# Generalized Predictive Coding: Bayesian Inference in Static and Dynamic models

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Predictive coding networks (PCNs) have an inherent degree of biological plausibility and perform approximate backpropagation of error in supervised settings. It is less clear how predictive coding compares to state-of-the-art architectures, such as VAEs in unsupervised and probabilistic settings. We propose a generalized PCN that, like its' inspiration in neuroscience, parameterizes hierarchical latent distributions under the Laplace approximation and maximises model evidence via iterative inference using local precision weighted error signals. Unlike its inspiration it uses multi-layer networks with nonlinearities between latent distributions. We compare our model to VAE and VLAE baselines on three different image datasets and find that generalized predictive coding shows performance comparable to variational autoencoders with exact error backpropagation. Finally, we evaluate the possibility of learning temporal dynamics via static prediction by encoding sequences of states in generalized coordinates of motion.

## 1 Introduction

Predictive coding is an influential theory in neuroscience that describes brain function as maintaining a generative model of the world by minimising prediction errors about sensory and internal states [3, 19]. Static PCNs are organised hierarchically, where top-down signals from higher layers predict the activity of the layer below and bottom-up signals convey prediction errors. In dynamical predictive coding models each layer additionally predicts temporal changes of expected neural activity in the lower layer. Given these dynamics, Hebbian weight updates can be defined that minimize the prediction error at each layer of the network.

The weight and activity update dynamics of PCNs can be interpreted as performing variational inference (VI) by iteratively refining an inferred distribution over possible causes $p(z|o)$ of observed sensory data $o$ [6, 5, 22]. In variational inference, an approximate distribution $q(z;\lambda)$ is fit to the generally intractable posterior $p_\theta(z \mid o)$ by optimizing the variational Free Energy $\mathcal{F}$, also known as evidence lower-bound (ELBO) [6, 7]: $\mathcal{F}_\theta(o;\lambda) = \mathbb{E}_{q(z;\lambda)}\left[\ln p_\theta(o,z) - \ln q(z;\lambda)\right]$

In predictive coding, we define $q(z;\lambda)$ to be a simple diagonal or full-covariance Gaussian distribution with $\lambda$ as the sufficient parameters, i.e. the mean and covariance. Given the generative model $\theta$ (decoder) of a particular hierarchical layer, inference in predictive coding models proceeds by estimating the optimal variational parameters $\lambda^*$ that maximize model evidence given observed data and current parameterization. Learning of the parameters of the generative model $\theta$ can be achieved by performing a gradient descent on $\mathcal{F}_\theta\left(o;\lambda^*\right)$ with respect to $\theta$ which results in Hebbian weight updates. Crucially, learning and inference in PCNs are driven by locally generated predictions and prediction errors. In hierarchical PCNs, the predicted distributions of higher layers foster empirical

Submitted to 4th Workshop on Shared Visual Representations in Human and Machine Visual Intelligence (SVRHM) at NeurIPS 2022. Do not distribute.

priors for the next lower layer: $p(z, o) = p(o \mid z_1)\, p(z_1 \mid z_2) \ldots p(z_{L-1} \mid z_L)$, such that a layer's inference model can be interpreted as the next higher layer's generative model.

The variational autoencoder (VAE) is a highly influential class of deep neural networks that performs amortized inference of $\lambda$ using an inference model $\phi$ (encoder) [10]. The inference model in VAEs learns to predict the approximate posterior $q_\phi(z \mid o)$ by learning the parameters $\phi$ of the variational mapping over a dataset. In contrast to the Hebbian updates in PCNs, VAEs are trained using exact backpropagation of error through the entire model [21]. More recently, the notion of iterative inference has been adapted for VAEs to improve the posterior distribution [16, 13]. While VAEs and static PCNs have striking similarities in terms of architecture and optimisation scheme, there is still a lack of quantitative comparison. Similarly, various deep recurrent models for predicting sequential stimuli have been developed, but lack exhaustive comparison to dynamical PCNs [2, 20, 8].

## 2 Generalized predictive coding

Generalized predictive coding (GPC) describes a particularly influential class of PCNs that covers static and dynamic models in combination with generalized coordinates of state motion and the Laplace approximation [5, 6, 4, 1]. Static GPC networks maintain cause states $v$ and hidden states $x$. Each hierarchical layers predicts the expected activity in the next lower layer using non-linear function $g$: $y = g(x, v, \theta) + z$. Dynamical GPC networks additionally predict the motion of hidden states $\dot{x} = f(x, v, \theta) + w$ using a non-linear transition function $f$[1]. $z$ and $w$ denote observation noise and transition noise respectively. While cause states are predicted hierarchically, hidden states are usually not observed by higher layers.

Under the assumption of local linearity, GPC uses states in generalized coordinates of motion $\tilde{y} = [y, y', y'', \ldots]^{\mathrm{T}}$, where $y'$ denotes the temporal derivative at $y$. Similarly, for cause and hidden states:

$$
\begin{aligned}
y &= g(x, v) + z & x' &= f(x, v) + w \\
y' &= g_x x' + g_v v' + z' & x'' &= f_x x' + f_v v' + w'
\end{aligned}
$$

Using Gaussian priors $p(z) = \mathcal{N}(z; \bar{\mu}, \Sigma)$, GPC infers posterior distributions of the causes $p(\tilde{x} \mid \tilde{v}) = N\left(D\tilde{x} : \tilde{f}, \tilde{\Sigma}^z\right)$ and the hidden states $p(\tilde{y} \mid \tilde{x}, \tilde{v}) = N\left(\tilde{y} : \tilde{g}, \tilde{\Sigma}^z\right)$. Here, $D$ denotes a derivative operator that replaces each order of state motion with the next higher order: $x \leftarrow x', x' \leftarrow x'', \ldots$.

While mean parameters $\mu$ are encoded explicitly, the covariance $\Sigma$ is encoded implicitly as a function of the mean using the Laplace approximation (LA). Under the LA, the covariance is determined by the local curvature of $\log p_\theta(y, v, x)$ at the inferred mode of $p_\theta(v, x \mid y)$. Figure 1 shows dynamical and static GPC in comparison to the VAE architecture.

GPC proceeds by expressing the VFE of each hierarchical layer $l$ as a function of precision weighted prediction errors $\xi^{(l)o} = \Sigma^{(l)o-1}\epsilon^{(l)o}$ for outgoing predictions and for top-down predictions $\xi^{(l)v} = \Sigma^{(l)v-1}\epsilon^{(l)v}$ from the next higher layer. Depending on the chosen prior, the distance between prior and posterior distribution is measured by $\xi^{(l)p} = \Sigma^{(l)p-1}\epsilon^{(l)p}$. Here, $\epsilon^p = \mu - \bar{\mu} = \mu - 0$, is the prediction error between posterior and prior and $\epsilon^{(l)v} = \mu^{(l)} - g(\mu^{(l+1)})$ is the hierarchical prediction error between layers (or the sensory prediction error at the lowest layer). For dynamical models, the generalized predictions $\tilde{y}$ result in generalized errors $\tilde{\epsilon} = \tilde{y} - \tilde{o} = [\epsilon, \epsilon', \epsilon'', \ldots]^{\mathrm{T}}$. Inference in each layer is done via gradient descent on $\xi = \Sigma^{-1}\epsilon$ for cause states $\dot{\tilde{\mu}}^{(l)v} = \tilde{\mu}^{(l)v} - \tilde{\varepsilon}_v^{(l)T}\xi^{(l)} - \xi^{(l+1)v}$ [5]. Within each hierarchical layer, the motion of hidden states is inferred as $\dot{\tilde{\mu}}^{(l)x} = D\tilde{\mu}^{(l)x} - \tilde{\varepsilon}_x^{(l)T}\xi^{(l)}$. Section G of the Appendix provides a more in-depth explanation of the simplified VFE objective.

Our particular implementation uses Gauss-Newton updates $\dot{\mu} = \left(W_z^T W_z + I\right)^{-1} W_z^T \varepsilon \mid_{z=\mu}$ where prediction errors are weighted by $\Sigma^{-1} = {W_z}^T W_z + I \mid_{z=\mu}$, and $W_z$ is the Jacobian of the decoder network $W$. We perform exact error propagation strictly locally within each hierarchical layer using automatic differentiation in PyTorch [7, 17]. This is in contrast to the exact backpropagation pass

[1] Since static models lack dynamical predictions, hidden states can usually be ignored in the static case.

over all network parameters, such as in VAEs, where the decoder's prediction errors directly drive updates of encoder parameters.

A particular appeal of GPC is that it provides a simple model inversion scheme that might be implemented in the brain [5, 1]. Encoding state motion in generalized coordinates casts dynamical predictions (which normally require complex recurrent connectivity) as a static prediction mechanism. Similarly, the Laplace approximation provides a simplified scheme to represent uncertainty.

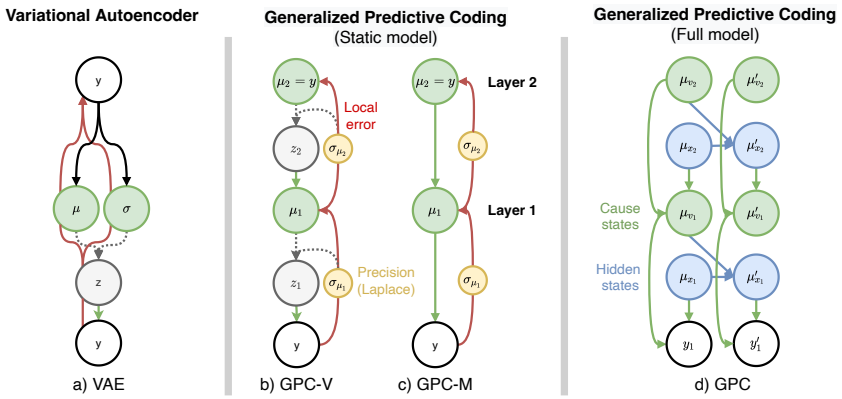



Figure 1: Variational autoencoders (a) encode mean and variance of their latent distribution. Error signals are propagated through the entire network via the backpropagation algorithm. Generalized predictive coding (a-d) propagates local errors and encodes only the mean under the Laplace approximation. The variance is a function of the mean and can be explicitly sampled (b) or appears only as error weighting terms (c).

# 3 Implemented models and baselines

## 3.1 VAE and VLAE baseline

We use the conventional VAE architecture with fully factorized normal distribution, reparameterization of the latent distribution and trained via backpropagation of error [10]. It is trained using a single sample from the latent distribution as input to the decoder and the regularization with a standard normal. The VLAE is a VAE with iterative mode seeking using the Laplace approximation, i.e. infers a full covariance matrix [16]. The VLAE uses a single sample from the latent distribution at the inferred mode as input to the decoder. Similarly a single sample is used for the regularization with a standard normal. Unlike the cited model, we do not use a decaying learning rate for mode seeking. For VAE and VLAE, encoder and decoder consist of two layers with 256 hidden units and parameterize 16 latent units via weights and biases.

## 3.2 Static GPC model

We implement a static GPC with a hierarchy of two latent distributions and fix the mean of the second layer's distribution to the data. In this setup, the output of the second hierarchical layer provides empirical priors via amortized inference on the first hierarchical layer's cause states $p(v_1)$ [2]. The resulting structure resembles that of an autoencoder by interpreting the second layer's generative model as the first layer's inference model. The PC architectures differ only in terms of sampling and loss function: Both decoders consist of two layers with 256 hidden units that parameterize 16 latent units. Unlike VAE and VLAE, the PC models do not use biases. Inspired by the work of Park et al. [16], we employ ReLU nonlinearity after the first two hidden layers, followed by a linear output layer. Similar to Park et al. [16] we compute the precision (inverse covariance) of cause states $\Sigma^{-1} = -\nabla_z^2 \log p_\theta(o, z)\big|_{z=\mu}$ by computing activation masks $\mathrm{ReLU}(Wz) = O(Wz)$ during the decoder's forward pass and recursively multiplying with the decoder's weights. We compare

[2] Note that the cause state of the second hierarchical layer is fixed, i.e. iterative inference is restricted to the first hierarchical layer.

stochastic updates using a single sample (like the VAE baseline) and deterministic updates using only the analytical posterior. Figure 1 shows the resulting models in detail.

### 3.3 Dynamical GPC model

We also implement dynamical GPC model with two hierarchical layers. Unlike the full GPC model, we do not infer separate cause states and predict hidden states top-down [3]. Again, the data serves as fixed input to the second dynamical layer. The model features 16 latent units. In addition to decoding the states $y = g(x)$, we decode the encoded state motion $y' = g_x x'$ using the Jacobian of the first hierarchical layer's decoder $\partial g_\theta(x)/\partial x$. This requires a simple forward pass through the masked decoder network. Similarly, we decode the predicted hidden state change $x' = f(x)$ for the lowest embedding order and use the Jacobian of the transition network for higher embedding orders $x'' = f_x(x')$, ..., $x^N = f_x(x^{N-1})$. Amortized inference on the hidden states and their motion is possible by repeating the process with the second hierarchical layer's decoder. The resulting GPC model maps each order of state motion to the corresponding order of change in sensory data. We interpret the Taylor series expansion underlying the forward and inverse embedding of sequential data as convolution operation along the temporal axis, which can efficiently be computed using convolutional kernels, which we provide along we the model's source code. More details and examples for such kernels are shown in section A of the Appendix.

## 4 Static predictive coding on MNIST, FashionMNIST and OMNIGLOT

We train and evaluate models with varying amounts of inference steps on MNIST [12], FashionMNIST [23] and OMNIGLOT [11]. Unlike the VLAE baseline, we do not initialise the decoder output variance based on dataset statistics. Instead we add noise from a standard normal distribution and apply a logit transformation for all datasets. Table 1 shows test results on all datasets for 3 and 6 iterative inference steps using the conventional train and test splits. Listed are mean and standard deviation for 10 runs. We trained for $1e+4$ steps with the ADAM optimiser at a learning rate of 1e-2 [9] and inference learning rate of 0.5, the default setting of the VLAE baseline [16]. In almost all configurations, GPC-S and GPC-M slightly outperform the VAE, while the VLAE model consistently outperforms both PCNs. This indicates that PCNs, despite lacking exact error signals for the inference network learn a generative model that is comparable to the VAE. The GPC-M model without explicit sampling consistently outperformed the sample-based GPC-S model, except for one configuration on OMNIGLOT. In terms of divergence from the prior, the GPC models consistently showed posterior complexity that is comparable to, but slightly higher than, VAE complexity. For more details on model complexity, see Section B of the Appendix. For all tested models, increasing the number of inference steps is beneficial only for low numbers of steps. We found that reducing the inference learning rate or adding a decay term can improve stability, but did not include it in our experiments.

Table 1: Negative evidence lower bound (test set)

| | MNIST | OMNIGLOT | fMNIST | MNIST | OMNIGLOT | fMNIST |
|---|---|---|---|---|---|---|
| Steps | 3 | 3 | 3 | 6 | 6 | 6 |
| VAE | 901.2±1.4 | 1019.4±1.2 | 881.3±0.3 | N/A | N/A | N/A |
| GPC-S | 892.9±1.2 | 1001.1±0.8 | 882.0±1.5 | 896.4±1.1 | 1004.7±0.8 | 883.0±1.0 |
| GPC-M | 892.9±1.0 | 1002.2±0.8 | 880.0±0.4 | 894.7±0.8 | 1003.0±1.4 | 878.4±0.4 |
| VLAE | 881.4±1.2 | 989.3±1.0 | 870.1±0.4 | 877.4±1.4 | 983.1±3.8 | 869.3±0.2 |

## 5 Dynamical predictive coding on the dSprites dataset

In order to assess the possibility to learn dynamics from video we train a dynamical GPC model on a variant of the Disentanglement testing sprites dataset [15]. Most conventional video benchmarks have relatively low sampling rates, where the local linearity assumption does not hold. We generate high

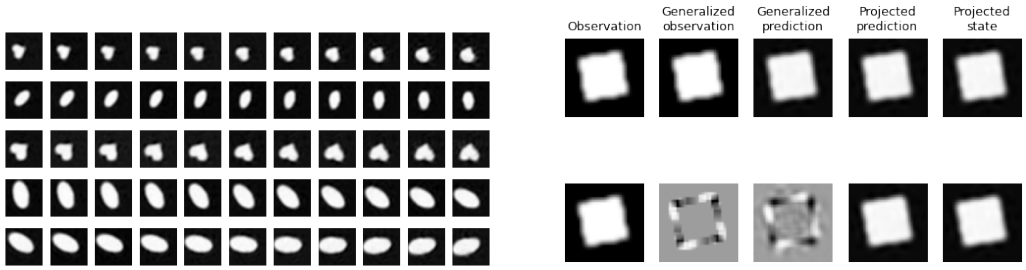
[3] Future work could evaluate more elaborate dynamical models that dynamically infer causes for the observed hidden state dynamics.

resolution videos for a single direction of rotation (counterclockwise) and use random, but constant, values for the remaining latent factors (shape, size, x  y position). We applied Gaussian blur to all images and cropped the videos to 32x32 pixel resolution, making sure that no sprites appear outside the area.

Table 2: Accuracy (dSprites)

| | MSE | Layer 1 | Layer 2 |
|---|---|---|---|
| GPC-all | 0.432±0.124 | 0.768±0.257 | 0.097±0.013 |
| GPC-L1 | 0.476±0.204 | 0.779±0.422 | 0.173±0.031 |

Table 2 shows mean and standard deviation of the mean squared error (MSE) for 10 runs over 3e+4 updates using two different variants of the dynamical model: The GPC-all model was trained using the prediction error from both dynamical layers, while GPC-L1 only considers the error in the lowest dynamical layer. Both models smoothly predict the constant rotation across latent factors. GPC-all shows improved MSE in terms of total and per-layer prediction. This indicates that including higher-order predictions errors propagated through the locally computed Jacobian of the networks indeed improves their accuracy. We found that GPC-L1 reacts poorly to increased latent dimensionality and stops predicting any meaningful state motion when 32 or more latent units were used. In contrast, GPC-all showed meaningful transitions for larger embeddings.




(a) Extrapolation (b) Generalized states

Figure 2: a) Extrapolation with the learned dynamical weights after training both dynamical layers. Shown is every fifth of 50 steps b) Generalized states and their projection to sensory space via decoder weights ("Generalized prediction"). Projection to sequences after ("Projected state") and before mapping to sensory space ("Projected prediction").

Figure 2 a) shows the result of extrapolating hidden states over 50 steps using the learned dynamical weights $f(x)$. While local changes between subsequent frames are intentionally small, both models show smooth rollouts up to 100 steps. Figure 2 b) provides shows adjacent frames and their temporal embedding. Additionally shown are the model's prediction as well as the resulting inverse embedding back to sequential data.

## 6 Conclusion

We presented a generalized predictive coding network (GPC) that uses Hebbian updates and the Laplace approximation with deep generative networks to infer posterior distributions. We have shown that the model performs comparable to VAEs trained with exact error backpropagation. We extended the model to cover dynamical predictions of simple video sequences and demonstrated the possibility to learn dynamics via static prediction using generalized coordinates of motion. Video datasets often have relatively low sampling rate and the local linearity assumption does not generally hold [4]. We found that in many cases, GPC still recovers meaningful dynamics. For this work however, we focus on high resolution data. Important steps for future work could be to use convolutional neural networks or a comparison to deep recurrent models, such as RNNs or LSTMs [8, 20].

[4]Note that physical and biological data such as processed by the brain do not suffer from low sampling rates, but generally are locally smooth.

# References


[1] Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

[2] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

[3] Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

[4] Karl Friston. Hierarchical models in the brain. *PLoS computational biology*, 4(11):e1000211, 2008.

[5] Karl Friston and Stefan Kiebel. Predictive coding under the free-energy principle. *Philosophical transactions of the Royal Society B: Biological sciences*, 364(1521):1211–1221, 2009.

[6] Karl Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of physiology-Paris*, 100(1-3):70–87, 2006.

[7] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, 1993.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[12] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[13] Joseph Marino. Predictive coding, variational autoencoders, and biological connections. *Neural Computation*, 34(1):1–44, 2022.

[14] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[15] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset, 2017. *URL https://github. com/deepmind/dsprites-dataset*, 2020.

[16] Yookoon Park, Chris Kim, and Gunhee Kim. Variational laplace autoencoders. In *International conference on machine learning*, pp. 5032–5041. PMLR, 2019.

[17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[19] Rajesh PN Rao and Dana 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.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[22] Alexander Tschantz, Beren Millidge, Anil K Seth, and Christopher L Buckley. Hybrid predictive coding: Inferring, fast and slow. *arXiv preprint arXiv:2204.02169*, 2022.

[23] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See section F of the Appendix
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code for training, evaluation and preprocessing routines for all experiments are provided as supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section C of the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report mean and standard deviation over ten different runs from random seeds in all experiments.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section E of the appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See section C of the appendix
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Generalized coordinates from discrete sequential data

We compute temporal embeddings of discrete sequential data into derivatives using a Taylor series expansion via central finite differences. We compute the temporal embedding according to a discrete Taylor expansion of form

$$f(x) = \sum_{k=0}^{\infty} \frac{\Delta^k[f](a)}{k!}(x-a)_k \tag{1}$$

for points $x$ around a point $a$ assuming a fixed step size e.g. $h = 1$. $\Delta^k[f](a)$ is the $kth$ finite difference operator. We use central differences expect for order 1 embeddings, which use forward differences. We interpret the resulting differencing coefficients as convolutional kernels, which can be applied to any sequential data with sufficiently high sampling rate either online or during preprocessing. Mapping back from the network's states to sequential data can easily be done using the inverse kernel. Figure 3 shows examples for forward and inverse embedding kernels.

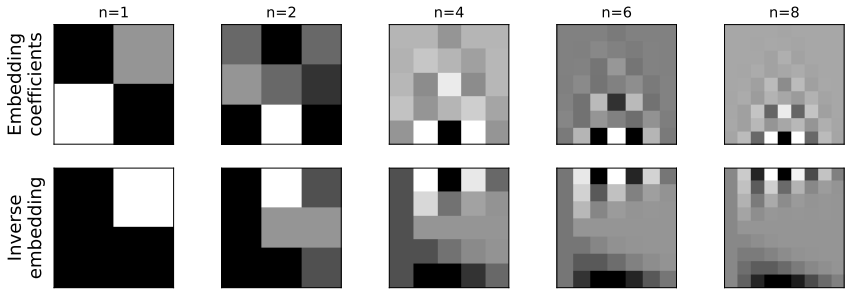



Figure 3: Forward and inverse embedding kernels for five different embedding orders.

# B Model complexity results for static predictive coding

Table 3: Complexity (Test set)

| | MNIST | OMNIGLOT | fMNIST | MNIST | OMNIGLOT | fMNIST |
|---|---|---|---|---|---|---|
| Steps | 3 | 3 | 3 | 6 | 6 | 6 |
| VAE | 37.7±0.4 | 32.8±0.3 | 30.3±0.6 | N/A | N/A | N/A |
| GPC-S | 37.5±0.1 | 34.1±0.1 | 30.9±0.3 | 39.0±0.1 | 36.2±0.1 | 31.2±0.2 |
| GPC-M | 37.8±0.1 | 33.3±0.1 | 35.7±0.1 | 38.0±0.1 | 34.5±0.1 | 33.4±0.1 |
| VLAE | 36.8±0.1 | 34.8±0.1 | 29.8±0.1 | 37.0±0.1 | 36.0±0.1 | 30.0±0.1 |

Table 3 shows the posterior complexity of models trained on the static prediction task for MNIST, OMNIGLOT and Fashion MNIST in terms of mean and standard deviation over ten runs. The predictive coding models GPC-S and GPC-M show complexity that is comparable to, but slightly higher than the complexity of the baseline VAE. The VLAE shows complexity values that are smaller than the baseline VAE in four out of the 6 tested configurations. For VLAE and GPC models, increasing the amount of inference steps from 3 to 6 slightly increases the complexity of encoded states.

# C Data sources

We employ three popular datasets for unsupervised learning on images: The MNIST dataset (Creative Commons Attribution-Share Alike 3.0 license), Fashion MNIST (MIT license) and OMNIGLOT (MIT License). Evaluation of the dynamical predictive coding model is based on the Disentanglement

testing Sprites dataset (Apache License 2.0). Table 4 shows sources and licenses for all datasets. MNIST and FashionMNIST contain 60000 train and 10000 test images (with 28 × 28 pixels) while the OMNIGLOT dataset contains 24345 train and 8070 test images (also with 28 × 28 pixels).

Table 4: Data sources

| Dataset | Source | License |
|---|---|---|
| MNIST [12] | yann.lecun.com/exdb/mnist | CC BY-SA 3.0 |
| OMNIGLOT [11] | github.com/brendenlake/omniglot | MIT license |
| Fashion MNIST [23] | github.com/zalandoresearch/fashion-mnist | MIT license |
| dSprites [14] | deepmind.com/open-source/dsprites | Apache License 2.0 |

To generate discrete video sequences with high temporal resolution for the dynamical GPC model we use a customized version of the dSprites dataset [14]. We generate 128000 random samples from the original dataset and apply Gaussian blur along both spatial axes with kernel size 3 and Standard deviation of 10 before applying normalization. We restrict x and y positions to six values respectively, such that all sprites appear within a center crop of 32 x 32 pixels. Starting with the noisy version of the sprite, we apply a single direction of rotation (counterclockwise) by rotating the sprite by a single degree. All remaining aspects, such as shape, size, horizontal position and vertical position stay constant. The Gaussian noise was applied only to the first frame of each sequence. We then projected the resulting discrete video sequences into generalized coordinates using the embedding kernels discussed in Section A of the appendix.

# D Code and dependencies

Our codebase was developed in PyTorch 1.12.1 [18]. The code including preprocessing routines for all presented experiments is available as supplementary material and will be published as URL.

# E Compute resources

We ran all experiments on a NVIDIA A40 GPU. Training a two-layer GPC model with 3 inference steps takes around 230 seconds for $1e + 4$ weight updates. In comparison, training the same configuration with a single inference steps takes about 160 seconds.

# F Societal Impact

Predictive coding models enable training of complex neural network architectures with increased parallelism. The relevant quantities for weight updates are computed locally for each layer and can be distributed among small compute devices, potentially making large-scale architectures accessible to a broader audience. Similarly, iterative inference enables increased performance without increasing model complexity. The potential applications are broad, ranging from artificial vision to drug or protein design or autonomous driving. Such broad applicability is always connected to potential negative impacts, such as the illegal usage of neural networks.

# G Inference and learning with precision weighted prediction errors

For a hierarchical layer l, the variational free energy F decomposes into an accuracy term, that measures the quality of the outgoing prediction $g^{(l)}(\mu^{(l)})$, and a complexity term between top down predicted state $g^{(l+1)}(\mu^{(l+1)})$ and inferred state $\mu^{(l)}$. Since we want to compare the GPC model to variational autoencoders, we regularize by the distance between a standard normal prior distribution and the inferred posterior distribution. Note that under the Laplace approximation, this simplifies to the divergence from zero mean.

$$\mathcal{F}(o, q^{(l)}, \hat{q}^{(l)}) = \mathbb{E}_q[\log p(o \mid z)] - \mathrm{KL}(q^{(l)}(z) \| \hat{q}^{(l)}(z)) - \mathrm{KL}(q^{(l)}(z) \| p(z)) \tag{2}$$

Note that hidden hierarchical layers predict the mean of activity of the layer below, i.e. observations $o$ are replaced by the sufficient parameters $\mu^{l-1}$ and $\Sigma^{l-1}$. We can also express this lower bound with respect to the optimal inferred posterior distribution $q^{*(l)}(z)$ that is inferred during inference:

$$\mathcal{F}(o, q^{(l)}, \hat{q}^{(l)}) = \mathbb{E}_q[\log p(o \mid z)] - \mathrm{KL}(q^{*(l)}(z) \| \hat{q}^{(l)}(z)) \tag{3}$$

In our GPC model, the optimal inferred posterior distribution $q^{*(l)}(z)$ is inferred using an approximation of the full KL divergences that rests on precision weighted errors that are simple to compute locally for each layer:

$$\begin{aligned} \mathcal{F}(o, q^{(l)}, \hat{q}^{(l)}) &= \mathbb{E}_q[\log p(o \mid z)] - \mathrm{KL}(q^{*(l)}(z) \| \hat{q}^{(l)}(z)) \\ &\approx -\epsilon^{(l)o}\xi^{(l)o} - \epsilon^{(l)v}\xi^{(l)v} \end{aligned} \tag{4}$$

Where the corresponding weighted prediction errors $\xi$ are computed as:

$$\begin{aligned} \xi^{(l)p} &= \Sigma^{(l)p-1}\epsilon^{(l)p}, & \epsilon^{(l)p} &= (\mu^{(l)} - 0) \\ \xi^{(l)v} &= \Sigma^{(l)v-1}\epsilon^{(l)v}, & \epsilon^{(l)v} &= (\mu^{(l)} - g^{(l+1)}(\mu^{(l+1)})) \\ \xi^{(l)o} &= \Sigma^{(l)o-1}\epsilon^{(l)o}, & \epsilon^{(l)o} &= (g^{(l)}(\mu^{(l)}) - o) \end{aligned} \tag{5}$$

To make use of the amortized inference, at the start of iterative inference, the inferred posterior is initialized with its top-down prediction $\mu_0 = \hat{\mu}$. During inference the optimal posterior distribution $q^{*(l)}(z)$ with respect to distance from the prior and decoder accuracy is then computed using a simple gradient descent or using Gauss-Newton updates on $\epsilon^{(l)p}$ and $\epsilon^{(l)o}$. After inference, the covariance parameters of the top-down predicted distribution $\hat{q(z)}$ and the inferred posterior distribution $q^*(z)$ are inferred following the routine described in Park et al. [16] using $\Sigma^{-1} = -\nabla_z^2 \log p_\theta(o, z)\big|_{z=\mu}$, which can be efficiently computed for ReLU activations. We then compute weights updates using the Adam optimiser by replacing the full KL divergence in Formula 3 with the precision weighted error $\epsilon^{(l)v}\xi^{(l)v}$. Note that for model evaluation and comparison to VAE and VLAE baselines, we use the full analytical KL divergence. We found that training the model using the full KL terms leads to similar results, although with increased numerical instability when latent vectors have large dimensions or when the amount and size of inference steps is insufficient.

## H Comparing gradient descent and Gauss-Newton updates for iterative inference

We compared Gauss-Newton updates during iterative inference with simple gradient descent steps that do not consider precision weighting. The gradient descent based updates perform relatively well, in many cases outperforming the ELBO of the VAE baseline. Table 5 shows model evidence results for 1 and 10 updates using a learning rate of 0.01 and 0.001 respectively.

Table 5: Negative evidence lower bound (test set)

| | MNIST | OMNIGLOT | fMNIST | MNIST | OMNIGLOT | fMNIST |
|---|---|---|---|---|---|---|
| Steps | 10 | 10 | 10 | 1 | 1 | 1 |
| Learning rate | 0.001 | 0.001 | 0.001 | 0.01 | 0.01 | 0.01 |
| VAE | 901.7±1.6 | 1018.3±1.5 | 881.2±0.6 | N/A | N/A | N/A |
| GPC-S | 905.1±3.0 | 1016.21±6.6 | 893.2±1.7 | 910.4±4.0 | 1010.7±2.8 | 899.7±1.6 |
| GPC-M | 894.6±4.0 | 1003.5±0.9 | 882.1±2.6 | 900.4±1.0 | 1009.0±1.1 | 887.0±1.1 |
| VLAE | 877.0±1.4 | 986.9±3.9 | 869.6±0.2 | 888.9±0.9 | 1000.5±1.2 | 873.8±0.4 |

As visible in the Table 5 and Figure 4, the gradient based updates are much more sensitive to the inference learning rate. We found that values around 0.001 work best, while higher rates lead to degraded performance. In contrast, the Gauss-Newton based updates consider the precision at the

currently inferred mode, leading to more stable updates that are adaptively weighted and are less sensitive to the inference learning rate.

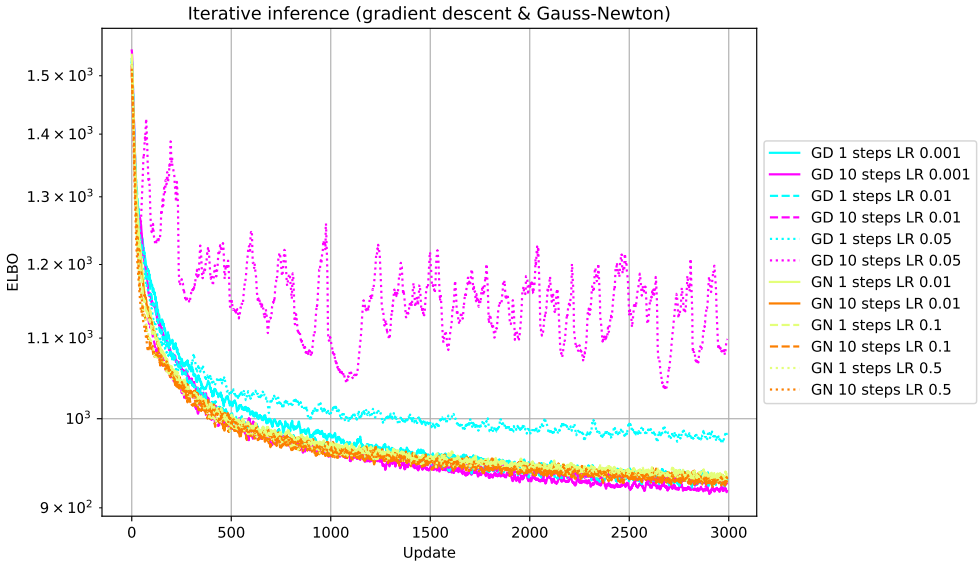



Figure 4: Comparison of static predictive coding with gradient descent and Gauss-Newton updates during iterative inference of the optimal posterior distribution. Shown are the first 3000 weight updates on the MNIST dataset.