RANDOM NEURAL NETWORK TUTORIAL

Random Neural Network (RANN) is a mathematical model that has been proposed by Erol Gelenbe in 1989 [1]. It has been declared as a composition of queueing theory and artificial neural networks. It's working principle relies on the circulation of positive and negative spikes with unit amplitude throughout the network.

1.1 Mathematical Model

In order to express the model clearly, a supermarket example is widely used in literature. In this example, assume that there is a single queue where customers arrive and leave. The arriving process of customers are regulated by a Poisson process with a rate defined as $\lambda > 0$. In order to express leaving process, a parameter called r has been defined to represent service times which is exponentially distribute and greater than zero. There exist two kind of customers: positive and negative ones. As a positive customer arrives the queue, the customers in the network which is defined as potential of the queue increases by 1. In contrast, arrival of a negative customer creates an inhibition effect on the queue and decreases the potential by 1, if only the potential is already greater than zero. Otherwise, negative customers disappear without any impact on the queue. Beside the arrival of negative customers, departing customers that has been served by the service rate r also have an inhibition effect on the potential if only there is at least one customer in the queue. In other words, departure process is run only with a positive potential. When a customer has been served in the queue i, it can either leave the network with a probability of d_i or it can move to another queue j with a probability p_{ij}. This probability is a compose of two sub-probabilities p_{ji}^{+} and p_{ji}^{-} where each denotes a possibility of moving as a positive or a negative customer respectively. In a Random Neural Network with N interconnected neuron the relationship between these probabilities can be expressed as it has been shown in the equation 1.1.

$$d_i + \sum_{j=1}^{N} (p_{ij}^{+} + p_{ij}^{-}) = 1 \qquad (1.1)$$

Considering the neural network structure, it can be said that the term d_i is equal to 1 only for the output neurons, otherwise it is equal to 0, since neurons in the input and hidden layers do not emit signals to the outside environment. In other words, the term d_i enable output neurons to transmit signals only outside by making the probabilities p_{ij}^{+} and p_{ij}^{-} zero. As a contrast, in the case of input layer, signals can only be received by the outside environment. In addition to the explanations about arrival processed at the paragraph above, the Poisson process rates that regulates these arrivals can be defined with the symbols λ_i^{+} and λ_i^{-}. A common practice about the RANN in the literature is to use the notations $W_{ij}^{+} = r_j p_{ij}^{+}$ and $W_{ij}^{-} = r_j p_{ij}^{-}$ that will help to establish an analogy with the conventional neural networks. In these representations the terms W_{ij}^{+} and W_{ij}^{-} denotes the positive and negative weights between the neurons i and j. The difference between the conventional weights and these ones, can be observed from their definition. As it can be seen, W_{ij}^{+} and W_{ij}^{-} are a product of two probabilities. Since both of the components of the product operation are defined as rates which are at least 0, the weights cannot take negative values.

In [2], the stability conditions for a Random Neural Network has been presented. According to theorem in [2], for a neuron i, the potential of the neuron which is denoted as q_i can be found with the following expression:

$$Q_i = \frac{T_i^{+}}{r_i + T_i^{-}} \qquad (1.2)$$

where the terms T_i^{+} and T_i^{-} denote total positive and negative contribution to neuron i from inside and outside environment respectively and the term r_i denotes the service rate of the particular neuron i. In the calculation of this formula, the total positive and negative arrival rates can be evaluated as follows:

$$T_i^{-} = \lambda_i^{-} + \sum_{j=1}^{N} Q_j W_{ij}^{-} \qquad (1.3)$$

$$T_i^- = \lambda_i^- + \sum_{j=1}^{N} Q_j W_{ij}^- \tag{1.4}$$

In the special case of input layer, T_i^+ and T_i^- are directly equal to λ_i^+ and λ_i^- respectively, since there is no input to this layer from the inside of the network.

The final term, needed in equation 1.2 is r_i. Using the manipulations $W_{ij}^+ = r_j p_{ij}^+$ and $W_{ij}^- = r_j p_{ij}^-$, in the equation 1.2, r_i can be evaluated with the formula:

$$r_i = \frac{1}{1 - d_i} + \sum_{j=1}^{N} (W_{ij}^+ + W_{ij}^-) \tag{1.5}$$

1.2 Gradient Based Optimization

Training a neural network requires a forward and a backward pass. Above, generalized forward pass formulas are given with extensive descriptions regarding the logic behind the Random Neural Network. In 1993, first attempt to implement optimization algorithm to Random Neural Network, has been made by Erol Gelenbe.

Assume a data set $S = \{(x^k, y^k), k = 1, \dots, K\}$ exists where each vector x^k and y^k represent a subset of X and Y. The duty of neural networks in supervised learning, is to create a mapping function, that imitates the relationship between X and Y. In order to establish this relationship with the interior parameters of a neural network, all the trainable parameters get updated after each forward pass. The magnitude and the direction of the gradient of a trainable parameter with respect to a loss function, determines the effect of update over that individual trainable parameter. As the network iterate over the data set S, it is expected from network to optimize trainable parameters to reach an optimal point that approximation results are close enough to mimic original relationship between X and Y. This general working principle of supervised learning algorithms, is also valid for Random Neural Networks as well.

Suppose a forward pass has been performed for a Random Neural Network with one hidden layer, using I units in input layer, H units in hidden layer and O units in output layer. As an output, a vector Q has been created for each neuron i where $i \in N = I + H + O$. Using the mean squared error loss function, loss gets calculate as follows:

$$L = \frac{1}{2}\sum_{i=1}^{N} d_i\,(q_i - y_i)^2 \qquad (1.6)$$

The reason behind the use of the factor $\frac{1}{2}$ is to eliminate the factor of 2 when taking the derivative of the loss function with respect to each output q_n. Beside the extra element $\frac{1}{2}$, another extra term is d_n which can be explained as a designator that distinguish output neurons from the rest. As it is noted at section 3.1, d_n is taken as 1 for the output neuron and 0 for the rest, in general practice.

In the case of Random Neural Networks, the trainable parameters are positive and negative weights. The update procedure of trainable parameters runs with the calculation:

$$W_{uv}^{*(k)} = W_{uv}^{*(k-1)} + \delta_{uv}^{*(k)} \qquad (1.7)$$

In order to not to write equations twice with the same structure, the term W_{uv}^{*} is used as a representation of the both positive and negative weights at the between neuron u and v, where k denotes the iteration number. Another point the notice is the size of the matrix $W_{uv}^{*(k)}$. In [2], it has been defined as NxN matrix that contains all the connections between all the neurons. It means there is no layer-wise discrimination in this most commonly used representation. The term $\delta_{uv}^{*(k)}$ that performs the update is calculated as follows:

$$\delta_{uv}^{*(k)} = -\eta\sum_{i=1}^{N} d_i\,\frac{\partial L}{\partial q_i^{(k)}}\frac{\partial q_i^{(k)}}{\partial W_{uv}^{*}} \qquad (1.8)$$

In the equation 1.8, the term η is called the learning rate which takes value in a range $[0,1]$. Another term $\frac{\partial L}{\partial q_i^{(k)}}$ can be expressed as $\left(q_i^{(k)} - y_i^{(k)}\right)$, when the derivative of the equation 3.6 is taken with respect to $q_i^{(k)}$. Finally, the calculation of the remaining gradient term $\frac{\partial q_i^{(k)}}{\partial W_{uv}^{*}}$ is actually the main focus of backpropagation phase in the Random Neural Networks.

In order to formulate backpropagation phase, Erol Gelenbe has proposed a formula structure to update positive and negative weight matrices in [2]. As it can be observed in the equations 1.9 and 3.10, there exist two new parameters that have not been explained in the previous chapters.

$$\frac{\partial q_i^{(k)}}{\partial W_{uv}^{+}} = \gamma_{uv}^{+} q_u [I - \Omega]^{-1} \tag{1.9}$$

$$\frac{\partial q_i^{(k)}}{\partial W_{uv}^{-}} = \gamma_{uv}^{-} q_u [I - \Omega]^{-1} \tag{1.10}$$

In equations 1.9 and 1.10, the terms γ_{uv}^{+} and γ_{uv}^{-} represent results of the two multiple conditional equalities, as it can be seen in the equations 1.11 and 1.12. The values that these terms can take are actually the results of basic derivatives of neurons q_u and q_v with respect to the nominators and denominators. But as a simplified rule, it is formulated as follows.

$$\gamma_{uv;i}^{+} = \begin{cases} -\dfrac{1}{r_i + T_i^{-}}, & if\ u = i, v \neq i \\ \dfrac{1}{r_i + T_i^{-}}, & if\ u \neq i, v = i \\ 0, & otherwise \end{cases} \tag{1.11}$$

$$\gamma_{uv;i}^{-} = \begin{cases} -\dfrac{1 + q_i}{r_i + T_i^{-}}, & if\ u = i, v = i \\ -\dfrac{1}{r_i + T_i^{-}}, & if\ u = i, v \neq i \\ -\dfrac{q_i}{r_i + T_i^{-}}, & if\ u \neq i, v = i \\ 0, & otherwise \end{cases} \tag{1.12}$$

As a beneficial feature of this representation in the equation 1.12, the first condition is $i = u = v$. This equality implies a recurrent connection that a neuron may have with itself, which leads to structure that has potential to be used with a “Recurrent Topology” which will be investigated in further chapters.

Another unexplained term is a matrix which is a subject to a process of taking inverse. In the formal notation it is symbolized with "W”, but in order to prevent confusion it is shown with Ω, using the notation in [3].

$$\Omega = \frac{W_{ij}^{+} - W_{ij}^{-} q_j}{r_j + T_j^{-}} \tag{1.13}$$

Just like the terms $\gamma_{uv;i}^{+}$ and $\gamma_{uv;i}^{-}$, the idea behind the equation 1.13 also based on the derivatives of neuron i with respect to neuron j. As an interpretation of the matrix Ω, its shape is NxN. This representation of connections enables the Random Neural Network to perform as a recurrent structure as well as the feedforward one. As useful tool, there is a MATLAB implementation of the calculations regarding forward and backward passes, which is written by Hossam Abdelbaki in 1999 [4].

1.3 Feedforward Topology

In the feedforward topology, connections can only exist between the neighbor layers such as, input and hidden, hidden and output, or hidden and hidden in deep learning applications. In such cases, as the neurons are numerated incrementally starting from the first neuron of input layer to the last neuron of output layer, the positive and negative weight matrices would have a shape of an NxN upper triangular matrix. Because of the shape of the weight matrices, the term Ω also have this upper triangular look, which facilitates the inverse operation in the backpropagation phase. These situations can be clearly observed in the implementation of Hossam Abdelbaki. The effect of numeration process, the weight matrices and the parameters in the backpropagation phase can be examined, to have a better understanding about the interior dynamics of the Random Neural Network. Unfortunately, this toolbox has no quick option to increase the number of hidden layers, which makes it not suitable for deep structures.

In the backpropagation there also exist another method to calculate derivatives, which is a simple implementation of classical derivative rules. Consider a Random Neural Network with N hidden layer. Thus, there exist $N - 1$ weight matrices between the layers. As the network yield an output from the layer N, the loss function creates a loss value to update the values of the weights. Since the function is trivial in terms of explaining the process, it is just denoted as a generic function L which takes the output of the network and real output value as inputs. Notice that q_N and y are vectors with the same size.

$$L^k = L(q_N, y) \tag{1.14}$$

For each weight matrices n in the k^{th} iteration, the delta update rule is applied as follows:

$$W_n^{*(k+1)} = W_n^{*(k)} + \delta_n^{*(k)} \tag{1.15}$$

The term $\delta^{*(k)}$ in equation 3.15 is calculated with the equation 3.16.

$$\delta^{*(k)} = -\eta \frac{\partial L^k}{\partial q_N} \frac{\partial q_N}{\partial W_n^*} \tag{1.16}$$

Normally in the feedforward neural networks, a weight can only affect the next layers output. In the case of RANN, considering the equation 1.2, a weight matrix affects the both of the layers that it connects. This situation can be seen in the equation 1.17. As a written expression, if the layers and weights have been numbered starting from the same number, the gradient of a weight matrix with the index n, will be depending on the layers q_{n+1} and q_n.

$$\frac{\partial q_N}{\partial W_n^*} = \frac{\partial q_N}{\partial q_{n+1}} \left(\frac{\partial q_{n+1}}{\partial W_n^*} + \frac{\partial q_{n+1}}{\partial q_n} \frac{\partial q_n}{\partial W_n^*} \right) \tag{1.17}$$

The last unexplained term in order to perform the calculation 1.15 is the term $\frac{\partial q_N}{\partial q_{n+1}}$. As it can be seen in the equation 1.18, this term can be evaluated using the chain rule:

$$\frac{\partial q_N}{\partial q_n} = \prod_{N > j \geq n} \frac{\partial q_{j+1}}{\partial q_j} \tag{1.18}$$

1.4 Recurrent Topology

The recurrent topology of Random Neural Network is proposed with the backpropagation algorithm at section 1.2. The name of the article was “Leaning in The Recurrent Random Neural Network”. What is meant by the word “Recurrent” was actually the capability of establishing connections among the neurons without any restriction of layer. This definition of “Recurrent” is completely different than the meaning of “Recurrent” for the Simple RNN, LSTM or GRU. The reason for these 3 models to be called as “Recurrent Neural Networks” is actually about their ability of having dynamical connections that can be unfold through time. Considering this difference in the approach of the word “Recurrent”, Recurrent Random Neural Network (RERANN) is definitely not a part of this “Recurrent Neural Networks Family” in the sense of recurrence connections expanding through time. Furthermore, considering the nature of RERANN, it is also not a similar model to ELMAN-JORDAN type networks neither unless some special modifications are performed. In contrast to examples above, it can be used in the associative memory applications, due to its ability to be used as a Hopfield Network.

REFERENCES

[1] **Gelenbe, Erol.** (1989). Random Neural Networks with Negative and Positive Signals and Product Form Solution. Neural Computation - NECO. 1. 502-510. 10.1162/neco.1989.1.4.502.

[2] **Gelenbe, Erol.** (1993). Learning in the Recurrent Random Neural Network. Neural Computation. 5. 154-164. 10.1162/neco.1993.5.1.154.

[3] **Basterrech, S., & Rubino, G.** (2015). Random Neural Network Model for Supervised Learning Problems. *Neural Network World, 25*, 457-499.

[4] **Abdelbaki H.** (1999). rnnsimv2.zip. Retrieved September 9, 1999. Available from https://www.mathworks.com/matlabcentral/fileexchange/91-rnnsimv2-zip