bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Learning in cortical networks through error back-propagation

James C.R. Whittington[a] and Rafal Bogacz[a,b]

[a]*MRC Unit for Brain Network Dynamics, University of Oxford, Mansfield Road, Oxford, OX1 3TH, UK*

[b]*Nuffield Department of Clinical Neurosciences, University of Oxford, John Radcliffe Hospital, Oxford, OX3 9DU, UK*

Short title: Error back-propagation in the cortex

Corresponding author: Rafal Bogacz

Office Tel: +44 (0)1865 231 903

E-mail: rafal.bogacz@ndcn.ox.ac.uk

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Abstract

To efficiently learn from feedback, the cortical networks need to update synaptic weights on multiple levels of cortical hierarchy. An effective and well-known algorithm for computing such changes in synaptic weights is the error back-propagation. It has been successfully used in both machine learning and modelling of the brain's cognitive functions. However, in the back-propagation algorithm, the change in synaptic weights is a complex function of weights and activities of neurons not directly connected with the synapse being modified. Hence it has not been known if it can be implemented in biological neural networks. Here we analyse relationships between the back-propagation algorithm and the predictive coding model of information processing in the cortex. We show that when the predictive coding model is used for supervised learning, it performs very similar computations to the back-propagation algorithm. Furthermore, for certain parameters, the weight change in the predictive coding model converges to that of the back-propagation algorithm. This suggests that it is possible for cortical networks with simple Hebbian synaptic plasticity to implement efficient learning algorithms in which synapses in areas on multiple levels of hierarchy are modified to minimize the error on the output.

Author Summary

When an animal learns from feedback, it should minimize its error, i.e., the difference between the desired and the produced behavioural output. Neuronal networks in the cortex are organized in multiple levels of hierarchy, and to best minimize such errors, the strength of synaptic connections should not only be modified for the neurons that directly produced the output, but also on other levels of cortical hierarchy. Theoretical work proposed an algorithm for such synaptic weight update known as error back-propagation. However, in this algorithm, the changes in weights are computed on the basis of activity of many neurons not directly connected with the

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

synapses being modified, hence it has not been known if such computation could be performed by biological networks of neurons. Here we show how the weight changes required in the back-propagation algorithm could be achieve in a model with realistic synaptic plasticity. Our results suggest how the cortical networks may learn efficiently from feedback.

Introduction

An efficient learning from feedback often requires changes in synaptic weights in many cortical areas. For example, when a child learns to recognize letters, after receiving a feedback from a parent, the synaptic weights need to be modified not only in speech areas, but also in visual areas. An effective algorithm for computing weight changes in network with hierarchical organization is the error back-propagation [1]. It allows network with multiple layers of neurons to learn desired associations between activity in the input and the output layers. Artificial neural networks (ANNs) employing back-propagation have been used extensively in machine learning [2, 3, 4], and have become particularly popular recently, with the newer deep networks having some spectacular results, now able to equal and outperform humans in many tasks [5, 6]. Furthermore, models employing the back-propagation algorithm have been successfully used to describe learning in the real brain during various cognitive tasks [7, 8, 9].

However, it has not been known if natural neural networks could employ an algorithm analogous to the back-propagation used in ANNs. In ANNs, the change in each synaptic weight during learning is calculated by a computer as a complex, global function of activities and weights of many neurons (often not connected with the synapse being modified). In the brain however, the network must perform its learning algorithm locally, on its own without external influence, and the change in each synaptic weight must depend just on the activity of pre-synaptic and post-

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

synaptic neurons. This led to a common view of the biological implausibility of this algorithm, e.g. "despite the apparent simplicity and elegance of the back-propagation learning rule, it seems quite implausible that something like equations [...] are computed in the cortex" (p. 162) [10].

Considering the success of models of the brain involving the back-propagation algorithm and that recent technological advances have allowed the capabilities of ANNs to progress at an extremely fast rate, it seems timely to examine if neural networks in the cortex could approximate the back-propagation algorithm.

Here we show how predictive coding, a well known model of information processing in cortical circuits [11, 12], can closely approximate the back-propagation algorithm. The predictive coding model was developed to describe unsupervised learning in the sensory cortex about the statistical structure of incoming stimuli. When trained on natural images the model learned features closely resembling the receptive fields of neurons in primary visual cortex [11]. Importantly, in this model, the neurons make computations only on the basis of inputs they receive from other neurons, and the changes in synaptic weights are based on Hebbian plasticity. The extended version of the model, known as the free-energy framework, explains how the network can learn about reliability of different sensory inputs [12, 13, 14, 15] and has been proposed to be a general framework for describing different computations in the brain [16, 17]. It has been discussed before that ANNs are related to predictive coding [18] and other unsupervised learning algorithms [19]. In this paper we show explicitly how the predictive coding model can be used for supervised learning of desired associations between inputs and outputs. We point out that for certain architectures and parameters, learning in the predictive coding model converges to the back-propagation algorithm. Furthermore, we characterize the performance of the predictive coding model in supervised learning for other architectures and parameters, and highlight that it allows learning bidirectional associations between inputs and outputs.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

	Back-propagation	Predictive coding
Activity of a node	$y_i^{(l)}$	$x_i^{(l)}$
Synaptic weight	$w_{i,j}^{(l)}$	$\theta_{i,j}^{(l)}$
Synaptic input	$v_i^{(l)}$	$u_i^{(l)}$
Utility function	E	F
Prediction error	$\delta_i^{(l)}$	$\varepsilon_i^{(l)}$
Activation function	f	
Number of neurons in a layer	$n^{(l)}$	
Highest index of a layer	l_{max}	
Input from the training set	s_i^{in}	
Output from the training set	s_i^{out}	

Table 1: Corresponding and common symbols used in describing ANNs and predictive coding models.

In the next section we review back-propagation in ANNs and predictive coding models. In the Results section we show how a predictive coding model can be used in a supervised setting, and how the resulting weight update rules compare to conventional back-propagation. We illustrate this comparison with simulations. Finally, we discuss experimental predictions of the predictive coding model in the supervised learning setting.

Models

While we review ANNs and predictive coding below, we use a slightly different notation than in their original description to highlight the correspondence between the variables in the two models. The notation will be introduced in detail as the models are described, but for reference it is summarized in Table 1. To make the dimensionality of variables explicit, we denote single numbers in italics (e.g. A), vectors with a bar ($\bar{A}$) and matrices in bold ($\mathbf{A}$).

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Back-propagation algorithm

ANNs [1] have been developed for supervised learning, i.e. they learn to produce a desired output for a given input. The network is configured in layers, with multiple neuron-like nodes in each layer as illustrated in Figure 1A. We denote by $y_i^{(l)}$ the activity of i^{th} node in the l^{th} layer. To make link with predictive coding more visible, we change the direction in which layers are numbered, and index output layer by 0 and input layer by l_{max}.

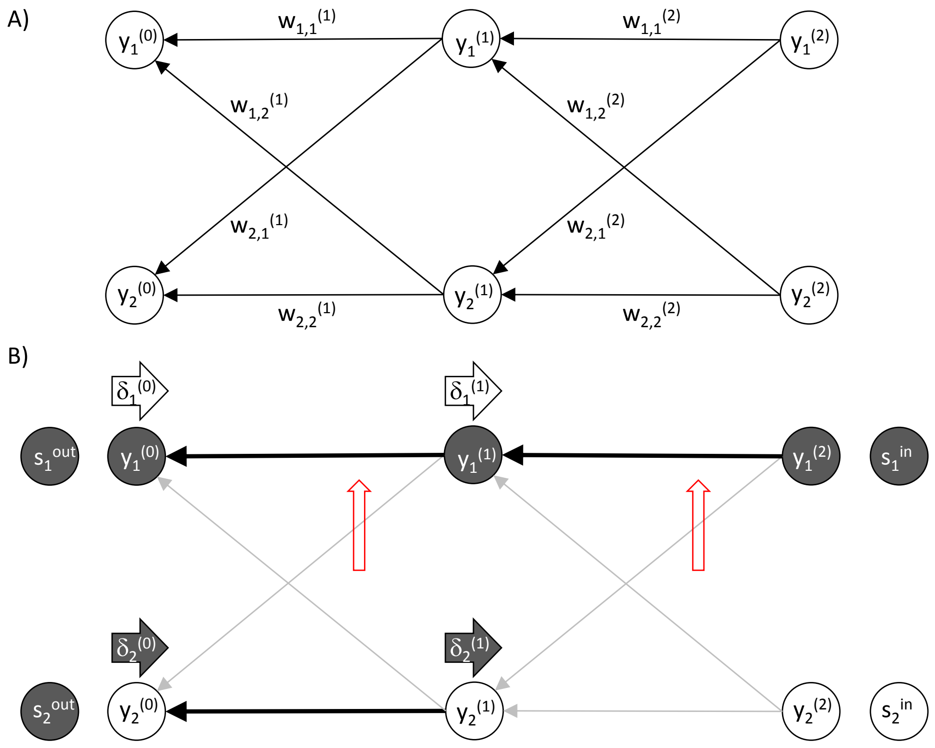

Figure 1: Back-propagation algorithm. A) Architecture of an ANN. Circles denote nodes and arrows denote connections. B) An example of activity and weight changes in an ANN. Thick black arrows between the nodes denote connections with high weights, while thin grey arrows denote the connections with low weights. Filled and open circles denote nodes with higher and lower activity, respectively. Rightward pointing arrows labelled $\delta_i^{(l)}$ denote error terms and their darkness indicates how large the errors are. Upward pointing red arrows indicate the weights that would most increase according to the back-propagation algorithm.

Each node receives an input $v_i^{(l)}$ which is equal to the weighted sum of the activities of nodes in the previous layer:

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$$v_i^{(l)} = \sum_{i=1}^{n^{(l+1)}} w_{i,j}^{(l+1)} y_j^{(l+1)} \tag{1}$$

where $w_{i,j}^{(l)}$ is the weight from the j^{th} node in the l^{th} layer to the i^{th} node in the $(l-1)^{th}$ layer, and $n^{(l)}$ is the number of nodes in layer l. Each node applies an activation function f to its input, and so each node's activity is computed as:

$$y_i^{(l)} = f\left(v_i^{(l)}\right) \tag{2}$$

The output the network produces for a given input depends on the values of weight parameters. This can be illustrated in an example of ANN shown in Figure 1B: The output node $y_1^{(0)}$ has a high activity as it receives an input from the active input node $y_1^{(2)}$ via strong connections. By contrast, for the other output node $y_2^{(0)}$ there is no path leading to it from the active input node via strong connections, so its activity is low.

The weight values are found during the following training procedure. At the start of each iteration, the activities in the input layer $y_i^{(l_{max})}$ are set to values from input training sample, which we denote by s_i^{in}. The network first makes a prediction, i.e. the activities of nodes are updated layer by layer according to Equation 2. The predicted output in the last layer $y_i^{(0)}$ is then compared to the output training sample s_i^{out}. We wish to minimize the difference between the actual and desired output, so we define the following utility function:

$$E = -\frac{1}{2} \sum_{i=1}^{n^{(0)}} \left(s_i^{out} - y_i^{(0)}\right)^2 \tag{3}$$

The training set contains many pairs of training vectors $(\overline{s}^{in}, \overline{s}^{out})$, which are iteratively presented to the network, but for simplicity of notation we will consider just changes in weights after presentation of a single training pair. We wish to modify the weights to maximize the utility function, so we update the weights proportionally to the gradient of the utility function:

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$$\Delta w_{b,c}^{(a)} = \alpha \frac{\partial E}{\partial w_{b,c}^{(a)}} \tag{4}$$

where α is a parameter describing the learning rate.

Since weight $w_{b,c}^{(a)}$ determines activity $y_b^{(a-1)}$, the derivative of the utility function over the weight can be found by applying the chain rule:

$$\frac{\partial E}{\partial w_{b,c}^{(a)}} = \frac{\partial E}{\partial y_b^{(a-1)}} \frac{\partial y_b^{(a-1)}}{\partial w_{b,c}^{(a)}} \tag{5}$$

The first partial derivative on the right hand side of the above equation expresses by how much the utility function can be increased by increasing the activity of node b in layer $a-1$, and we will denote it by:

$$\delta_b^{(a-1)} = \frac{\partial E}{\partial y_b^{(a-1)}} \tag{6}$$

The values of these error terms for the sample network in Figure 1B are indicated by the darkness of the arrows labelled $\delta_i^{(l)}$. The error term $\delta_2^{(0)}$ is high because there is a mismatch between the actual and desired network output, so by increasing the activity in the corresponding node $y_2^{(0)}$ the utility function can be increased. By contrast the error term $\delta_1^{(0)}$ is low, because the corresponding node $y_1^{(0)}$ already produces the desired output, so changing its activity will not increase the utility function. The error term $\delta_2^{(1)}$ is high because the corresponding node $y_2^{(1)}$ projects strongly to the node $y_2^{(0)}$ producing too low output, so increasing value of $y_2^{(1)}$ can increase the utility function. For analogous reasons, the error term $\delta_1^{(1)}$ is low.

Now let us calculate the error terms $\delta_b^{(a-1)}$. It is straightforward to evaluate them for the output layer:

$$\frac{\partial E}{\partial y_b^0} = s_b^{out} - y_b^0 \tag{7}$$

If we consider a node in an inner layer of the network then we must consider all possible routes through which the utility function is modified when the activity of

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

the node changes, i.e. we must consider the total derivative

$$\frac{\partial E}{\partial y_b^{(a-1)}} = \sum_{i=1}^{n^{(a-2)}} \frac{\partial E}{\partial y_i^{(a-2)}} \frac{\partial y_i^{(a-2)}}{\partial y_b^{(a-1)}} \tag{8}$$

Using the definition of Equation (6), and evaluating the last derivative of the above equation using the chain rule, we obtain the recursive formula for the error terms:

$$\delta_b^{(a-1)} = \begin{cases} s_b^{out} - y_b^{(a-1)} & \text{if } a-1=0 \\ \sum_{i=1}^{n^{(a-2)}} \delta_i^{(a-2)} f'\left(v_i^{(a-2)}\right) w_{i,b}^{(a-1)} & \text{if } a-1>0 \end{cases} \tag{9}$$

The fact that the error terms in layer $l > 0$ can be computed on the basis of the error terms in the next layer $l-1$ gave the name: "error back-propagation" algorithm.

Substituting the definition of error terms from Equation 6 into Equation 5 and evaluating the second partial derivative on the right hand side of Equation 5 using chain rule we obtain:

$$\frac{\partial E}{\partial w_{b,c}^{(a)}} = \delta_b^{(a-1)} f'\left(v_b^{(a-1)}\right) y_c^a \tag{10}$$

According to the above equation, the change in weight $w_{b,c}^{(a)}$ is proportional to the product of the activity of pre-synaptic node y_c^a, and the error term $\delta_b^{(a-1)}$ associated with post-synaptic node. Red upward pointing arrows in Figure 1B indicate which weights would be most increased in this example, and it is evident that the increase in these weights will indeed increase the utility function.

In summary, after presenting to the network a training sample, each weight is modified proportionally to the gradient given in Equation 10 with the error term given by Equation 9. The expression for weight change (Equations 10 and 9) is a complex global function of activities and weights of neurons not connected to the synapse being modified. In order for real neurons to compute it, the architecture of

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

the model would have to be extended to include nodes computing the error terms, that could affect the weight changes. As we will see, analogous nodes are present in the predictive coding model.

Predictive coding

Predictive coding [11, 12] is a model of unsupervised learning in the cortex, i.e. the model is presented with sensory data only and it tries to build a statistical model for these data. In this section we provide a succinct description of the model, but for readers interested in a gradual and more detailed introduction to the model, we recommend reading sections 1-2 of a tutorial on this model [15] before reading this section.

The idea behind predictive coding is that the neural networks in our brain learn the statistical regularities of the natural world, and then only propagate forward the error between the network's learned prediction and actual natural input. By only considering the deviations from learned regularities, these networks effectively remove the "already learned" and redundant components of the signal, thus the network only learns signals that it is not already able to predict.

The model also considers a hierarchy, where each level attempts to predict the signal from the previous level. The prediction error (prediction - actual signal) in each level is then fed forwards to the higher level to adjust its future prediction for the lower level.

We first formalize this hierarchy in terms of a probabilistic model, then we describe the inference in the model, its neural implementation, and finally learning of model parameters. Let $\bar{X}^{(0)}$ be a vector of random variables from which sensory inputs are sampled (it is a vector because the sensory input is multi-dimensional, e.g. we have multiple receptors in our eyes). Let us assume that the value of these variables depends on variables on the higher level $\bar{X}^{(1)}$, which in turn depend on even higher levels, etc. Let us denote a sample from random variable $\bar{X}^{(l)}$ by $\bar{x}^{(l)}$.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Let us assume the following relationship between the random variables on adjacent levels (for brevity of notation we write $P(\bar{x}^{(l)})$ instead of $P(\bar{X}^{(l)} = \bar{x}^{(l)})$):

$$P\left(x_i^{(l)} \mid \bar{x}^{(l+1)}\right) = \mathcal{N}\left(x_i^{(l)}; g_i\left(\bar{x}^{(l+1)}, \mathbf{\Theta}^{(l+1)}\right), \Sigma_i^{(l)}\right) \tag{11}$$

In the above equation $\mathcal{N}(x; \mu, \Sigma)$ is the probability density of a normal distribution with mean μ and variance Σ. Hence the mean of probability density on level l depends on the values on the higher level through function $g_i(\bar{x}^{(l+1)}, \mathbf{\Theta}^{(l+1)})$ parametrized by a matrix of parameters $\mathbf{\Theta}^{(l+1)}$. In this paper we will consider two versions of the predictive coding model with different functions g: We will consider a model with function g used in the original paper [11] to which we will refer as the Original Predictive Coding (OPC) model. We will also analyse a model with an adjusted function g which will make the link to ANNs clearer, and we will refer to it as the Adjusted Predictive Coding (APC) model. To avoid confusion between these two quite similar models we will start with the APC model, and later in the paper consider the OPC model (which has a simpler synaptic plasticity rule).

Function g in the APC model has the following form:

$$g_i\left(\bar{x}^{(l+1)}, \mathbf{\Theta}^{(l+1)}\right) = f\left(u_i^{(l)}\right) \tag{12}$$

where f is a function analogous to the activation function in ANNs, and

$$u_i^{(l)} = \sum_{j=1}^{n^{(l+1)}} \theta_{i,j}^{(l+1)} x_j^{(l+1)} \tag{13}$$

In the above equation $n^{(l)}$ denotes the number of random variables on level l (and $\theta_{i,j}^{(l+1)}$ are elements of matrix $\mathbf{\Theta}^{(l+1)}$). For such assumptions we can explicitly write the dependence of the probability density of each individual random variable:

$$P\left(x_i^{(l)} \mid \bar{x}^{(l+1)}\right) = \frac{1}{\sqrt{2\pi\Sigma_i^{(l)}}} \exp\left(-\frac{\left(x_i^{(l)} - f\left(u_i^{(l)}\right)\right)^2}{2\Sigma_i^{(l)}}\right) \tag{14}$$

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

For simplicity in this paper we do not consider how $\Sigma_i^{(l)}$ are learned [12, 15], but treat them as fixed parameters. Finally, we define the prior probability for the random variables on the highest level of hierarchy:

$$P\left(x_i^{(l_{max})}\right) = \mathcal{N}\left(x_i^{(l_{max})}; p_i^{(l_{max})}, \Sigma_i^{(l_{max})}\right) \tag{15}$$

where $p_i^{(l_{max})}$ denotes the mean of the prior distribution of $X_i^{(l_{max})}$.

This completes the description of the assumed probabilistic model, so let us now move to describing the inference in the model. Once the model is presented with a particular sensory input $\bar{x}^{(0)}$, we wish to find the most likely values of the random variables on higher levels of the hierarchy which maximize the joint probability $P(\bar{x}^{(0)}, \bar{x}^{(1)}, ..., \bar{x}^{(l_{max})})$. To simplify calculations we define the utility function equal to the logarithm of the joint distribution (since the logarithm is a monotonic operator, a logarithm of a function has the same maximum as the function itself):

$$F = \ln\left(P(\bar{x}^{(0)}, \bar{x}^{(1)}, ..., \bar{x}^{(l_{max})})\right) \tag{16}$$

Since we assumed that the variables on one level just depend on variables of the level above, we can write the utility function as:

$$F = \sum_{l=0}^{l_{max}-1} \ln\left(P(\bar{x}^{(l)} \mid \bar{x}^{(l+1)})\right) + \ln\left(P(\bar{x}^{l_{max}})\right) \tag{17}$$

Substituting Equations 14 and 15 into the above equation we obtain:

$$\begin{aligned} F = & \sum_{l=0}^{l_{max}-1} \sum_{i=1}^{n^{(l)}} \left[\ln \frac{1}{\sqrt{2\pi\Sigma_i^{(l)}}} - \frac{\left(x_i^{(l)} - f\left(u_i^{(l)}\right)\right)^2}{2\Sigma_i^{(l)}} \right] \\ & + \sum_{i=1}^{n_i^{(l_{max})}} \left[\ln \frac{1}{\sqrt{2\pi\Sigma_i^{(l_{max})}}} - \frac{\left(x_i^{(l_{max})} - p_i^{(l_{max})}\right)^2}{2\Sigma_i^{(l_{max})}} \right] \end{aligned} \tag{18}$$

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

To make the notation more compact, let us define $u_i^{(l_{max})}$ such that $p_i^{(l_{max})} = f(u_i^{(l_{max})})$. Then ignoring constant terms we can write the utility function as:

$$F = -\frac{1}{2}\sum_{l=0}^{l_{max}}\sum_{i=1}^{n^{(l)}} \frac{\left(x_i^{(l)} - f\left(u_i^{(l)}\right)\right)^2}{\Sigma_i^{(l)}} \tag{19}$$

Recall that we wish to find the values $x_i^{(l)}$ that maximize the above utility function. This can be achieved by modifying $x_i^{(l)}$ proportionally to the gradient of the utility function. To calculate the derivative of F over $x_i^{(l)}$ we note that each $x_i^{(l)}$ influences F in two ways: it occurs in Equation 19 explicitly, but it also determines the values of $u_j^{(l-1)}$. Thus the derivative contains two terms:

$$\frac{\partial F}{\partial x_b^{(a)}} = -\frac{x_b^{(a)} - f\left(u_b^{(a)}\right)}{\Sigma_b^{(a)}} + \sum_{i=1}^{n^{(a-1)}} \frac{x_i^{(a-1)} - f\left(u_i^{(a-1)}\right)}{\Sigma_i^{(a-1)}} f'\left(u_i^{(a-1)}\right)\theta_{i,b}^{(a)} \tag{20}$$

In the above equation, there are terms that repeat, so let us denote them by:

$$\varepsilon_i^{(l)} = \frac{x_i^{(l)} - f\left(u_i^{(l)}\right)}{\Sigma_i^{(l)}} \tag{21}$$

These terms describe by how much the value of a random variable on a given level differs from the mean predicted by a higher level, so let us refer to them as prediction errors. Substituting the definition of prediction errors into Equation 20 we obtain the following rule describing changes in $x_b^{(a)}$ over time:

$$\dot{x}_b^{(a)} = -\varepsilon_b^{(a)} + \sum_{i=1}^{n^{(a-1)}} \varepsilon_i^{(a-1)} f'\left(u_i^{(a-1)}\right)\theta_{i,b}^{(a)} \tag{22}$$

The computations described by Equations 21-22 could be performed by a simple network illustrated in Figure 2A with nodes corresponding to prediction errors $\varepsilon_i^{(l)}$ and values of random variables $x_i^{(l)}$. The prediction errors $\varepsilon_i^{(l)}$ are computed on the basis of excitation from corresponding variable nodes $x_i^{(l)}$, and inhibition from the

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

nodes on the higher level $x_j^{(l+1)}$ weighted by strength of synaptic connections $\theta_{i,j}^{(l+1)}$. Conversely, the nodes $x_i^{(l)}$ make computations on the basis of the prediction error from the corresponding level, and the prediction errors from the lower level weighted by synaptic weights.

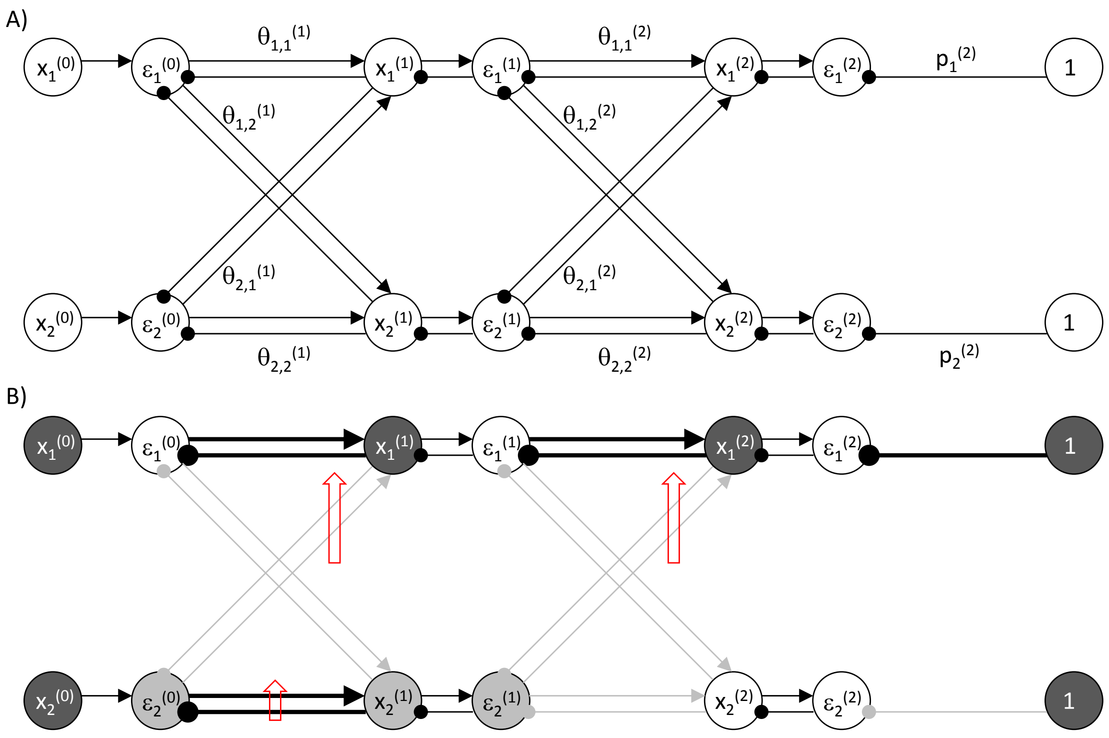

Figure 2: Predictive coding model. A) Architecture of the network. Arrows and lines ending with circles denote excitatory and inhibitory connections respectively. Connections without labels have weights fixed to 1. B) An example of activity and weight changes in the predictive coding model. Notation as in Figure 1.

It is important to emphasize that for a linear function $f(x) = x$, the non-linear terms in Equations 21-22 would disappear, and these equations could be fully implemented in the simple network shown in Figure 2A. To implement Equation 21, a prediction error node would get excitation from the corresponding variable node and inhibition equal to synaptic input from higher level nodes, thus it could compute the difference between them. Scaling the activity of nodes encoding prediction error by a constant $\Sigma_i^{(l)}$ could be implemented by self-inhibitory connections with weight $\Sigma_i^{(l)}$ (we do not consider them here for simplicity - but for details see [12, 15]). Analogously to implement Equation 22, a variable node would need to change its

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

activity proportionally to its inputs.

To provide an intuition for how the non-linear transformations could be performed when function f is non-linear, Figure 3A shows a sample network (magnification of a part on network in Figure 2A), in which each node performs a simple computation just on the basis of inputs it gets from the nodes it is connected to. To implement Equation 21, a prediction error node (e.g. $\varepsilon_1^{(1)}$) gets excitation from the corresponding variable node and inhibition equal to synaptic input ($u_1^{(1)}$) transformed by a function f. To implement Equation 22, a variable node (e.g. $x_1^{(2)}$) would need to change its level of activity proportionally to its input composed of inhibition from the corresponding prediction error node and excitation from the prediction error nodes from a lower level weighted by synaptic weights ($\theta_{1,1}^{(2)}$) and a non-linear function ($f'(u_1^{(1)})$).

In the predictive coding model, after the sensory input is provided, all nodes are updated according to Equations 21-22, until the network converges to a steady state. We label variables in the steady state with an asterisk e.g. $x_i^{*(l)}$ or F^*.

Figure 2B illustrates an example of operation of the model. The model is pre-

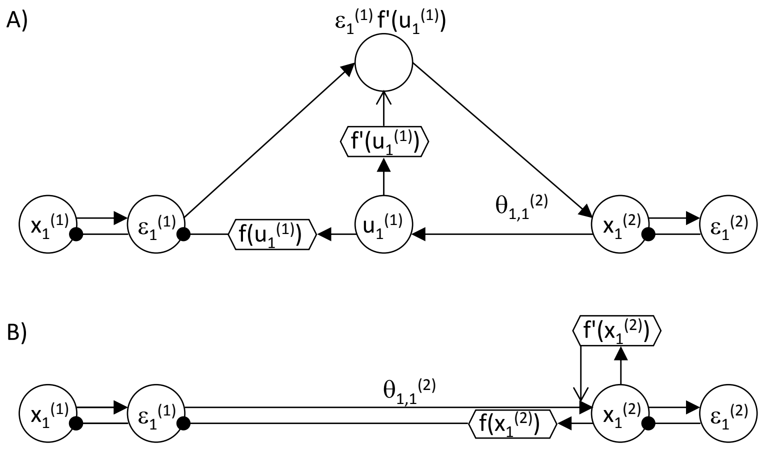

Figure 3: Details of architectures of A) the APC and B) the OPC models. Filled arrows and lines ending with circles denote excitatory and inhibitory connections respectively. Open arrows denote modulatory connections with multiplicative effect. Circles and hexagons denote nodes performing linear and non-linear computations respectively.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

sented with the sensory input in which both nodes $x_1^{(0)}$ and $x_2^{(0)}$ are active. Nodes $x_1^{(1)}$ and $x_1^{(2)}$ become activated, as they receive both top down and bottom up input (note that double inhibitory connection from higher to lower levels has overall excitatory effect). There is no mismatch between these inputs, so the corresponding prediction error nodes ($\varepsilon_1^{(0)}$, $\varepsilon_1^{(1)}$ and $\varepsilon_1^{(2)}$) are not active. By contrast, the node $x_2^{(1)}$ gets bottom up but no top down input, so its activity has intermediate value, and the prediction error nodes connected with it ($\varepsilon_2^{(0)}$ and $\varepsilon_2^{(1)}$) are active.

Once the network has reached its steady state, the parameters of the model $\theta_{i,j}^{(l)}$ are updated so the model better predicts sensory inputs. This is achieved by modifying $\theta_{i,j}^{(l)}$ proportionally to the gradient of utility function over the parameters. To compute the derivative of the utility function over $\theta_{i,j}^{(l)}$, we note that $\theta_{i,j}^{(l)}$ affects the value of function F of Equation 19 by influencing $u_i^{(l-1)}$, hence

$$\frac{\partial F^*}{\partial \theta_{b,c}^{(a)}} = \varepsilon_b^{*(a-1)} f'\left(u_b^{*(a-1)}\right) x_c^{*(a)} \tag{23}$$

According to the above equation, the change in a synaptic weight $\theta_{b,c}^{(a)}$ of connection between levels a and $a-1$ is proportional to the product of quantities encoded on these levels. For a linear function $f(x) = x$, the non-linear term in the above equation would disappear, and the weight change would simply be equal to the product of the activities of pre-synaptic and post-synaptic nodes (Figure 2A). Let us consider in more detail how the above equation could be implemented when f is non-linear in the sample network in Figure 3A. Recall that for each parameter $\theta_{b,c}^{(a)}$ there are two connections in this network equal to it, e.g. both connections next to label $\theta_{1,1}^{(2)}$ in Figure 3A, so both of them need to get modified according to the above equation. For the top connection labelled $\theta_{1,1}^{(2)}$, the weight change in the above equation is simply proportional to the product of activities of pre-synaptic and post-synaptic nodes, hence it corresponds to Hebbian plasticity.

For the bottom connection labelled $\theta_{1,1}^{(2)}$, the weight change in the above equa-

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

tion is no longer proportional to the product of activities of pre-synaptic and post-synaptic nodes. However, one could imagine mechanisms implementing Equation 23, for example, one could add a connection from the top node in Figure 3A to node $u_1^{(1)}$ that would transmit information only during plasticity and "set" the activity in the node labelled $u_1^{(1)}$ to the value desired for Hebbian plasticity. Furthermore, we will see later that by choosing different functions g, the synaptic plasticity rule can be simplified.

Finally, we can find the update rules for the parameters describing the prior probabilities by finding the derivative over Equation 18:

$$\frac{\partial F^*}{\partial p_b^{(l_{max})}} = \varepsilon_b^{*(l_{max})} \tag{24}$$

In the example of Figure 2B, red upward arrows indicate the weights which are most increased (i.e. between active pre-synaptic and post-synaptic neurons). It is evident that after these weight changes the activity of prediction error nodes would be reduced indicating that the sensory input is better predicted by the network.

Results

Already after describing the two models, one can notice the parallels between their architecture (cf. Figures 1A and 2B), activity update rules (cf. Equations 1 and 13) and weight update rules (cf. Equations 10 and 23). We now explore these parallels by considering how the predictive coding model can be used for supervised learning, and under what conditions the computations in the two models become equivalent. We start with the simplest architecture of the predictive coding model approximating back-propagation algorithms, and later discuss other architectures.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Predictive coding in supervised learning

The predictive coding model is normally used for unsupervised learning so it is presented with just a single stream of data: the sensory inputs, whereas the ANNs are presented with two streams $\bar{s}^{in}$ and $\bar{s}^{out}$. Thus while thinking about the relationship between the models we need to first ask which of the pair $(\bar{s}^{in}, \bar{s}^{out})$ corresponds to sensory input of the predictive coding model. Please note that both models make predictions: An ANN attempts to predict $\bar{s}^{out}$ while a predictive coding model attempts to predict sensory input. Therefore, while using the predictive coding model for supervised learning, $\bar{s}^{out}$ needs to be provided to nodes where normally the sensory input is presented, i.e. to $\bar{x}^{(0)}$.

Let us now consider which nodes in the predictive coding network used for supervised learning could be set to $\bar{s}^{in}$. An ANN tries to predict the output on the basis of $\bar{s}^{in}$, while in the predictive coding model the sensory input is assumed to depend on the variables on the higher levels of hierarchy. So eventually the predictive coding model attempts to predict sensory input on the basis of variables on the highest level, thus while using predictive coding for supervised learning, $\bar{s}^{in}$ can be provided to nodes on the highest level of the hierarchy, i.e. to $\bar{x}^{(l_{max})}$.

The effect of such initialization of predictive coding network can be observed by comparing the examples of operation of a predictive coding model and an ANN in Figures 2B and 1B. In Figure 2B we set the sensory input $x_i^{(0)}$ to the values of output training sample s_i^{out} from Figure 1B, and we set the prior parameters $p_i^{(2)}$ to values such that the nodes on highest level $x_i^{(2)}$ in Figure 2B have the same values as the input training sample s_i^{in} in Figure 1B. Furthermore, the corresponding synaptic weights in both examples have the same values. Please note the similarity in the pattern of prediction errors in both models, and the corresponding weight changes.

ANN has two modes of operation: during prediction it computes its output on the basis of $\bar{s}^{in}$, while during learning it updates its weights on the basis of $\bar{s}^{in}$ and $\bar{s}^{out}$. Let us now consider how the predictive coding model can operate in these modes.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Its architecture during prediction is shown in Figure 4A. The values of the nodes on the highest level are fixed to $\bar{x}^{(l_{max})} = \bar{s}^{in}$. The values of other nodes are found by maximizing the utility function proportional to $\ln P(\bar{x}^{(0)}, ..., \bar{x}^{(l_{max}-1)} \mid \bar{x}^{(l_{max})})$, which differs from the original utility function of Equation 19 in that the summation goes only until $l_{max} - 1$:

$$F = -\frac{1}{2} \sum_{l=0}^{l_{max}-1} \sum_{i=1}^{n^{(l)}} \frac{\left(x_i^{(l)} - f\left(u_i^{(l)}\right)\right)^2}{\Sigma_i^{(l)}} \tag{25}$$

All nodes, except for the highest level are modified proportionally to the gradient of the above utility. Thus the nodes on levels $l \in \{1, ..., l_{max} - 1\}$ are modified in the same way as described before (Equation 22). Additionally, the nodes on the lowest level $l = 0$ are not fixed (as the model is generating own prediction) but instead they are also modified proportionally to the gradient of the above utility, which is simply:

$$\dot{x}_b^{(0)} = \frac{\partial F}{\partial x_b^{(0)}} = \varepsilon_b^{(0)} \tag{26}$$

Thus the nodes on the lowest level change their activity proportionally to the input from the corresponding prediction error nodes, which is represented in Figure 4A by an additional connections from $\varepsilon_i^{(0)}$ to $x_i^{(0)}$.

We now show that the network with such dynamics has a stable fixed point at the state where all nodes have the same values as the corresponding nodes in the ANN receiving the same input $\bar{s}^{in}$. Since all nodes change proportionally to the gradient of F, the value of function F always increases. The maximum value F can reach is 0, because F is a negative of sum of squares, and this maximum is achieved if all terms in the summation of Equation 25 are equal to 0, i.e. when:

$$x_i^{*(l)} = f\left(u_i^{*(l)}\right) \tag{27}$$

The above equation has the same form as the update Equation 2 for ANNs,

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

and additionally $u_i^{(l)}$ is defined in analogous way as $v_i^{(l)}$ (cf. Equations 1 and 13). Therefore nodes in the prediction mode have the same values at the fixed point as the corresponding nodes in the ANN, i.e. $x_i^{*(l)} = y_i^{(l)}$.

The above property is illustrated in Figure 4A, in which weights are set to the same value as for the ANN in Figure 1B, and the network is presented with the same input sample. The activity in this case propagates from node $x_1^{(2)}$ through the connections with high weights resulting the same pattern of activity on level $l = 0$ as for the ANN in Figure 1B.

During learning the values of the nodes on the lowest level are set to output sample, i.e. $\bar{x}^{(0)} = \bar{s}^{out}$, as illustrated in Figure 4B. Then the values of all nodes on levels $l \in \{1, ..., l_{max}-1\}$ are modified in the same way as described before (Equation 22). In the example in Figure 4B it leads to the same pattern of activity on levels $l \in \{0, 1\}$ as in Figure 2B for the reasons discussed previously.

Once the network converges, all synaptic weights are modified (Equation 23). In the example of Figure 4B, the weights that increase most are indicated by long red upward arrows. There would be also increase in weight between $\varepsilon_2^{(0)}$ and $x_2^{(1)}$, indicated by a shorter arrow, but it would be not as large as node $x_2^{(1)}$ has lower activity. This pattern of weight change is similar as in back-propagation algorithm (Figure 1B). In the next section we analyse under what conditions such weight changes converge to that in the back-propagation algorithm.

Convergence of predictive coding to back-propagation

The weight update rules in the two models (Equations 10 and 23) have the same form, however, the prediction error terms $\delta_i^{(l)}$ and $\varepsilon_i^{(l)}$ were defined differently. To see the relationship between these terms, we will now derive the recursive formula for prediction errors $\varepsilon_i^{(l)}$ analogous to that for $\delta_i^{(l)}$ in Equation 9. We note that once the network reaches the steady state in the learning mode, the change in activity of each node must be equal to zero. Setting the left hand side of Equation 22 to 0 we

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

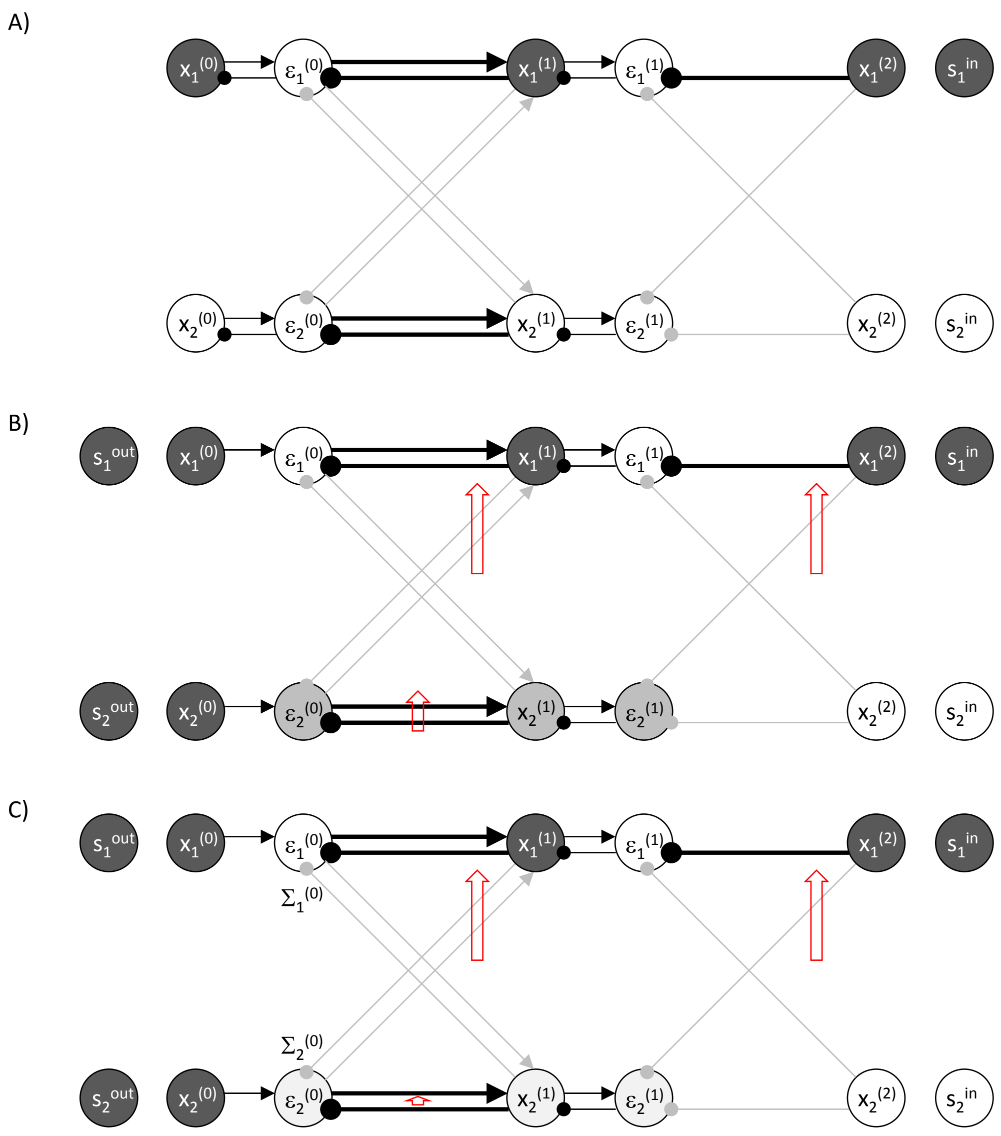

Figure 4: Example of a predictive coding network for supervised learning. A) Prediction mode. B) Learning mode. C) Learning mode for a network with high value of parameter describing sensory noise. Notation as in Figure 2B.

obtain:

$$\varepsilon_b^{*(a)} = \sum_{i=1}^{n^{(a-1)}} \varepsilon_i^{*(a-1)} f'\left(u_i^{*(a-1)}\right) \theta_{i,b}^{(a)} \tag{28}$$

We can now write a recursive formula for the prediction errors:

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$$\varepsilon_b^{*(a-1)} = \begin{cases} \left(s_b^{out} - f\left(u_b^{*(a-1)}\right)\right)/\Sigma_b^{(0)} & \text{if } a-1=0 \\ \sum_{i=1}^{n^{(a-2)}} \varepsilon_i^{*(a-2)} f'\left(u_i^{*(a-2)}\right)\theta_{i,b}^{(a-1)} & \text{if } a-1>0 \end{cases} \tag{29}$$

Let us first consider the case when all variance parameters are set to $\Sigma_i^{(l)} = 1$ (as in [11]). Then the above formula has exactly the same form as for the back-propagation algorithm (Equation 9). Therefore, it may seem that weight change in the two models is identical. However, for the weight change to be identical, the values of the corresponding nodes must be equal, i.e. $x_i^{*(l)} = y_i^{(l)}$ (it is sufficient for this condition to hold for $l > 0$, because $x_i^{*(0)}$ do not directly influence weight changes). Although we have shown in the previous subsection that $x_i^{*(l)} = y_i^{(l)}$ in the prediction mode, it may not be the case in the learning mode, because the nodes $x_i^{(0)}$ are fixed (to s_i^{out}), and thus function F may not reach the maximum of 0, so Equation 27 may not be satisfied.

Let us now consider under what conditions $x_i^{*(l)}$ is equal or close to $y_i^{(l)}$. First, when the networks are trained such that they correctly predict the output training samples, then utility function F can reach 0 during the relaxation and hence $x_i^{*(l)} = y_i^{(l)}$, and the two models have exactly the same weight changes. In particular, the change in weights is then equal to 0, thus the weights resulting in perfect prediction are a fixed point for both models.

Second, when the networks are trained such that their predictions are close to the output training samples, then fixing $x_i^{(0)}$ will only slightly change the activity of other nodes in the predictive coding model, so the weight change will be similar.

To illustrate this property we compare the weight changes in back-propagation and predictive coding models with very simple architecture. In particular, we consider a network with just three layers ($l_{max} = 2$) and one node in each layer ($n^{(0)} = n^{(1)} = n^{(2)} = 1$). Such network has only 2 weight parameters ($w_{1,1}^{(1)}$ and $w_{1,1}^{(2)}$), so the utility function of the ANN can be easily visualized. Figure 5A shows the utility function for a training set in which input training samples were generated

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

randomly from uniform distribution $s_1^{in} \in [-5, 5]$, and output training samples were generated as $s_1^{out} = \tanh(W^{(1)} \tanh(W^{(2)} s_i^{in}))$, where $W^{(1)} = W^{(2)} = 1$. Thus ANN with weights equal to $w_{1,1}^{(l)} = W^{(l)}$ perfectly predicts all samples in the training set, so the utility function is equal to 0. There are also other combinations of weights resulting in good prediction, which create a "ridge" of the utility function.

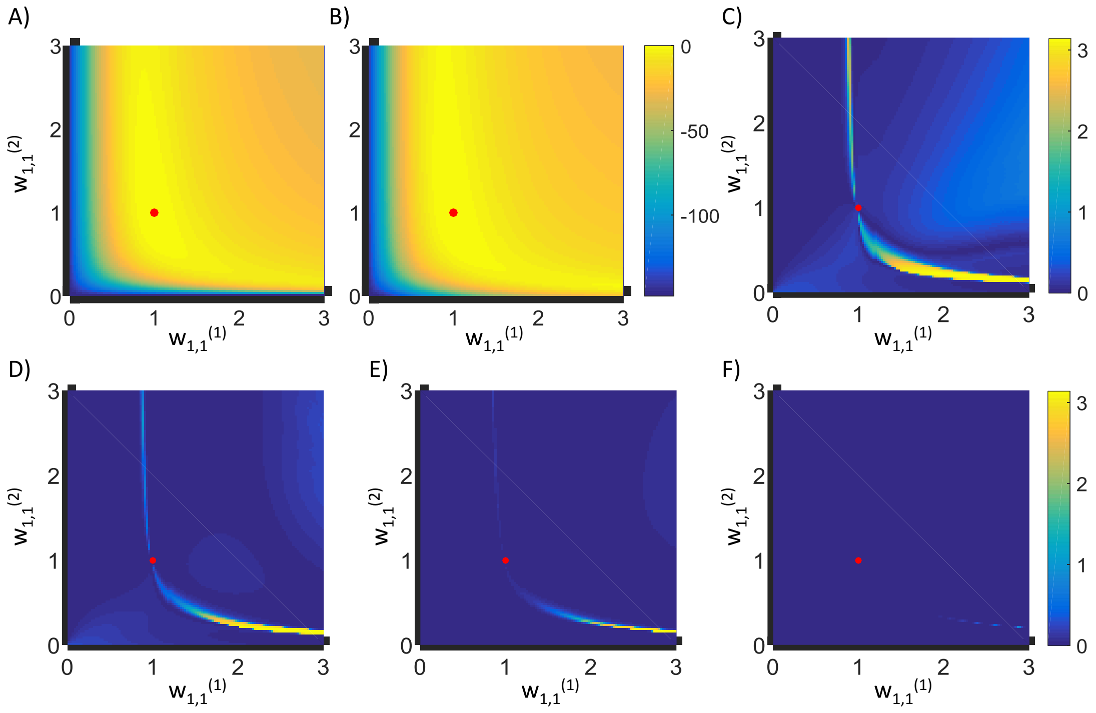

Figure 5: Comparison of weight changes in back-propagation and predictive coding models. A) The utility function of an ANN for a training set with 300 samples generated as described in main text. The utility function is equal to sum of 300 terms given by Equation 3 corresponding to individual training samples. The red dot indicates weights that maximize the utility function. B) The utility function of the APC model at the fixed point. For each set of weights and training sample, to find the state of predictive coding network at the fixed point, the nodes in layers 0 and 2 were set to training examples, and the node in layer 1 was updated according to Equation 22. This equation was solved using Euler method with integration step of 0.02, until the total change of the activity of the nodes was smaller that 10^{-8} or 100,000 iterations have been performed. C) The angle difference between the gradients of the cost function of the ANN and the cost function of the OPC model. To find the gradient, the network was first allowed to settle to a fixed point, and then the gradient was found from Equation 35. D-F) Angle difference between the gradient for the ANN and the gradient for the APC model found from Equation 23. Different panels correspond to different values of parameter describing sensory noise: D) $\Sigma_1^{(0)} = 1$. E) $\Sigma_1^{(0)} = 8$. F) $\Sigma_1^{(0)} = 256$.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Figure 5D shows the angle between the direction of weight change in back-propagation and the predictive coding model. The directions of the gradient for the two models are very similar except for the weights along the "ridge" of the utility function. This difference is caused by slight misalignment of "ridges" of utility functions E and F^* (cf. Figures 5 A and B). Nevertheless close to the maximum of the utility function (indicated by a red dot), the directions of weight change become similar and the angle decreases towards 0.

There is also a third condition under which the predictive coding network approximates the back-propagation algorithm. Namely, when the value of parameters $\Sigma_i^{(0)}$ is increased, then the impact of fixing $x_i^{(0)}$ on the activity of other nodes is reduced, because $\varepsilon_i^{(0)}$ becomes smaller (Equation 21) and its influence on activity of other nodes is reduced. Thus $x_i^{*(l)}$ is closer to $y_i^{(l)}$ (for $l > 0$), and the weight change in the predictive coding model becomes closer to that in the back-propagation algorithm (recall that the weight changes are the same when $x_i^{*(l)} = y_i^{(l)}$ for $l > 0$).

Multiplying $\Sigma_i^{(0)}$ by a constant will also reduce all $\varepsilon_i^{(l)}$ by the same constant (see Equation 29), and consequently all weight changes will be reduced by this constant. This can be compensated by multiplying the learning rate α by the same constant, so the magnitude of the weight change remains constant.

Figures 5E and F show that as $\Sigma_i^{(0)}$ increases the angle between weight changes in the two models decreases towards 0. Thus as the parameters $\Sigma_i^{(0)}$ are increased, the weight changes in the predictive coding model converge to those in the back-propagation algorithm.

In the example of Figure 4, panel C illustrates the impact of increasing $\Sigma_i^{(0)}$. It reduces $\varepsilon_2^{(0)}$, which in turn reduces $x_2^{(1)}$ and $\varepsilon_2^{(1)}$. This decreases all weight changes, but particularly the change of the weight between nodes $\varepsilon_2^{(0)}$ and $x_2^{(1)}$ (indicated by a short red arrow) as both of these nodes have reduced activity. After compensating for the learning rate these weight changes become more similar to those in back-propagation algorithm (compare Figures 4B, C and 1B).

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Original predictive coding model

Let us now consider a version of the predictive coding model where function g is defined as in the original model [11], because, as mentioned before, such definition results in a simpler synaptic plasticity rule for the model.

$$g_i\left(\bar{x}^{(l)}, \mathbf{\Theta}^{(l)}\right) = u_i^{(l)} \tag{30}$$

where we redefine $u_i^{(l)}$ as:

$$u_i^{(l)} = \sum_{j=1}^{n^{(l+1)}} \theta_{i,j}^{(l+1)} f\left(x_j^{(l+1)}\right) \tag{31}$$

Following the same logic as for the APC model, the utility function becomes:

$$F = -\frac{1}{2}\sum_{l=0}^{l_{max}}\sum_{i=1}^{n^{(l)}} \frac{\left(x_i^{(l)} - u_i^{(l)}\right)^2}{\Sigma_i^{(l)}} \tag{32}$$

Consequently we redefine the prediction errors as:

$$\varepsilon_i^{(l)} = \frac{x_i^{(l)} - u_i^{(l)}}{\Sigma_i^{(l)}} \tag{33}$$

Analogously as before, the dynamics of the nodes resulting in maximizing the utility function is:

$$\dot{x}_b^{(a)} = -\varepsilon_b^{(a)} + \sum_{i=1}^{n^{(a-1)}} \varepsilon_i^{(a-1)} f'\left(x_b^{(a)}\right)\theta_{i,b}^{(a)} \tag{34}$$

The computation described by the above two equations can also be implemented in the architecture of Figure 2A and the details of implementation of non-linear computations are shown in Figure 3B [15]. To implement Equation 33, a prediction error node (e.g. $\varepsilon_1^{(1)}$) must receive excitation from corresponding variable node ($x_1^{(1)}$) and inhibition (scaled by synaptic weights) from nodes in the higher level transformed through the non-linear function f.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

To implement Equation 34 a variable node (e.g. $x_1^{(2)}$ in Figure 3B) needs to receive input from the prediction error nodes on the corresponding and lower levels. Additionally, the input from the lower level needs to be scaled by a non-linear function of the activity of variable node itself ($f'(x_1^{(2)})$). Such scaling could be implemented either by a separate node or by intrinsic mechanisms within the variable node that would make it react to excitatory inputs differentially depending on its own activity level.

The weight update rule in this OPC model is equal to the gradient of model's utility function:

$$\frac{\partial F^*}{\partial \theta_{b,c}^{(a)}} = \varepsilon_b^{*(a-1)} f\left(x_c^{*(a)}\right) \tag{35}$$

The above update rule has a simpler form than for the APC model (Equation 23) and just depends on the activities of the nodes the synapse connects. In particular, for the bottom connection labelled $\theta_{1,1}^{(2)}$ in Figure 3B, the change in a synaptic weight is simply equal to the product of the activity of nodes it connects. For the top connection, the change in weights in equal to the product of activity of the pre-synaptic node and a function of activity of the post-synaptic node.

While using the above OPC model for supervised learning we could set during learning $\bar{x}^{(l_{max})} = \bar{s}^{in}$ and $\bar{x}^{(0)} = \bar{s}^{out}$, but to make the comparison with ANNs easier, it is helpful to instead set $f(\bar{x}^{(l_{max})}) = \bar{s}^{in}$ and $f(\bar{x}^{(0)}) = \bar{s}^{out}$. In particular, when the model is used in the prediction mode, and we set $f(\bar{x}^{(l_{max})}) = \bar{s}^{in}$, then the nodes converge to values corresponding to those in an ANN in a sense that: $y_i^{(l)} = f(x_i^{*(l)})$.

Following the same methods as before, we obtain the recursive formula for the prediction errors at the fixed point (the second case below comes from setting the left hand side of Equation 34 to 0):

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$$\varepsilon_b^{*(a-1)} = \begin{cases} \left(f^{-1}\left(s_b^{(out)}\right) - u_b^{*(0)}\right)/\Sigma_b^{(0)} & \text{if } a-1=0 \\ \sum_{i=1}^{n^{(a-2)}} \varepsilon_i^{*(a-2)} f'\left(x_i^{*(a-1)}\right)\theta_{i,b}^{(a-1)} & \text{if } a-1>0 \end{cases} \tag{36}$$

We see that the above recursive formula for the prediction error has an analogous form to that for APC model (Equation 29), so in the OPC model, the errors are also "back-propagated" to higher layers. However, the weight change in the OPC model differs more from that in back-propagation algorithm than for the APC model, because the equation for weight change is simper (cf. Equations 10, 23 and 35). Figure 5C compares the direction of weight change in the OPC model and the back-propagation algorithm. As for the APC model, the direction is similar apart from the weights on the "ridge" of the utility function, and it converges to the weight change of the back-propagation algorithm as the weights become closer to the maximum of the utility function. Importantly, due to the same maximum of the utility function and similar direction of gradient, the OPC model converges to the same values of weights in the learning problem of Figure 5 as the back-propagation algorithm.

Performance on more complex learning tasks

To efficiently learn in more complex tasks, ANNs include a "bias term" or an additional node in each layer which does not receive any input, but has activity equal to 1. Let us define this node as the node with index 0 in each layer, so $y_0^{(l)} = 1$. With such node, the definition of synaptic input (Equation 1) is extended to include one additional term $w_{i,0}^{(l+1)}$, which is referred to as the "bias term". The weight corresponding to the "bias term" is updated during learning according to the same rule as all other weights (Equation 10).

An equivalent "bias term" can be easily introduced to the predictive coding models. This would just be a node with a constant value of $x_0^{(l)} = 1$ which projects to the next layer, but does have an associated error node. The activity of such node would not change after the training inputs are provided, and corresponding

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

weights $\theta_{i,0}^{(l+1)}$ would be modified as all other weights (Equation 23 for the APC and Equation 35 for the OPC model).

To assess the performance of the predictive coding models on more complex learning tasks, we tested them on standard benchmark datasets obtained from University of California Irvine Machine Learning Repository [20]. We chose 4 datasets for which the learning task involved prediction of a single numerical value from a vector of numerical values. For each dataset, we compared error of predictive coding models and ANN with one hidden layer (i.e. $l_{max} = 2$) and $n^{(1)} = 10$ nodes in that layer.

Figure 6 compares the error in 10-fold cross validation of different models with different learning rates. The lowest value of the learning rate used was 0, and so the leftmost point on each panel corresponds to the performance of untrained network. In this case all models had exactly the same error, because the computations of predictive coding models in the prediction mode are the same as for ANN. For larger learning rates, the error decreased for all models, including the OPC model which has particularly simple neural implementation. For the dataset used in Figures 6C and D all models achieved a very similar level of accuracy.

For a large value of parameter $\Sigma_i^{(0)}$ the performance of the APC model was very similar to back-propagation algorithm, in agreement with earlier analysis showing that then the weight changes in the APC model converge to those in the back-propagation algorithm. These two models achieved the highest accuracy among other models in Figures 6A and B, because the back-propagation algorithm explicitly minimizes the error in model's predictions.

For the largest learning rate employed, the error of the ANN started to differ from the error of the APC model with high $\Sigma_i^{(0)}$ in Figures 6B and D. This happened because for such a large learning rate, the average absolute values of weights became very high (as we did not employ any regularization). Interestingly, in this case the predictive coding models achieved higher accuracy than the ANN, suggesting that

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

they are more robust to instability due to too high learning rate.

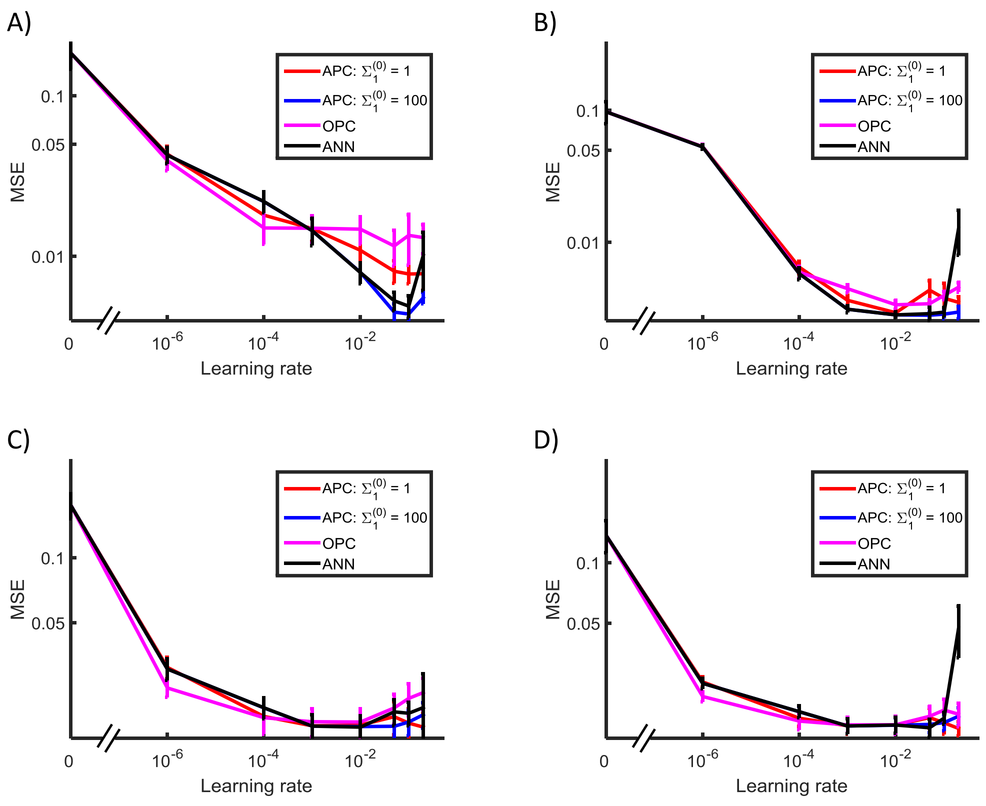

Figure 6: Comparison of Mean Squared Error (MSE) in 10-fold cross-validation for different models (indicated by colours - see key) and different datasets. A) Airfoil Self-Noise dataset: 1503 samples with 5 input attributes. B) Combined Cycle Power Plant dataset: 9568 samples with 4 input attributes [21, 22]. C-D) Wine Quality datasets: both datasets had 11 input attributes; white wine dataset (panel C) had 1599 samples and red wine dataset (panel D) had 4898 samples [23]. At the start of each iteration of 10-fold cross-validation the weights were initialized randomly but in the same way across models. The weights were initialised from a uniform random distribution between 0 and 1, these values were then divided by the number of neurons in the layer that the weights project from. In each iteration of 10-fold cross-validation, each algorithm was trained on 1.5 million data samples. For each sample, the predictive coding networks were simulated until convergence. A dynamic form of the Euler integration step was used where its size was allowed to reduce by a factor of 10 should the system not be converging (i.e. the maximum change in node activity increases from the previous step). The relaxation was performed until the maximum value of $\partial F/\partial x_i^{(l)}$ was lower than $10^{-6}/\Sigma_i^{(0)}$. The weights were then modified with a learning rate corresponding to the value on horizontal axes. Error bars show standard error of MSE across 10 folds.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Effects of architecture of the predictive coding model

In the architecture of predictive coding networks considered so far, the input samples s_i^{in} were provided to the nodes at the highest level of hierarchy. However, while mapping the predictive coding networks on the brain organization, the nodes on the highest level correspond to higher associative areas, while the nodes on the lowest level correspond to primary sensory areas to which the sensory inputs are provided.

This discrepancy can be resolved by considering an architecture in which there are two separate sensory areas receiving s_i^{in} and s_i^{out}, which are both connected with higher areas. For example, in case of learning associations between visual stimuli and desired motor responses, s_i^{in} and s_i^{out} could be provided to primary visual and primary somatosensory/motor cortices, respectively. Both of these primary areas project through a hierarchy of sensory areas to common higher associative cortex.

To understand the potential benefit of such an architecture over standard back-propagation, we analyse a simple example of learning the association between one dimensional samples shown in Figure 7A. Since there is a simple linear relationship (with noise) between samples in Figure 7A, we will consider predictions generated by a very simple network shown in Figure 7B. During training of this network the samples are provided to the nodes on the lowest level ($x_1^{(0)} = s_1^{out}$ and $x_2^{(0)} = s_1^{in}$).

For simplicity, we will assume a linear relationship between variables ($f(x) = x$). Under this assumption the APC and OPC models become equivalent, because they only differ in how the non-linearity was introduced to function g. Furthermore, we will ignore the prior for the highest level, so the node on the highest level will be updated according to (linearised Equation 22 without the term corresponding to the prior):

$$\dot{x}_1^{(1)} = \sum_{i=1}^{n^{(0)}} \varepsilon_i^{(0)} \theta_{i,1}^{(1)} \tag{37}$$

During testing, we only set $x_2^{(0)} = s_1^{in}$, and let both nodes $x_1^{(1)}$ and $x_1^{(0)}$ to be

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

updated according to Equations 37 and 26.

Solid lines in Figure 7A show the values predicted by the model (i.e. $x_1^{*(0)}$) after providing different inputs (i.e $x_2^{(0)} = s_1^{in}$), and different colours correspond to different noise parameters. When equal noise is assumed in input and output (red line), the network simply learns the probabilistic model that explains the most variance in the data, so the model learns the direction in which the data is most spread out. This direction is the same as the first principal component shown in dashed red line (any difference between the two lines is due the iterative nature of

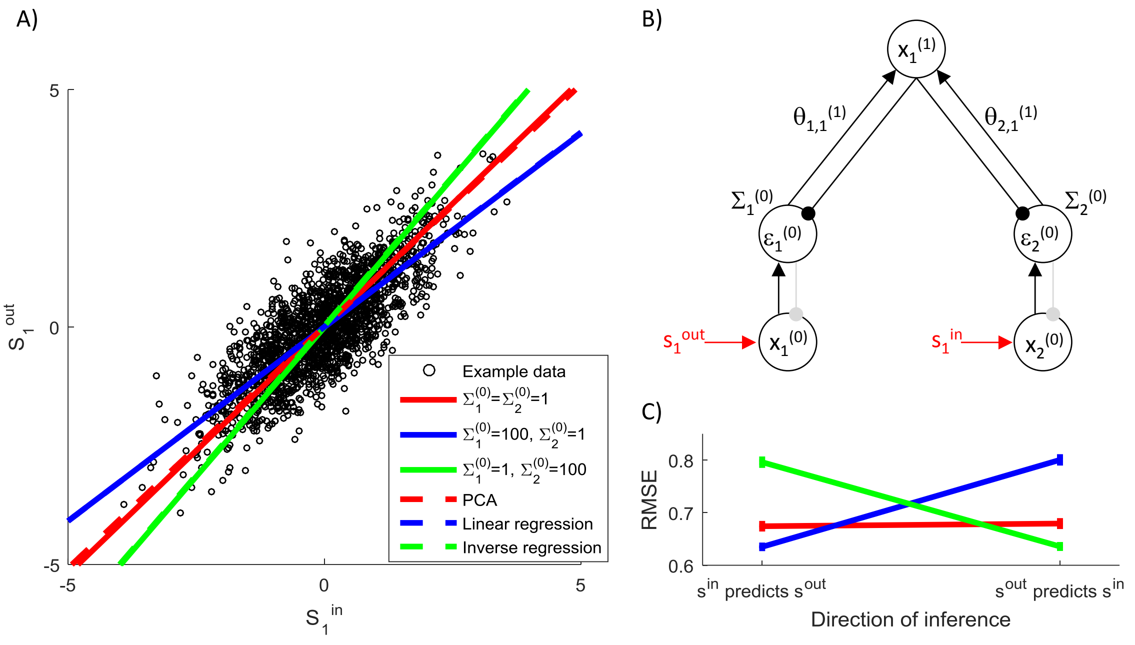

Figure 7: The effect of variance associated with different inputs on network predictions. A) Sample training set composed or 2000 randomly generated samples, such that $s_1^{in} = a + b$ and $s_1^{out} = a - b$ where $a \sim \mathcal{N}(0, 1)$ and $b \sim \mathcal{N}(0, 1/9)$. Lines compare the predictions made by the model with different parameters with predictions of standard algorithms (see key). B) Architecture of the simulated predictive coding network. Notation as in Figure 4. Additionally, connections shown in grey are used if the network predicts the value of the corresponding sample. C) Root Mean Squared Error (RMSE) of the models with different parameters (see key of panel A) trained on data as in panel A and tested on further 100 samples generated from the same distribution. During the training, for each sample the network was allowed to converge to the fixed point as described in caption of Figure 5 and the weights were modified with learning rate $\alpha = 1$. The entire training and testing procedure was repeated 50 times, and the error bars show the standard error.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

learning in the predictive coding model).

When the noise at the node receiving output samples is large (blue line in Figure 7A), the dynamics of the network will lead to the node at the top level converging to the input sample (i.e. $x_1^{*(1)} \approx s_1^{in}$). Given the analysis in the previous section, the model converges then to the back-propagation algorithm, which in the case of linear $f(x)$ simply corresponds to linear regression, shown by dashed blue line.

Conversely, when the noise at the node receiving input samples is large (green line in Figure 7A), the dynamics of the network will lead to the node at the top level converging to the output sample (i.e. $x_1^{*(1)} \approx s_1^{out}$). The network in this case will learn to predict the input sample on the basis of the output sample. Hence its predictions correspond to that obtained by finding linear regression in inverse direction (i.e. the linear regression predicting s^{in} on the basis of s^{out}), shown by the dashed green line.

Different predictions of the models with different noise parameters will lead to different amounts of error when tested, which are shown in the left part of Figure 7C (labelled s^{in} predicts s^{out}). The network approximating the back-propagation algorithm is most accurate, as the back-propagation algorithm explicitly minimizes the error in predicting output samples. Next in accuracy is the network with equal noise on both input and output, followed by the model approximating inverse regression.

Due to the flexible structure of the predictive coding network, we can also test how well it is able to infer the likely value of input sample s^{in} on the basis of the output sample s^{out}. In order to test it, we provide the trained network with the output sample ($x_1^{(0)} = s_1^{out}$), and let both nodes $x_1^{(1)}$ and $x_2^{(0)}$ to be updated according to Equations 37 and 26. The value $x_2^{*(0)}$ to which the node corresponding to the input converged is the network's inferred value of the input. We compared these values with actual s^{in} in the testing examples, and the resulting root mean squared errors are shown in the right part of Figure 7C (labelled s^{out} predicts s^{in}). This time the model approximating the inverse regression is most accurate.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Figure 7C illustrates that when noise is present in the data, there is a trade-off between accuracy of inference in the two directions. Nevertheless, the predictive coding network with equal noise parameters for inputs and outputs is predicting relatively well in both directions, being just slightly less accurate than the optimal algorithm for the given direction.

It is also important to emphasize that the models we analysed in this subsection generate different predictions, only because the the training samples are noisy. If the amount of noise were reduced, the models' predictions would become more and more similar (and their accuracy would increase). This parallels the property discussed earlier that the closer the predictive coding models predict all samples in the training set, the closer their computation to ANNs with back-propagation algorithm.

The networks in the cortex are likely to be non-linear and include multiple layers, but predictive coding models with corresponding architectures are still likely to retain the key properties outlined above. Namely, they would allow learning bidirectional associations between inputs and outputs, and if the mapping between the inputs and outputs could be perfectly represented by the model, the networks could be able to learn them and make accurate predictions.

Discussion

In this paper we have proposed how the predictive coding models can be used for supervised learning. We showed that then they perform the same computation as ANNs in the prediction mode, and weight modification in the learning mode has a similar form as for the back-propagation algorithm. Furthermore, in the limit of parameters describing the noise in the layer where output training samples are provided, the learning rule in the APC model converges to that for the back-propagation algorithm.

As the predictive coding network, which was originally developed for unsuper-

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

vised learning, can also be used for supervised learning, a single network can be used for both. This suggests that cortical circuits with the same plasticity rules can learn from both unlabelled sensory stimuli, which are experienced very often, and from less frequent situations when the correct response to a given stimulus is known. As for ANNs, such exposure to the unsupervised learning is likely to speed up supervised learning [24].

In the paper we presented two versions of the predictive coding model differing in the assumed form of the non-linear relationship between random variables on different levels of hierarchy. In both models, after error in prediction of the training sample, the weights were modified in all layers to minimize the prediction errors. Only in the APC model the weight change for certain parameters converged to that in the back-propagation algorithm, but the synaptic plasticity rule in the APC model was more complex for neural implementation than in the OPC model. Nevertheless, it is useful to point out that in all simulations in this paper we used $f(x) = \tanh(x)$ for consistency with classical ANNs, but it has been noted that ANNs with rectified linear activation function $f(x) = \max(x, 0)$ work equally good or better [25]. Such function may well describe the non-linearity of cortical neurons, as it approximately equal to firing-Input relationship (or f-I curve) of the most popular model of a neuron - the integrate and fire model [26]. The rectified linear function is piecewise-linear and, as we emphasized earlier, the neural implementation of the predictive coding models simplifies considerably for a linear function f.

The conditions under which the predictive coding model converges to the back-propagation algorithm provide a probabilistic interpretations of the back-propagation algorithm. This allows the comparison of the assumptions made of the back-propagation algorithm with the probabilistic structure of learning tasks. First, the predictive coding model corresponding to back-propagation assumes that output samples are generated from a probabilistic model with multiple layers of random variables, but most of the noise is added only at the level of output samples (i.e.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$\Sigma_i^{(0)} >> \Sigma_i^{(l>0)}$). By contrast, probabilistic models corresponding to most of real-world datasets have variability entering on multiple levels. For example, if we consider classification of images of hand-written digits, the variability is present in both high level features like length or angle of individual strokes, and low level features like the colors of pixels.

Second, the predictive coding model corresponding to back-propagation assumes layered structure of the probabilistic model. By contrast probabilistic models corresponding to many problems may have other structures. For example, consider a little kitten which, by observing its mother, learns what hunting strategies to use for preys of different types. This situation could be described by a probabilistic model including a higher level variable corresponding to a type of prey, which determines both the visual input perceived by a kitten, and movements of its mother (such model is appropriate as there will be variability across trials in both how a prey of a given type looks, and in how the mother reacts to it). It would be very interesting to investigate how the performance of the predictive coding model with architecture reflecting the probabilistic structure of a datasets would compare to the ANN with back-propagation algorithm.

We hope that the proposed extension of the predictive coding framework to supervised learning will make easier to experimentally test this framework. With the previous unsupervised formulation, it was difficult to state detailed predictions on neural activity, as it was difficult to measure the values of inferred higher level features, on the basis of which the model computes prediction errors. In experiments with participants performing supervised learning tasks, the model can be used to estimate prediction errors, and one could analyse which cortical regions or layers have activity correlated with model variables. Inspection of the neural activity could in turn refine the predictive coding models, so they better reflect information processing in cortical circuits.

Different versions or parametrizations for the predictive coding model make dif-

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

ferent predictions on both behaviour and neural activity. For example, in Figure 7A we illustrated how the predictions of the model approximating the back-propagation algorithm differ from the predictive coding model with its default parameters assuming equal noise on input and output. It would be interesting to test which of these models describes better behaviour of humans trained on and performing an analogous task.

The proposed predictive coding models are still quite abstract and it is important to investigate if different linear or non-linear nodes can be mapped on particular anatomically defined neurons within cortical micro-circuit [27]. Iterative refinements of such mapping on the basis of experimental data (such as f-I curves of these neurons, their connectivity and activity during learning tasks) may help understand how supervised and unsupervised learning is implemented in the cortex.

Acknowledgements

The authors thank Tim Vogels and Eduardo Martin Moraud for reading the previous version of the manuscript and for providing very useful comments.

Author contributions

JW and RB did analytic work. JW performed simulations. RB wrote the paper.

References

[1] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323:533–536.

[2] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, et al. Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation. 1989;1:541–551.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

[3] Chauvin Y, Rumelhart DE. Backpropagation: Theory, Architectures, and Applications. Lawrence Erlbaum Associates; 1995.

[4] Bogacz R, Markowska-Kaczmar U, Kozik A. Blinking artefact recognition in EEG signal using artificial neural network. In: Proceedings of 4th Conference on Neural Networks and Their Applications, Zakopane (Poland); 1999. p. 502–507.

[5] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in Neural Information Processing Systems 25. Curran Associates, Inc.; 2012. p. 1097–1105.

[6] Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. Signal Processing Magazine, IEEE. 2012;29:82–97.

[7] Seidenberg JL Mark S ; McClelland. A distributed, developmental model of word recognition and naming. Psychological Review. 1989;96:523–568.

[8] McClelland JL, McNaughton BL, O'Reilly RC. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. Psychological Review. 1995;102:419–457.

[9] Plaut DC, McClelland JL, Seidenberg MS, Patterson K. Understanding normal and impaired word reading: Computational principles in quasi-regular domains. Psychological Review. 1996;103:56–115.

[10] O'Reilly RC, Munakata Y. Computational Explorations in Cognitive Neuroscience. MIT Press; 2000.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

[11] Rao RPN, Ballard DH. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature Neuroscience. 1999;2:79–87.

[12] Friston K. A theory of cortical responses. Philosophical Transactions of the Royal Society B. 2005;360:815–836.

[13] Friston K. Hierarchical Models in the Brain. PLoS Computational Biology. 2008;4:e1000211.

[14] Feldman H, Friston K. Attention, Uncertainty, and Free-Energy. Frontiers in Human Neuroscience. 2010;4:215.

[15] Bogacz R. A tutorial on the free-energy framework for modelling perception and learning. Journal of Mathematical Psychology. 2016;resubmitted.

[16] Friston K. The free-energy principle: a unified brain theory? Nature Reviews Neuroscience. 2010;11:127–138.

[17] Friston K, Schwartenbeck P, FitzGerald T, Moutoussis M, Behrens T, Dolan RJ. The anatomy of choice: active inference and agency. Frontiers in Human Neuroscience. 2013;7:598.

[18] Friston K. Learning and inference in the brain. Neural Networks. 2003;16:1325–1352.

[19] Hyvarinen A. Regression using independent component analysis, and its connection to multi-layer perceptrons. In: Proceedings of the 9th International Conference on Artificial Neural Networks, Edinburgh; 1999. p. 491–496.

[20] Lichman M. UCI Machine Learning Repository, http://archive.ics.uci.edu/ml; 2013.

bioRxiv preprint doi: https://doi.org/10.1101/035451; this version posted December 28, 2015. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

[21] Tufekci P. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power and Energy Systems. 2014;60:126–140.

[22] Kaya H, Tufekci P, Grgen SF. Local and Global Learning Methods for Predicting Power of a Combined Gas and Steam Turbine. In: Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering, Dubai. ICETCEE; 2012. p. 13–18.

[23] Cortez P, Cerdeira A, Almeida F, Matos T, Reis J. Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems. 2009;47:547–553.

[24] Erhan D, Bengio Y, Courville A, Manzagol PA, Vincent P, Bengio S. Why Does Unsupervised Pre-training Help Deep Learning? J Mach Learn Res. 2010;11:625–660.

[25] Zeiler MD, Ranzato M, Monga R, Mao M, Yang K, Le QV, et al. On rectified linear units for speech processing. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE; 2013. p. 3517–3521.

[26] Dayan P, Abbott LF. Theoretical Neuroscience. MIT Press; 2001.

[27] Bastos AM, Usrey WM, Adams RA, Mangun GR, Fries P, Friston KJ. Canonical Microcircuits for Predictive Coding. Neuron. 2012;76:695–711.