

On the relationship between predictive coding and backpropagation

Robert Rosenbaum

Department of Applied and Computational Mathematics and Statistics

University of Notre Dame

Notre Dame, IN 46556 USA

Abstract

Artificial neural networks are often interpreted as abstract models of biological neuronal networks, but they are typically trained using the biologically unrealistic backpropagation algorithm and its variants. Predictive coding has been offered as a potentially more biologically realistic alternative to backpropagation for training neural networks. In this manuscript, I review and extend recent work on the mathematical relationship between predictive coding and backpropagation for training feedforward artificial neural networks on supervised learning tasks. I discuss some implications of these results for the interpretation of predictive coding and deep neural networks as models of biological learning and I describe a repository of functions, Torch2PC, for performing predictive coding with PyTorch neural network models.

The backpropagation algorithm and its variants are widely used to train artificial neural networks. While artificial and biological neural networks share some common features, a direct implementation of backpropagation in the brain is often considered biologically implausible in part because of the non-local nature of parameter updates: The update to a parameters in one layer depends on activity in all deeper layers. In contrast, biological neural networks are believed to learn largely through local synaptic plasticity rules for which changes to a synaptic weight depend on neural activity local to that synapse. Backpropagation can be performed using local updates if gradients of neurons' activations are passed upstream through feedback connections, but this interpretation implies other biologically implausible properties of the network, like symmetric feedforward and feedback weights. See previous work [1, 2] for a more complete review of the biological plausibility of backpropagation.

Several approaches have been proposed for achieving or approximating backpropagation with more biologically realistic learning rules [1–13]. One such approach [10–13] is derived from the theory of “predictive coding” or “predictive processing” [14–22]. When applied to artificial neural networks trained on supervised learning tasks, predictive coding can produce weight updates that are similar to or the same as the exact gradients computed by backpropagation [10–13].

In this manuscript, I first extensively review this previous work [10–13] from an algorithmic perspective and explore its implications on empirical examples. I next prove a theorem that modestly extends results from this previous work and discuss some implications of these results on the interpretation of predictive coding and artificial neural networks as models of biological learning. Finally, I introduce a repository of Python functions, Torch2PC, that can be used to perform predictive coding on any PyTorch Sequential model. Torch2PC can be found at

<https://github.com/RobertRosenbaum/Torch2PC>

A Google Drive folder with Colab notebooks that produce all figures in this text can be found at

https://drive.google.com/drive/folders/1m_y0G_sTF-pV9pd2_sysWt1nvRvHYzX0

A copy of the same code is also stored at

<https://github.com/RobertRosenbaum/PredictiveCodingVsBackProp>

1 A review of the relationship between backpropagation and predictive coding from previous work

For completeness, let us first review the backpropagation algorithm. Consider a feedforward deep neural network (DNN) defined by

$$\begin{aligned}\hat{v}_0 &= x \\ \hat{v}_\ell &= f_\ell(\hat{v}_{\ell-1}; \theta_\ell), \quad \ell = 1, \dots, L\end{aligned}\tag{1}$$

where each \hat{v}_ℓ is a vector or tensor of activations, each θ_ℓ is a set of parameters for layer ℓ , and L is the network's depth. In supervised learning, we seek to minimize a loss function $\mathcal{L}(\hat{y}, y)$ where y is a label associated with input, x , and

$$\hat{y} = f(x; \theta) = \hat{v}_L$$

is the network's output, which depends on parameters $\theta = \{\theta_\ell\}_\ell$. The loss is typically minimized using gradient-based optimization methods with gradients computed using automatic differentiation tools based on the backpropagation algorithm. For completeness, backpropagation is reviewed in the pseudocode below.

Algorithm 1 A standard implementation of backpropagation.

Given: Input (x) and label (y)

forward pass

$$\hat{v}_0 = x$$

for $\ell = 1, \dots, L$

$$\hat{v}_\ell = f_\ell(\hat{v}_{\ell-1}; \theta_\ell)$$

backward pass

$$\delta_L = \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L}$$

for $\ell = L - 1, \dots, 1$

$$\delta_\ell = \delta_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}$$

$$d\theta_\ell = -\delta_\ell \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell}$$

A direct application of the chain rule and mathematical induction shows that backpropagation computes the gradients,

$$\delta_\ell = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \quad \text{and} \quad d\theta_\ell = -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \theta_\ell}.$$

The negative gradients, $d\theta_\ell$, are then used to update parameters, either directly for stochastic gradient descent or indirectly for other gradient-based learning methods [23].

I next review algorithms derived from the theory of predictive coding and their relationship to backpropagation. I focus on the approach described in [10], but the approach in other work is similar [11–13]. For clarity, I consider the special case of a sequential, feedforward neural network (as in Eq. (1) and Algorithm 1), whereas the approach in [10] applies to the more general case of networks that can be represented by directed acyclic graphs.

1.1 A strict interpretation of predictive coding does not accurately compute gradients.

Predictive coding can be derived from a hierarchical, Gaussian probabilistic model in which each layer, ℓ , is associated with a Gaussian random variable, V_ℓ , satisfying

$$p(V_\ell = v_\ell | V_{\ell-1} = v_{\ell-1}) = \mathcal{N}(v_\ell; f_\ell(v_{\ell-1}; \theta_\ell), \Sigma_\ell)$$

where $\mathcal{N}(v; \mu, \Sigma) \propto \exp(-[v - \mu]^T \Sigma^{-1} [v - \mu]/2)$ is the multivariate Gaussian distribution with mean, μ , and covariance matrix, Σ , evaluated at v . Following previous work [10–13], I take $\Sigma = I$ to be the identity matrix, but later discuss the potential implications of relaxing this assumption [20].

If we condition on an observed input, $V_0 = x$, then a forward pass through the network described by Eq. (1) corresponds to setting $\hat{v}_0 = x$ and then sequentially computing the conditional expectations or, equivalently, maximizing conditional probabilities,

$$\begin{aligned} \hat{v}_\ell &= E[V_\ell | V_{\ell-1} = \hat{v}_{\ell-1}] \\ &= \underset{v_\ell}{\operatorname{argmax}} p(V_\ell = v_\ell | V_{\ell-1} = \hat{v}_{\ell-1}) \\ &= f_\ell(\hat{v}_{\ell-1}; \theta_\ell) \end{aligned}$$

until reaching an inferred output, $\hat{y} = \hat{v}_L$. Note that this forward pass does not necessarily maximize the global conditional probability, $p(V_L = \hat{y} | v_0 = x)$ and it does not account for any prior belief about V_L .

One interpretation of a forward pass is that each \hat{v}_ℓ is the network’s “belief” about the state of V_ℓ , when only $V_0 = x$ has been observed. Now suppose that we condition on both an observed input, $V_0 = x$, and its label, $V_L = y$. In this case, generating beliefs about the hidden states, V_ℓ , is more difficult because we need to account for potentially conflicting information at each end of the network. We can proceed by initializing a set of beliefs, v_ℓ , about the state of each V_ℓ , and then updating our initial beliefs to be more consistent with the observations, x and y , and parameters, θ_ℓ .

The error made by a set of beliefs, $\{v_\ell\}_\ell$, under parameters, $\{\theta_\ell\}_\ell$, can be quantified by

$$\epsilon_\ell = f_\ell(v_{\ell-1}; \theta_\ell) - v_\ell$$

for $\ell = 1, \dots, L-1$ where $v_0 = V_0 = x$ is observed. It is not so simple to quantify the error, ϵ_L , made at the last layer in a way that accounts for arbitrary loss functions. In the special case of a squared-Euclidean loss function,

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2} \|\hat{y} - y\|^2,$$

where $\|u\|^2 = u^T u$. Standard formulations of predictive coding [19, 20] use

$$\epsilon_L = f_L(v_{L-1}; \theta_L) - y \tag{2}$$

where recall that y is the label. In this case, ϵ_L satisfies

$$\epsilon_L = \frac{\partial \mathcal{L}(\tilde{v}_L, y)}{\partial \tilde{v}_L} \tag{3}$$

where

$$\tilde{v}_L = f_L(v_{L-1}; \theta_L).$$

We use the $\tilde{\cdot}$ to emphasize that \tilde{v}_L is different from \hat{v}_L (which is defined by a forward pass starting at $\hat{v}_0 = x$) and is defined in a fundamentally different way from the v_ℓ terms (which do not necessarily satisfy $v_\ell = f_\ell(v_{\ell-1}; \theta_\ell)$). We can then define the total summed magnitude of errors as

$$F = \frac{1}{2} \sum_{\ell=1}^L \|\epsilon_\ell\|^2$$

Under a formulation of the model in terms of approximate Bayesian inference, F approximates the variational free energy [10, 15, 19, 20].

Under a more heuristic interpretation, v_ℓ represents the network’s “belief” about V_ℓ , and $f_\ell(v_{\ell-1}; \theta_\ell)$ is the “prediction” of v_ℓ made by the previous layer. Under this interpretation, ϵ_ℓ is the error made by the previous layer’s prediction, so ϵ_ℓ is called a “prediction error.” Then F quantifies the total magnitude of prediction errors given a set of beliefs, v_ℓ , parameters, θ_ℓ , and observations, $V_0 = x$ and $V_L = y$.

In predictive coding, beliefs, v_ℓ , are updated to minimize the error, F . This can be achieved by gradient descent, *i.e.*, by making updates of the form

$$v_\ell \leftarrow v_\ell + \eta dv_\ell$$

where η is a step size and

$$\begin{aligned} dv_\ell &= -\frac{\partial F}{\partial v_\ell} \\ &= \epsilon_\ell - \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(v_\ell; \theta_{\ell+1})}{\partial v_\ell} \end{aligned} \tag{4}$$

In this expression, $\partial f_{\ell+1}(v_\ell; \theta_{\ell+1})/\partial v_\ell$ is a Jacobian matrix and $\epsilon_{\ell+1}$ is a row-vector to simplify notation, but a column-vector interpretation is similar. If x is a mini-batch instead of one data point, then v_ℓ is an $m \times n_\ell$ matrix and derivatives are tensors. These conventions are used throughout the manuscript. The updates in Eq. (4) can be iterated until convergence or approximate convergence. Note that the prediction errors, $\epsilon_\ell = v_\ell - f_\ell(v_{\ell-1}; \theta_\ell)$, should also be updated on each iteration.

Learning can also be phrased as minimizing F with gradient descent on parameters. Specifically,

$$\theta_\ell \leftarrow \theta_\ell + \eta_\theta d\theta_\ell$$

where

$$\begin{aligned} d\theta_\ell &= -\frac{\partial F}{\partial \theta_\ell} \\ &= -\epsilon_\ell \frac{\partial f_\ell(v_{\ell-1}; \theta_\ell)}{\partial \theta_\ell}. \end{aligned} \tag{5}$$

Note that some previous work uses the negative of the prediction errors used here, *i.e.*, they use $\epsilon_\ell = v_\ell - f_\ell(v_{\ell-1}; \theta_\ell)$. While this choice changes some of the expressions above, the value of F and its dependence on θ_ℓ is not changed because F is defined by the norms of the ϵ_ℓ terms. The complete algorithm is defined more precisely by the pseudocode below:

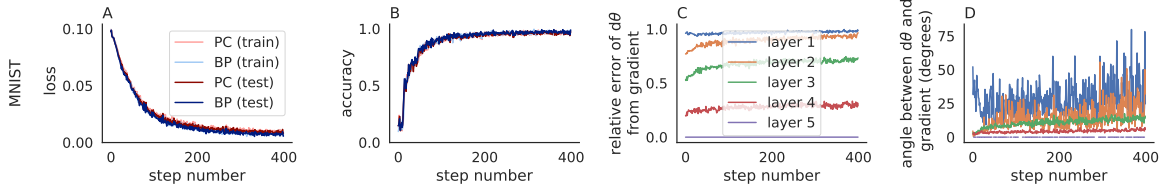


Figure 1: **Predictive coding compared to backpropagation in a convolutional neural network trained on MNIST.** **A,B)** The loss (A) and accuracy (B) on the training set (pastel) and test set (dark) when a 5-layer network was trained using a strict implementation of predictive coding (Algorithm 2 with $\eta = 0.1$ and $n = 20$; red) and backpropagation (blue). **C,D)** The relative error (C) and angle (B) between the parameter update, $d\theta$, computed by Algorithm 2 and the negative gradient of the loss at each layer.

Algorithm 2 A direct interpretation of predictive coding.

Given: Input (x), label (y), and initial beliefs (v_ℓ)

error and belief computation

for $i = 1, \dots, n$

$$\tilde{v}_L = f_L(v_{L-1}; \theta_L)$$

$$\epsilon_L = \frac{\partial \mathcal{L}(\tilde{v}_L; y)}{\partial \tilde{v}_L}$$

for $\ell = L - 1, \dots, 1$

$$\epsilon_\ell = v_\ell - f_\ell(v_{\ell-1}; \theta_\ell)$$

$$dv_\ell = -\epsilon_\ell + \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(v_\ell; \theta_{\ell+1})}{\partial v_\ell}$$

$$v_\ell = v_\ell + \eta dv_\ell$$

parameter update computation

for $\ell = 1, \dots, L$

$$d\theta_\ell = -\epsilon_\ell \frac{\partial f_\ell(v_{\ell-1}; \theta_\ell)}{\partial \theta_\ell}$$

The choice of initial beliefs is not specified in the algorithm above, but previous work [10–13] uses the results from a forward pass, $v_\ell = \hat{v}_\ell$, as initial conditions and I do the same in all numerical examples.

I tested Algorithm 2 on MNIST using a 5-layer convolutional neural network. To be consistent with the definitions above, I used a mean-squared error (squared Euclidean) loss function, which required one-hot encoded labels [23]. Algorithm 2 performed similarly to backpropagation (Fig. 1A,B) even though the parameter updates did not match the true gradients (Fig. 1C,D). Algorithm 2 was slower than backpropagation (31s for Algorithm 2 versus 8s for backpropagation when training metrics were not computed on every iteration) in part because Algorithm 2 requires several inner iterations to compute the prediction errors ($n = 20$ iterations used in this example). Algorithm 2 failed to converge on a larger model. Specifically, the loss grew consistently with iterations when trying to use Algorithm 2 to train the 6-layer CIFAR-10 model described in the next section.

Fig. 1C,D shows that predictive coding does not update parameters according to the true gradients, but it is not immediately clear whether this would be resolved by using more iterations (larger n) or different values of the step size, η . I next compared the parameter updates, $d\theta_\ell$, to the true gradients, $\partial \mathcal{L} / \partial \theta_\ell$ for different values of n and η (Fig. 2). For small values of η and large values of n , parameter updates were similar to the true gradients in deeper layers, but they differed substantially in shallower layers. Larger values of η caused the iterations in Algorithm 2 to diverge.

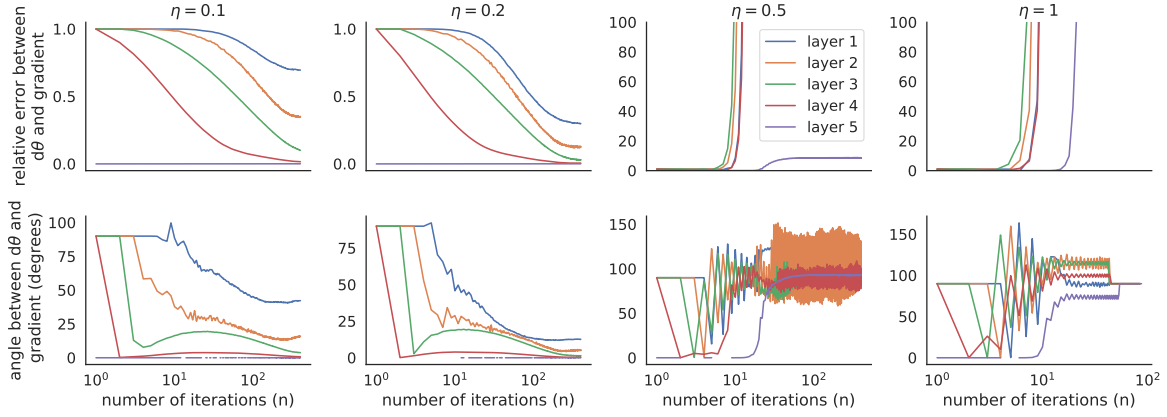


Figure 2: **Comparing parameter updates from predictive coding to true gradients.** Relative error and angle between $d\theta_\ell$ produced by predictive coding (Algorithm 2) as compared to the exact gradients, $\partial\mathcal{L}/\partial\theta_\ell$ computed by backpropagation (relative error defined by $\|d\theta_{pc} - d\theta_{bp}\|/\|d\theta_{bp}\|$). Updates were computed as a function of the number of iterations, n , used in Algorithm 2 for various values of the step size, η , using the model from Fig. 1 applied to one mini-batch of data. Both models were initialized identically to the pre-trained parameter values from the trained model in Fig. 1.

Some choices in designing Algorithm 2 were made arbitrarily. For example, the three updates inside the inner for-loop over ℓ could be performed in a different order or the outer for-loop over i could be changed to a while-loop with a convergence criterion. For any initial conditions and any of these design choices, if the iterations over i are repeated until convergence or approximate convergence of each v_ℓ to a fixed point, v_ℓ^* , then the increments must satisfy $dv_\ell = 0$ at the fixed point and therefore the fixed point values of the prediction errors, ϵ_ℓ^* , must satisfy

$$\epsilon_\ell^* = \frac{\partial f_{\ell+1}(v_\ell^*; \theta_{\ell+1})}{\partial v_\ell^*} \epsilon_{\ell+1}^* \quad (6)$$

for $\ell = 1, \dots, L-1$. By the definition of ϵ_L , we have

$$\epsilon_L^* = \frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial \tilde{v}_L^*}. \quad (7)$$

where

$$\tilde{v}_L^* = f_L(v_{L-1}^*; \theta_L).$$

Combining Eqs. (6) and (7) gives the fixed point prediction errors of the penultimate layer

$$\begin{aligned} \epsilon_{L-1}^* &= \frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial \tilde{v}_L^*} \frac{\partial f_L(v_{L-1}^*; \theta_L)}{\partial v_{L-1}^*} \\ &= \frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial v_{L-1}^*} \end{aligned}$$

where I used the fact that $\tilde{v}_L^* = f_L(v_{L-1}^*; \theta_L)$ and the chain rule. The error in layer $L-2$ is then given by

$$\epsilon_{L-2}^* = \frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial v_{L-1}^*} \frac{\partial f_{L-1}(v_{L-2}^*; \theta_{L-1})}{\partial v_{L-2}^*}.$$

Note that we cannot apply the chain rule to reduce this product further because it is not necessarily true that $v_{L-1}^* = f_{L-1}(v_{L-2}^*; \theta_{L-1})$. We revisit this point below. Generalizing to earlier layers, we have

$$\epsilon_\ell^* = \frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial \tilde{v}_{L-1}^*} \prod_{\ell'=\ell}^{L-2} \frac{\partial f_{\ell'+1}(v_{\ell'}^*; \theta_{\ell'+1})}{\partial v_{\ell'}^*}. \quad (8)$$

for $\ell = 1, \dots, L-2$. Therefore, if the inference loop converges to a fixed point, then the subsequent parameter update obeys

$$d\theta_\ell = -\frac{\partial \mathcal{L}(\tilde{v}_L^*, y)}{\partial \tilde{v}_{L-1}^*} \left[\prod_{\ell'=\ell}^{L-2} \frac{\partial f_{\ell'+1}(v_{\ell'}^*; \theta_{\ell'+1})}{\partial v_{\ell'}^*} \right] \frac{\partial f_\ell(v_{\ell-1}^*; \theta_\ell)}{\partial \theta_\ell} \quad (9)$$

by Eq. (5). It is not clear whether there is a simple mathematical relationship between these parameter updates and the negative gradients, $d\theta_\ell = -\partial \mathcal{L} / \partial \theta_\ell$, computed by backpropagation.

It is tempting to assume that $v_\ell^* = f_\ell(v_{\ell-1}^*; \theta_\ell)$, in which case the product terms would be reduced by the chain rule. Indeed, this assumption would imply that $v_\ell^* = \hat{v}_\ell$ and $\tilde{v}_L^* = \hat{v}_L$ and, finally, that $\epsilon_\ell = \partial \mathcal{L} / \partial \hat{v}_\ell$ and $d\theta_\ell = -\partial \mathcal{L} / \partial \theta_\ell$, identical to the values computed by backpropagation. However, we cannot generally expect to have $v_\ell^* = f_\ell(v_{\ell-1}^*; \theta_\ell)$ because this would imply that $\epsilon_\ell^* = 0$ and therefore $\partial \mathcal{L} / \partial v_\ell^* = \partial \mathcal{L} / \partial \theta_\ell = 0$. In other words, Algorithm 2 is only equivalent to backpropagation in the case where parameters are at a critical point of the loss function, so all updates are zero. Nevertheless, this thought experiment suggests a modification to Algorithm 2 for which the fixed points *do* represent the true gradients [10, 11]. We review that modification in the next section.

Note also that the calculations above rely on the assumption of a Euclidean loss function, $\mathcal{L}(\hat{y}, y) = \|\hat{y} - y\|^2 / 2$. If we want to generalize the algorithm to different loss functions, then Eqs. (2) and (3) could not both be true, and therefore Eqs. (6) and (7) could not both be true. This leaves open the question of how to define ϵ_L when using loss functions that are not proportional to the squared Euclidean norm. If we defined ϵ_L by (2), at the expense of losing (3), then the algorithm would not account for the loss function at all, so it would effectively assume a Euclidean loss, *i.e.*, it would compute the same values that are computed by Algorithm 2 with a Euclidean loss. If we instead defined ϵ_L by Eq. (3) at the expense of (2), then Eqs. (4) and (6) would no longer be true for $\ell = L-1$ and Eq. (5) would no longer be true for $\ell = L$. Instead, all three of these equations would involve second-order derivatives of the loss function, and therefore the fixed point equations (8) and (9) would also involve second order derivatives. The interpretation of the parameter updates is not clear in this case. One might instead try to define ϵ_L by the result of a forward pass,

$$\begin{aligned} \epsilon_L &= f_L(\hat{v}_{L-1}; \theta_L) - y \\ &= \hat{v}_L - y \end{aligned}$$

but then ϵ_L would be a constant with respect to v_{L-1} , so we would have $\partial \epsilon_L / \partial v_{L-1} = 0$, and therefore Eq. (4) at $\ell = L-1$ would become

$$\begin{aligned} dv_{L-1} &= -\frac{\partial F}{\partial v_{L-1}} \\ &= \epsilon_{L-1} \end{aligned}$$

which has a fixed point at $\epsilon_{L-1}^* = 0$. This would finally imply that all the errors converge to $\epsilon_\ell^* = 0$ and therefore $d\theta_\ell = 0$ at the fixed point.

We next discuss a modification of Algorithm 2 that converges to the same gradients computed by backpropagation, *and* is applicable to general loss functions [10, 11].

1.2 Predictive coding modified by the fixed prediction assumption converges to the gradients computed by backpropagation.

Previous work [10, 11] proposes a modification of the predictive coding algorithm described above called the “fixed prediction assumption.” Motivated by the considerations in the last few paragraphs of the previous section, we can selectively substitute some terms of the form v_ℓ and $f_\ell(v_{\ell-1}; \theta_\ell)$ in Algorithm 2 with \hat{v}_ℓ (or, equivalently, $f_\ell(\hat{v}_{\ell-1}; \theta_\ell)$) where \hat{v}_ℓ are the results of the original forward pass starting from $\hat{v}_0 = x$. Specifically, the following modifications are made to the quantities computed by Algorithm 2

$$\begin{aligned}\epsilon_\ell &= \hat{v}_\ell - v_\ell \\ \epsilon_L &= \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L} \\ dv_\ell &= \epsilon_\ell - \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ d\theta_\ell &= -\epsilon_\ell \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell}\end{aligned}$$

for $\ell = 1, \dots, L-1$. This modification can be interpreted as “fixing” the predictions at the values computed by a forward pass and is therefore called the “fixed prediction assumption” [10, 11]. Additionally, the initial conditions of the beliefs are set to the results from a forward pass, $v_\ell = \hat{v}_\ell$ for $\ell = 1, \dots, L-1$. The complete modified algorithm is defined by the pseudocode below:

Algorithm 3 Supervised learning with predictive coding modified by the fixed prediction assumption. Adapted from the algorithm in [10] and similar to the algorithm from [11].

```
Given: Input ( $x$ ) and label ( $y$ )
# forward pass
 $\hat{v}_0 = x$ 
for  $\ell = 1, \dots, L$ 
     $\hat{v}_\ell = f_\ell(\hat{v}_{\ell-1}; \theta_\ell)$ 
     $v_\ell = \hat{v}_\ell$ 
# error and belief computation
 $\epsilon_L = \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L}$ 
for  $i = 1, \dots, n$ 
    for  $\ell = L-1, \dots, 1$ 
         $\epsilon_\ell = v_\ell - \hat{v}_\ell$ 
         $dv_\ell = \epsilon_\ell - \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}$ 
         $v_\ell = v_\ell + \eta dv_\ell$ 
# parameter update computation
for  $\ell = 1, \dots, L$ 
     $d\theta_\ell = -\epsilon_\ell \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell}$ 
```

Note, again, that some choices in Algorithm 3 were made arbitrarily. The three updates inside the inner for-loop over ℓ could be performed in a different order or the outer for loop over i could be changed to a while-loop with a convergence criterion. Regardless of these choices, the fixed points, ϵ_ℓ^* , can again

be computed by setting $dv_\ell = 0$ to obtain

$$\epsilon_\ell^* = \epsilon_{\ell+1}^* \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}.$$

Now note that ϵ_L is fixed, so

$$\epsilon_L^* = \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L}$$

and we can combine these two equations to compute

$$\begin{aligned} \epsilon_{L-1}^* &= \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L} \frac{\partial f_L(\hat{v}_{L-1}; \theta_L)}{\partial \hat{v}_{L-1}} \\ &= \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_{L-1}} \end{aligned}$$

where I used the chain rule and the fact that $\hat{v}_L = f_L(\hat{v}_{L-1}; \theta_L)$. Continuing this approach we have,

$$\begin{aligned} \epsilon_\ell^* &= \epsilon_{\ell+1}^* \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ &= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \end{aligned}$$

for all $\ell = 1, \dots, L$ (where recall that $\hat{y} = \hat{v}_L$ is the output from the feedforward pass). Combining this with the modified definition of $d\theta_\ell$, we have

$$\begin{aligned} d\theta_\ell &= -\epsilon_\ell^* \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell} \\ &= -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \frac{\partial \hat{v}_\ell}{\partial \theta_\ell} \\ &= -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \theta_\ell} \end{aligned}$$

where I used the chain rule and the fact that $\hat{v}_\ell = f_\ell(\hat{v}_{\ell-1}; \theta_\ell)$. We may conclude that, if the inference step converges to a fixed point ($dv_\ell = 0$), then Algorithm 3 computes the same values of $d\theta_\ell$ as backpropagation and also that the prediction errors, ϵ_ℓ , converge to the gradients, $\delta_\ell = \partial \mathcal{L} / \partial \hat{v}_\ell$, computed by backpropagation. As long as the inference step *approximately* converges to a fixed point ($dv_\ell \approx 0$), then we should expect the parameter updates from Algorithm 3 to *approximate* those computed by backpropagation. In the next section, I extend this result to show that a special case of the algorithm computes the true gradients in a fixed number of steps.

I next tested Algorithm 3 on MNIST using the same 5-layer convolutional neural network considered above. I used a cross-entropy loss function, but otherwise used all of the same parameters used to test Algorithm 2 in Fig. 1. The modified predictive coding algorithm (Algorithm 3) performed similarly to backpropagation in terms of the loss and accuracy (Fig. 3A,B). Parameter updates computed by Algorithm 3 did not match the true gradients, but pointed in a similar direction and provided a closer match than Algorithm 2 (compare Fig. 3C,D to Fig. 1C,D). Algorithm 3 was similar to Algorithm 2 in terms of training time (29s for Algorithm 3 versus 31s for Algorithm 2 and 8s for backpropagation).

To see how well these results extend to a larger model and more difficult benchmark, I next tested Algorithm 3 on CIFAR-10 [24] using a six-layer convolutional network. While the network only has one more layer than the MNIST network used above, it has 141 times more parameters (32,695 trainable parameters in the MNIST model versus 4,633,738 in the CIFAR-10 model).

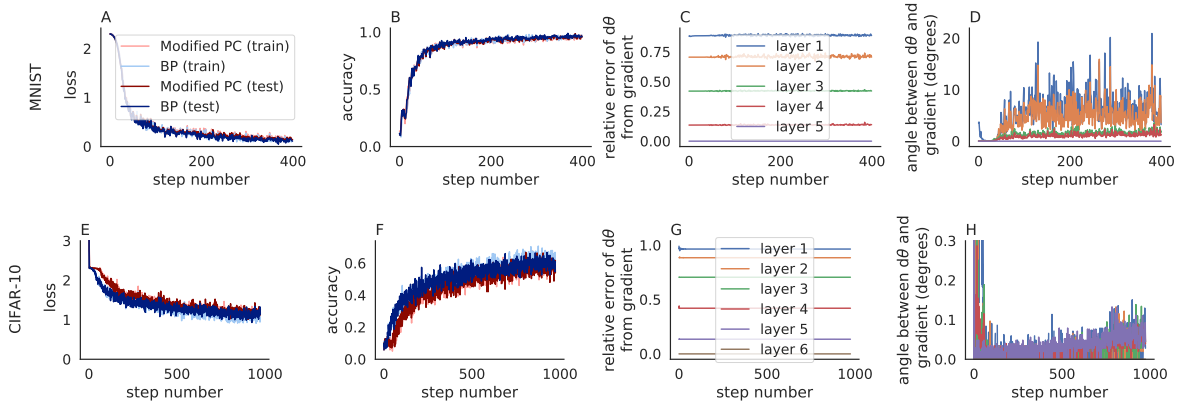


Figure 3: **Predictive coding modified by the fixed prediction assumption compared to backpropagation in convolutional neural networks trained on MNIST and CIFAR-10.** Same as Fig. 1 except Algorithm 3 was used (with $\eta = 0.1$ and $n = 20$) in place of Algorithm 2.

Algorithm 3 performed similarly to backpropagation in terms of loss and accuracy during learning (Fig. 3E,F) and produced parameter updates that pointed in a similar direction, but still did not match the true gradients (Fig. 3G,H). Algorithm 3 was substantially slower than backpropagation (848s for Algorithm 3 versus 58s for backpropagation when training metrics were not computed on every iteration).

I next compared the parameter updates computed by Algorithm 3 to the true gradients for different values of n and η (Fig. 4). For $\eta < 1$, the parameter updates, $d\theta_\ell$, appear to converge, but do not converge exactly to the true gradients. This is likely due to numerical floating point errors accumulated over iterations. When $\eta = 1$, the parameter updates at each layer remained constant for the first few iterations, then immediately jumped to become very near the updates from backpropagation. In the next section, I provide a mathematical analysis of this behavior and show that when $\eta = 1$, Algorithm 3 computes the true gradients in a fixed number of steps.

2 Predictive coding modified by the fixed prediction assumption using a step size of $\eta = 1$ computes exact gradients in a fixed number of steps.

A major disadvantage of the approach outlined above – when compared to standard backpropagation – is that it requires iterative updates to v_ℓ and ϵ_ℓ . Indeed, previous work [10] used $n = 100 - 200$ iterations, leading to substantially slower performance compared to standard backpropagation. Other work [11] used $n = 20$ iterations like I did above. In general, there is a tradeoff between accuracy and performance when choosing n , as demonstrated in Fig. 4. However, more recent work [12, 13] showed that, under the fixed prediction assumption, predictive coding can compute the exact same gradients computed by backpropagation in a fixed number of steps. That work used a more specific formulation of the neural network which can implement fully connected layers, convolutional layers, and recurrent layers. They also used an unconventional interpretation of neural networks in which weights are multiplied outside the activation function, *i.e.*, $f_\ell(x; \theta_\ell) = \theta_\ell g_\ell(x)$, and inputs are fed into the last layer instead of the first. Next, I show that their result holds for arbitrary feedforward neural networks as formulated in Eq. (1) (with arbitrary functions, f_ℓ) and this result has a simple interpretation in terms of Algorithm 3. Specifically, the following theorem shows that taking a step size of $\eta = 1$ yields an exact computation of gradients using just $n = L$ iterations (where L is the depth of the network).

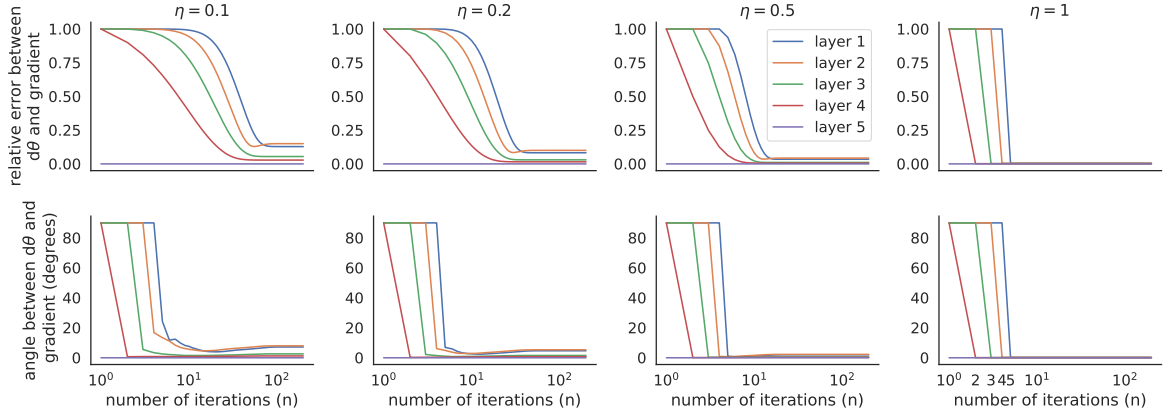


Figure 4: **Comparing parameter updates from predictive coding modified by the fixed prediction assumption to errors computed by backpropagation.** Relative error and angle between $d\theta$ produced by predictive coding modified by the fixed prediction assumption (Algorithm 3) as compared to the exact gradients computed by backpropagation (relative error defined by $\|d\theta_{pc} - d\theta_{bp}\|/\|d\theta_{bp}\|$). Updates were computed as a function of the number of iterations, n , used in Algorithm 3 for various values of the step size, η , using the model from Fig. 3 applied to one mini-batch of data. Both models were initialized identically to the pre-trained parameter values from the backpropagation-trained model in Fig. 3.

Theorem 1. *If Algorithm 3 is run with step size $\eta = 1$ and at least $n = L$ iterations then the algorithm computes*

$$\epsilon_\ell = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell}$$

and

$$d\theta_\ell = -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \theta_\ell}$$

for all $\ell = 1, \dots, L$ where $\hat{v}_\ell = f_\ell(\hat{v}_{\ell-1}; \theta_\ell)$ are the results from a forward pass with $\hat{v}_0 = x$ and $\hat{y} = \hat{v}_L = f(x; \theta)$ is the output.

Proof. For the sake of notational simplicity within this proof, define $\delta_\ell = \partial \mathcal{L}(\hat{v}_L, y)/\partial \hat{v}_\ell$. So we first need to prove that $\epsilon_\ell = \delta_\ell$. First, rewrite the inside of the error and belief loop from Algorithm 3 while explicitly keeping track of the iteration number in which each variable was updated,

$$\begin{aligned} \epsilon_\ell^i &= v_\ell^{i-1} - \hat{v}_\ell \\ dv_\ell^i &= \epsilon_\ell^i - \epsilon_{\ell+1}^i \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ v_\ell^i &= v_\ell^{i-1} + dv_\ell^i. \end{aligned}$$

Here, v_ℓ^i , ϵ_ℓ^i , and dv_ℓ^i denote the values of v_ℓ^i , ϵ_ℓ^i , and dv_ℓ^i respectively at the end of the i th iteration, $v_\ell^0 = \hat{v}_\ell$ corresponds to the initial value, and all terms without superscripts are constant inside the inference loop. There are some subtleties here. For example, we have v_ℓ^{i-1} in the first line because v_ℓ is updated after ϵ_ℓ in the loop. More subtly, we have $\epsilon_{\ell+1}^i$ in the second equation instead of $\epsilon_{\ell+1}^{i-1}$ because the for loop goes backwards from $\ell = L - 1$ to $\ell = 1$, so $\epsilon_{\ell+1}$ is updated before ϵ_ℓ . First note that

$$\epsilon_\ell^1 = 0$$

for $\ell = 1, \dots, L - 1$ because $v_\ell^0 = \hat{v}_\ell$. Now compute the change in ϵ_ℓ across one step,

$$\begin{aligned}\epsilon_\ell^{i+1} - \epsilon_\ell^i &= v_\ell^i - v_\ell^{i-1} \\ &= dv_\ell^i \\ &= \epsilon_\ell^i - \epsilon_{\ell+1}^i \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}.\end{aligned}$$

Note that this equation is only valid for $i \geq 1$ due to the $i - 1$ term (v_ℓ^{-1} is not defined). Adding ϵ_ℓ^i to both sides of the resulting equation gives

$$\epsilon_\ell^{i+1} = \epsilon_{\ell+1}^i \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}.$$

We now use induction to prove that $\epsilon_\ell = \delta_\ell$ after $n = L$ iterations. Indeed, we prove a stronger claim that $\epsilon_\ell^i = \delta_\ell$ at $i = L - \ell + 1$. First note that $\epsilon_L^i = \delta_L$ for all i because ϵ_L^i is initialized to δ_L and then never changed. Therefore, our claim is true for the base case $\ell = L$.

Now suppose that $\epsilon_{\ell+1}^i = \delta_{\ell+1}$ for $i = L - (\ell + 1) + 1 = L - \ell$. We need to show that $\epsilon_\ell^{i+1} = \delta_\ell$. From above, we have

$$\begin{aligned}\epsilon_\ell^{i+1} &= \epsilon_{\ell+1}^i \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ &= \delta_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ &= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_{\ell+1}} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell} \\ &= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_{\ell+1}} \frac{\partial \hat{v}_{\ell+1}}{\partial \hat{v}_\ell} \\ &= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \\ &= \delta_\ell.\end{aligned}$$

This completes our induction argument. It follows that $\epsilon_\ell^i = \delta_\ell$ at iteration $i = L - \ell + 1$ at all layers $\ell = 1, \dots, L$. The last layer to be updated to the correct value is $\ell = 1$, which is updated on iteration number $i = L - 1 + 1 = L$. Hence, $\epsilon_\ell = \delta_\ell$ for all $\ell = 1, \dots, L$ after $n = L$ iterations. This proves the first statement in our theorem. The second statement then follows from the definition of $d\theta_\ell$,

$$\begin{aligned}d\theta_\ell &= -\epsilon_\ell \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell} \\ &= -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \frac{\partial f_\ell(\hat{v}_{\ell-1}; \theta_\ell)}{\partial \theta_\ell} \\ &= -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{v}_\ell} \frac{\partial \hat{v}_\ell}{\partial \theta_\ell} \\ &= -\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \theta_\ell}.\end{aligned}$$

This completes the proof. □

This theorem ties together the implementation and formulation of predictive coding from [10] (*i.e.*, Algorithm 3) to the results in [12, 13]. The results of the theorem are illustrated empirically by the

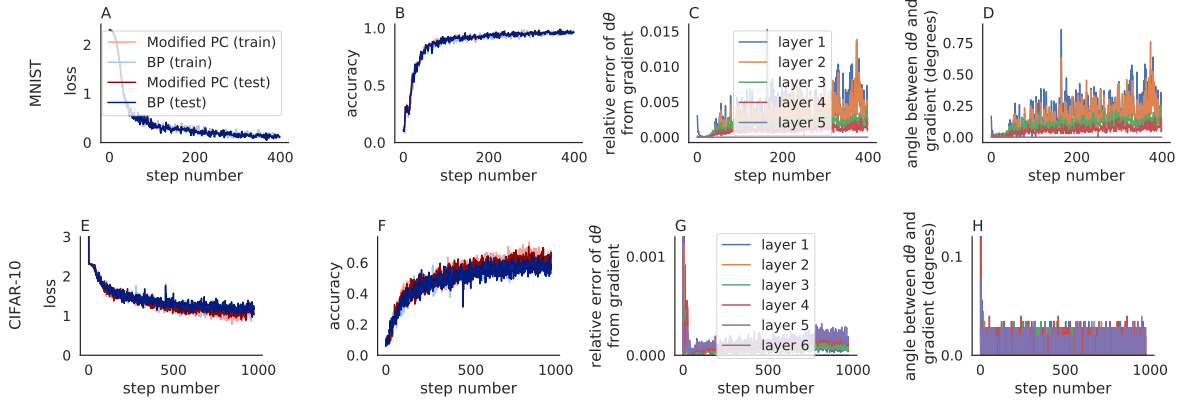


Figure 5: **Predictive coding modified by the fixed prediction assumption with $\eta = 1$ compared to backpropagation in convolutional neural networks trained on MNIST and CIFAR-10.** Same as Fig. 3 except I set $\eta = 1$ and $n = L$.

convergence of the gradients shown in the rightmost panels of Fig. 4. To further test the result empirically, I repeated Fig. 3 with $\eta = 1$ and $n = L$ (in contrast to Fig. 3 which used $\eta = 0.1$ and $n = 20$). The loss and accuracy closely match those computed by backpropagation (Fig. 5A,B,E,F). More importantly, the parameter updates closely matched the true gradients (Fig. 5C,D,G,H), as predicted by Theorem 1.

The differences between predictive coding and backpropagation in Fig. 5 are due floating point errors and the non-determinism of computations performed GPUs. For example, similar differences to those seen in Fig. 5A,B are present when the same training algorithm is run twice with the same random seed. The smaller number of iterations ($n = L$ in Figure 5 versus $n = 20$ in Figure 3) resulted in a shorter training time (13s for MNIST and 300s for CIFAR-10 for modified PC in Fig. 5, compare to 29s and 848s in Fig. 3, and compare to 8s and 58s for backpropagation).

In summary, a review of the literature shows that a strict interpretation of predictive coding (Algorithm 2) does not converge to the true gradients computed by backpropagation. To compute the true gradients, predictive coding must be modified by the fixed prediction assumption (Algorithm 3). Further, I proved that Algorithm 3 computes the exact gradients if we set $\eta = 1$ and $n \geq L$, which ties together results from previous work [10, 12, 13].

3 Predictive coding with the fixed prediction assumption and $\eta = 1$ is functionally equivalent to a direct implementation of backpropagation.

The proof of Theorem 1 and the last panel of Fig. 4 give some insight into how Algorithm 3 works. First note that the values of v_ℓ in Algorithm 3 are only used to compute the values of ϵ_ℓ and are not otherwise used in the computation of $d\theta_\ell$ or any other quantities. Therefore, if we only care about understanding parameter updates, $d\theta_\ell$, we can ignore the values of v_ℓ and only focus on how ϵ_ℓ is updated on each iteration, i . Secondly, note that when $\eta = 1$, each ϵ_ℓ is updated only once: $\epsilon_\ell^i = 0$ for $i < L - \ell + 1$ and $\epsilon_\ell^i = \epsilon_{\ell+1}^i \partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1}) / \partial \hat{v}_\ell$ for $i \geq L - \ell + 1$, so ϵ_ℓ is only changed on iteration number $i = L - \ell + 1$. In other words, the error computation in Algorithm 3 when $\eta = 1$ is equivalent to

$$\begin{aligned} & \# \text{ error computation} \\ \epsilon_L &= \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L} \end{aligned}$$

```

for  $i = 1, \dots, n$ 
  for  $\ell = L - 1, \dots, 1$ 
    if  $\ell == L - i + 1$ 
       $\epsilon_\ell = \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}$ 

```

The two computations are equivalent in the sense that they compute the same values of the errors, ϵ_ℓ^i , on every iteration. The formulation above makes it clear that the nested loops are unnecessary because for each value of i , ϵ_ℓ is only updated at one value of ℓ . Therefore, the nested loops and if-statement can be replaced by a single for-loop. Specifically, the error computation in Algorithm 3 when $\eta = 1$ is equivalent to

```

# error computation
 $\epsilon_L = \frac{\partial \mathcal{L}(\hat{v}_L, y)}{\partial \hat{v}_L}$ 
for  $\ell = L - 1, \dots, 1$ 
   $\epsilon_\ell = \epsilon_{\ell+1} \frac{\partial f_{\ell+1}(\hat{v}_\ell; \theta_{\ell+1})}{\partial \hat{v}_\ell}$ 

```

This is *exactly* the error computation from the standard backpropagation algorithm, *i.e.*, Algorithm 1. Hence, if we use $\eta = 1$, then Algorithm 3 is just backpropagation with extra steps and these extra steps do not compute any non-zero values. If we additionally want to compute the fixed point beliefs, then they can still be computed using the relationship

$$v_\ell = \epsilon_\ell + \hat{v}_\ell.$$

We conclude that, when $\eta = 1$, Algorithm 3 can be replaced by an exact implementation of backpropagation without any effect on the results or effective implementation of the algorithm.

4 Prediction errors do not necessarily represent surprising or unexpected features of inputs.

Deep neural networks are often interpreted as abstract models of cortical neuronal networks. To this end, the activations of units in deep neural networks are compared to the activity (typically firing rates) of cortical neurons [1, 25, 26]. This approach ignores the representation of errors within the network. More generally, the activations in one particular layer of a feedforward deep neural network contain no information about the activations of deeper layers, the label, or the loss. On the other hand, the activity of cortical neurons can be modulated by downstream activity and information believed to be passed upstream by feedback projections. Predictive coding provides a precise model for the information that deeper layers send to shallower layers, specifically prediction errors.

Under the fixed prediction assumption (Algorithm 3), prediction errors in a particular layer are approximated by the gradients of that layers' activations with respect to the loss function, $\epsilon_\ell = \delta_\ell = \frac{\partial \mathcal{L}}{\partial \hat{v}_\ell}$, but under a strict interpretation of predictive coding (Algorithm 2), prediction errors do not necessarily reflect gradients. I next empirically explored how the representations of images differ between the activations from a feedforward pass, \hat{v}_ℓ , the prediction errors under the fixed prediction assumption, $\epsilon_\ell = \delta_\ell$, as well as the beliefs, v_ℓ , and prediction errors, ϵ_ℓ , under a strict interpretation of predictive coding (Algorithm 2). To do so, I computed each quantity in VGG-19 [27], which is a large, feedforward convolutional neural network (19 layers and 143,667,240 trainable parameters) pre-trained on ImageNet [28].

The use of convolutional layers allows us to visualize the activations and prediction errors in each layer. Specifically, I took the Euclidean norm of each quantity across all channels and plotted them as two-dimensional images for layers $\ell = 1$ and $\ell = 10$ and for two different input images (Fig. 6).

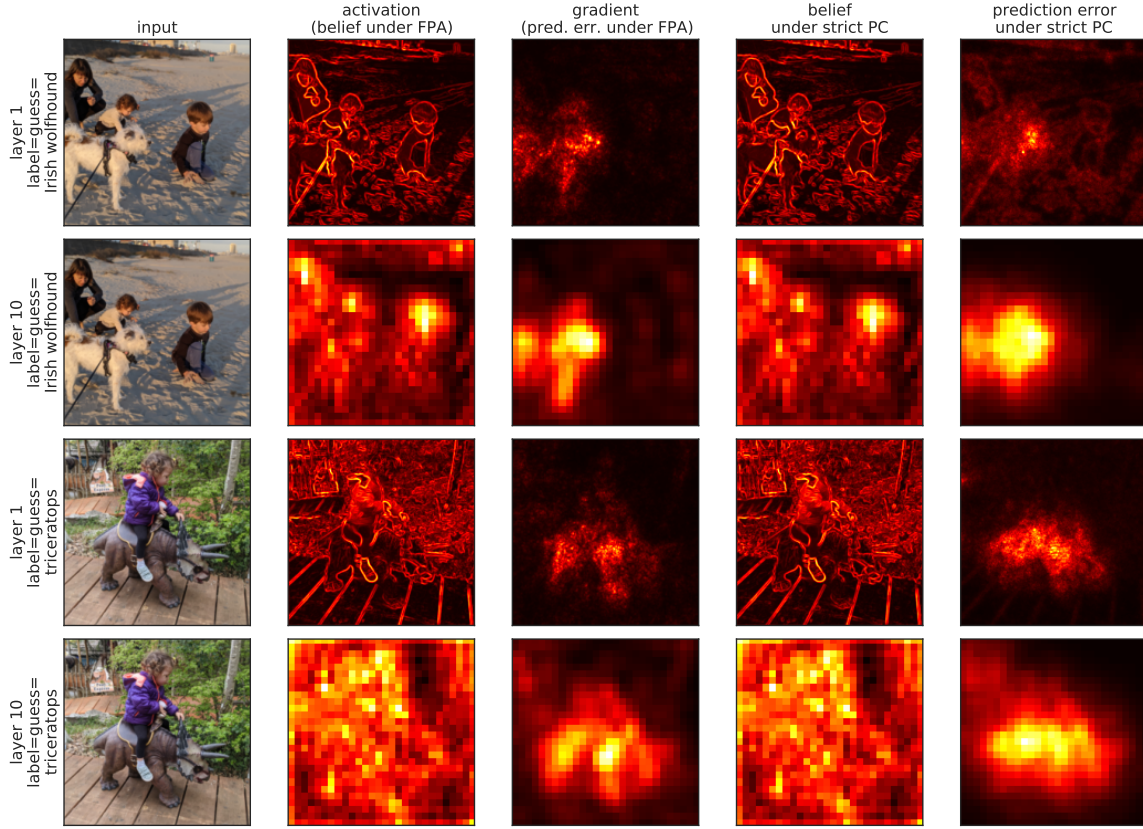


Figure 6: **Magnitude of activations, beliefs, and prediction errors in a convolutional neural network pre-trained on ImageNet.** The Euclidean norm of feedforward activations (\hat{v} , interpreted as beliefs under the fixed prediction assumption), gradients of the loss with respect to activations ($\delta_\ell = \partial \mathcal{L} / \partial \hat{v}$, interpreted as prediction errors under the fixed prediction assumption), beliefs (v) under strict predictive coding, and prediction errors (ϵ_ℓ) under strict predictive coding computed from the VGG-19 network [27] pre-trained on ImageNet [28] with two different photographs as inputs at two different layers. The vertical labels on the left (“triceratops” and “Irish wolfhound”) correspond to the guessed label which was also used as the “true” label (y) used to compute the gradients.

Under predictive coding with the fixed prediction assumption, we can interpret the activations, \hat{v}_ℓ , as “beliefs” and the gradients, δ_ℓ , as “prediction errors.” Strictly speaking, there is a distinction between the beliefs, \hat{v}_ℓ , from a feedforward pass and the beliefs, $v_\ell = \hat{v}_\ell + \epsilon_\ell$, when labels are provided. Either could be interpreted as a “belief.” However, I found that the difference between them was negligible for the examples considered here.

Overall, the activations, \hat{v}_ℓ , from a feedforward pass were qualitatively very similar to the beliefs, v_ℓ , computed under a strict interpretation of predictive coding (Algorithm 2). To a slightly lesser degree, the gradients, δ_ℓ , from a feedforward pass were qualitatively similar to the prediction errors computed under a strict interpretation of predictive coding (Algorithm 2). Since \hat{v}_ℓ and δ_ℓ approximate beliefs and prediction errors under the fixed prediction assumption, these observations confirm that the fixed prediction assumption does not make large qualitative changes to the representation of beliefs and errors in these examples. Therefore, in the discussion below, I use “beliefs” and “prediction errors” to refer to the quantities from both models.

Interestingly, prediction errors are non-zero even when the image and the network’s “guess” is consistent with the label (no “mismatch”). Indeed, the prediction errors are largest in magnitude at pixels corresponding to the object predicted by the label, *i.e.*, at the most predictable regions. While this observation is an obvious consequence of the fact that prediction errors are approximated by the gradients, $\delta_\ell = \frac{\partial \mathcal{L}}{\partial \hat{v}_\ell}$, it seems contradictory to the heuristic or intuitive interpretation of prediction errors as measurements of “surprise” in the colloquial sense of the word [15].

As an illustrative example from Fig. 6, it is not surprising that an image labeled by “triceratops” contains a triceratops, but this does not imply a lack of prediction errors because the space of images containing a triceratops is large and any one image of a triceratops is not wholly representative of the label. Moreover, the pixels to which the loss is most sensitive are those pixels containing the triceratops. Therefore those pixels give rise to larger values of $\epsilon_\ell \approx \delta_\ell = \partial \mathcal{L} / \partial \hat{v}_\ell$. Hence, in high-dimensional sensory spaces, predictive coding models do not necessarily predict that prediction error units encode “surprise” in the colloquial sense of the word.

In both examples in Fig. 6, I used an input, y , that matched the network’s “guessed” label, *i.e.*, the label to which the network assigned the highest probability ($\text{argmax}(\hat{y})$). Prediction errors are often discussed in the context of mismatched stimuli in which top-down input is inconsistent with bottom-up predictions [29–34]. Mismatches can be modeled by taking a label that is different from the network’s guess. In Fig. 7, I visualized the prediction errors in response to matched and mismatched labels. The network assigned a probability of $p = 0.9991$ to the label “carousel” and a probability of $p = 3.63 \times 10^{-8}$ to the label “bald eagle”. When I applied the mismatched label “bald eagle,” prediction errors were larger in pixels that are salient for that label (*e.g.*, the bird’s white head, which is a defining feature of a bald eagle). Moreover, the prediction errors as a whole are much larger in magnitude in response to the mismatched label (see the scales of the color bars in Fig. 7).

In summary, the relationship between prediction errors and gradients helps demonstrate that prediction errors sometimes, but do not always conform to their common interpretation as unexpected features of a bottom-up input in the context of a top-down input. Also, beliefs and prediction errors were qualitatively similar with and without the fixed prediction assumption for the examples considered here.

5 Torch2PC software for predictive coding with PyTorch models

The figures above were all produced using PyTorch [35] models combined with custom written functions for predictive coding. The functions are collected in the GitHub repository Torch2PC (“torch to predictive coding”) which contains functions for predictive coding with PyTorch models. Currently, the only

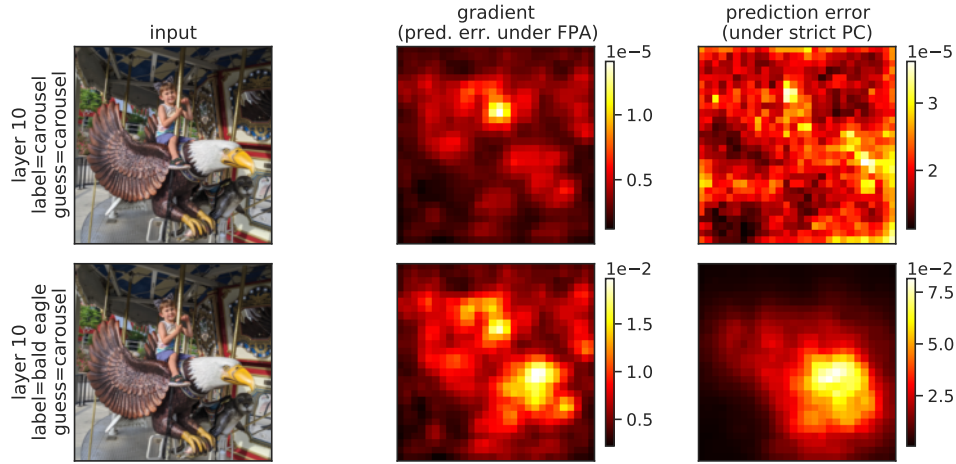


Figure 7: **Magnitude of activations, beliefs, and prediction errors in response to matched and mismatched inputs and labels.** Same as Fig. 6, but for the bottom row the label did not match the network's guess.

available functions are intended for models built using the Sequential class, but more general functions will be added to Torch2PC in the future. The functions can be imported using the following commands

```
!git clone https://github.com/RobertRosenbaum/Torch2PC.git
from Torch2PC import TorchSeq2PC as T2PC
```

The primary function in TorchSeq2PC is PCInfer, which performs one predictive coding step (computes one value of $d\theta$) on a batch of inputs and labels. The function takes an input ErrType, which is a string that determines whether to use a strict interpretation of predictive coding (Algorithm 2; ErrType="Strict"), predictive coding with the fixed prediction assumption (Algorithm 3; "FixedPred"), or to compute the gradients exactly using backpropagation (Algorithm 1; "Exact"). Algorithm 2 can be called as follows,

```
vhat, Loss, dLdy, v, epsilon =
    T2PC.PCInfer(model, LossFun, X, Y, "Strict", eta, n, vinit)
```

where model is a Sequential PyTorch model, LossFun is a loss function, X is a mini-batch of inputs, Y is a mini-batch of labels, eta is the step size, n is the number of iterations to use, and vinit is the initial value for the beliefs. If vinit is not passed, it is set to the result from a forward pass, vinit=vhat. The function returns a list of activations from a forward pass at each layer as vhat, the loss as Loss, the gradient of the output with respect to the loss as dLdy, a list of beliefs, v_ℓ , at each layer as v, and a list of prediction errors, ϵ_ℓ , at each layer as epsilon. The values of the parameter updates, $d\theta_\ell$, are stored in the grad attributes of each parameter, model.param.grad. Hence, after a call to PCInfer, gradient descent could be implemented by calling

```
with torch.no_grad():
    for p in modelPC.parameters():
        p-=eta*p.grad
```

Alternatively, an arbitrary optimizer could be used by calling

```
optimizer.step()
```

where `optimizer` is an optimizer created using the PyTorch `optim` class, *e.g.*, by calling `optimizer=optim.Adam(model.parameters())` before the call to `T2PC.PCInfer`.

The input `model` should be a PyTorch Sequential model. Each layer is treated as a single predictive coding layer. Multiple functions can be included within the same layer by wrapping them in a separate call to `Sequential`. For example the following code:

```
model=model=nn.Sequential(
    nn.Conv2d(1,10,3),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(10,10,3),
    nn.ReLU()
)
```

will treat each item as its own layer (5 layers in all). To treat each "convolutional block" as a separate layer, instead do

```
model=nn.Sequential(
    nn.Sequential(
        nn.Conv2d(1,10,3),
        nn.ReLU(),
        nn.MaxPool2d(2)
    ),
    nn.Sequential(
        nn.Conv2d(10,10,3),
        nn.ReLU()
    )
)
```

which has just 2 layers.

Algorithm 3 can be called as follows,

```
vhat, Loss, dLdy, v, epsilon=
    T2PC.PCInfer(model, LossFun, X, Y, "FixedPred", eta, n)
```

The input `vinit` is not used for Algorithm 3, so it does not need to be passed in. The exact values computed by backpropagation can be obtained by calling

```
vhat, Loss, dLdy, v, epsilon=
    T2PC.PCInfer(model, LossFun, X, Y, "Exact")
```

The inputs `vinit`, `eta`, and `n` are not used for computing exact gradients, so they do not need to be passed in. Theorem 1 says that

```
T2PC.PCInfer(model, LossFun, X, Y, "FixedPred", eta=1, n=len(model))
```

computes the same values as

```
T2PC.PCInfer(model, LossFun, X, Y, "Exact")
```

up to numerical floating point errors. The inputs `eta`, `n`, and `vinit` are optional. If they are omitted by calling

```
T2PC.PCInfer(model, LossFun, X, Y, ErrType)
```

then they default to `eta=.1`, `n=20`, `vinit=None` which produces `vinit=vhat` when `ErrType="Strict"`. More complete documentation and a complete example is provided as `SimpleExample.ipynb` in the GitHub repository and in the code accompanying this paper. More examples are provided by the code accompanying each figure above.

6 Discussion

I reviewed and slightly extended previous work [10–13] on the relationship between predictive coding and backpropagation for learning in neural networks. I found that a strict interpretation of predictive coding does not accurately approximate backpropagation, but is still capable of learning (Figs. 1 and 2). Previous work proposed a modification to predictive coding called the “fixed prediction assumption” which causes predictive coding to converge to the same parameter updates produced by backpropagation, under the assumption that the predictive coding iterations converge to fixed points.

Hence, the relationship between predictive coding and backpropagation identified in previous work relies critically on the fixed prediction assumption. Formal derivations of predictive coding in terms of approximate variational inference [19] does not produce the fixed prediction assumption. It is possible that an alternative probabilistic model or alternative approaches to the variational formulation could help formalize a model of predictive coding under the fixed prediction assumption.

I proved analytically and verified empirically that taking a step size of $\eta = 1$ in the modified predictive coding algorithm computes the exact gradients computed by backpropagation in a fixed number of steps (modulo floating point numerical errors). This result is consistent with similar, but slightly less general, results in previous work [12, 13].

A closer inspection of the algorithm that results from taking $\eta = 1$ under the fixed prediction assumption shows that it is functionally equivalent to a direct implementation of backpropagation. As such, the neural architecture and machinery needed to implement predictive coding with the fixed prediction assumption could also implement backpropagation directly. These results call into question whether predictive coding with the fixed prediction assumption is any more biologically plausible than a direct implementation of backpropagation.

Visualizing the beliefs and prediction errors produced by predictive coding models applied to a large convolutional neural network pre-trained on ImageNet showed that beliefs and prediction errors were activated by distinct parts of input images, and the parts of the images that produced larger prediction errors were not always consistent with an intuitive interpretation of prediction errors as representing surprising or unexpected features of inputs.

When interpreting artificial deep neural networks as models of biological neuronal networks, it is common to compare activations in the artificial network to biological neurons’ firing rates [25, 26]. However, under predictive coding models and other models in which errors are propagated upstream by feedback connections, many biological interpretations posit the existence of “error neurons” that encode the errors sent upstream. In most such models (including predictive coding), error neurons reflect or approximate the gradient of the loss function with respect to artificial neurons’ activations, δ_ℓ . Any model that hypothesizes the neural representation of backpropagated errors would predict that some recorded

neural activity should reflect these errors. Therefore, if we want to draw analogues between artificial and biological neural networks, the activity of biological neurons should be compared to both the activations *and* the gradients of artificial neurons.

The Brain-Score project provides an ideal environment for such a comparison. Brain-Score is a framework for directly comparing the activity of deep neural networks to cortical recordings from visual cortex [25, 26]. Several deep neural networks have been tested in Brain-Score, including VGG-19, but these tests only compare activations to the activity of biological neurons. In upcoming work, I will repeat the testing of VGG-19 in Brain-Score, but also include the gradients, δ_ℓ , and (where possible) prediction errors, ϵ_ℓ , in the comparisons.

Following previous work [10, 11], I took the covariance matrices underlying the probabilistic model to be identity matrices, $\Sigma_\ell = I$, when deriving the predictive coding model. Even with this assumption, predictive coding produces similar learning performance to backpropagation, which suggests that relaxing this assumption could lead to improvements over backpropagation. Future work will focus on these possibilities. Previous work derives local learning rules that account for non-diagonal covariance matrices [20].

Predictive coding and deep neural networks (trained by backpropagation) are often viewed as competing models of brain function. However, their relationships studied here suggest that they can be viewed as two different ways of looking at the same or similar learning algorithms. Understanding their relationship can help in the interpretation and implementation of each algorithm and their mutual relationships to biological neuronal networks.

References

- [1] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.
- [2] J. C. Whittington and R. Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019.
- [3] R. Urbanczik and W. Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.
- [4] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):1–10, 2016.
- [5] B. Scellier and Y. Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.
- [6] J. Aljadeff, J. D’amour, R. E. Field, R. C. Froemke, and C. Clopath. Cortical credit assignment by hebbian, neuromodulatory and inhibitory plasticity. *arXiv preprint arXiv:1911.00307*, 2019.
- [7] D. Kunin, A. Nayebi, J. Sagastuy-Brena, S. Ganguli, J. Bloom, and D. Yamins. Two routes to scalable credit assignment without weight symmetry. In *International Conference on Machine Learning*, pages 5511–5521. PMLR, 2020.
- [8] A. Payeur, J. Guerguiev, F. Zenke, B. A. Richards, and R. Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature Neuroscience*, pages 1–10, 2021.
- [9] D. G. Clark, L. Abbott, and S. Chung. Credit assignment through broadcasting a global error vector. *arXiv preprint arXiv:2106.04089*, 2021.

- [10] B. Millidge, A. Tschantz, and C. L. Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*, 2020.
- [11] J. C. 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.
- [12] T. Salvatori, Y. Song, T. Lukasiewicz, R. Bogacz, and Z. Xu. Predictive coding can do exact backpropagation on convolutional and recurrent neural networks. *arXiv preprint arXiv:2103.03725*, 2021.
- [13] Y. Song, T. Lukasiewicz, Z. Xu, and R. Bogacz. Can the brain do backpropagation?—exact implementation of backpropagation in predictive coding networks. *Advances in Neural Information Processing Systems*, 33:22566, 2020.
- [14] R. P. 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.
- [15] K. Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010.
- [16] Y. Huang and R. P. Rao. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593, 2011.
- [17] 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.
- [18] A. Clark. *Surfing uncertainty: Prediction, action, and the embodied mind*. Oxford University Press, 2015.
- [19] C. L. Buckley, C. S. Kim, S. McGregor, and A. K. Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017.
- [20] R. Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017.
- [21] M. W. Spratling. A review of predictive coding algorithms. *Brain and cognition*, 112:92–97, 2017.
- [22] G. B. Keller and T. D. Mrsic-Flogel. Predictive processing: a canonical cortical computation. *Neuron*, 100(2):424–435, 2018.
- [23] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep Learning*. MIT press Cambridge, 2016.
- [24] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.
- [25] M. Schrimpf, J. Kumbusi, H. Hong, N. J. Majaj, R. Rajalingham, E. B. Issa, K. Kar, P. Bashivan, J. Prescott-Roy, F. Geiger, K. Schmidt, D. L. K. Yamins, and J. J. DiCarlo. Brain-score: Which artificial neural network for object recognition is most brain-like? *bioRxiv preprint*, 2018.
- [26] M. Schrimpf, J. Kumbusi, M. J. Lee, N. A. R. Murty, R. Ajemian, and J. J. DiCarlo. Integrative benchmarking to advance neurally mechanistic models of human intelligence. *Neuron*, 2020.

- [27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [29] L. Hertäg and H. Sprekeler. Learning prediction error neurons in a canonical interneuron circuit. *Elife*, 9:e57541, 2020.
- [30] C. J. Gillon, J. E. Pina, J. A. Lecoq, R. Ahmed, Y. Billeh, S. Caldejon, P. Groblewski, T. M. Henley, E. Lee, J. Luviano, et al. Learning from unexpected events in the neocortical microcircuit. *bioRxiv*, 2021.
- [31] G. B. Keller, T. Bonhoeffer, and M. Hübener. Sensorimotor mismatch signals in primary visual cortex of the behaving mouse. *Neuron*, 74(5):809–815, 2012.
- [32] P. Zmarz and G. B. Keller. Mismatch receptive fields in mouse visual cortex. *Neuron*, 92(4):766–772, 2016.
- [33] A. Attinger, B. Wang, and G. B. Keller. Visuomotor coupling shapes the functional development of mouse visual cortex. *Cell*, 169(7):1291–1302, 2017.
- [34] J. Homann, S. A. Koay, A. M. Glidden, D. W. Tank, and M. J. Berry. Predictive coding of novel versus familiar stimuli in the primary visual cortex. *BioRxiv*, page 197608, 2017.
- [35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019.