Introduction of Reflex Based Neural Network

Liang Yi (liang.y.jlu@hotmail.com)

Abstract

This paper introduces a new neural network that works quite different from current neural network model. The RBNN model is based on the concept of conditioned reflex, which widely exists in real creatures. In RBNN, all learning procedure are executed by the neural network itself, which makes it not a complex mathematic model but simple enough to be implemented in real brain. In RBNN, information is organized in a clear way, which makes the whole network a white box rather than a black box, so we can teach the network knowledge easily and fast. This paper shows the power of conditioned reflex as a search tool which can be used as states transfer function in state machine. Using combinations of neurons as symbols which plays the role of letters in traditional state machine, a RBNN can be treated as a state machine with small number of memory unit but huge number of letters.

Introduction

Current artificial neural network model is just a mathematic tool. It can solve a lot of problems, but some of its features make it impossible to be the way how a real brain works.

The first undesired feature is that the whole leaning process needs out-of-system computation. The learning process changes the weight of inputs of neurons. And calculating those changes needs quite complex out of system calculations which is so complex to be done by real neurons. Also, the whole procedure takes too much time. There doesn't seem to be an easy way to implement a pure neural network to do the learning process fast.

Even when we get those changes, the way to execute the changes is also quite undesired. In current model, these changes are also out-of-system action. Of cause, in computer it is quite easy to increase or decrease a number in memory unit. But in real brain, the weight unit seems only connect to an input dendrite. If you want to change the weight, you'll have to encode the change (or new value) to a single wire of dendrite. It is a float number, and it can be positive or negative. Also, you must make a distinction between working mode and learning mode to tell the neuron if current signal represents a signal to respond or a signal that should be decoded to a change of weight. To do all these works there must be an extremely complex structure in neuron for each input.

Current training model usually requires high precision float computation. And some trained networks are quite sensitive to input value. Very small changes in input may greatly change the output (maybe partly due to over-fitting). But in real brain you can't expect high precision at all. To make things worse, neurons die and mis-work every second. The network model in real brain must be robust and redundant.

Also, compare to real brains, current artificial neural network seems to be too powerful. Given a network big and deep enough, you can train it to do any job, maybe not quite well, but it will work. But for real brain, the network seems to be quite specialized. Real brain can do some jobs extremely well and fast, but for a job that it is not born for, it usually simply doesn't work at all. A well-trained dog can understand some simple words, but if you want to teach it gramma, even the simplest ones, you are wasting your time. And a baby, on the other hand, can learn gramma quite naturally.

Of cause, currently artificial neural network model has made a lot of great achievements. But it is just a mathematic tool. In our brain, there must be a different way to do the job.

In real brain, there must be no out of system magic. The whole network should be able to do the learning job as a black box. The only way to interact with outworld is its inputs and outputs. To train a network, we give it some inputs, then we observe the outputs. Depends on the outputs, we give it some other inputs. It should be an interactive procedure like a teacher and a student. This is just like how we learn. We can read, see, talk, or do actions. But there aren't wires which connect to the neurons in our brain manipulate them. A neuron changes its input weights only depends on its state and inputs. And each neuron should be simple enough to be implement in real cell.

What a neuron can be

Let's focus on single wire communication problem first. The weight of an input can be positive or negative. When people drink alcohol, the negative signals that usually keep them from do something are deactivated first, so they become bold, and willing to talk. If they drink more, the positive signals that usually let them do something are also deactivated, so they lose control of their body and fall asleep. Inspired by this fact, in RBNN, there are two kinds of inputs, positive ones and negative ones. The positive inputs always have positive weight, and the negative signals always have negative weight. In learning procedure, the weights can be enhanced, but can never be weaken. With such limits, the communication between neurons can be greatly simplified.

Now let's think about how a neuron calculate the total input $\sum x_i w_i$. x_i are always 0 or 1. Discreate value are hard to analyze, so we usually treat them as continues value and we invented a lot of activation functions. But in real cell, a switch value is much easier to deal with. If we treat x_i as a switch, $\sum x_i w_i$ become the sum of all w_i that x_i is on. There's only addition left. Use the number of some molecules to represent the

value, addition and subtraction operation can be implemented by mixing two kinds of molecules. To invent a small machine to do this job in a cell, the solution is quite straight forward. Fig 1 shows the idea.

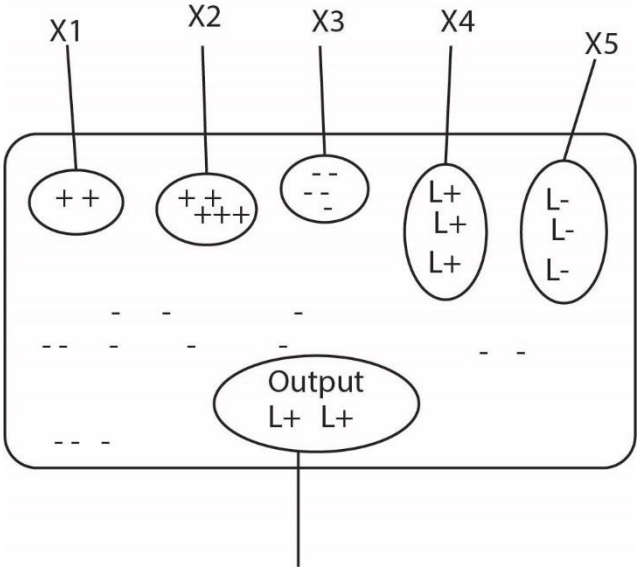

Fig 1

It's an organelle that do the calculation. The outer bubble contained some alkali-like molecules and usually the neuron maintains them at a fixed concentration which represent the base value. The input bubbles, each connected to an input signal, contained some acid-like or alkali-like molecules. When input signals come, the carriers socked in the corresponding input bubbles' membranes will be activated by current and start to transport some molecules to outer environment. Then two kinds of molecules mix and react in outer environment. If the concentration of acid-like molecules in outer bubble reaches a certain value, the output structure will generate an output pulse. The process of generate output pulse also consume some acid-like molecules, so the environment value is reduced, and the cell is deactivated for next wave of inputs. It's quite simple and seems reasonable. We know that there are quite a lot of chemicals works in pairs to control something in human body.

If this is how a neuron work, then how does a neuron learning? If we have out-of-system magic, we can simply read and change the number of molecules each time the input bubble releases. But we don't want out-of-system magic, so all learning command should go through the only input wire which link to input bubble. Since the releasing of molecules is controlled by the carriers in input bubbles' membranes, we can change the number of those carries to change the weight of each input. To do this, we introduce two more kinds of molecules: L+ and L-. In Fig 1, x4 and output bubble contains L+, and x5 bubble contains L-. When x4 or output is activated, L+ will be released. And when L+ reaches a positive input's bubble, it will increase the number of carriers of that babble, so the weight of that input is enhanced. L- works like L+, but they only enhance negative inputs' bubbles.

I think the most possible thing that happens in real cell is like this: Each bubble has a fixed number of carriers. Most of the carriers are deactivated at the beginning, they don't respond to input current, so the weight of the input is small. And the inactivate carriers can be activated when L molecules and current both exist. Such process can be very fast and suitable for short memory. Increasing the number of carriers is another plan, but it seems to be a slow process which may good for long term memory.

In mathematic view, the learning procedure is simple enough: when learning signal appears, the weighs of activated inputs are enhanced. More learning signals are activated, more value of the weights is enhanced. For some neurons, those weights will grow weaker by time, which is how you forgot things. This is not quite interesting for our model.

The last thing about learning is the base value that a neuron should generate an output pulse. For some neurons, a fixed base value is enough to the job. For many of them, the base value is dynamic. Fig.2 shows the idea.

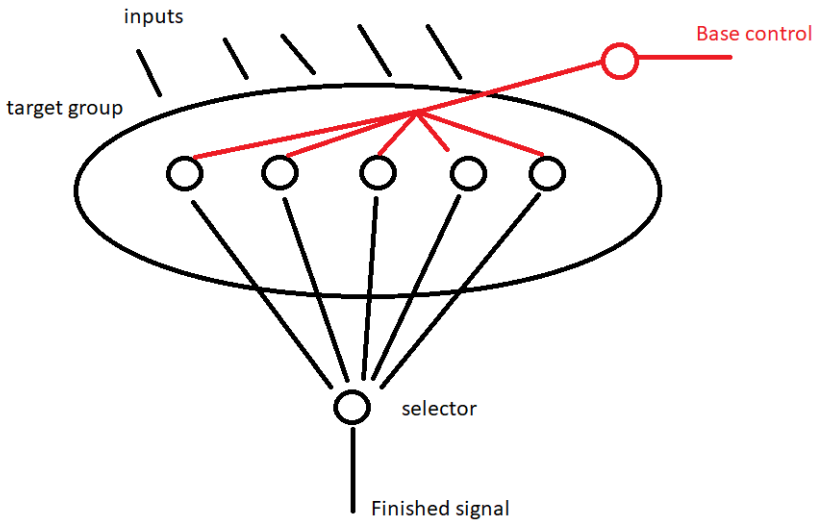

Fig 2

Let there be another type of input called base control, which has small positive weight. If you active such inputs at a certain frequency, the neuron will respond fast and release acid-like molecules at a certain speed, which will gradually change the base concentration of alkali-like molecules, which will equivalently change the base value.

The target group contains a lot of neurons, each have some inputs and they share same base control signal. Given a set of inputs, different neurons have different total inputs. At the beginning, the base value is so high that none of them are activated. Then we

active base control signal and keep generate pulse which will increase the target neurons' total input and they will be activated by the order of their initial total inputs. When enough neurons are activated, the selector will be activated and generate a signal to end the process, so we can select a fixed number of neurons in a group.

This is the basic picture of neurons in RBNN. I don't have any direct biological evidence. But we do see similar things in real cells. Chemicals work in pair, carriers on cell membrane that transport molecules, electric charge sensitive protein, they are all found somewhere in real cells. I try to combine them in a simplest way to make things work, and I get the picture above.

It's a weak version of current artificial neuron network. It forbids a lot of operations. But if you must implement it in a real cell, these limits are necessary. Later will see that with some other types of neurons, it shows great power. Before introducing other types of neurons, let's discuss how to implement conditioned reflex.

Conditioned reflex

Conditioned reflex is one the most basic action of neuron network. The most famous experiment is Ivan Pavlov's dogs. If you keep on ring a bell before giving some meat to a dog for some days, the dog will respond to ring to start to salivate.

Fig 3 show the basic concept of how to do this in RBNN.

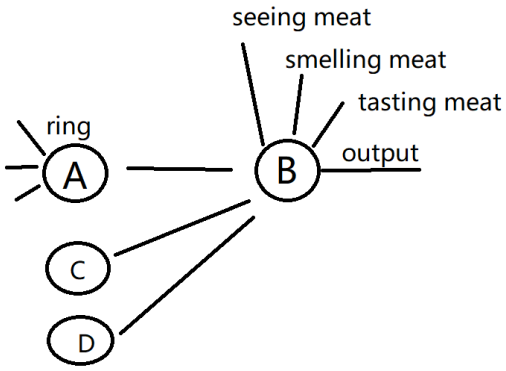

Fig 3

Neuron A represent the ring signal. When it hears the ring, Neuron A is activated. Neuron B controls the salivating process. If B is activated, the dog begins to salivate. B has many inputs. Seeing meat, smelling meat and tasting meat all can activate B. But at the beginning, hearing the ring will only active A. The weight of the link A-B is zero, so B will not respond to A.
Now, let the learning begin. We ring the bell, and A is activated. We feed the dog meat, and B is also activated. When B output the pulse, its output bubble also releases some molecules L+ which will enhance the weight of activated inputs. In this case, that are the input of seeing, smelling, tasting signals, and the most important one: ring signal A.

For other signals like C and D, they are not activated, so they are left unchanged. When we repeat the training, the weight of link A-B will grow bigger and bigger. When it's big enough to activate B only by the input A, we say the reflex is established. If you stop the training for a long time. The input bubble of A-B will gradually loose some kind of its chemical and the weight grow smaller. That's how the dog forgets the reflex.

As you can see, in RBNN, establishing reflex is quite simple. All you need is a direct link between two neurons. There's no out-of-system operation.

When the number of neurons grows, the required number of direct links will soon be unacceptable. In real brain, the number of neurons that represent some sound can be millions. And there are other signal sources, like some shape, some color, some feeling of some part of body. Not only the sources, the actions like salivating can also be millions. Barking, standing up, sitting down, moving to certain position, they can also be the target action and each of them involves quite a lot of neurons. Although all these links are possible, only quite a few of them will be really established. Which ones are useful depends on the training, but in the end, the established links won't be too many. To reduce the number of links, we need an extra reflex layer.

Reflex layer

In real brain, the source and target neurons of a reflex usually locate at different areas. The connections between them usually need to pass a long distance. Too many connections will take too much space. With an extra reflex layer, then space requirement can be greatly reduced. Fig.4 shows the idea.

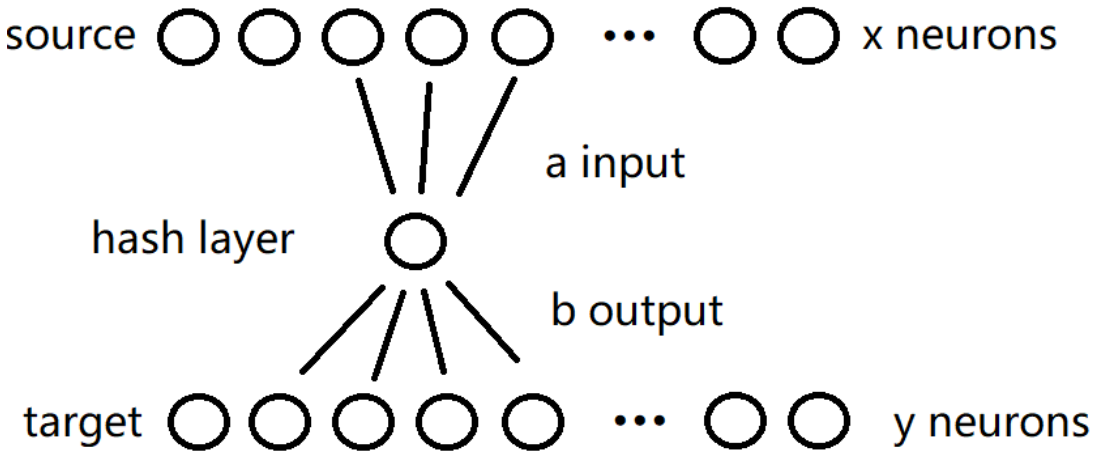

Fig 4

We want to map x source neurons to y target neurons. Full connection needs x * y links. Now, let's introduce an intermedia layer, called reflex layer, which contains neurons with a inputs and b outputs. To ensure there's a path between each pair of source and

target neuron, you only need at least (x * y) / (a * b) neurons in reflex layer. In a real neuron, a and b can be more than a thousand, so with only one reflex layer, the connections between source and target layer can be reduced to an acceptable number. We should notice that it's not necessary to make sure there's a path between each pair of source and target neuron. In real brain, neurons are redundant. When a dog hears ring, a lot of neurons (set S), in source area will be activated. And a lot of neurons (set T), control salivating process. To establish the reflex, we only need to make sure that most neurons in set S and T have several paths to the other set. By this way, the number of hash layer neurons can be further reduced. Theoretically, the minimums number of reflex layer neurons mostly depends on the number of function blocks of system rather number real neurons. You can increase the system's redundancy without increasing the number of connections dramatically.

Now we still have a lot of connections between reflex layer to source/target layer. But if we arrange them correctly, they won't take much space, like in fig 5.

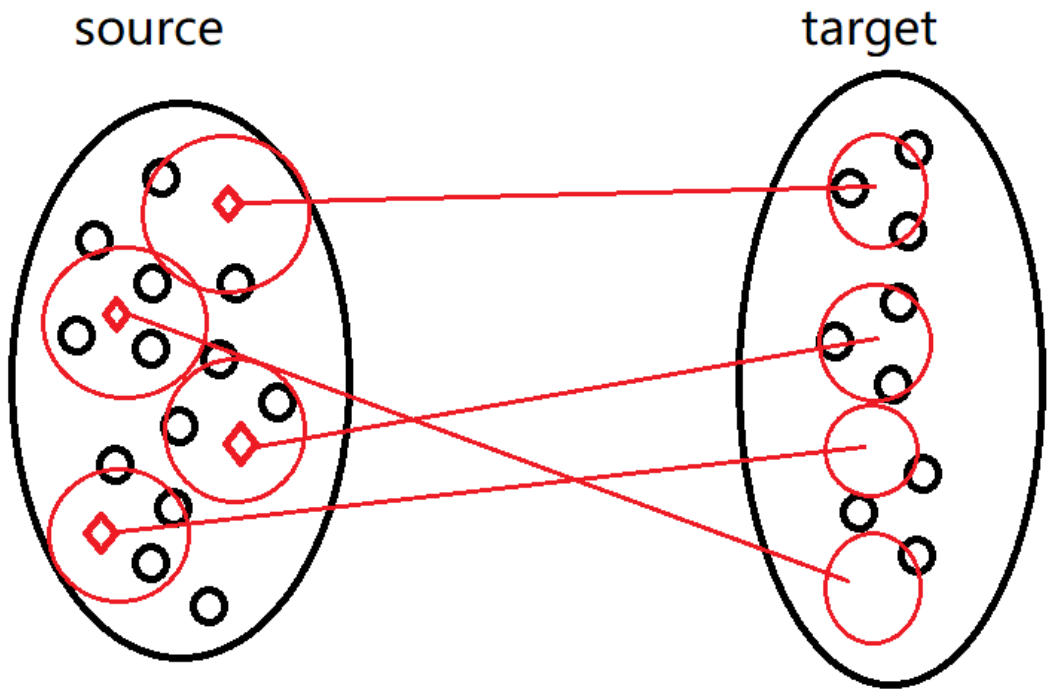

Fig 5

In fig 5, red diamonds are the reflex neurons. Let's put them inside the source area. A hash neuron gets inputs from the surrounding area inside the red circle. It still needs a lot of connections to source neurons, but these connections are very short and won't take much space. Each reflex neuron has only one output that travels a long way to target area, so these long connections also won't take much space. When an output reaches target area, it forks, and the terminals connect to all neurons in surrounding area marked by red circle. By this way the connections to target also won't take much space.

All reflex neurons only need to be located randomly in their area. For you can't expect

neurons in brain grow follow some regular pattern. And since they are randomly grown, there must be more than (x * y) / (a * b) hash neurons. Increased by 10 or 100 times maybe enough. Some source / target neurons may still be missed, but the source / target neurons are also redundant. If you don't miss too many connections, there will be no real problem.

Now, let's see how these neurons learn. The target neurons learn as the old way: when it is activated, enhance activated inputs. On the other hand, the reflex neurons don't need learn at all. If they receive any input, they output 1. Let's take an example to discuss this. Let there be 1000 reflex neurons, and the correct input signals actives 10 neurons in source layer. Because all neurons are grown randomly, we can expect that 10 neurons in reflex layer will be activated. Let their weights on target layer to be 1. A correct signal will generate a total input of 10 on a target neuron. Now given incorrect signals which also active 10 neurons in source layer, which will active 10 hash neurons which are chosen randomly. On average, these hash neurons will only generate 10 * 1 * (10 / 1000) = 0.1 total input which will be filtered out by a selector. Sometimes, it may still make mistakes. We can't expect it works perfectly and some procedure to check the result is needed.

One important thing is that I put reflex neurons in source area. What if we put them on target area and use the forking skill in source area? Forking of input signal has a problem. To assign weight for each input, we need each input wire connects to an input bubble. If two input wires join before linking to input bubble, they must share one weight. That is a wire or operation. If you put two bubbles near the join site, the following wire must transfer chemical signal rather than electrical signal, which is totally unacceptable for long distance. Maybe this is the reason that dendrites are often short. But for reflex neurons, since they do perform the or operation, this plan seems also acceptable (And it may be better because of the decreasing of number of input bubbles. Whatever, they are same in mathematic.)

Salivating reflection is one-way reflection. But there's no limit to stop us to establish bidirectional reflection. For example, when we hear the sound of a letter, some neurons in our brain are activated. And when we see that letter, some other neurons are activated. We can establish a bidirectional reflection between the two group of neurons. When we hear the letter, we can recall how it looks like. And when we see the letter, we can recall how it sounds. Fig.6 shows the idea.

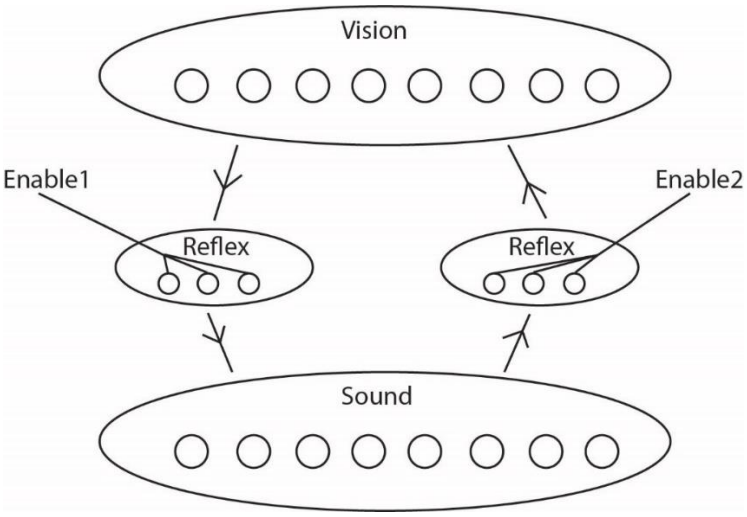

Fig 6

There are two enable signals each link to a group of reflex neurons to enable them. Two group of reflex neurons can't be activated at the same time. By this way we can avoid feedback, which may cause the network unstable.

Register

Long term memory can hold for years or even a whole life. Short term memory can hold for hours or minutes. Some memories can only hold seconds. When we are thinking, words and ideas flow in our mind, and we can intentionally remember something and recall them later. This fact suggests that there must be some neurons that works like the register in computer which can save the state of other neuron and restore them later. There are many ways to do this. Fig.7 shows a plan.

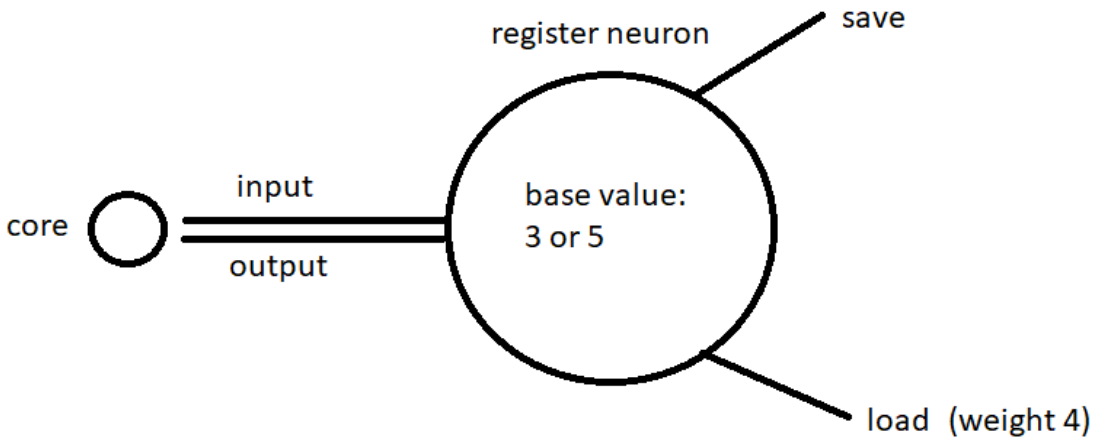

Fig 7

It has one input and one output both connect to a same neuron called core, and it has 2 other inputs, one called save and one called load. When save signal is activated, it changes its base value depending on the state of input signal. When input signal is 1,

the base value is set to 3. And when input signal is 0, the base value is set to 5. By this way, it can save the state of core neuron which is a one-bit data. The load signal has a fixed weight of 4. When load signal is activated, the output will be the state it saved before so core neuron can accept this output to restore its state.
Fig 8 shows how to organize such neurons in groups.

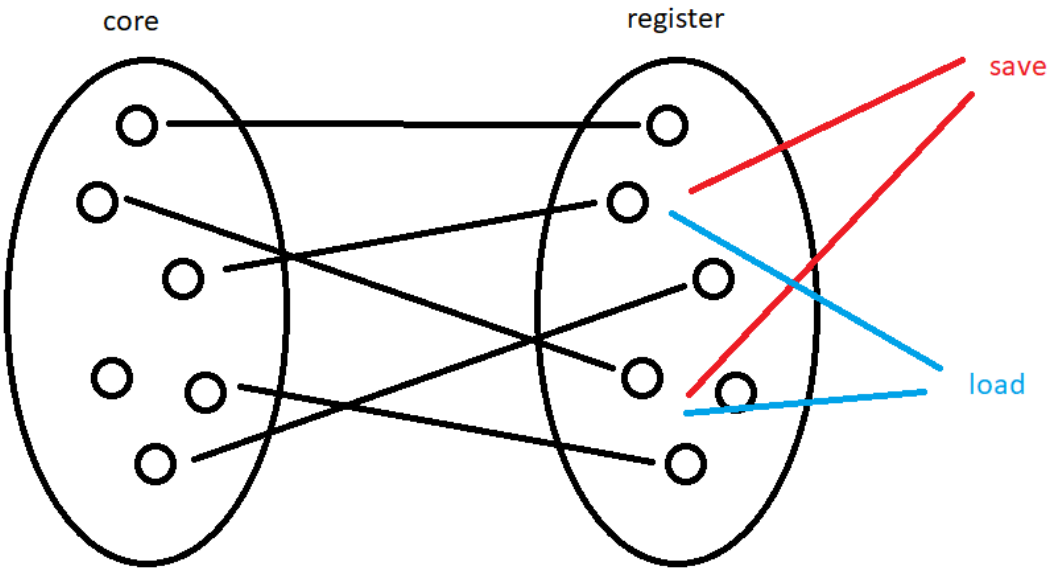

Fig 8

Left Fig.8 is A group of core neurons. And a group of register neurons in another area. All register neurons in a group share same save and load signals. The core group use different combination of activated neurons to represent different data, and the register can save and restore the data.

Register can be linked to a core, or another register. A chain of register can be quite useful to deal with sequence. A word is a sequence of letters. When a new letter gets into core, the register chain shift one cell and accept the new letter as new top. Not only a chain, all registers can form a tree with a core as root. Fig. 9 shows the idea.

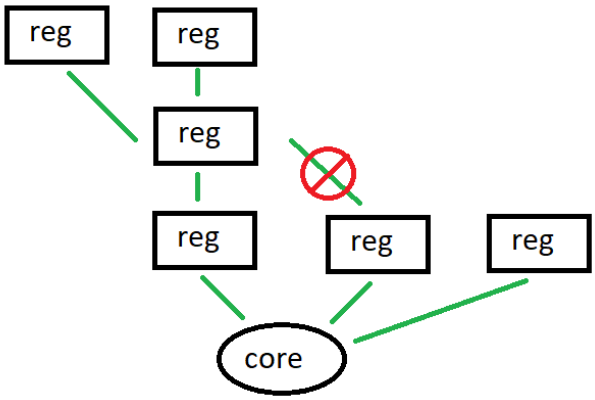

Fig 9

Some connections in the tree may be one direction way. That is, you can store data into them for reflex, but you can't restore them back to their source. Without the need of copy back, those registers don't need to be a full copy of data. Instead, they can just hold a hash of source. This will decrease the number of neurons.

The connections between registers are all randomly grown. We can't expect regularly one to one connection. For this reason, ring of register is not acceptable in real brain. When we work with computer, ring cause no problem and that will make some data transfer much easier. But I suggest that we focus on the things that can happen in real brain first. Because there must be a way to make things work. And with fewer options, the path to answer is also limited.

State transfer and calculation

Conditioned reflex can be used as state transfer function to construct state machine. Fig 10 shows the basic idea of how to calculate 1 + 1 = 2.

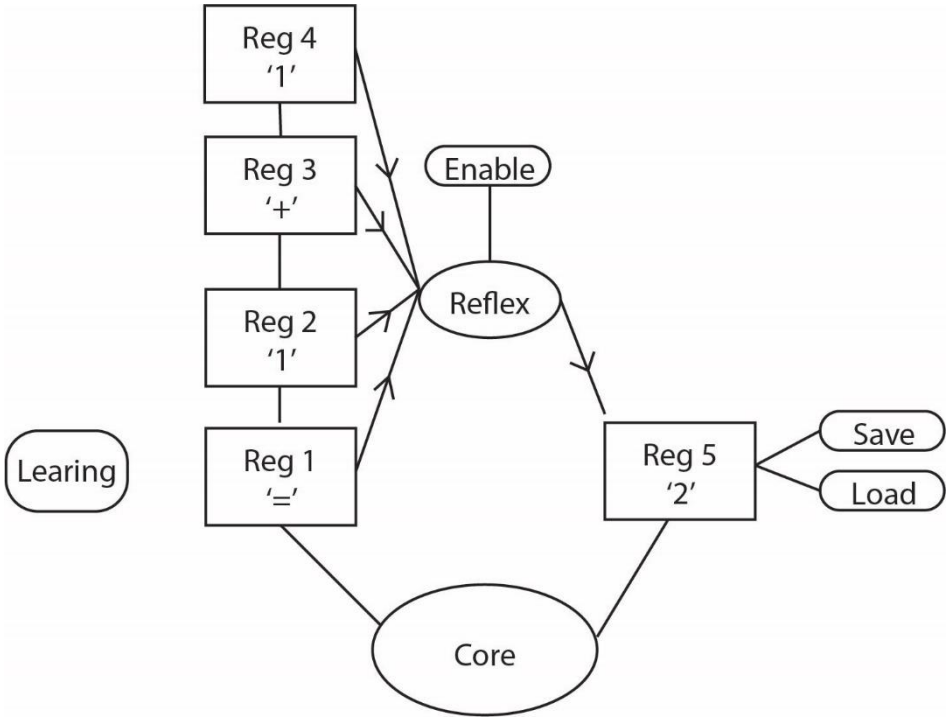

Fig 10

Each time the network receives a character, certain combination in core will be activated. Each combination is called a symbol. Here let's just say that some symbols in core are 1, 2, plus and equal. When a new symbol is activated in core, the network should transfer it into the register chain on the left which represent question. After we put "1+1=" into question chain, a reflex should active the neurons that represent number 2 in register 5 which holds answer. Then it should be copy to core for output. The procedure to establish such reflex is just the learning procedure. It is also a simple procedure: The network should transfer the input in core into Reg5 and enhance the

reflex.

All these works involve some action, like transfer data, shift register list. These works can all be done by the network itself. These actions are executed by activating enable signal in Fig.6 and save/load signals in Fig.7 by correct order. Let's call neurons who output these signals signal delegate. Since they are just ordinary neurons, we can establish reflex form other neurons to them. In Fig.10, there are some of such neurons like Learning, Instruction, Save/Load of Reg5, Enable of Reflex. The Learning signal indicate whether the network is learning. A reflex indicate next action has the form of (Reg1…4, Learning, …) -> (Enable, Save, Load, …).

We can treat the network as a state machine, which uses combinations of activated neurons as states. Then the reflexes just become the state transfer function. It works quite like a Turing machine. Traditional Turning machines usually have quite limited dictionary but large number of memory unit (Technically speaking, it should be infinite. But usually we can treat a computer with large enough memory as a Turning machine). A RBNN network usually have quite large dictionary but a small number of memory unit. In RBNN, the unit of calculation which plays the role of letter in traditional state machine is called a symbol.

Symbol

In RBNN, a symbol plays the role of letter in traditional Turning machine. It is a certain combination in core neuron group that works like pointer or reference in traditional programing language. It may refer to some letter, some word, some people or some action sequence. It is the basic unit in calculation and in each time, only one symbol can be activated in a core neuron group. Different symbols may share same neurons, but as long as the number of neurons is large enough, the conflict will not be a problem. Let's assume the core contains 1 million neurons, and each symbol contains 1 thousand neurons. If they distribute randomly, two different symbols will share only 1 neuron on average. The shared neurons will produce noise when we do reflection, but later we'll see the noise can be filtered out.

The symbols in a core group always have same number of neurons. In our example, the number is 1000. When our brain receives some inputs, like hearing a letter, or seeing a letter, some neurons that represent some features are activated first. Those feature neurons may represent some line, some color, some shape or some voice feature. The basic way to generate symbols is from features. A direct way is to select 1000 feature neurons with highest total input as a symbol. But that will not work well. The feature neurons, are often of different types. Some of them may represent lines, and some of them may represent corners. They have different number of input, different input weight, and different base value. Their total input can't be simply compared. And they are redundant. One feature, like a corner or a line, may activate quite a lot of neurons, and all these neurons have relatively high total input. If we select a small number of feature

neurons with highest total input, they may all belong to same feature. The symbol should be generated from a overview of features rather than a small number of most significant features. To make things works, we need to introduce an extra hash layer. Following figure shows how to generate a symbol in core.

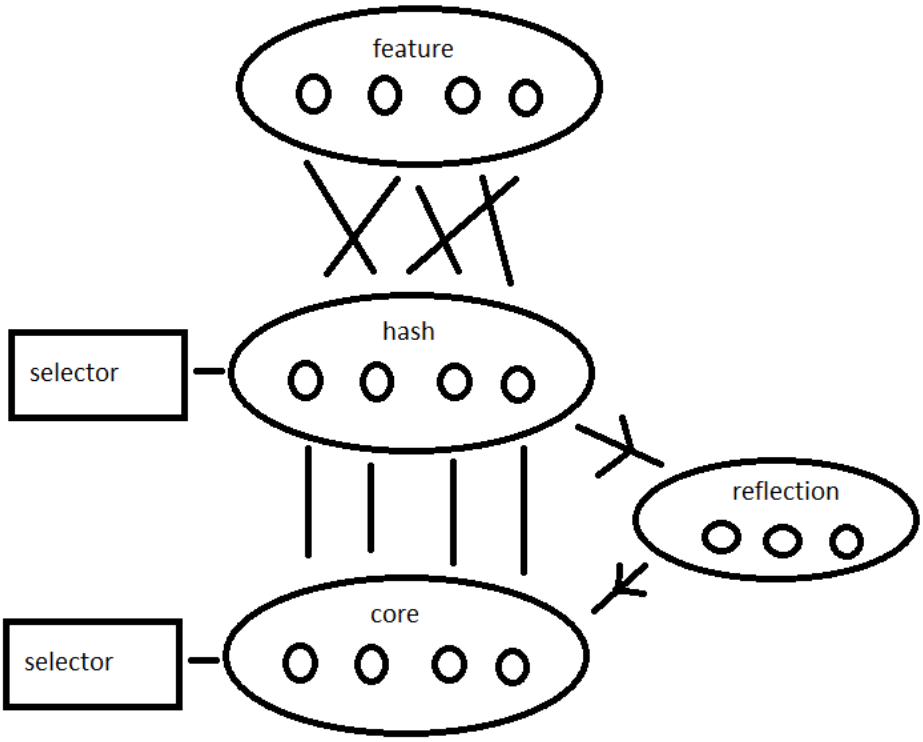

Fig 11

The hash layer is formed by neurons which have a great number of inputs that randomly connected to feature layer. Let feature group contains 10 million neurons for example. When we see something, like a letter, the number of activated neurons in feature group is not a fixed number, but it must be large enough to get our attention. In this example, let it to be 10 thousand. The chance that a neuron is activated in feature group is 1 in 1000. Let a hash neuron have 100 thousand inputs randomly connect to feature group, and each input has a fixed weight of 1. The total input of hash neuron follows the binomial distribution. The mean is 100 and the variance is 100. In real brain, the connection can't be fully randomized, so the distribution of total input may be quite complex. But the detail of distribution is not important. As long as some hash neurons connect to more activated feature neurons, we can use a selector to select 1000 hash neurons with highest total input as a symbol. The connection from hash layer to core is a one to one mapping, so we can copy the symbol from hash layer to core.

One important character of symbol is that a symbol can be fully activated by a small set of feature neurons. When we see a blur letter at first sight, we may have no idea what it is. But suddenly we may find that it looks like a letter A. Then the symbol A

appears in our brain. We may still not sure whether it really is a letter A, be we can clear know we are talking about A. To make this happen, let's add a reflex from hash group to core. When we generate a symbol in hash group, we do not only copy them to core, but also establish a reflection from hash to core. This reflection works as an amplifier. Given a small set of symbol neurons in hash layer, the reflection can activate the whole set of that symbol's neurons in core. It also increases the total input of the activated symbol. Then, with a selector to select top 1000 neurons in core, the noise which is not amplified by reflect will be filtered out.

When two symbols share quite a lot of feature neurons, they may be amplified at the same time, so two symbols are both activated in core. Luckily, it's not quite difficult to detect. If the base value is too high for the selector of core to select top 1000 neurons, a signal can be raised to notify that a conflict may happen. Then the brain can react to the abnormal signal to do some procedure to solve the conflict. Also, if the base value is too low. It may be a new symbol which's reflection is not fully established yet, so some learning procedure should be executed.

Context register

Store operation can save a symbol from core to a register. As we can see in Fig.10, a serial of register can be used to save a sequence of symbols as the source of reflex. By this way, the result depends the order of symbols. In many case, order is not quite important. Such information is often call context.

We can do store operation several times into a register without clear it, so that several symbols may exist in one register. Such a register can be a good reflex source to provide context information. A reflex that take such a register as source will take a lot of symbols into account without considering their order.

Options and negative register

As we already see, reflex can be treat as search operation. We active source conditions first, then select the most possible answer. However, reflex is a rough operation. You can't expect it always gives correct answer. In real world, when hearing a question, we often get an answer by reflex without thinking, especially when we are doing sports. But if we take more thinking, we may find that answer is not correct, or least, not the best one. We need other plans. While in RBNN, reflex structure can give not only the most possible answer, but also second, third possible answer. To do this, we need a negative register.

A negative register is usually a context register which output negative signals. For example, we can save letter B and letter C from core to it. And when we active the load signal to write the data bake to core, it doesn't active the core neurons the belong to

letter B and C. Instead, it inhibits them from being activated. The weight of those signal should be high enough that the neurons belong to symbol B or symbol C in core have no chance to be activated.

Now, when a reflex gives the best answer in core, let's store it into a negative register. To get the second option, we do the reflect again, and in this time, we also active the negative register which stores the first answer. Because the first answer is suppressed, the second answer will be activated. Then we store the second answer to negative register too. Repeat the procedure, we can enumerate the answers given by a reflex.

By this way, the reflex doesn't need to always give correct answer. Instead, it only need to make sure that the correct answer appears in several top options. We can check them one by one until we find the correct one.

If we link a negative register to feature group, we can easily implement set subtraction. Also, two positive registers link to feature group with a proper selector can be used to do set union. These set operations can be very useful to find the key feature of certain concept.

Sequence

Sequence plays an important role in RBNN. A word is a sequence of letters. A music is a sequence of notes. And above we introduced many actions like transfer data which are also sequence. From the networks view, all these sequences are the sequences of symbols.

A sequence must have a handle, which is a symbol that represent the sequence. When the network refers to the sequence, it uses that handle symbol rather than all symbols in that sequence. For example, there's symbol for the word 'Apple', which is the handle of the sequence of letters 'start-A-p-p-l-e-end'. Here I introduced the symbol of start and end, which are important to indicate the network to do special works.

A handle can be any existing symbol. It can also be generated by the sequence itself. With the symbol of start and end, we can easily put the member symbols into register tree. Then we take the whole register tree as feature group in Fig.11. By this way we can get a handle which is the hash of its contents.

To learn a sequence, we need to establish such reflexes:

(handle, start) > (first symbol)
(handle, start, first symbol) > (second symbol)
(handle, start, first symbol, second symbol) > (third symbol)
…
(handle, start, first symbol, … last symbol) > (end)

Apparently, if the handle exists, we only need to feed the sequence into the network once to generate the reflex. Sometimes these reflexes may conflict with other exiting reflexes. If that happens, complex procedure is needed to solve the conflict.

We referred many procedures in the whole paper. These procedures are sequences too. They can also be learned or changed by the learning procedure. The whole network can learn knowledge, and it can also learn how to learn.

Conclusion

In RBNN model, we can see that the ability of a network heavily depends on two things: the structure of the network and the initial knowledge of the network. For a real brain, they are all decided by the host's DNA. Without correct structure, the network won't work at all. That's why a dog can't read at all. But the initial knowledge is a bit different. As long as there's minimal knowledge, it may be able to learn new learning method. On the other hand, the number of neurons only decides how much knowledge it can store. Before a network reaches its limit, the number of neurons is not very important.

RBNN model works quite different from traditional artificial neural network. In works in a way which is simple enough to be implemented in real brain. And the way it works also quite like a real brain. I am quite confident that this is just the way how human's brain works except some details. It organizes knowledge in the form of reflex and apply knowledge in certain context. When the pieces of knowledge fully connect to each other, we can say the network really understand the knowledge.

In this paper, I just introduced some basic ideas. However, to construct a network really works still need great effort with many details to consider. To construct and analyze RBNN, you don't need complex mathematic. It is much like to construct a computer and write program for it. I think it will not be too difficult to make some toys. And to construct a network like a real human may also not quite difficult, as long as it just a stupid one. But once we create a smart network which is smart enough to understand the whole theory of itself, it will be able to do research by itself without human's help.

Reference

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville (November 18, 2016): *Deep Learning (Adaptive Computation and Machine Learning series)* The MIT Press
2. Richard O. Duda, Peter E. Hart, and David G. Stork (2001): *Pattern Classification, Second Edition* John Wiley & Sons, Inc.
3. 吴岸城 (2016.6): 神经网络与深度学习 北京：电子工业出版社

4. 黄安埠 (2017.6): 深入浅出深度学习:原理剖析与 Python 实践 北京：电子工业出版社
5. Peter Flach (2012): *Machine Learning: The Art and Science of Algorithms that Make Sense of Data* first edition Cambridge University Press
6. 叶韵 (2017.6): 深度学习与计算机视觉: 算法原理、框架应用于代码实现 北京：机械工业出版社