

Learning inside the Rete algorithm

YKY (Yan King Yin)

general.intelligence@gmail.com

February 17, 2019

Abstract

1 Introduction to the Rete algorithm

The general form of a **production rule** is like this:

$$\overbrace{\text{green circle} \wedge \text{green circle} \wedge \text{green circle} \dots}^{\text{condition}} \rightarrow \overbrace{\text{red circle}}^{\text{action}} \quad (1)$$

where \wedge denotes logical conjunction (AND).

Typically, we would be trying to match a relatively small number of **facts** (that represent the current **state**, or **working memory**) against a very large number of **rules**:

$$\underbrace{\dots \text{yellow circle} \text{yellow circle} \text{yellow circle} \dots}_{\text{queue of WMEs}} \quad \text{match against} \quad \begin{array}{ccccccc} \text{green circle} & \text{green circle} & \text{green circle} & \dots & \rightarrow & \text{red circle} \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & \rightarrow & \text{red circle} \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & \rightarrow & \text{red circle} \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & \rightarrow & \text{red circle} \\ \dots & \dots & \dots & \dots & & \dots \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & \rightarrow & \text{red circle} \end{array} \quad (2)$$

where $\text{yellow circle} = \text{WME} = \text{working memory element} = \text{fact} = \text{grounded logic formula} = \text{formula not containing variables}$.

Obviously, if the number of rules is large, it would be time-consuming to test each rule one by one to see if they apply.

It would be much more efficient if we could look at each yellow circle and immediately see which rule(s) may apply to it. This is the idea behind Rete.

In other words, we want to *compile* the rule conditions green circle into a **decision tree**:

$$\begin{array}{ccccccc} \text{green circle} & \text{green circle} & \text{green circle} & \dots & & & \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & & & \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & & & \\ \dots & \dots & \dots & & & & \\ \text{green circle} & \text{green circle} & \text{green circle} & \dots & & & \end{array} \xRightarrow{\text{Rete algorithm}} \begin{array}{c} \text{green circle} \\ \swarrow \quad \searrow \\ \text{green circle} \quad \text{green circle} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{green circle} \quad \text{green circle} \quad \text{green circle} \quad \text{green circle} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{green circle} \quad \text{green circle} \quad \text{green circle} \quad \text{green circle} \end{array} \quad (3)$$

The actions red circle of the rules do not figure in the decision process.


1.1 How the Rete graph is constructed

To construct the decision tree in (3), it may appear that we just need to “collect like items”, but this would be wrong due to a peculiarity of **predicate logic**.

That is, there exist **linkages** across atoms in a conjunction:

$$\forall X \forall Y \forall Z. \text{ father}(X, Y) \wedge \text{ father}(Y, Z) \rightarrow \text{grandfather}(X, Z) \quad (4)$$

For example, the variable X in 2 different **positions** must be substituted with the *same* object, otherwise the logic formula is not satisfied.

Rete’s idea is to *decompose* each condition  into 2 parts:

- one pertaining to the **internal** structure (“schema”) of the condition itself
- one pertaining to the **inter-relations** between this condition and the others in the conjunction (the “linkages”)

This gives rise to the famous **alpha** and **beta** nodes:

$$\text{green circle} = \alpha\text{-condition} \wedge \beta\text{-condition(s)} \quad (5)$$

Each α -node is responsible for matching only 1 atom in the conjunction:

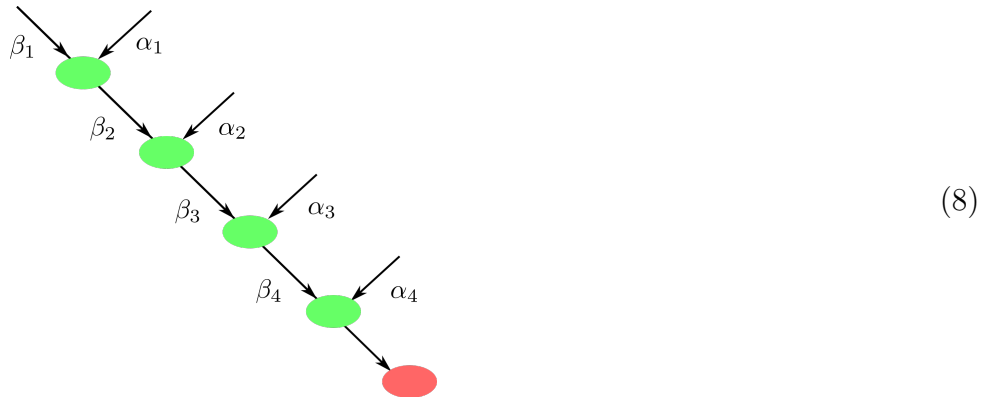
$$\begin{array}{c} \alpha_1 \quad \alpha_2 \quad \alpha_3 \\ \text{green circle} \wedge \text{green circle} \wedge \text{green circle} \dots \rightarrow \text{red circle} \end{array} \quad (6)$$

A β -node is responsible for the relations between the latest atom and the previous atoms:

$$\begin{array}{c} \beta_3 \\ \beta_2 \\ \beta_1 \\ \text{green circle} \wedge \text{green circle} \wedge \text{green circle} \dots \rightarrow \text{red circle} \end{array} \quad (7)$$

where β_1 is empty as it contains only 1 atom.

So the conditions of a rule are re-structured as a chain as follows:



Now that all the rules are converted into such chains as (8), the Rete decision network can be constructed simply by identifying common nodes.

My presentation may be slightly different from those found on the internet, but I guess the main idea is the same.

2 Learning inside Rete

The Rete network completely replaces the set of rules, so we no longer need the latter. In AI applications, the rules don't even exist initially. This means that we could learn the Rete network directly. What is the advantage, you may ask?

2.1 Relation to deep learning

The standard **neural network**:

weights
matrix

total # layers

$$F(\vec{x}) = \bigcirc(W_1 \bigcirc(W_2 \dots \bigcirc(W_L \vec{x}))) \tag{9}$$

is just a composition of layers of differentiable functions. This allows the use of the **chain rule** to compute derivatives, which is what **TensorFlow** is all about. The power of deep learning comes from the hierarchical composition of layers.

Now Rete is also a network (the latin root *rete* means “web” or “net”). If we could make its nodes *differentiable*, perhaps we could turn Rete into some form of deep learning?