Versatile predictive estimator without weight copying

David Xu, Cameron Seth, Jeff Orchard

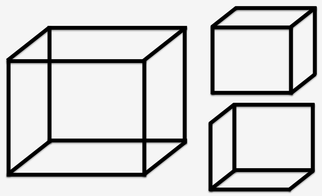
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

In predictive coding, cortical circuits called predictive estimators (PEs) participate in a hierarchy, passing predictions to lower layers, which send back the prediction error. These PE units learn a set of connection weights that translate between the layers. However, the standard implementation is not biologically plausible, since it uses the same weight matrix for both feed-forward and feed-back projections. We propose a more general PE unit, called a *symmetric predictive estimator* (*SPE*) that can learn more complex, non-linear transformations in a biologically plausible manner. Our connection weights are adjusted using an error signal that is local to the connections themselves, and there is no need for connection-weight copying. Error is evaluated from both feed-forward and feed-back connections, making the unit symmetric. Our new architecture is demonstrated by learning a Cartesian-to-Polar transformation, and exhibits non-classical receptive fields akin to end-stopping.

Motivation

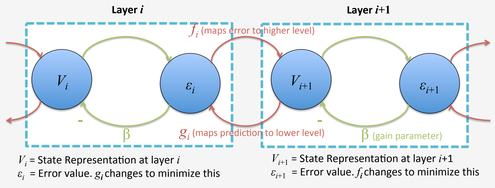
The Necker cube is an optical illusion that provides ambiguous visual input [3]. What the eye sees as a set of connected lines, the mind interprets as a cube. This cube can be perceived in two different orientations, as shown on the right.



Perception is an active process, looking for a solution linking input and expectation. For example, each perspective of the cube could be represented by a stable state in a dynamic network. Some perceptual feedback networks have been constructed of predictive estimators.

Predictive Estimators

Predictive coding posits that the cortex consists of many predictive estimator (PE) circuits arranged in a hierarchy; they send predictions down the hierarchy, and receive errors back [1, 2].



Previous models of predictive estimators have two serious shortcomings:

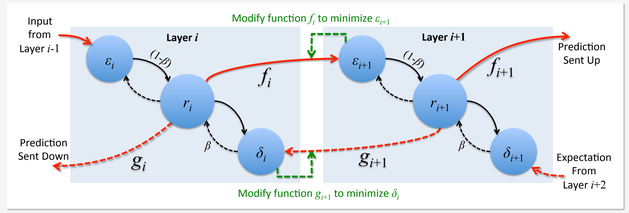
1. they are not biologically realistic because they tend to copy the weight matrix for forward and backward projections, and
2. the set of non-linear transformations between layers is limited because many transformations cannot be made stable with the PE architecture.

SPE Model

Each symmetric predictive-estimator (SPE) unit consists of 3 nodes:

- r the percept
- ε the error between the bottom-up signal and the percept
- δ the error between the top-down expectation and the percept

If the percept r matches the input from below and above, then ε and δ will be zero.



Note that the r, ε, and δ can be vectors, with dimensions indexed using square brackets. Their behaviour is governed by the differential equations

$$\tau \frac{dr_i}{dt} = (1-\beta)\varepsilon_i - \beta\delta_i$$
$$\tau \frac{d\varepsilon_i}{dt} = f_{i-1}(r_{i-1}) - r_i - \varepsilon_i$$
$$\tau \frac{d\delta_i}{dt} = r_i - g_{i+1}(r_{i+1}) - \delta_i$$

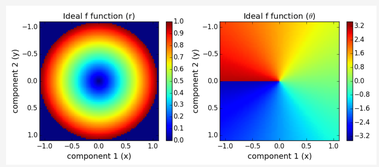
Learning rules:

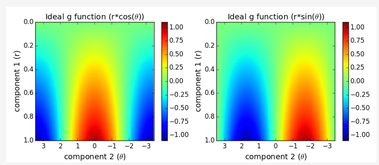
$$\gamma \frac{df_i^{(n)}}{dt} = -\kappa\, \varepsilon_{i+1}\, r_i^{(n)}$$
$$\gamma \frac{dg_i^{(n)}}{dt} = \kappa\, \delta_{i-1}\, r_i^{(n)}$$

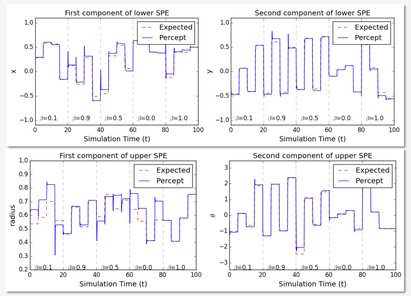
Cartesian/Polar Experiment

We simulated networks of SPE units in Python using Euler-stepping at 1ms time steps.

We generated a two-layer network and fed it randomly-chosen 2-D Cartesian co-ordinates (x,y) from the domain $[-1,1]^2$ as the lower-layer input, and corresponding polar coordinates (r,θ) [max radius of 1] as input from above. The f and g mappings were initially set to zero.







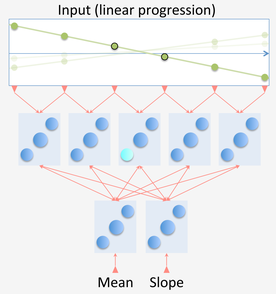
Using a simple delta learning rule, the f and g mappings were updated based on the values of ε and δ.

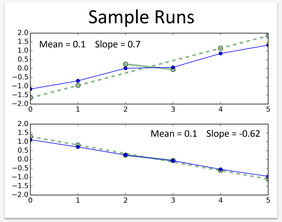
The above figures show that the percepts match the expectations in the lower and upper SPEs.

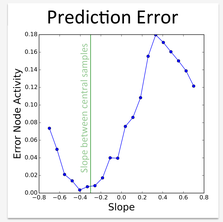
The figures on the left represent the learned f and g mappings. For the f mapping, the first component is the radius, and looks like a cone near the origin, while the second component is the angle, and correctly depicts a spiral ramp.

Linear progression experiment

The lower-layer input was a set of 6 numbers forming a linear progression. The upper-layer input was the corresponding mean and slope. After training, each layer sends predictions to the other layer. We held the two central points fixed, but varied the linear progression for the other points.



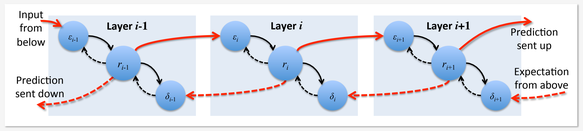




The δ-node highlighted in the network exhibits reduced activity when the lower-layer input matches the higher-layer expectation (slope=-0.3).

Future Work

After mastering 2-layer learning, we wanted to see if we could learn the mappings in a 3-layer network

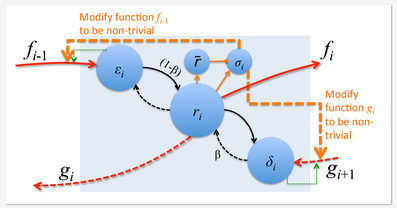


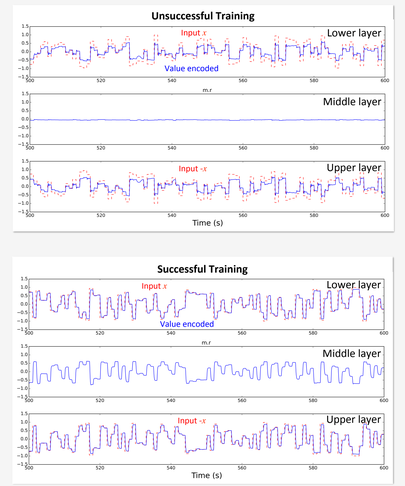
However, we found that the middle layer was under-constrained, which often lead to a disconnect. The plots on the right illustrate the middle layer's unhelpful representation.

We wanted the middle unit to show more variability so we added two nodes, $\overline{r}_i$ and σ_i to track the time-windowed variability of r_i.

We saw better results when we supplemented the learning rule with:

- ✓ destabilizing feedback when σ_i is low,
- ✓ smoothing, and
- ✓ a zero-mean requirement.





The activity in the middle layer facilitates the translation between the lower and upper layers.

[1] D. Mumford, "On the computational architecture of the neocortex - II The role of cortico-cortical loops", *Biological Cybernetics*, **66**(1987):241–251, 1992. [2] R. Rao and D. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects", *Nature Neurosci*, **2**(1):79-87, 1999. [3] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge, MA: MIT Press, 2003.