Why Aren't All Predictive Coding Networks Generative?

Jeff Orchard, Wei Sun, Neil Liu
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
{jorchard,w55sun}@uwaterloo.ca

Abstract

Predictive coding (PC) networks are bidirectional networks that have been shown to approximate backprop, and can be trained on discriminative tasks. But even though "predictive" is in the name, PC classifier networks have not been convincingly shown to generate input samples. Simply clamping the desired class vector and running the network to an equilibrium state does not generate a sensible state in the input layer. The problem is ill-posed; it has an infinite number of solutions. However, we show that a linear network can reconstruct sensible inputs by imposing a minimum-norm constraint, which can be done using simple weight decay. It also works on nonlinear networks, as demonstrated on MNIST.

1 Introduction

Predictive coding (PC) is a processing strategy hypothesized to take place in cortical networks [1, 2, 7, 9]. Inputs to the network are propagated up through a hierarchy of layers. Each layer only communicates with its adjacent layers. In this way, the network creates a progression of representations in its stacked layers, such that layers are linked together by a chain of feedback loops. The feedback connections in predictive coding (PC) enable the architecture to approximate backprop [3, 11].

But can the feedback connections enable the network to be generative? Classification PC networks have not been demonstrated to have very remarkable generative capabilities. Some attempts have been made, but with limited success. A convolutional PC network generated blurry versions of the input images [10] after allowing the network state to settle to an equilibrium. The paper really only demonstrated blurry deconvolution for one layer. In another study, a PC network that was trained as an autoencoder, not as a classifier, was able to generate blurry reconstructions of the input [4]. However, since the network was not a classifier network, it remains to be seen how a classifier network can be made generative.

An implementation of predictive coding was proposed and shown to approximate backpropagation using local Hebbian-like learning rules, achieving learning performance comparable to backprop [3, 11]. In this paper, we study the generative properties of their method.

2 Methods

A biologically plausible learning algorithm requires that changes in each synaptic weight depend only on the local activity of pre-synaptic and post-synaptic neurons. The architecture proposed in [11] solves the problem, describing a local learning rule that relies only on the activity of locally connected neurons.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

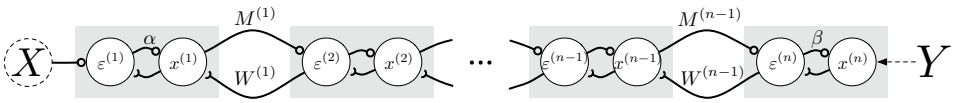

Figure 1: Architecture of the Whittington and Bogacz PC network [11]. Note that $\varepsilon^{(i)}$ and $x^{(i)}$ each represent the vector of activities of an array of nodes. Likewise for other layers.

Figure 1 shows a PC network. The state nodes and error nodes of layer i are updated according to,

$$\tau \frac{d\varepsilon^{(i)}}{dt} = x^{(i)} - M^{(i-1)}\sigma(x^{(i-1)}) - \nu^{(i)}\varepsilon^{(i)} \tag{1}$$

$$\tau \frac{dx^{(i)}}{dt} = W^{(i)}\varepsilon^{(i+1)} \odot \sigma'(x^{(i)}) - \varepsilon^{(i)} \tag{2}$$

where $x^{(i)}$ is the (vector) state of layer i, $\varepsilon^{(i)}$ is the corresponding (vector) error for layer i, $W^{(i)}$ is the backward weight matrix, $M^{(i-1)}$ is the forward weight matrix, $\nu^{(i)}$ is a scalar variance parameter, the $\odot$ operator represents the Hadamard (element-wise) product, $\sigma(\cdot)$ is the activation function, and τ is a time constant. Our method differs from that in [11] in that we do not include a bias in (1). Note that we are also using the common convention in which the predictions are sent *up* the network, in contrast to the original formulation of predictive coding [8].

For PC networks, the cost function, F, is proportional to the sum of the squares of the error nodes. See [3] or [11] for an explanation of how F is the negative log-likelihood of the input conditioned on the class (output). Gradient descent on F with respect to the connection weights, M and W, approximate the backprop algorithm, yielding the weight update dynamics,

$$\gamma \frac{dM^{(i)}}{dt} = -\varepsilon^{(i+1)} \otimes \sigma(x^{(i)}) \tag{3}$$

where $\otimes$ is the outer product. A similar rule exists for $W^{(i)}$. The time constant γ is larger than τ so that x and ε reach their equilibrium states much faster than M and W.

To allow the bottom and top inputs to be clamped or free, we have introduced the parameters α and β which simply modulate (multiply) their corresponding connections. The parameter α controls whether or not the input X has an influence on the bottom layer of the network. The parameter β controls whether the value in the top layer is influenced by the penultimate layer (otherwise it is constant).

Training: To train our PC network on a discriminative task, we simultaneously feed the input vector X (eg. a digit from the MNIST dataset) into layer 1, and set the output layer $x^{(n)} = Y$ (eg. the corresponding one-hot classification vector), as shown in Fig. 1. We set $\alpha = 1$ and $\beta = 0$ to clamp both inputs.

Simulating the network in continuous time while holding each input for a few simulation seconds causes the states of the nodes to converge to their corresponding equilibrium values rather quickly, thereby delivering the error gradients to the error nodes, where they are used to update the connection weights.

Discriminative Mode: When we test the discriminative capabilities of our network, we present the input X to the bottom layer, and set $\alpha = 1$ and $\beta = 1$, thereby enabling information to flow up the network. The resulting equilibrium value of $x^{(n)}$ is the network's output. It is fairly simple to show that all the error nodes will converge to zero in this case.

Our version of the PC network performs as well as that reported in [11], achieving 98% test accuracy on MNIST using two fully-connected hidden layers of 600 nodes each, after 10 training epochs.

Generative Mode: To run the network in generative mode, we set $x^{(n)}$ to the desired class vector (denoted Y in Fig. 1), and set $\beta = 0$ so that $x^{(n)}$ does not change. At the same time, we unclamp $x^{(1)}$ from by setting $\alpha = 0$.

Figure 2 shows the ten images generated by the network trained on MNIST. Unfortunately, the generated images do not look like MNIST digits. Why is that? To get a better understanding of what is happening, we will study a much simpler network.

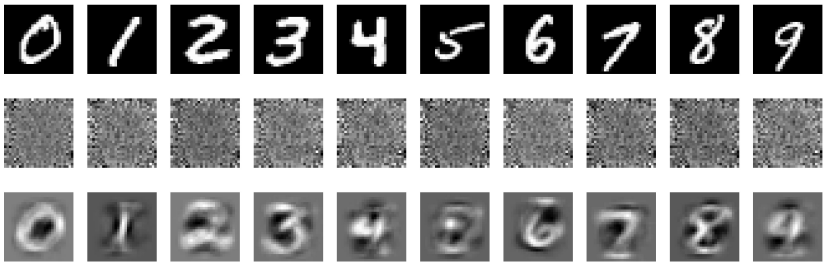

Figure 2: Images generated by the network trained on MNIST. The top row shows a sample of each digit class. The middle row shows the images generated by a trained PC network. These generated images do not resemble actual digits. The bottom row shows images generated by a PC network trained with weight decay and value-node decay.

Analysis of the Network: Consider a simple 2-layer discriminative network in which the input layer has m nodes, the output layer has n nodes, and the dataset has r different classes, with $r \leq n < m$. The forward weight matrix, $M^{(1)}$, has dimensions $n \times m$.

During *learning*, we clamp the bottom and top layers to their respective inputs, X and Y. Then, once the system reaches equilibrium (including $M^{(1)}$ and $W^{(1)}$), equation (1) gives

$$Y = M^{(1)}\sigma(X)\,. \tag{4}$$

After training, running the network in *discriminative* mode ($\alpha = 1$ and $\beta = 1$, which clamps only the bottom layer), it is easy to show that the equilibrium solution yields $\varepsilon^{(2)} = 0$, and $\varepsilon^{(1)} = 0$, and thus $x^{(1)} = X$, and hence

$$x^{(2)} = M^{(1)}\sigma(X)\,,$$

which then implies that $x^{(2)} = Y$, the desired target. In this way, the network has learned to solve the discriminative task; given the input X, the output matches the target Y. This works even with deeper networks, as demonstrated in [11].

Now consider the *generative* mode of the network. In that case, we set $x^{(2)} = Y$, and $\alpha = \beta = 0$. At equilibrium, one can show that,

$$M^{(1)}\sigma(x^{(1)}) = Y\,. \tag{5}$$

We know that $x^{(1)} = X$ is a solution, but is it the only solution? Even though the network quickly converges to an equilibrium, and its error nodes report very small values, usually $x^{(1)}$ is not very close to X.

To understand what is happening here, we will further simplify the problem.

Linear Network: Let us suppose, for the sake of simplicity, that $\sigma(x) \equiv x$. Thus, (5) becomes

$$M^{(1)}x^{(1)} = Y\,. \tag{6}$$

This system is under-determined, since $M^{(1)}$ is $n \times m$, with $n < m$. Thus, if $\bar{x}$ is a solution to (6), then so is $\bar{x} + c\hat{x}$ for any scalar c, as long as $\hat{x} \in \mathrm{null}(M^{(1)})$.

How can we get a unique solution? And can we hope to generate samples that are close to the vectors in the training dataset?

Theorem 1 *Given a matrix of r linearly-independent m-vectors, $X = [X_1|\cdots|X_r] \in \mathbb{R}^{m\times r}$, and a corresponding matrix of n-vectors, $Y = [Y_1|\cdots|Y_r] \in \mathbb{R}^{n\times r}$, with $r \leq n < m$, there is an $n \times m$ matrix, A, such that the minimum 2-norm solution x^* to $Ax = Y_i$ is $x^* = X_i$. Moreover, the jth row of A is the minimum 2-norm solution to $aX = Y$ for $a \in \mathbb{R}^{1\times m}$.*

(See the Appendix for a proof.) The theorem tells us that we can find unique solutions for the connection weights and network states that match our data, as long as we additionally seek their minimum-norm solutions.

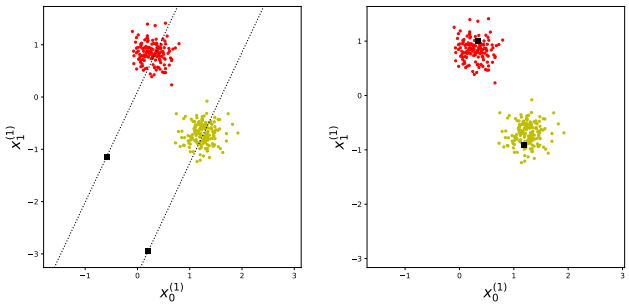

Figure 3: Generative output without decay (left), and with decay (right). The left plot also shows the corresponding solution space. Note that the plots depict a 2-D projection of a 3-D space. The generated samples fall within the data cluster when weight- and value-decay is used (right).

One way to implement the min-norm constraint is to add a term to the objective function that penalizes for the square of the L2 norm. The gradient of that penalty term turns into a linear decay term in the gradient. Hence, the min-norm constraint can easily be implemented in the PC network dynamics as an additional linear decay. For example, equation (2), governing the activity of $x^{(i)}$, will include a term of the form $-\lambda_x x^{(1)}$, and equation (3), governing the update of $M^{(i)}$, will include a term $-\lambda_M M^{(i)}$, where λ_x and λ_M are positive scalars. We use $\frac{-\lambda_M}{2} W^{(1)}$ for W's update rule.

3 Experiments

To demonstrate this effect, we created a dataset $\{X_n, Y_n\}$ with $X_n \in \mathbb{R}^3$ and $Y_n \in \mathbb{R}^2$. The samples in X were drawn from just two classes, each represented by a Gaussian distribution around a fixed centroid. The target for one class is $[1, 0]$, and $[0, 1]$ for the other class. The full dataset consists of 300 samples, with 150 samples of each class. A linear network was trained for 4 epochs, and achieved 100% training accuracy.

In generative mode, we set the class vector to either $[1, 0]$ or $[0, 1]$ and run the network to equilibrium with $\alpha = \beta = 0$. The values in x and ε are zero at the beginning of each simulation. Each class vector generates a different sample, which we will denote $\bar{x}_{[1,0]}$ and $\bar{x}_{[0,1]}$, respectively. A sample result is shown in Fig. 3(a). The generated samples (black squares in the Figure) do not fall within the clusters, even though the equilibrium network state yields very small values in the error nodes, $\varepsilon^{(2)}$ (around 10^{-6}). The solution spaces are shown as dotted lines; they pass through the generated points, as well as the cluster centres. Every point in the solution space is a solution to (6) and yields very small (or zero) errors.

However, introducing decay ($\lambda_x = \lambda_M = 0.05$) to x and M yields a network that generates the black squares in Fig. 3(b). The generated samples fall neatly inside the cluster of training samples.

We also tried training a 4-layer PC network (784-600-600-10) on MNIST using decay. The bottom row of Fig. 2 shows the generated samples for each of the 10 digit classes. Note that this was not a linear network, but used $\tanh$ activation functions.

4 Conclusions

Running a PC classifier network with its output class clamped tends not to generate meaningful input samples. This is likely because the generative problem is ill-posed. We analyzed a linear version of the problem, and confirmed that it is under-determined.

However, we showed that regularizing using decay on the network value nodes (x) and the forward connection weights (M) yield a unique solution for the linear network. We proposed (and proved) a theorem that guarantees that our solution will be similar to the training inputs.

While the theorem technically only applies to linear networks, adding the decay to our nonlinear ($\tanh$) PC network made it generate far more sensible sample MNIST images.

References

[1] A. Bartels. Visual Perception: Early Visual Cortex Fills in the Gaps. *Current Biology*, 24(13):R600–R602, 2014.

[2] A. M. Bastos, W. M. Usrey, R. A. Adams, G. R. Mangun, P. Fries, and K. J. Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

[3] R. Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017.

[4] S. Dora, C. Pennartz, and S. Bohte. A Deep Predictive Coding Network for Learning Latent Representations, 2018.

[5] G. H. Golub and C. F. Van Loan. *Matrix computations*. 1996.

[6] D. P. Kingma and L. J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, May 2015.

[7] D. Mumford. On the computational architecture of the neocortex - II The role of cortico-cortical loops. *Biological Cybernetics*, 66(1987):241–251, 1992.

[8] R. P. N. Rao and D. H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999.

[9] S. Shipp. Neural Elements for Predictive Coding. *Frontiers in Psychology*, 7:1792, 2016.

[10] H. Wen, K. Han, J. Shi, Y. Zhang, E. Culurciello, and Z. Liu. Deep Predictive Coding Network with Local Recurrent Processing for Object Recognition. In *Proc. International Conference on Machine Learning*, pages 5266–5275, 2018.

[11] J. C. R. Whittington and R. Bogacz. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural computation*, 29(5):1229–1262, 2017.

Appendix

Proof of Theorem 1: Consider the system $X^{\mathrm{T}}A^{\mathrm{T}} = Y^{\mathrm{T}}$, with r equations with m unknowns. Let the j row of A be denoted A_j, so that

$$A = \left[\begin{array}{c} A_1 \\ \hline \vdots \\ \hline A_n \end{array}\right].$$

Let y_j be the jth row of Y. Then $X^{\mathrm{T}}A_j^{\mathrm{T}} = y_j^{\mathrm{T}}$. This system is under-determined, since $r < m$. Thus, there are infinitely many solutions (since the columns of X are linearly independent). However, we can seek the minimum-norm solution for A_j^{T} using the SVD [5].

Let $U\Sigma V^{\mathrm{T}} = X^{\mathrm{T}}$, where U is an $r \times r$ orthogonal matrix, V^{T} is $r \times m$ with orthonormal rows, and Σ is a diagonal $r \times r$ matrix containing the r non-zero singular values. The minimum-norm solution of $X^{\mathrm{T}}A_j^{\mathrm{T}} = y_j^{\mathrm{T}}$ is

$$A_j^{\mathrm{T}} = V\Sigma^{-1}U^{\mathrm{T}}y_j^{\mathrm{T}}$$

We can construct all n columns of A^{T} using $A^{\mathrm{T}} = V\Sigma^{-1}U^{\mathrm{T}}Y^{\mathrm{T}}$.

Now we show that X is a solution of $AX = Y$. Substituting the above expression for A, as well as the SVD for X^{T}, we get

$$\begin{aligned} AX &= YU\Sigma^{-1}V^{\mathrm{T}}X \\ &= YU\Sigma^{-1}V^{\mathrm{T}}\left(V\Sigma U^{\mathrm{T}}\right) \\ &= YUU^{\mathrm{T}} \quad \text{since } V^{\mathrm{T}}V = I \text{ and } \Sigma^{-1}\Sigma = I \\ &= Y \quad \text{since } UU^{\mathrm{T}} = I \end{aligned}$$

Thus, X is a solution of $AX = Y$.

Now, we want to show that each column of X is the minimum-norm solution. Consider the ith column of X, and suppose we find a different solution, $X_i + \tilde{x}$, where $\tilde{x} \neq 0$. Then,

$$\begin{aligned} A\left(X_i + \tilde{x}\right) &= Y_i \\ AX_i + A\tilde{x} &= Y_i \\ Y_i + A\tilde{x} &= Y_i \\ A\tilde{x} &= 0 \end{aligned}$$

Thus, $\tilde{x} \in \text{null}(A)$, which tells us that $V^{\text{T}}\tilde{x} = 0$. But $X^{\text{T}} = U\Sigma V^{\text{T}}$, so $\tilde{x} \in \text{null}(X^{\text{T}})$ too. Thus, $X_i \perp \tilde{x}$.

Consider $\|X_i + \tilde{x}\|$. Since $X_i \perp \tilde{x}$, we can use Pythagoras, and conclude that

$$\begin{aligned} \|X_i + \tilde{x}\|^2 &= \|X_i\|^2 + \|\tilde{x}\|^2 \\ \|X_i + \tilde{x}\|^2 &> \|X_i\|^2 \ \text{ since } \tilde{x} \neq 0 \\ \implies \|X_i + \tilde{x}\| &> \|X_i\| \end{aligned}$$

Therefore, X_i is the minimum-norm solution to $Ax = Y_i$. ■