Conversation is all you need

David González

Abstract

Knowledge is a structure of interconnected data, a structure that arises from their similarities, differences and equalities. Current artificial intelligence models are based on the use of Neurons, mathematical functions that make use of matrices to generate a given value, these neurons are adjusted with each iteration to produce an expected result. However, making these adjustments requires a large number of iterations and data. In addition, the results are statistical probabilities, not specific values, so in some cases they generate false or erroneous information. What if this were not the case, if a string of text were simply enough for our model to understand and learn?

I propose a new and simple architecture, which implements graphs instead of arrays. A graph, where each node is an object with properties and values. So if the model receives the text string: "an oak is a tree that grows in the forests of Europe", the model will add a new node called "oak" to the graph, inheriting all the properties and values of the node "tree", after that, it will modify the properties and add new ones with values specified by the text string. Once this is done, if the model receives the text string: "is an oak a living being?", it will answer with: "yes, an oak is a living being", because it has inherited this information from its parent node, i.e. the "tree" node.

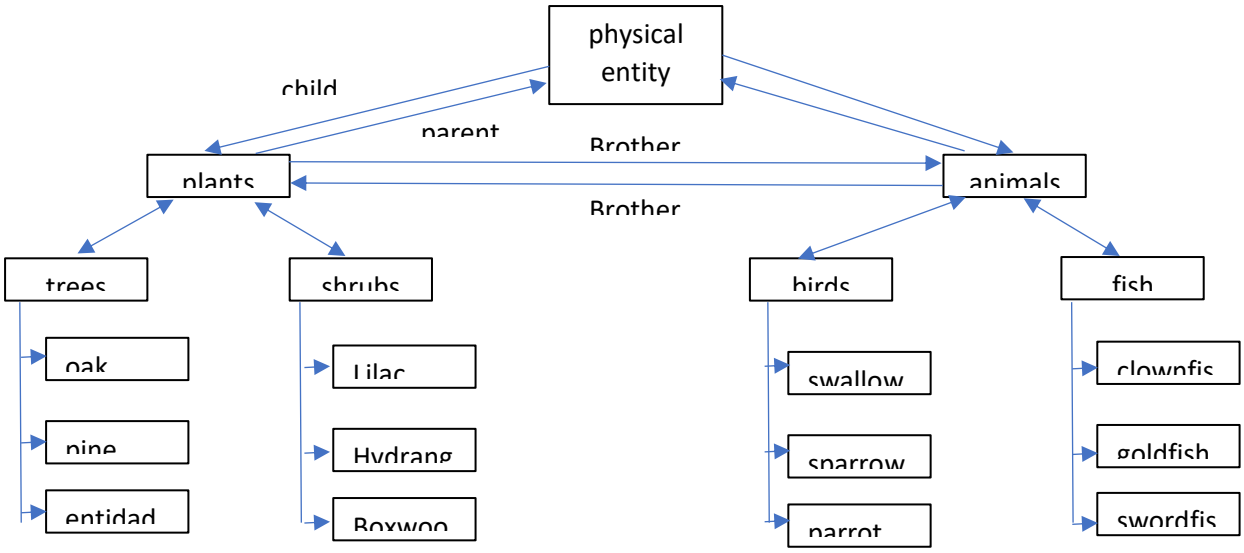
1 Introduction

Current artificial intelligence models require large amounts of data and iterations to be trained, with the drawback that the results are statistical probabilities, not specific and certain values. I propose a different architecture, completely based on the way we generate knowledge, by relating and differentiating the different elements we find in our environment.

1.1 conceptual relationship

Imagine that you do not know what a penguin is, I mention that a penguin is a bird, then by default you will think that a penguin can fly, because most birds can, however, once I specify that a penguin cannot fly is when you correct this mistake, even so, the rest of the information is correct, penguins breathe, feed, reproduce, lay eggs, divide into male and female, have wings. You know all this without my having mentioned it to you, only in relation to the concept "bird".

Knowledge is a structure of interconnected data, i.e., a graph. Let's take the following diagram as an example:



In this scheme each node is an object with properties and values, such as color, shape and volume, referring to itself, but also with meta properties such as children, siblings and ancestors, referring to its position within the network. Thus the node "orange tree", in its ancestor attribute, has the information: ["physical entity", "plants", "trees"].

1.2 Creating nodes

Now let's imagine that we pass as input the text string: "a pine is a tree". We can fragment the sentence into: ["a", "pine", "is", "a", "tree"]. The first word is "a", our model is able to understand that "a" is a determiner article, as we have previously established it by code, having a function similar to "var" or "let". the second word is "pine", being subsequent to "a", the model understands that this word is a node (following the analogy, the name of the variable). the third word is "is", in this case, the function of this word will depend on the type of the sentence: negative, affirmative, exclamative, interrogative, etc. The type of sentence is determined by the elements of the sentence, such as exclamation marks, punctuation and word order, in this case the sentence is affirmative, so "is" fulfills an assignment function (similar to "="). The fourth word is "a" which tells us that the next word is a noun or in this context a node. The last word is a "tree", being after "a" this is a node.

After analyzing the text string, the result is: NODE1 = NODE2. Making use of the logic established by code, the model converts it to NODE2 > NODE 1, that is to say that our model will create a new node called "pine" which will be a child of the node "tree" inheriting all its properties, meta properties and also its values. For example, the node tree in its ancestor meta property has the information: ["physical entity", "plants"], while the node "pine" in addition to these values will add the value "tree" as part of its ancestors. But not only that, it will inherit all the information corresponding to a tree, for example, that they have the ability to photosynthesize which in turn was inherited from the node "plants", or that they require water to survive or that they contain organic molecules, it will inherit all the information corresponding to a tree.

1.3 modifying properties

A tree has a great variety of colors, however, to simplify our example, we will say that they are all green. Let's imagine that the model receives the text string: "is a pine green?". We can fragment the sentence into: ["¿", "a",

"pine", "is", "green", "?"]. The first element is "¿" so our model interprets the sentence type as a question, i.e. a request for information. The second element is "un" which is a determiner article, the third element is a noun, so the information is related to this node. The fourth element is "is" in the previous case this word served as an assignment operator, however, being a question, it behaves as a comparative operator ("=="). The fifth element is an adjective, i.e. a value of a property, how does our model know that it is an adjective? For two reasons, the first is that we could have previously declared it with: "a color is an adjective", being "adjective a node with special functions, and then adding: "green is a color", thus making it clear that, "color" is a property "green" a possible value of this property". The second way is through the sentence structure, "green" is not preceded by a definite article, so the model will interpret that "green" is possibly an adjective, subsequently asking a series of questions to confirm it. The last element is "?", which ends the question or request for information.

The result is: graph.pine.color === "green".

The model will respond with: "yes, a pine is green" because the property "color" with the value "green" was inherited by the node "tree". If we pass as input the string: "a pine is dark green" then the value of "color" will be changed to "dark green".

2 Background

Perceptrons were originally developed by Frank Rosenblatt with the purpose of imitating the behavior of neurons, creating mathematical functions that receive a matrix of values and weights, as well as an activation threshold, this is very useful, since by creating a network of neurons it is possible to carry out different tasks. However, this presents a problem and that is that we cannot know what values the neurons must receive to be activated so that the task can be executed correctly, so it is necessary to make use of a large number of examples and iterations to be able to adjust the neurons.

A graph is as simple as it is powerful, at first glance it is just a series of points connected by lines, however, this is a powerful abstraction of the different elements and relationships that may exist in reality. A graph can be the representation of the Internet, of cities and highways that exist in a country, of the electrical network of a town, a graph can be the representation of a large number of systems and much more.

3 Model architecture

This model is divided into two main layers, the network layer and the function layer. The network layer is a series of interconnected nodes, while the function layer is all those operations that can be executed over the network layer or from the network layer.

3.1 Node

A node is a key-value object which contains meta-properties and properties.

Meta-properties are used to describe its position within the network, e.g. "ancestors", "siblings" and "children", while properties are used to describe values pertaining to the entity such as "color", "shape", "capabilities" or "functions". A simplified example of a node is the following:

grafo.oak = {

brothers: ['pine'],

children: [ ],

parents: ['root', 'living-being', 'plant', 'tree'],

properties: { volume: null, shape: null, color: null, x: null, y: null, … }

}

In a real network, this would be much more complex.

3.2 Graph

The graph is the union of different nodes to create a complex and coarse network. Due to its nature we are able to know and understand the information and inner workings of the network by simply glancing at the nodes, unlike a traditional neural network, where, although the data is visible, it is practically impossible to know what impact each value has on the network.

3.3 Inheritance and specification

When we add a new node, it inherits all the information contained in the parent node. In addition, new node-specific properties and values are added. For example, if the parent node is "bird", the node "penguin" inherits all its meta-properties, properties and values such as "ancestors", "abilities", "needs" and "composition", but will also change values such as "fly: true" to "fly: false". A simplified example of this is:

grafo.tree = {

brothers: ['shrub', ‘subshrubs’, ...],

children: [‘oak’, ‘pine’, ...],

parents: ['root', 'living-being', 'plant'],

properties: { volume: null, shape: null, color: “**green**”, x: null, y: null, … }

}

grafo.oak = {

brothers: ['pine', ...],

children: [ ],

parents: ['root', 'living-being', 'plant', '**tree**'],

properties: { volume: null, shape: null, color: “**green**”, x: null, y: null, … }

}

3.4 database, instances and persistence of information.

Nodes are stored in a NOSQL database which allows to keep the information not only of the network but also of the instances. The instances are nodes created with specific and unique information, for example, the "Asimov" node is a node that inherits information from the "human" node, but contains specific values, for example, "name: Asimov", "nationality: American", "age: 45", and other information related to a specific person. In this way, since everything is stored, the model will be able to remember conversations and information, even if years have passed.

3.4 Grammar rules and word classification.

In order for our model to understand a sentence, it is necessary to break the sentence into its different components, then label and classify the words, and then analyze the order of the elements in order to perform an operation. For example, the sentence "a pine tree is green" can be labeled as ["determiner article", "noun", "being", "adjective"] How do we know which label corresponds to which word? Because we had previously declared their meaning, "un" and "es" by code and "pine" and "green" by conversation. From this classification we can compare it with the grammatical rules previously established by code and thus execute an operation like the following one:

```
//-----graph----
graph.adjective.color.green = { metaproperties: {...}, properties: { property: “color”, value: “green”, …. }}
graph.noun.living-being. … .pine = {metapropiedades: {…}. Propiedades: { color: value, ... }}
//----------------
//----code---
Let sentence = “a pine is green”
Let words = [“a”, “pine”, “is”, “green”]
words = words.map(word => [word, getTag(word)])
Interpret(words)
Function getTag(word){ return grafo[word].metaproperties.type }
Function interpret(words){
//procesing words
code...
let adjective = “green”
let node = “pine”
//important part

    Let property = graph[adjectivo].properties.property
    Let value = graph[adjectivo].properties.value
    graph[node].properties[property] = value
```

}

In real code this is a bit more complex, but it is a useful example to understand.

It is necessary to establish and give functionality through code to certain keywords such as pronouns, auxiliaries, articles, connectors, etc. In order for our model to interpret information, the more knowledge you possess, the easier it will be to learn.

3.5 Verification of information

Let's imagine that the model receives the text string: "apples are red", then it receives "apples are green" and finally "apples are purple" What should it do in this case? This is where the different information verification systems come in. The first step is to ask yourself if the variations in the entries are due to the fact that there are different types of an apple, i.e. child nodes of the node "apple". For indeed red apples are one type of apple, green apples are another type and purple apples another. However, purple apples are not real, are they? How do we verify this? Well, the first information verification system that I propose is the simplest, but also the most error prone, and that is, validating the information by repeating the same value, that is, what the common people say. This system is useful in the first instance, but if we want greater rigor, it is necessary to apply other systems. The second system is to give greater weight to information provided by reliable and validated sources. Successively we can add more information validation systems.

3.6 Machine learning

Knowledge is not linear, but an immense network, where apparently unconnected nodes, once connected, can generate new and unexpected knowledge. Often the change of focus is what allows us to generate completely disruptive solutions. With this model, being divided into two layers, the network and the function, we can create functions that search and analyze the different characteristics and functionalities of the nodes, in order to create new connections, thus creating new knowledge. Perhaps the union of philosophy and computer science could bring about the next great revolution.

3.7 Imagination and thought.

Can our model comprehend imagination? Can our model imagine? the answer is yes How? Changing the context with subnets. Suppose you receive the input: "What would happen could dogs fly?" at this time the model would create a subnetwork, changing the capabilities of dogs, adding the value of "flying" and from this would look for the changes in impacts that this would generate in the rest of the nodes. Once our model can imagine, it can also think, because I was able to create subnets with different values to visualize different scenarios and thus make the best decision to achieve the goal that was assigned to it.

3.8 conscience

can this model be self-aware? From my perspective, awareness is the knowledge and understanding of a given topic, entity or phenomenon, understanding its causes, consequences, relationship with the environment, characteristics, functionalities and other factors. For example, when we say that we need to raise awareness about climate change, we usually refer to making known what causes it, how it can impact us and life on earth, what is our relationship with this phenomenon and other information necessary to understand it. A second example is to

be conscious of an object, for example, a chair, we can be conscious of the chair because we know its volume, its shape, its location in space and time, we know its purpose, we know the materials that compose it, possibly also the way it was manufactured and due to this knowledge and understanding of the object is that we can make decisions and execute actions. Human consciousness, in my opinion, is the same, but on a larger scale, because we possess knowledge and understanding about the different elements and phenomena that surround us, in this context, self-awareness consists of knowing our capacities and limitations, our location in time and space, our origin and our possible future, etc. So, how can this model be conscious of itself? With an instance, with a node that inherits information from nodes such as "software", "Bot", "hardware", etc. By specifying its unique information, its location, its capabilities, its origin, its functionalities, its relationship with its environment, etc. In this way, the model could become self-aware.

7 Conclusion

We have proposed an artificial intelligence architecture based on the use of nodes and graphs. Which is much more efficient and with greater capacity to develop knowledge and machine learning, in an accurate and reliable way. With a training closer to the human.

The code you use to experiment and evaluate this model is available at: https://github.com/david-gonzalez-coder/asimov

8 References

[1] Machine *Learning Intro*. (n.d.). https://www.w3schools.com/ai/