1. Random Neurons

The aim of this sections was to provide necessary background information for Queueing RNN.

In order to explain Random Neurons, a queue analogy is commonly used in literature. Random Neuron structure represents a queue and a network with Random Neurons is named as Random Neural Network (RANN). Each neuron has a non-negative potential value which is the number of customers in this queue. The arrival of customers are regulated by a poisson process with a rate defined as $\lambda > 0$. There exist two kind of customers: positive and negative ones. As a positive customer arrives the queue, the customers in the queue which is defined as the 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. When a customer leaves the queue, the potential of the queue decreases by 1. In order to express leaving process, a parameter called r has been defined which is also a poisson process rate.

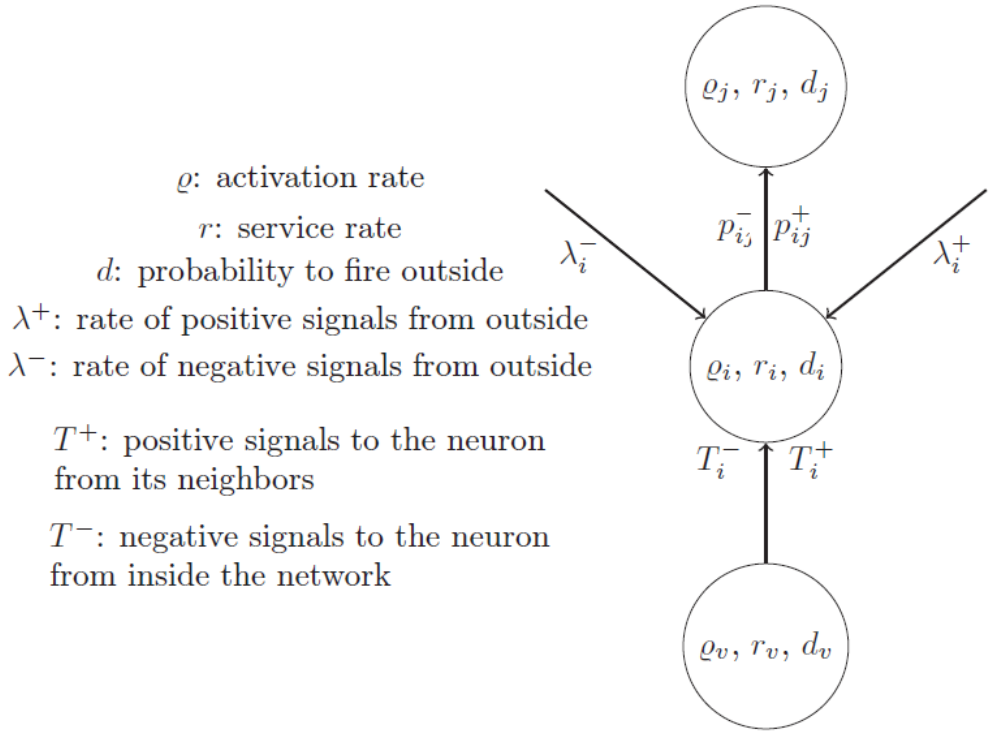

Figure 1: Random Neuron Scheme [1]

When a customer has been served with service rate r 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.

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

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 for a feedforward RANN structure. 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. So there exist two kind of weights and both are positive definite unlike conventional neural networks.

For a neuron i, the potential of the neuron which is denoted as q_i can be found with the following expression:

$$Q_i = \frac{T_i^{+}}{r_i + T_i^{-}} \tag{1.2}$$

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:

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

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

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:

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

2. Mathematical Model of QRNN

This section contains mathematical model of QRNN with 1 hidden layer for the sake of simplicity.

The architecture of QRNN with one hidden layer, consist of an input layer with I units, a hidden layer with H neuron and an output layer with O neuron. It is a common practice for most of the neural network applications to normalize or standardize the data in the preprocessing phase. Since rate of a poisson process can only take values in the interval $[0,1]$, after the preprocessing phase it makes sense the scale the input data to this interval and give it to the term $\lambda_{i_t}^+$ which is the input data of the neurons in the input layer. In this scenario, the term $\lambda_{i_t}^-$ actually become useless.

The output formula of the neurons in the input layer is quite similar to the output formula of any arbitrary neuron in RANN. A small difference is the index t, which denotes the time step of the

neuron. For this tutorial, each term with an index t means that this variable changes through the time and each term without the index means, this variable has no dependency with time.

$$q_{i_t} = \frac{\lambda_{i_t}^{+}}{\sum_{j=1}^{H}(W_{ih_j}^{+} + W_{ih_j}^{-}) + \lambda_{i_t}^{-}} \tag{2.1}$$

In the equation 2.1 total positive and negative inputs are represented directly with terms $\lambda_{i_t}^{+}$ and $\lambda_{i_t}^{-}$, since there is no connection for the input neurons with any others within the network. As a useful practice, the rate of the neuron r_i is written explicitly in the equation 2.1 and the following ones. The reason for that, is mainly facilitating the comprehension of the terms in the backpropagation phase. Notice that the term r_i do not have the index t. Since all the weights are the same through the time, in other words not time variant, the input layers from different time steps shares the same service rate r_i where W_{ih}^{+} and W_{ih}^{-} denotes the positive and negative weight matrices respectively between input and hidden layers. Considering the fact that the shape of the matrix W_{ih}^{+} is IxH, the term $W_{ih_j}^{+}$ actually shows a row of this two dimensional matrix which corresponds to a vector that contains all the connections that an input neuron have with the hidden neurons at the same time step. Additionally, notice that actually every term in the equation 2.1 is a vector. Length of the all the vectors $\lambda_{i_t}^{+}$, $\lambda_{i_t}^{-}$, q_{i_t}, $W_{ih_j}^{+}$ and $W_{ih_j}^{-}$ are the same which is I.

In the case hidden layers, the equations are slightly different than the formulation that is used by RANN. This situation is caused by the dynamic nature of RNNs. The connections of hidden layers change depending on not only the application, but also the positions of this layer in time. Since the rate parameter of each neuron is calculated based on the outputs of that particular neuron, dynamic structure of RNNs make changes in the fully explicit version of the formula:

$$q_{h_t} = \frac{T_{h_t}^{+}}{r_h + T_{h_t}^{-}} \tag{2.2}$$

$$T_{h_t}^{+} = \sum_{j=1}^{I}(q_{i_t} W_{ih_j}^{+}) + \sum_{j=1}^{H}(q_{h_{t-1}} W_{hh_j}^{+}) \tag{2.3}$$

$$T_{h_t}^{-} = \sum_{j=1}^{I} (q_{i_t} W_{ih_j}^{-}) + \sum_{j=1}^{H} (q_{h_{t-1}} W_{hh_j}^{-}) \tag{2.4}$$

$$r_h = \sum_{j=1}^{H} (W_{hh_j}^{+} + W_{hh_j}^{-}) + \sum_{j=1}^{O} (W_{ho_j}^{+} + W_{ho_j}^{-}) \tag{2.5}$$

In the equation 2.2 the most general form of hidden layer formulation is presented. Depending the architecture of the model (one to many, many to one or many to many), some of the terms may be directly 0. Imagine a forecasting application using the previous data using the "many to one" structure as it has been depicted in figure 2. Aim is the make a predictions based on the previous values of the given variables. In this architecture, a hidden layer at time step t, takes input from the input layer at time step t and hidden layer at time step $t-1$. This statement is valid for each hidden layer in time. Thus, total positive and negative inputs of hidden layers $T_{h_t}^{+}$ and $T_{h_t}^{-}$ are the same end equal to equations 2.3 and 2.4 for all the time steps. On the other hand, a hidden layer at time step t gives output only to the hidden layer at time step $t+1$ unless t is the last time step. Because at the last time step, the output of the hidden layer is given as an input to the output layer. It can be seen in the figure 2. Consequently, rate of last hidden layer is equals to $\sum_{j=1}^{O}(W_{ho_j}^{+} + W_{ho_j}^{-})$ while the rest of them equal to $\sum_{j=1}^{H}(W_{hh_j}^{+} + W_{hh_j}^{-})$. This dynamic formulation increases the complexity of not only the forward phase, but also the backward one.

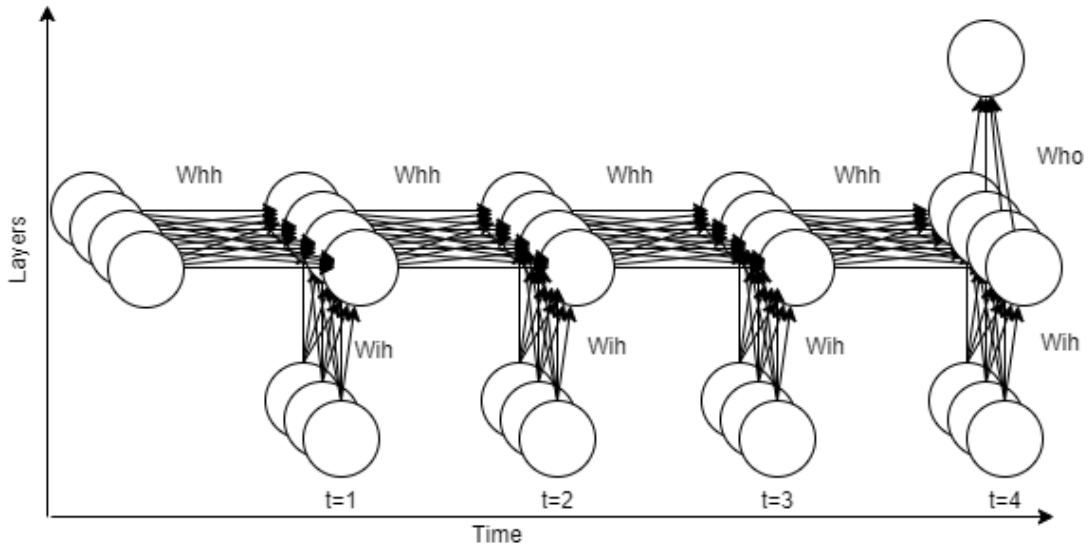

Figure 2: RNN architecture

In the case of output layer, since there is no connection between output layers through the time and only input comes from the hidden layers of the same time step, the equation of output layer is relatively simpler compare to the hidden layers.

$$q_{o_t} = \frac{\sum_{j=1}^{H}(q_{h_t} W_{ho_j}^{+})}{r_o + \sum_{j=1}^{H}(q_{h_t} W_{ho_j}^{-})} \tag{2.6}$$

Notice that the rate parameter of output layer, r_o, is not given explicitly. Considering the fact that the term r_o is about the output connections of a neuron, for the output layer there is an ambiguous situation. It is actually a free parameter. Thus, it can be given arbitrarily. But the experiments show that in some cases, the static value of r_o actually affect the convergence. This effect is not spectacular, but could be vital for an application where a small change matter.

3. Backpropagation

This section contains information about Backpropagation Through Time (BPTT) algorithm and how it is used to train QRNN.

Considering the fact that, QRNN is an algorithm which is based on general working principle of RNNs, its backpropagation mechanism is quite similar to RNNs. The implementation of BPTT algorithm to QRNN is quite similar to Simple RNN. However, the generic formulations of BPTT for QRNN is a bit more complex due to the number of trainable parameters of QRNN model.

The following expressions are derived for a simple QRNN with one hidden layer. For the sake of simplicity, in the following equations, some abbreviations and new notations are used. The denotations are as follows:

- All the positive and negative weight matrices: W^{*}
- Positive and negative weight matrices together: W_{ih}^{*}, W_{hh}^{*}, W_{ho}^{*}
- Nominator of the output formula of a neuron (which is ($T_{*_t}^{+}$)): N_{i_t}, N_{h_t}, N_{o_t}
- Denominator of the output formula of a neuron (which is (r_{*}+$T_{*_t}^{-}$)): D_{i_t}, D_{h_t}, D_{o_t}
- Input-Output pair at iteration k: (x_k, y_k)

In order to update weights, “The Generalized Delta Rule” will be used as it has been used in the previous backpropagation calculations before this section.

$$W^{*(k+1)} = W^{*(k)} + \delta^{*(k)} \tag{3.7}$$

First step is to calculate loss by comparing the network output with the real one. Loss function can be chosen arbitrarily according to the application. Since the method is trivial for the rest of the BPTT process it will be denoted with the vector L_t as a function of vectors q_{o_t} and y_t where the length of the each vector is the same and equal to the length of the output layer which is O.

$$\frac{\partial q_{o_t}}{\partial W_{ho}^*} = \frac{\partial q_{o_t}}{\partial W_{ho}^*} + \frac{\partial q_{o_t}}{\partial q_{h_t}} \frac{\partial q_{h_t}}{\partial W_{ho}^*} \tag{3.8}$$

As it can be seen in the forward calculation equations, the weights are a variable for both of the layers that they are located, unlike the rest of the RNNs. In the other RNN structures the weights are only the variable for the target layer that receives input from the other, not the layers that gives their output. From this point of view, QRNN has a unique position in RNNs because every change in the weight matrices, affects both of the layer that shares that weight matrix. Considering the general formulation in equation 1.2 negative weight matrices create inhibition effect for both of the layers that shares this matrix. In contrast, positive weight matrices create excitation effect on the layer that receive signal and inhibition effect for the layer that sends the signal. This characteristic feature enables QRNN, RERANN and RANN to learn highly non-linear correlations between inputs and outputs.

The update rule of W_{hh}^* is relatively more complex compare to the update rule of W_{ho}^*

$$\frac{\partial q_{o_t}}{\partial W_{hh}^*} = \frac{\partial q_{o_t}}{\partial q_{h_t}} \sum_{m=1}^{t} \frac{\partial q_{h_t}}{\partial q_{h_m}} \frac{\partial q_{h_m}}{\partial W_{hh}^*} \tag{3.9}$$

$$\frac{\partial q_{h_t}}{\partial q_{h_m}} = \prod_{t \geq j \geq m} \frac{\partial q_{h_j}}{\partial q_{h_{j-1}}} \tag{3.10}$$

The weight matrices W_{hh}^* and W_{ih}^* are get updated by the sum of the gradient for each time step output, which is calculated by iterating backward from the time step t to the time step 1 using the

counter m. In other words, there are two recursive "For Loops" where the range of inner one is determined by the value of the outer one. In this case, counter of the outer one is t with the upper limit T and the counter of the inner one is m with the upper limit t. This recursive structure is valid for all the RNNs since they all use BPTT as well.

In the case of W_{ih}^{*}, in terms of number of parameters, the BPTT becomes more complex than W_{hh}^{*}, since it affects the outputs of both input and hidden layers.

$$\frac{\partial q_{o_t}}{\partial W_{ih}^{*}} = \frac{\partial q_{o_t}}{\partial q_{h_t}} \sum_{m=1}^{t} \frac{\partial q_{h_t}}{\partial q_{h_m}} \left(\frac{\partial q_{h_m}}{\partial W_{ih}^{*}} + \frac{\partial q_{h_m}}{\partial q_{i_m}} \frac{\partial q_{i_m}}{\partial W_{ih}^{*}} \right) \tag{3.11}$$

In the case of QRNN, the weights and the other terms can be located in the nominator or the denominator of the calculations of a neuron's output. In some cases they appear in both. This situation allows the same weights to change the output of a neuron positively and negatively at the same time.

Considering the figure 2, one of direction that needs to be considered in the gradient calculation is the "$time$" axis and the other one is the "$layer$" axis. The formulation of gradient in the axis of "$time$" and "$layer$", is given in the equation 3.12 and 3.13 respectively.

$$\begin{aligned} \frac{\partial q_{h_j}}{\partial q_{h_{j-1}}} &= \frac{\partial q_{h_j}}{\partial N_{h_j}} \frac{\partial N_{h_j}}{\partial q_{h_{j-1}}} + \frac{\partial q_{h_j}}{\partial D_{h_j}} \frac{\partial D_{h_j}}{\partial q_{h_{j-1}}} \\ &= \frac{1}{D_{h_j}} W_{hh}^{+} - \frac{N_{h_j}}{(D_{h_j})^2} W_{hh}^{-} \\ &= \frac{W_{hh}^{+} - q_{h_j} W_{hh}^{-}}{D_{h_j}} \end{aligned} \tag{3.12}$$

$$\begin{aligned} \frac{\partial q_{\ell_j}}{\partial q_{\ell-1_j}} &= \frac{\partial q_{\ell_j}}{\partial N_{\ell_j}} \frac{\partial N_{\ell_j}}{\partial q_{\ell-1_j}} + \frac{\partial q_{\ell_j}}{\partial D_{\ell_j}} \frac{\partial D_{\ell_j}}{\partial q_{\ell-1_j}} \\ &= \frac{1}{D_{\ell_j}} W_{\ell,\ell-1}^{+} - \frac{N_{\ell_j}}{(D_{\ell_j})^2} W_{\ell,\ell-1}^{-} \\ &= \frac{W_{\ell,\ell-1}^{+} - q_{\ell_j} W_{\ell,\ell-1}^{-}}{D_{\ell_j}} \end{aligned} \tag{3.13}$$

Even if the form and evaluation method of formulations are quite similar, the equations that evaluate the gradients between layers are shown with two different but also similar representation. Since the calculations between time steps are only performed for hidden layers in time, for the term ∂q_{h_j} the index h is preserved during the calculation while the index j is decremented by 1 which denotes the hidden layer $j-1$. On the other hand, since the equation 3.13 shows the gradient calculation for the axis "$layer$" the time step j preserved and layer index ℓ is decremented by 1 in order to obtain the gradient.

REFERENCES

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