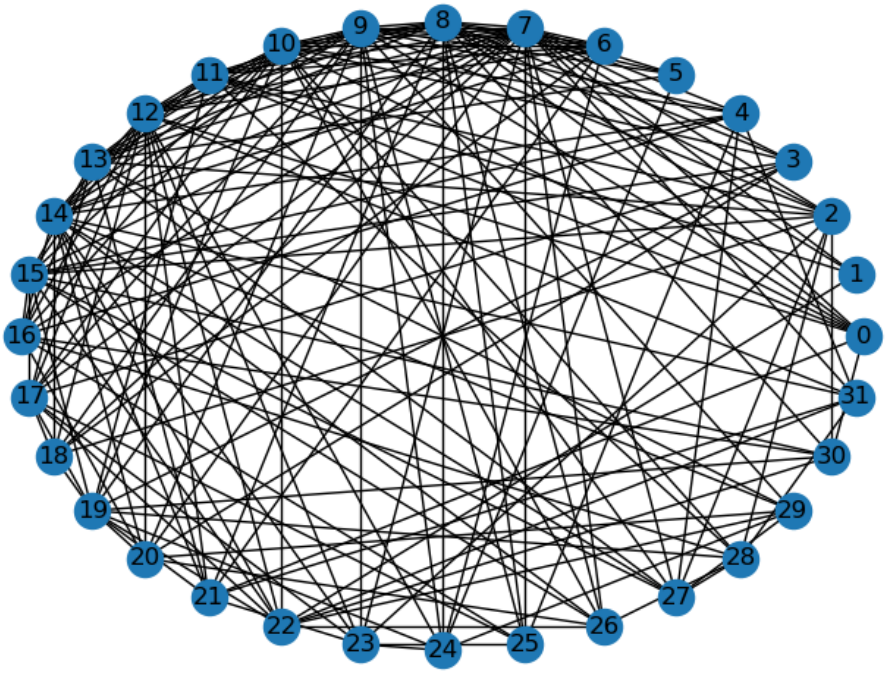


Figure 29. The image of the BA model by applying in the first DAG layer with 32 vertices and 7 wire-able edges

WS Graph Image. The images below are the illustrations of some of the WS graphs for the first DAG layer, we used for our neural network generator.

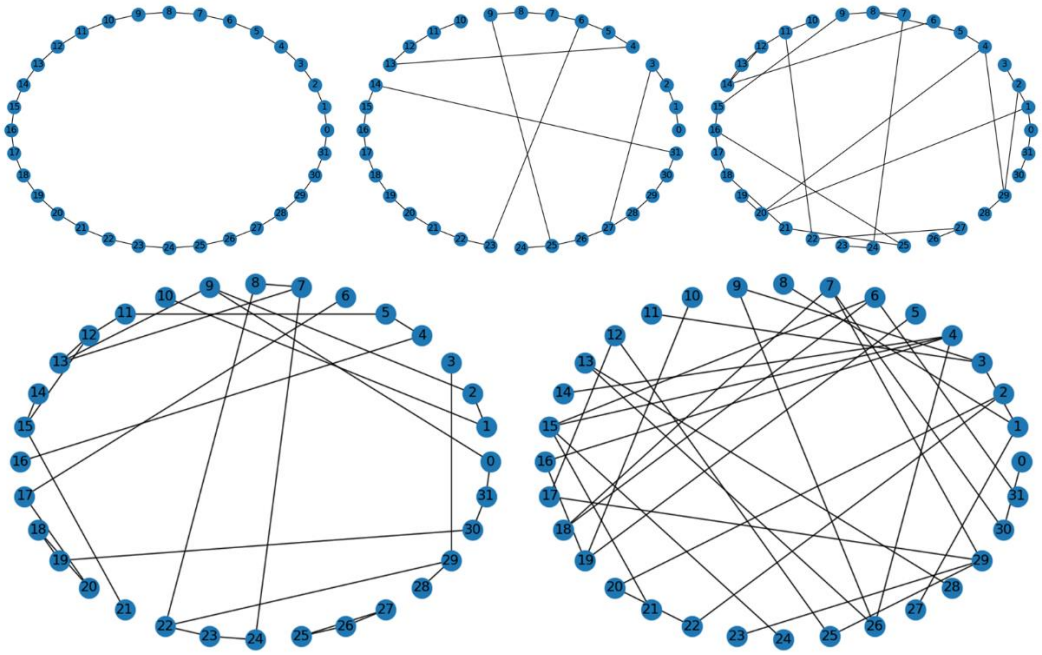


Figure 30. The image of the WS model by applying in the first DAG layer with 32 vertices and 2 nearest neighbors from left to right and top to bottom the value of the wire-able probabilities are 0, 0.25, 0.5, 0.75 and 1

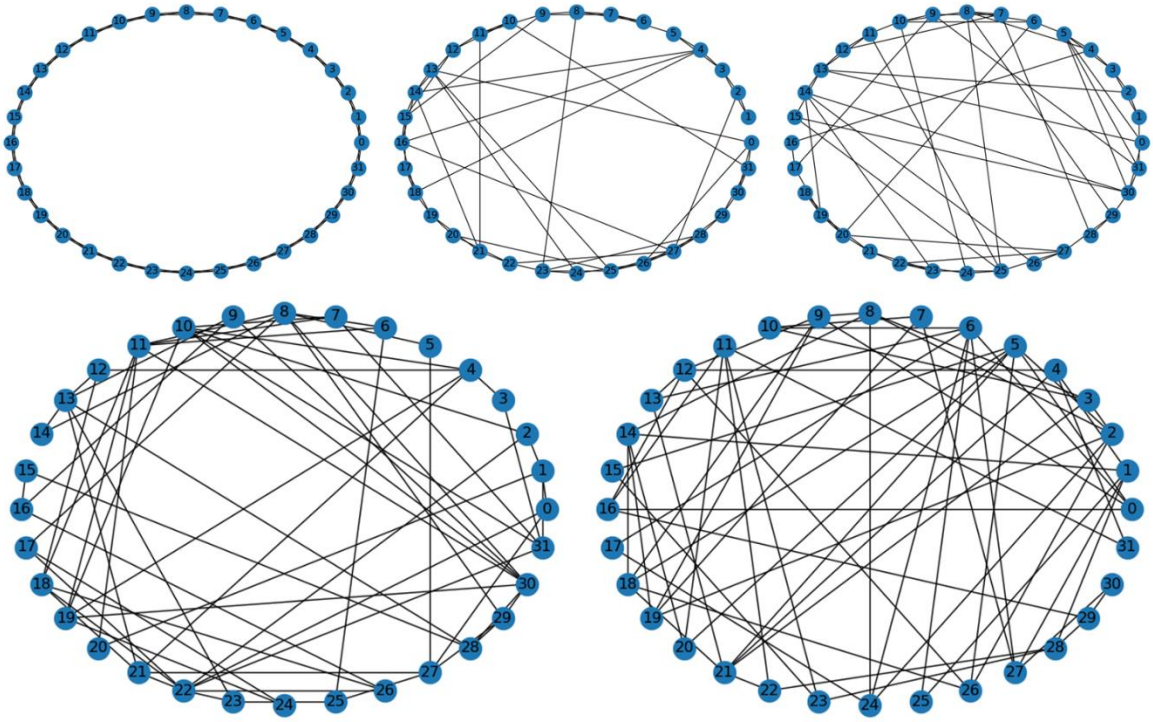


Figure 31. The image of the WS model by applying in the first DAG layer with 32 vertices and 4 nearest neighbors from left to right and top to bottom the value of the wire-able probabilities are 0, 0.25, 0.5, 0.75 and 1

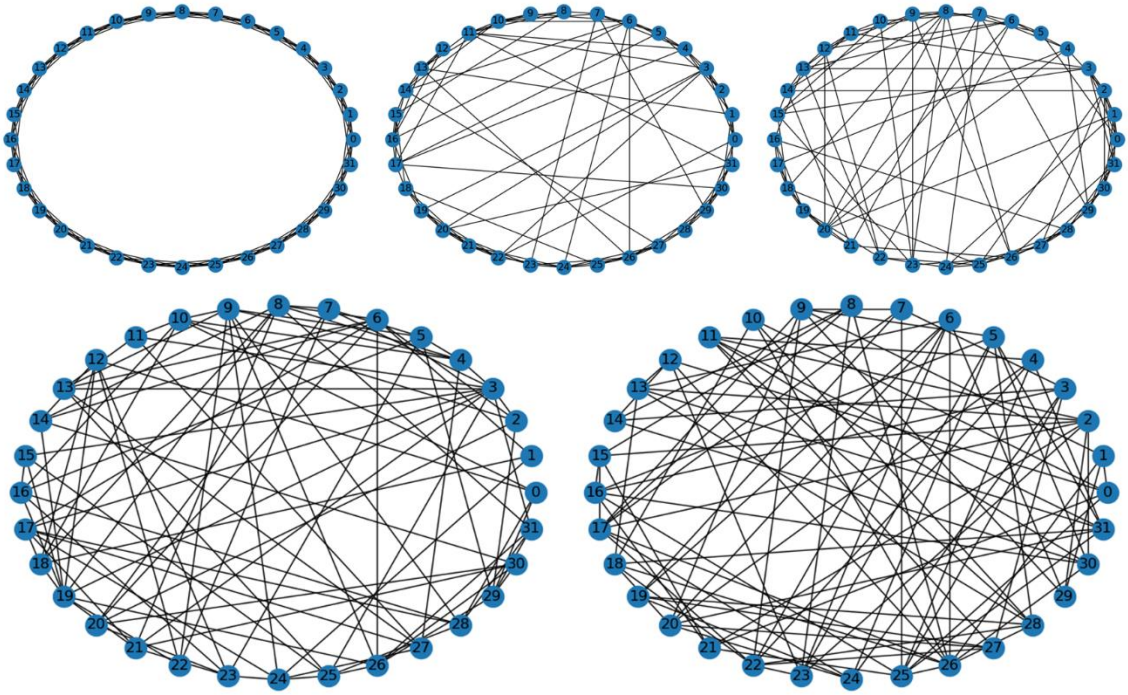


Figure 32. The image of the WS model by applying in the first DAG layer with 32 vertices and 6 nearest neighbors from left to right and top to bottom the value of the wire-able probabilities are 0, 0.25, 0.5, 0.75 and 1

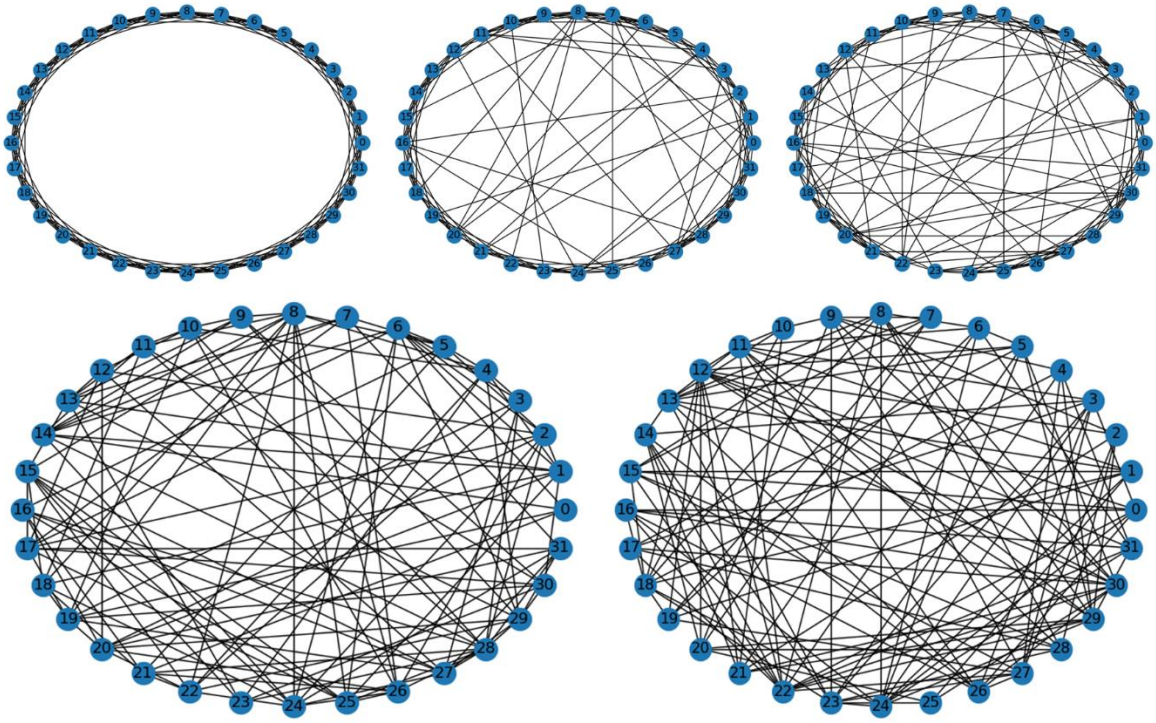


Figure 33. The image of the WS model by applying in the first DAG layer with 32 vertices and 8 nearest neighbors from left to right and top to bottom the value of the wire-able probabilities are 0, 0.25, 0.5, 0.75 and 1

Accuracy and Loss. The images below are the illustration of some training accuracy and loss logging information.

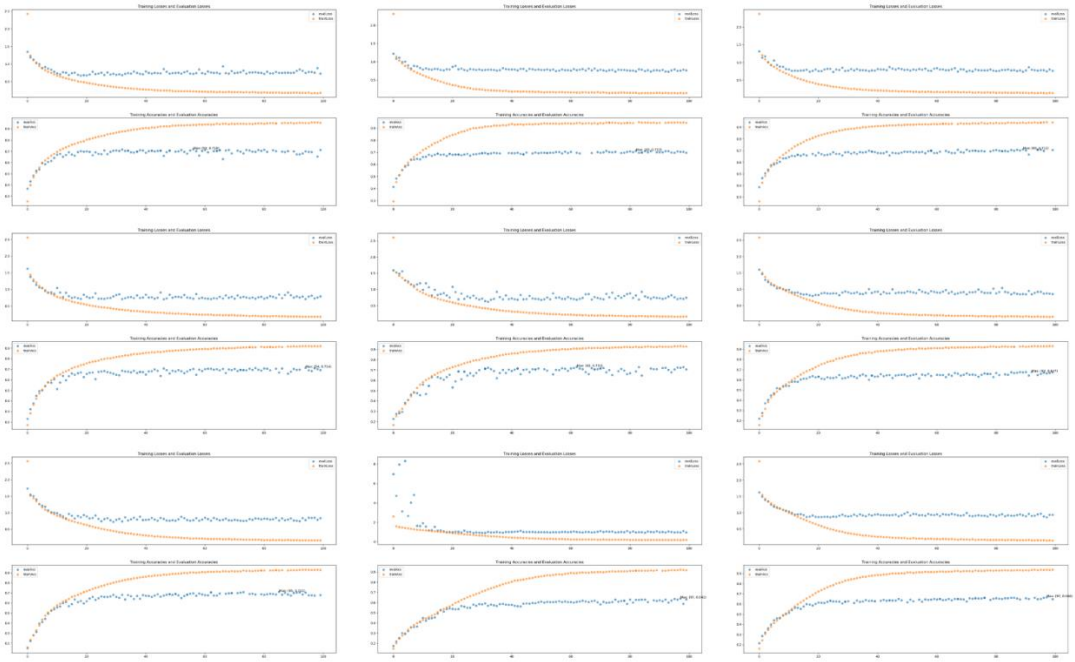


Figure 34. The image of the accuracy and loss for ER(0.2) model with random seed 1 from left to right and top to bottom the value of the number of the vertices are 4, 14, 24, 34, 44, 54, 64, 74 and another 74 w random seed 2

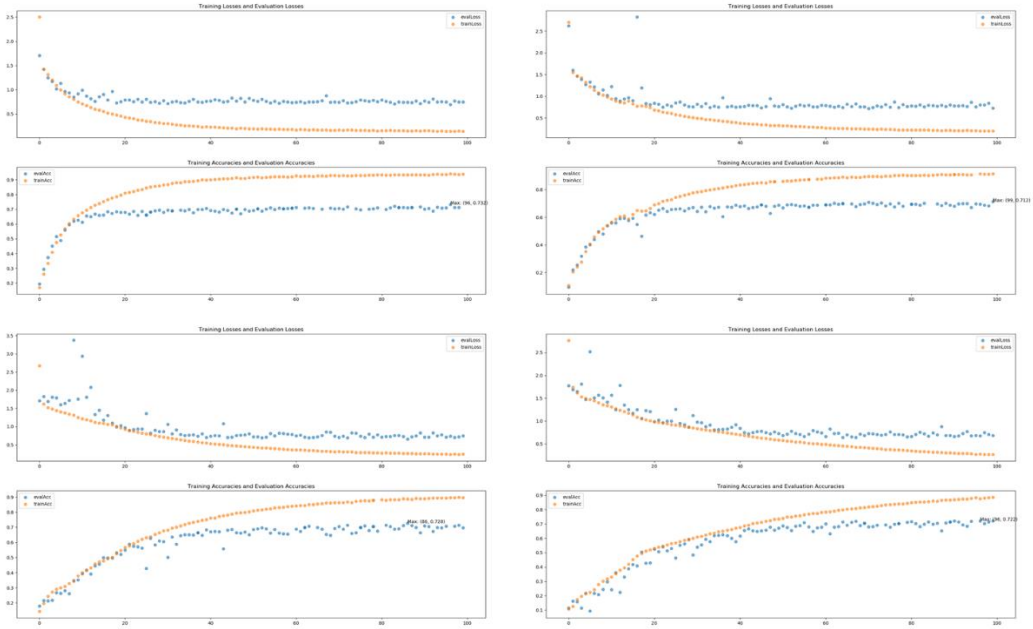


Figure 35. The image of the accuracy and loss for ER model with 32 vertices and from the left to right and top to bottom the value of the wire-able probabilities are 0.2, 0.4, 0.6 and 0.8

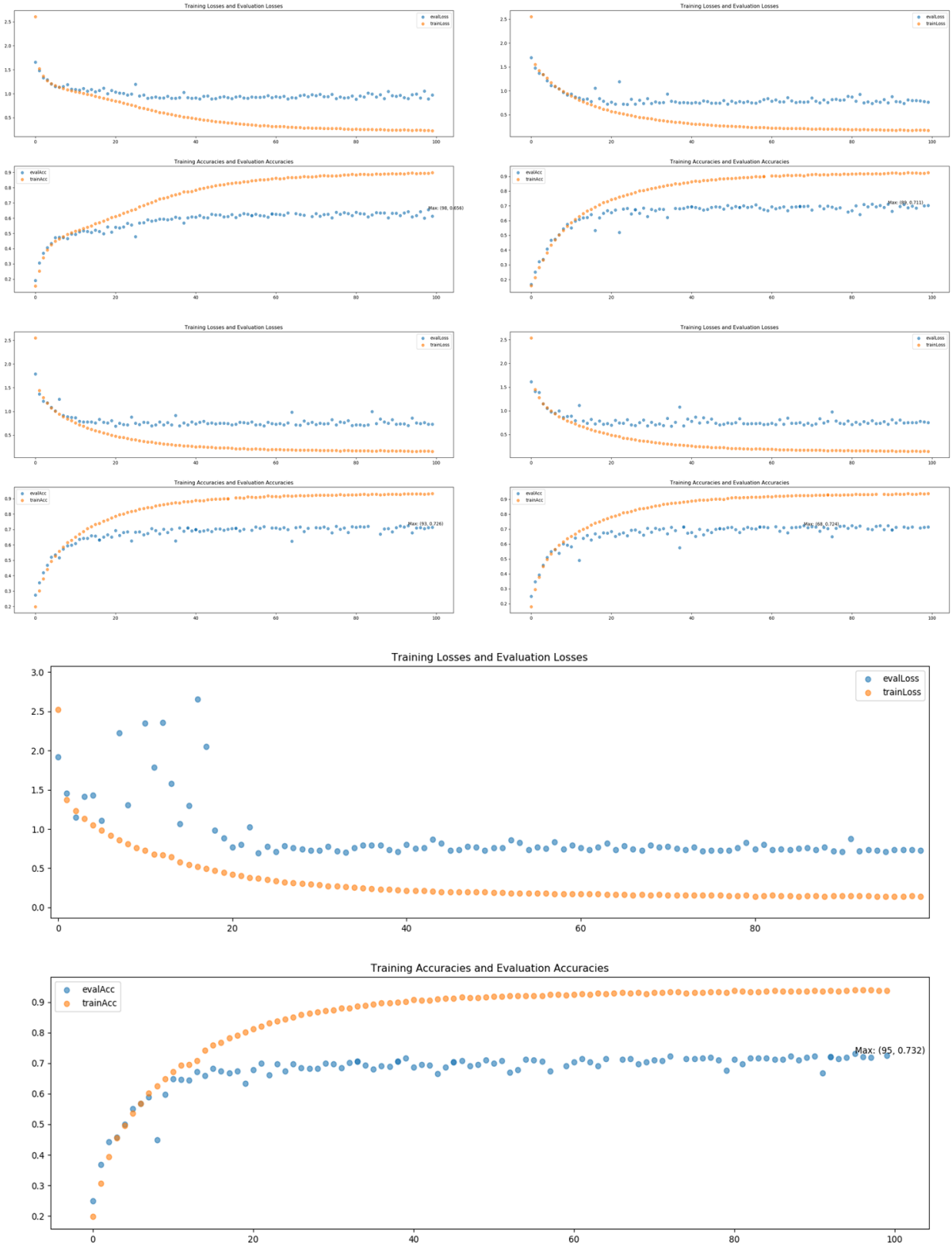


Figure 36. The image of the accuracy and loss for BA model with 32 vertices and from the left to right and top to bottom the number of the wire-able edges are 1, 2, 3, 5 and 7

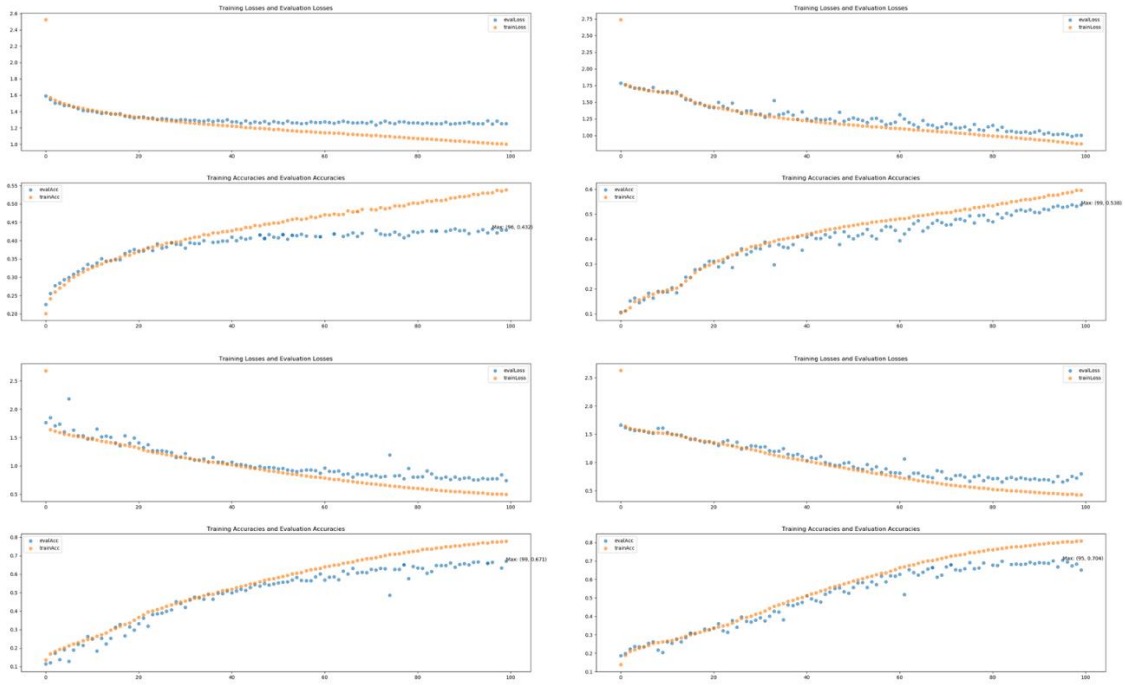


Figure 37. The image of the accuracy and loss for WS model with 32 vertices and 0 wire-able probability and from the left to right and top to bottom the number of the nearest neighbors are 2, 4, 6 and 8

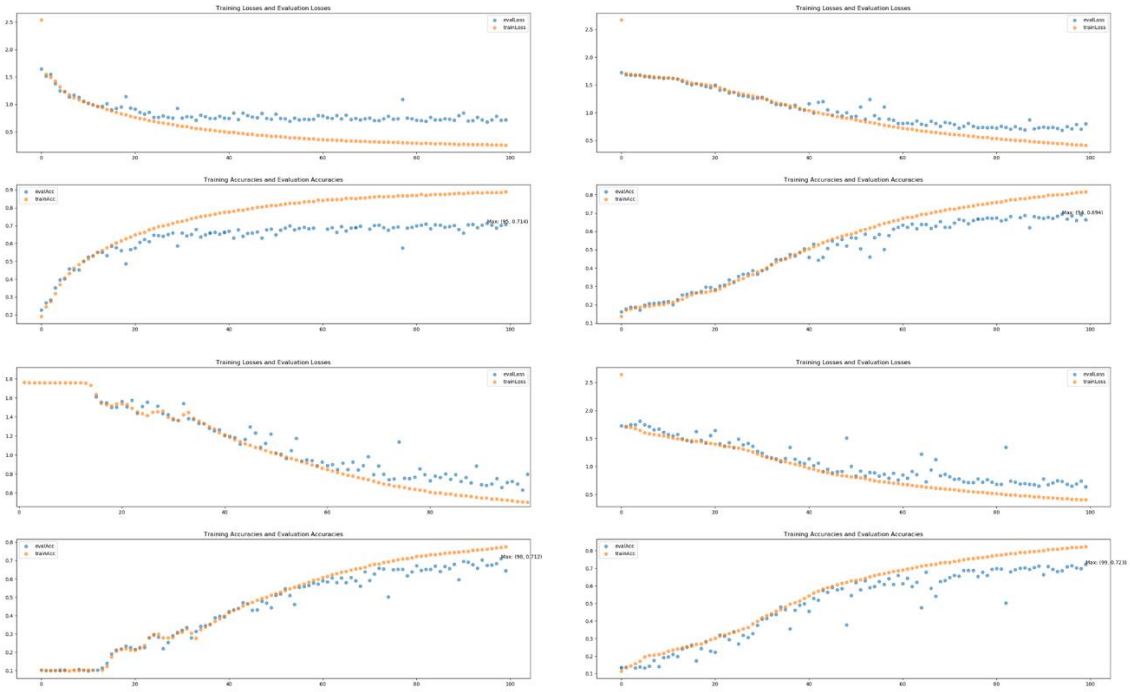


Figure 38. The image of the accuracy and loss for WS model with 32 vertices and 0.25 wire-able probability and from the left to right and top to bottom the number of the nearest neighbors are 2, 4, 6 and 8

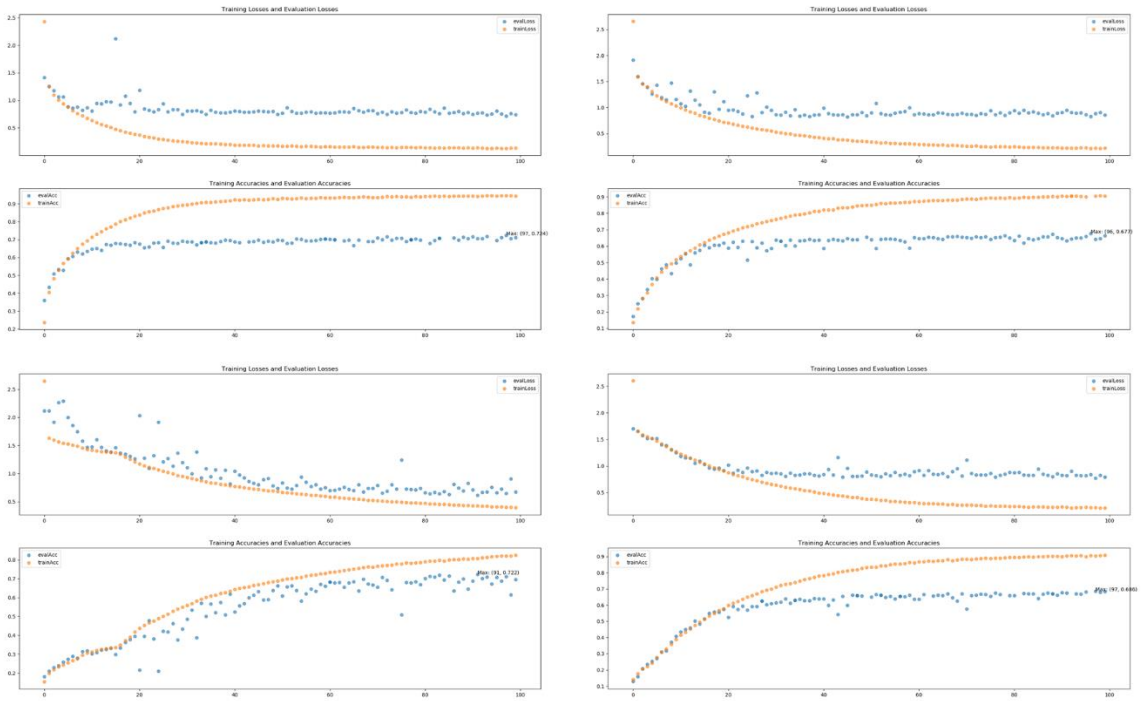


Figure 39. The image of the accuracy and loss for WS model with 32 vertices and 0.5 wire-able probability and from the left to right and top to bottom the number of the nearest neighbors are 2, 4, 6 and 8

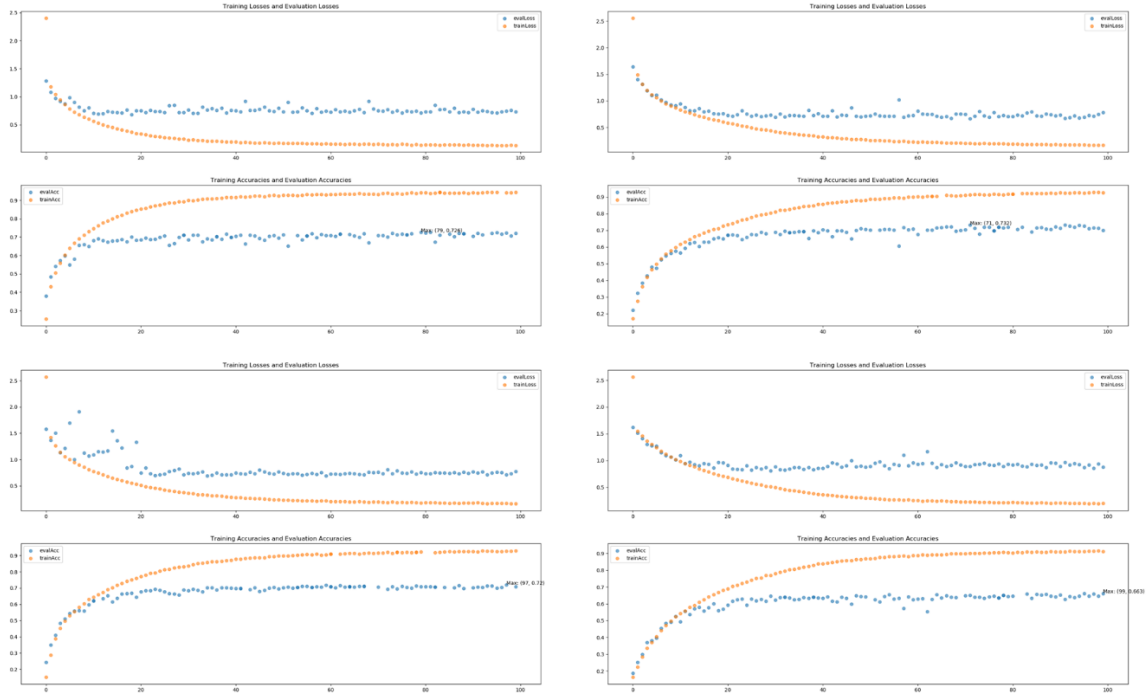


Figure 40. The image of the accuracy and loss for WS model with 32 vertices and 0.75 wire-able probability and from the left to right and top to bottom the number of the nearest neighbors are 2, 4, 6 and 8

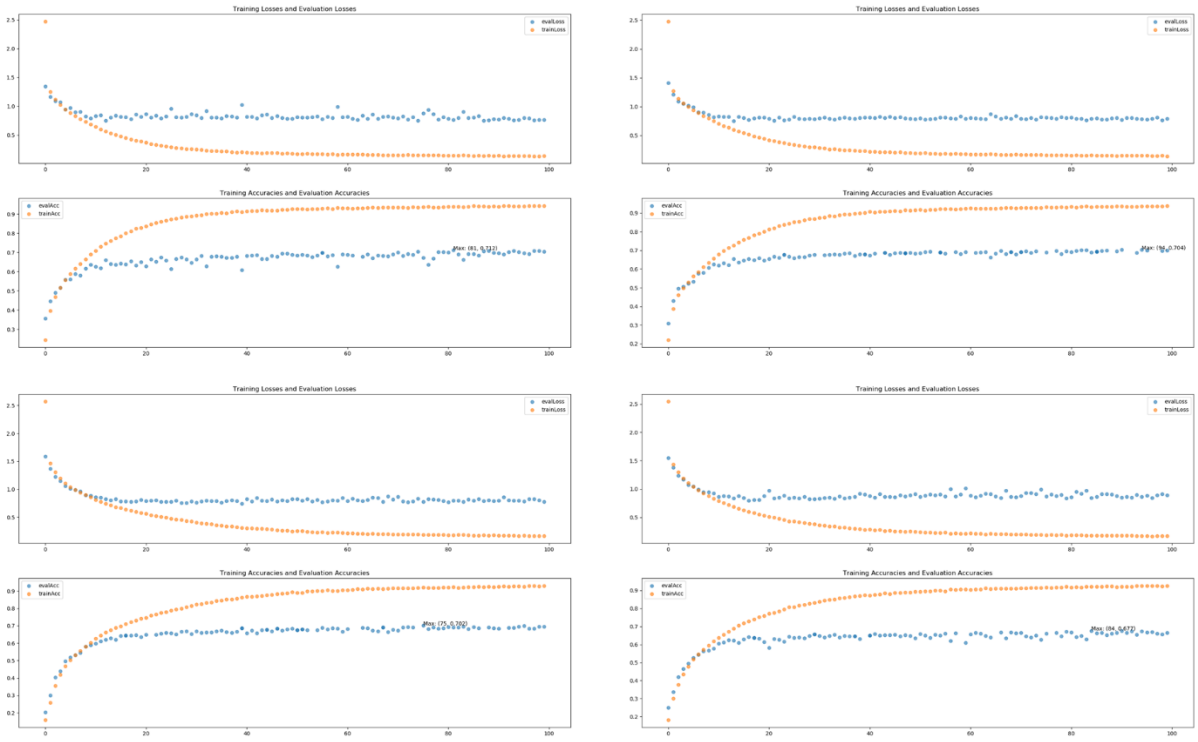


Figure 41. The image of the accuracy and loss for WS model with 32 vertices and 1 wire-able probability and from the left to right and top to bottom the number of the nearest neighbors are 2, 4, 6 and 8

Member Contribution. Jarvis (1730026042) did all the coding parts (Random Graph Generator, Graph Data Reader, Neural Network Generator, Training Optimization Components, Training Parameter Configurator and Training Information Logger), all the stage-two experiments, a part of the FAIR team's experiments repetition (ER), the space cost big-oh complexity, the Final Thesis First Draft version 1 and the Final Thesis First Draft version 3. Shawn (1730026116) did two parts of the FAIR team's experiments repetition (BA and WS), the time cost big-oh complexity, and the Final Thesis First Draft version 2.