AOgmaNeo Guide

Eric Laukien
Ogma Corp

April 2021

1 Introduction

1.1 What is AOgmaNeo?

AOgmaNeo is a biologically-inspired online machine learning system. It differentiates itself from the more popular Deep Learning methods (DL) through a very low compute footprint and the ability to learn online without forgetting.

The underlying theory behind AOgmaNeo is called SPH (Sparse Predictive Hierarchies).

This guide will not go into too much detail on how each of the components of AOgmaNeo actually work, but rather just describe them enough to know what is going on. Hopefully this will be sufficient for your own AOgmaNeo project.

1.2 When should I use AOgmaNeo?

AOgmaNeo can be employed on any task that involves predicting a stream of data points one point ahead of time. Given sample X_t, AOgmaNeo predicts X_{t+1}. As a result, it learns to model a stream as a time series. AOgmaNeo is equipped with short-term memory and optional reinforcement learning systems (as reinforcement learning tasks can by modeled as a stream of action-predictions).

2 Basic Concepts

2.1 The CSDR

A CSDR (Columnar Sparse Distributed Representation) is essentially the data format of AOgmaNeo. All systems in AOgmaNeo communicate with CSDRs. So what is a CSDR?

CSDRs are seen by the user as simply a list/array of integers. Each integer represents a *column* (named so due to its connections to the concept of the same name from neuroscience). A column is a vertical stack of cells (neurons). The integer indexes the currently active cell in the column (only one can be active

in the column at a time). A column is therefore referred to as "one-hot". The size of the columns in the list are all the same, and given by the user when initializing the system.

While the CSDR is stored as simply a list/array of integers, AOgmaNeo will actually interpret it as a 3D structure. CSDRs in AOgmaNeo are typically 2D grids of columns (so 3D overall). A column is addressed with standard row-major matrix ordering:

$$index_{array} = col_y + col_x * size_y$$

And in reverse:

$$col_x = index_{array} / size_y$$
$$col_y = index_{array} \pmod{size_y}$$

(The division is integer division, discarding remainder)

This may seem rather complicated, but it's actually the very similar to a grayscale image (represented by pixels) as used in most image processing libraries. It is a 2D array of integers, represented in a single 1D array of integers. The difference is that the "pixels" are not pixels but column indices.

Here is an image of a CSDR to aid in understanding it:

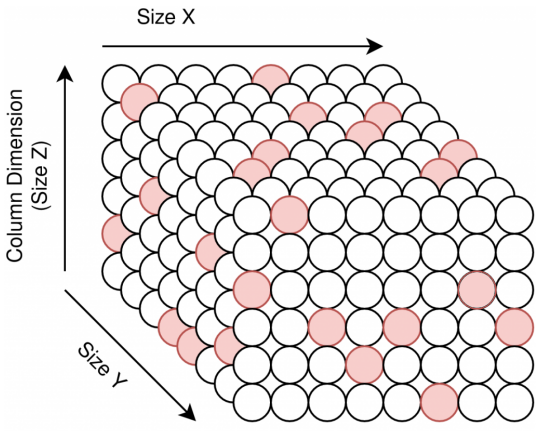

Figure 1: A CSDR

Note that along the vertical (column) dimension, exactly 1 cell is active (colored red).

2.2 The Hierarchy

An AOgmaNeo system consists of a hierarchy (stack) of layers, where each layer contains at least one *encoder* and *decoder*. Both input and output happen at the *bottom* of the hierarchy, through several IO "ports" (also called IO layers). Each layer's *hidden representation* is the CSDR of that layer's encoder. These can be retrieved at any time to monitor the activity of the hierarchy.

The input/output ports accept CSDRs, and produce CSDRs (or nothing, depending on the configuration). When a CSDR is produced, it will either be a *prediction* (for standard use) or an *action* (for reinforcement learning). The predictions are next-timestep $t + 1$ predictions of the input CSDR given to the same port.

AOgmaNeo hierarchies are bidirectional: There is an "up" pass, where information is encoded using the encoders, then a "down" pass where predictions are made with the decoders. These two passes occur once every simulation tick of the system.

Below is a diagram of a hierarchy with 3 layers.

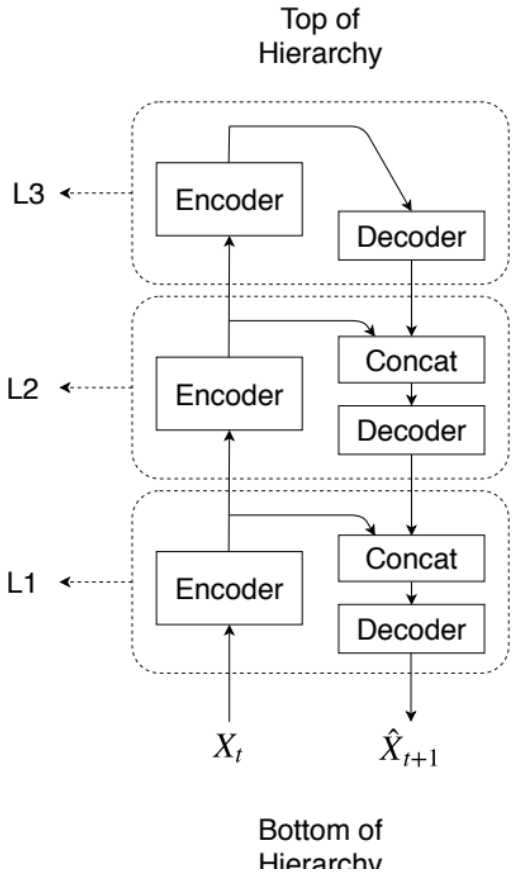

Figure 2: A Hierarchy

2.3 Encoders

AOgmaNeo currently uses a generative Exponential Sparse Reconstruction (ESR) encoder. It is a simple, well-rounded encoder that works for most tasks. It gets its name from the way it is updated - by reconstructing the "visible" portion from the "hidden" portion, passing it through an exponential activation function, and then learning the weights based on the error between that reconstruction and the input. There are many important implementation details, but one of particular importance is to not perform learning if the reconstruction is correct (after inhibition). This is a type of early stopping.

2.4 Decoders

AOgmaNeo uses a single type of decoder, which is essentially just an online learning variant of logistic regression. It takes the encoder representation as

input, as well as feedback from higher layers, which are all nonlinear and discontinuous.

2.5 Actor

An actor is similar to a decoder, and is used for reinforcement learning. Like a decoder, it produces a prediction of the next timestep of input. However, these predictions are modified in order to maximize the reward signal provided to the hierarchy.

2.6 Exponential Memory

AOgmaNeo hierarchies use a concept called "Exponential Memory" to handle short-term memory (working memory). Exponential Memory (EM, a type of Clockwork RNN) works by having each layer in the hierarchy "clock" at some multiplier slower than the previous lower layer. This multiplier is typically 2, but can be configured. Each layer covers a small temporal window of information of the layer below it, but since each layer clocks/ticks slower than the layer below it, its temporal "receptive field" increases as one ascends the hierarchy. In the case where the multiplier is 2, each layer takes 2 timesteps of information from the layer below it, and produces 2 predictions of the state of the layer below it. With a multiplier of 2, the memory capacity of the system is 2^L, where L is the number of layers. This is why it is called "exponential memory" - the final memory horizon can become very large after only a few layers.

Below is an image to aid in the understanding of exponential memory. Note that there are only 3 layers, they are shown over time. The red slice is the hierarchy at the current time, blue connections are connections to the "past", and green connections are predictions of the future.

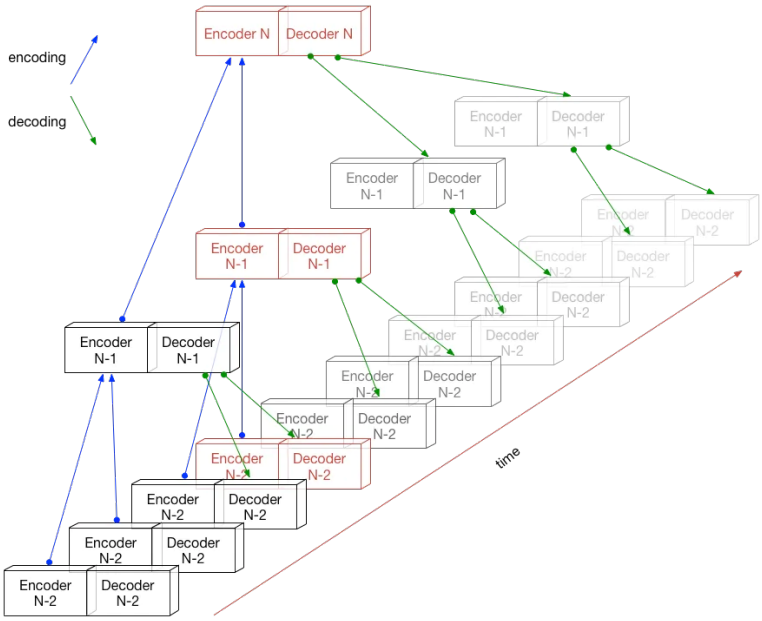

Figure 3: Exponential Memory

An interesting feature of exponential memory is that amount of computation needed for each additional layer decreases as one adds more layers. In the case of a multiplier of 2, each layer clocks/ticks half as often as the previous, so it only needs to be calculated half as often. In this case we theoretically do not need to exceed 2x the cost of the first (bottom-most) layer, since the series $1 + 1/2 + 1/4 + 1/8...$ converges to 2.

2.7 Online Learning

In AOgmaNeo, information is received in a stream, in temporal order. This is in contrast to most Deep Learning systems, where information is typically fed to the system i.i.d. (independent identically distributed). This combined with AOgmaNeo's capability to learn without forgetting means that AOgmaNeo is *online*, it learns from a sequence by seeing each sample once, in order.

In order to perform online learning, AOgmaNeo leverages its CSDR representation. CSDRs are a sparse representation - the 3D block of cells it represents are mostly 0's (cell off), with a few active 1 cells (the sparsity being defined by the column size).

Sparse representations are extremely beneficial for online learning. They may even be the only way of performing online learning. A sparse representation means that only parts of the network are updated at a time, and that the representations can easily be mostly *orthogonal*, meaning they do not overlap

by much. These two properties permit online learning, amongst several other benefits.

2.8 Performance

If you have used AOgmaNeo, you will likely note that it is fast, especially given how many parameters (synapses) it often uses. The speed is mostly due to the CSDR representation again, but also due to the use of online learning and exponential memory.

CSDRs are sparse, and as a result we don't have to compute the 0 entries. The 0 entries are not even looped through - we know which entries are 1 (and important) due to the column indices. We therefore never need to "check if a value is 0" - instead we just ignore most of the network at any time. However, all of the network is still used eventually, over time. Increasing the sparsity level (by increasing column sizes) will not greatly increase the amount of compute needed.

Aside from the sparsity (which is the main factor), AOgmaNeo also is performant due to its online learning (which is due to the CSDRs in large part, but we will consider it a separate property at the moment). Online learning means we do not need to revisit samples - we only need to see them once. In Deep Learning, typically data is revisited all the time, otherwise the system forgets. i.i.d. sampling and forgetting slow down Deep Learning a lot - in AOgmaNeo we do not have this problem.

Finally, exponential memory also speeds things up. Unlike other short-term memory systems such as recurrent connections, exponential memory requires less and less additional compute the larger the "remembering horizon" is, since we don't clock/tick the higher layers as often.

Due to the performance of AOgmaNeo, it can be readily used on small devices such as Raspberry Pis or even microcontrollers.

2.9 Receptive Fields

Connectivity between layers in AOgmaNeo (and also input/output ports/layers) is local - this means that a column only sees other columns near it. The area of all the other columns a column can "see" is called its receptive field. In AOgmaNeo, these are square, and their radius is set by the user. Larger receptive fields allow information to bridge larger gaps, but are more costly in terms of computation.

Given a receptive field radius of rad_{rf}, the diameter and areas of the receptive field are:

$$diam_{rf} = rad_{rf} * 2 + 1$$

$$area_{rf} = diam_{rf}^2$$

Below is an image of a column projecting its receptive field onto a layer layer. It has a radius of 1. The column and its receptive field are marked in green.

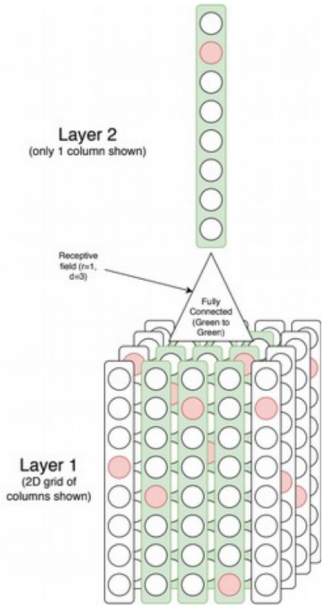

Figure 4: Receptive Field

2.10 Pre-Encoders and Pre-decoders

You may be wondering how data gets converted to a CSDR in order to be fed to the hierarchy (as it only accepts the CSDR format). This is done through the use of pre-encoders (which convert from your data to a CSDR) and pre-decoders (which convert from a CSDR to your data format again). These are application-specific, and can really be anything that is able to convert to/from a list of bounded integers (a CSDR).

By default AOgmaNeo includes one pre-made pre-encoder/pre-decoder, called the ImageEncoder (it contains the pre-decoder in it as well). It is used to map between CSDRs and images.

A general tip when designing your own pre-encoder/pre-decoder: Try to make it locally sensitive. This means that if the input data changes "a little bit", the output CSDR only changes a few of its columns. If the input data changes "a lot", then most or all of the output CSDR's column indices should change. This local sensitivity is important for AOgmaNeo to be able to quickly model and generalize your data.