Regularization for Predictive Coding Networks

Giacomo Camposampiero Anne Marx Franz Nowak Andrea Pinto

Department of Computer Science, ETH Zürich, Switzerland

Equal contributions. {gcamposampie, anmarx, fnowak, pintoa}@ethz.ch

ABSTRACT

Predictive Coding (PC) is picking up momentum as a biologically plausible alternative to Backpropagation. In this work, we implement PC networks for regression and classification tasks and investigate the impact of dropout and different PC activation initialization techniques on their training dynamics. Our results show that dropout in PC acts similarly to BP, and initialization techniques other than the standard forward initialization lead to smoother training and smaller network weights, hence also acting as a regularizer.

1 INTRODUCTION

Deep Learning with Backpropagation (BP) has shown increasing performance over the past decade, resulting in significant progress in a range of research areas, including Reinforcement Learning [26], Computer Vision [14], and Natural Language Processing [4, 6].

As Artificial Intelligence (AI) has progressed towards human level performances, there has been a growing interest in biological plausibility of Artificial Neural Networks (ANNs). This shift in perspective has led to biologically grounded approaches that could overcome limitations of the classical BP algorithm, including the implausible global function differentiation and its slow propagation of error signals through time.

One example is Predictive Coding (PC) [23, 29], a theory of brain computation from Neuroscience. In its adaptation to ANNs [31], each artificial neuron only interacts with its parents in the computational graph by predicting their activations, offering localized, highly parallelizable and more plausible computations from a biological perspective. PC appears to be a strong alternative to BP, as it has been shown to converge towards it under certain conditions [27], and can be adapted to arbitrary network topologies.

Despite these advantages, PC is a relatively young and understudied field in Machine Learning (ML). For example, regularization techniques that are used in many state-of-the-art BP models and have shown a considerable impact on the generalization capability [8, 15] have not been extensively explored for PC networks yet.

In this work, we implement simple to use and relatively efficient PC networks, capable of tackling regression and classification tasks, alongside corresponding standard BP neural networks with comparable architectures. We then exploit them to investigate whether regularization has positive effects on the generalization capability of PC networks, exploring the impact of dropout and different PC neuron initialization techniques (for which we formulate a hypothesis linking initialization to regularization) on their generalization power. The empirical evidence gathered on regression (function approximation) and classification (handwritten digit identification) tasks suggest that even though the generalization performance is not improved by dropout in our experiments, it has an effect on PC networks similar to that for BP networks. Our results also show that initialization techniques different from forward reduce the L_2-norm of the PC model's weights, indicating that they also have regularizing effects.

In the following sections, we briefly outline existing related works (§2), the PC framework (§3) and our hypothesis (§4). Then, we present our experimental setting (§5) and results (§6). Finally, we conclude with some remarks on our work (§7).

2 RELATED WORK

As pointed out by Bishop [2], there exist two ways of preventing overfitting, and thereby increasing generalizability in ANNs.

Firstly, a greater degree of generalization can be achieved by reducing the complexity of the network through a decrease in the number of its free parameters (structural stabilization). One way of doing this is by applying dropout [17, 20, 30], an established technique to regularize training in traditional ANNs. Dropout works by randomly setting a certain proportion of network weights to zero, thereby forcing other weights to make up for the loss caused and hence contribute to the prediction. Eventually, this enforces a more even distribution of contributions among neurons [30]. To our best knowledge, no prior work on PC investigated dropout yet.

Secondly, networks can be regularized by restricting the parameter space, as large weights tend to cause sharp transitions in the node functions and thus large changes in output for small changes in the inputs [24]. This is usually achieved in traditional ANNs using an additional penalty term in the error function, as for example in Thikonov regularization [1]. We speculate that a similar behavior could be obtained in PC networks using specific initialization techniques for the guessed activations (see Section 4). While current implementations of PC networks [18, 19, 22, 25] seem to exclusively focus on forward initialization (meaning the neuron activations are initialized by performing an initial forward pass through the network [28]), Millidge et al. [19] posits that future work on PC should relax this constraint, as it could lead to potential improvements of PC over BP.

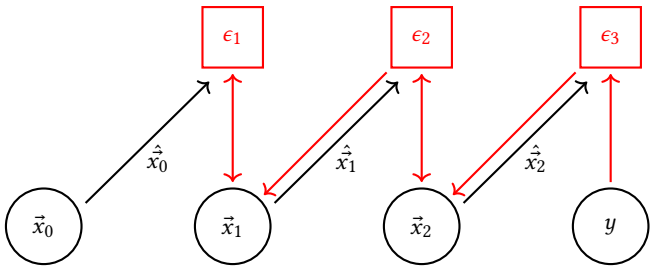

Figure 1: Local dependencies of PC error signals.

3 PREDICTIVE CODING

Predictive Coding as a framework for ML has its grounding in Neuroscience, with a strong empirical support as a theory modelling the brain [3, 9, 32]. Other than the benefits already mentioned, it has been shown to help tackle catastrophic forgetting [16], to be superior to other architectures in associative memory, and has since been implemented for CNNs, RNNs, and LSTMs [25], and even simple transformer networks [22].

In PC, the computation graph is augmented by introducing error nodes $\epsilon = ||\vec{x}_l - \hat{\vec{x}}_l||_2^2$, which contain the squared difference between predictions $\hat{\vec{x}}_l$ and true activations $\vec{x}_l$. The sum of these errors can also be interpreted as the free energy of the system [10], which we use as the loss function for PC:

$$\mathcal{L} = \sum_{l=1}^{L} \epsilon_l = \sum_{l=1}^{L} ||\vec{x}_l - \hat{\vec{x}}_l||_2^2 \tag{1}$$

The version of the PC algorithm we use to train our models is based on that proposed by Millidge et al. [18] and is presented in Algorithm 1. A key difference in our implementation is that we allow for different ways of initializing the activations, described in Section 5.3, instead of requiring forward initialization. The algorithm can be outlined as follows. First, the input activation $\vec{x}_0$ is fixed to the input, and the output activation is fixed to the label y (Line 6). Then all the other activations are initialized (Lines 7-8). Learning in PC is comprised of two phases, relaxation and weight update. In the first phase, we repeatedly compute the energy ϵ_l for each layer, sum them to obtain the total energy $\mathcal{L}$, and optimize the activations to minimize it while keeping the weights fixed. Afterwards, activations are updated using gradient descent (or more advanced optimizers, as outlined in Section 5.3) for a number of N iterations, or until convergence (Lines 9-15). Finally, for the second phase, a single update step is performed on the weights $\vec{\theta}_l$ while keeping the activations fixed (Lines 16-17), further minimizing the total energy.

Algorithm 1 Predictive Coding Training

1: **Data:** Dataset $\mathcal{D} = \mathbf{X}, \mathbf{y}$
2: $N \leftarrow$ number of iterations
3: $L \leftarrow$ number of layers
4: **def** TRAINING($\mathcal{D}$)
5: **for** $X, y \in \mathcal{D}$ **do**
6: $\vec{x}_0, \vec{x}_L \leftarrow X, y$
7: **for** $l = 1 \dots L-1$ **do**
8: $\vec{x}_l \leftarrow$ INITIALIZE()
9: **for** $t = 1 \dots N$ **do**
10: **for** $l = 1 \dots L$ **do**
11: $\hat{\vec{x}}_l \leftarrow f(\vec{x}_{l-1}, \vec{\theta}_l)$
12: $\epsilon_l \leftarrow ||\vec{x}_l - \hat{\vec{x}}_l||_2^2$
13: $\mathcal{L} \leftarrow \sum_{l=1}^{L} \epsilon_l$
14: **for** $l = 1 \dots L-1$ **do**
15: $\vec{x}_l^{t+1} \leftarrow \vec{x}_l^t + \eta_x^t \frac{\partial \mathcal{L}}{\partial \vec{x}_l{}^t}$
16: **for** $l = 1 \dots L$ **do**
17: $\vec{\theta}_l \leftarrow \vec{\theta}_l + \eta_\theta \frac{\partial \mathcal{L}}{\partial \vec{\theta}_l}$

As already mentioned, this training algorithm is highly parallelizable assuming that the predictions at each layer are fixed at the values assigned during the feedforward pass throughout the convergence step [18]. In fact, the error signals only propagate in local neighbors, as shown in Figure 1. This is further shown in our derivation of the gradients for $\mathcal{L}$, included in Appendix A.

4 MOTIVATION

In this section, we present a more formal argumentation to back our choice of experimenting with dropout and different PC initialization methods. For what concerns dropout, we expect it to restrict the expressivity of the model by reducing the number of free parameters, as it happens for BP, and hence reducing the generalization error. This is expected to happen for forward initialization, since the two paradigms are asymptotically equivalent [18] in this setting. The behavior with other initialization techniques, however, has to be empirically verified, and no previous assumptions can be made.

On the other hand, our hypothesis regarding different PC initialization techniques is that they might positively affect the value range of the weights, implicitly shrinking them and hence regularizing the network. When using forward initialization, all the energy is initially concentrated in the last layer and is equal to the squared difference between the predicted output and the ground truth [18]. For the previous layers, in contrast, the error signal is zero, as they are initialized to their forward values. Hence, the error signals are not evenly distributed and there will only be small changes between the initial and the converged energy distribution. If, on the other hand, the activations are initialized with either with 0 or randomly (e.g. sampled from a normal distribution), all the neurons will converge to equilibrium more uniformly, because the error signal will be more evenly spread across the network. If the activations are more equally scattered, this may also lead to the weight updates being more evenly distributed, which then would have the desired effect of preventing overly large weights, hence regularizing the network [2].

5 METHODS

This section describes how experiments are conceptualized and conducted to analyze our hypothesis. Due to the limited computational resources available for our experiments, we only experiment with two simple supervised regression and classification tasks, that allow us to investigate our hypothesis on small models.

5.1 Datasets

For the regression task, we generate a total of 800 training and 200 test observations from the noisy sinusoidal function:

$$f(x) = \sin(1 + x^2) + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(0, 0.1) \quad x \in [0, 4]$$

where the standard Gaussian noise is used to test the ability of the model to grasp the underlying structure of the data. The input and output data are scalar, which allows for a small network with few parameters and reasonable convergence time. For the classification task, we use the MNIST dataset [5] which contains 60,000 training images and 10,000 testing images of handwritten monochromatic digits, each of size 28x28 pixels. The small input dimensions and output size of 10 allow the use of a shallow linear network as well.

5.2 Models

Considering the limited complexity of the tasks and datasets, we opt to use simple fully-connected networks built in PyTorch [21]. We implement BP as well as PC models in order to compare the performance and training dynamics where possible.

BP Models For the regression task, the model consists of two hidden layers with 1024 neurons each, using ReLU activations and dropout. The output layer consists of a single neuron, with no activation function and no dropout. The model is optimized using a Mean Squared Error (MSE) loss. For the classification task, the model consists of a single hidden layer with 50 neurons, using ReLU activations and a dropout. The output layer consists of 10 neurons, that use a LogSoftmax activation function. The model is optimized using a Cross-Entropy (CE) loss.

PC Models The PC models are implemented like their BP equivalents, with the same number of linear layers, widths, and activation functions. To train the models using Algorithm 1, we develop the following additional components:

- PC Layer, which handles the previous layer's activation guessing and computes the error signals used to train the model; a visualization is included in Appendix B.
- PC Dropout, which allows the dropout mask to be consistent during the energy descent phase for each batch.
- PC Softmax (extension of PC Layer) which ensures that the guessed output values always sum up to 1.

The definition of these components allows us to build a flexible and generalizable framework, where the adaptation of a model to the PC setting only requires an addition of a limited amount of layers.

5.3 Experiments

We perform three sets of experiments, one on general performance, one on dropout and one on initialization. For all experiments, we take the median over five different PyTorch seeds to reduce noise. We train the regression models for 500 epochs and the classification models on 200 epochs, both with early stopping.

PC Implementation Performance: In order to validate our PC implementation and to find good optimizers and hyperparameters for subsequent experiments, we compare the performance of our PC network to a corresponding BP network with no dropout and using forward initialization. In order to allow for the best possible training settings, we do Bayesian hyperparameter search, optimizing on the test loss. We investigate Adam [13], Adagrad [7], RMSProp [12] and Stochastic Gradient Descent (SGD) with momentum for the weights' optimizer of the linear layers. For the PC activations, we only investigate Adam and BP with momentum to keep the size of the hyperparameter space manageable. We perform a total of 400 runs per optimizer and model combination.

Dropout Experiments: For the following experiments, we select and fix the best weight and activation optimizer combination with the highest-performing hyperparameters for the PC network and select the corresponding weight optimizer for the BP trainings. This allows us to compare the training dynamics between PC and BP models using a consistent optimizer setting, while analyzing the change in performance between the PC networks only based on one free parameter. We test dropout with dropout probabilities ranging between $[0.0, 0.9]$, with steps of 0.1.

PC initialization experiments: Thirdly, we test four different techniques for initializing the activations of the PC neurons:

- 'Forward' initializes the activation to the forward values.
- 'Zeros' initializes them as 0.
- 'Normal' samples them from a standard normal distribution.
- 'Xavier Normal' following [11] with a gain value of 1.

6 RESULTS AND DISCUSSION

PC Implementation Performance: We find that the PC model with forward initialization converges towards the BP model in terms of both training and test loss. BP against PC performances on the MNIST classification dataset is shown in Figure 2 (analog for sine can be found in Figure 8 in Appendix C). This suggests that PC models can achieve similar or even better performance compared to BP models on certain tasks given appropriate hyperparameter tuning, confirming our expectations and validating results and conclusions from Millidge et al. [18]. Based on the test loss, the best performing optimizers for the classification task were RMSProp for the weight optimization and SGD with momentum for the convergence optimization. As for the regression task, the best performing weight optimizer was Adam, while SGD with momentum proved to be the best optimizer for the convergence step. In the following experiments, these optimizers were fixed.

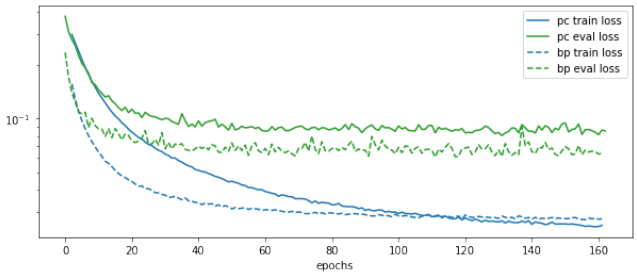

Figure 2: Train loss (blue) and test loss (green) of BP (dashed) and PC (full) model on MNIST data. Weight optimizer (both): RMSProp, activation optimizer (PC): SGD with momentum.

Dropout Experiments: As shown in Figure 4, a clear resemblance between the dependency of dropout on the test error can be observed between PC and BP models, even though dropout did not actually improve the generalization capabilities in this case for either model type. This might be explained by the hyperparameters being optimized on a dropout of zero in the previous experiments. For the MNIST classification task, the test errors for different dropout rates are generally similar between PC and BP. We observe a few outliers for higher dropout values in the PC model, indicating that with high dropout, PC model performance becomes more erratic than for the BP equivalent (see Appendix 6). Our results indicate that using dropout has a similar impact on the performance between BP and PC models, and with the right choice of hyperparameters and model we would expect it to have a positive impact on the generalization capability for PC networks as it usually has for BP.

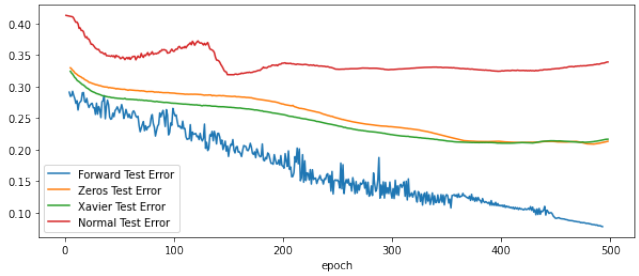

Figure 3: Test error for PC regression for different initializations

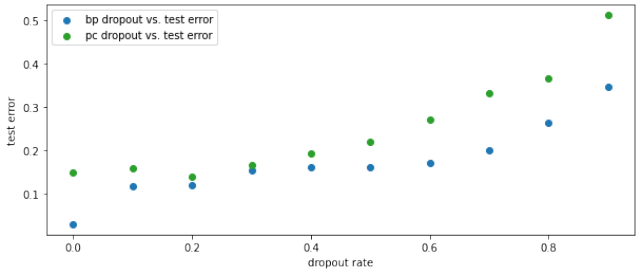

Figure 4: Test error for different dropout rates of BP and PC regression models on the sine dataset

PC initialization experiments: The results shown in Figure 3 highlight that in our experiments, forward initialization strongly outperforms all the other initialization techniques in terms of test loss, supporting its widespread use as the way to initialize PC activations in the literature. While different techniques have a worse test error than forward, Xavier and Zeros initialization do affect the L_2 norm of the model weights, leading to weight shrinkage as shown in Figure 5. Considering the regularization effect of weight shrinking, this seems to support our hypothesis presented in Section 4. Given that we only tuned the hyperparameters on forward initialization due to resource constraints, it is likely that they are not optimal for the other initialization regimes, and hence with more hyperparameter tuning or a different model architecture, this regularization might benefit the test accuracy compared to forward initialization.

Figure 3 also shows that forward initialization leads to more fluctuation in the test error, while the other initialization schemes show a smoother loss landscape, further supporting that they might be acting as a regularizer.

Init. Technique	Train Error	Test Error
Normal	0.3862	0.3391
Xavier	0.2467	0.2165
Zeros	0.2424	0.2137
Forward	**0.0753**	**0.0755**

Table 1: Top Sine Train Error and Test Error performance of PC networks as a function of the initialization technique

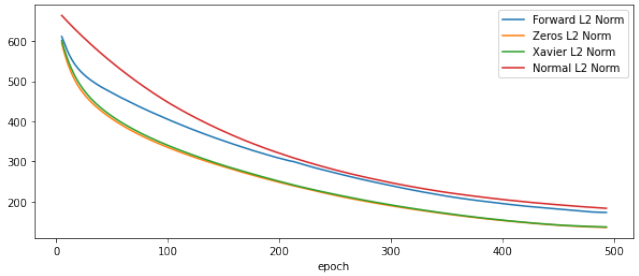

Figure 5: Effect of activation initialization on weight shrinkage

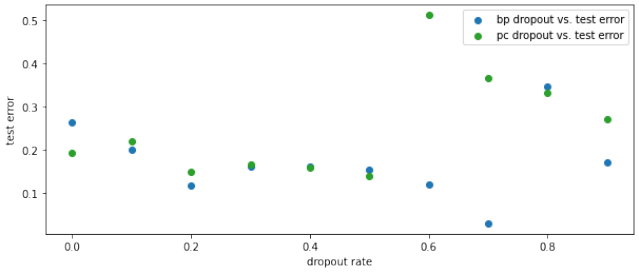

Figure 6: Classification test error by dropout rate for BP and PC

7 CONCLUSION

Our project's focus was finding and testing regularization techniques in Predictive Coding networks. For this, we implemented PC neural networks for classification and regression in PyTorch, as well as architecturally similar BP neural networks. We then leveraged our framework to study the effect of dropout and different activation initialization techniques as regularizers in PC networks. At the time of writing, there are, to our knowledge, no other PC frameworks that include dropout and different methods of initialization, so there are no baseline models with which to compare our findings.Therefore, one of our main contributions is adding our flexible and extendable PyTorch implementation for others to build on. First, we validated our implementation by conducting a series of experiments to verify that our PC networks learned similarly well compared to equivalent models trained with BP. We then investigated dropout in PC networks, finding that it has a similar effect on PC and BP networks, though in neither it improves the test accuracy. This could be explained by the lack of hyperparameter tuning and the small model sizes. For initializing the PC models' neuron activations, forward initialization technique consistently outperformed other initializations, though this is likely due to insufficient hyperparameter optimization as well. However, we found that some of the data-independent initializations contributed to reducing the L_2 norms of the weights and smoothing learning curves, indicating that they too act as regularizers. In future work further experiments could yield better results through optimizing hyperparameters for the different initialization techniques, and using more suitable training data that induces overfitting when training without regularization.

REFERENCES

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[2] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[3] Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017. ISSN 0022-2496. doi: https://doi.org/10.1016/j.jmp.2015.11.003. URL https://www.sciencedirect.com/science/article/pii/S0022249615000759. Model-based Cognitive Neuroscience.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[5] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

[8] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[9] Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 360:815–36, 05 2005. doi: 10.1098/rstb.2005.1622.

[10] Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010. doi: 10.1038/nrn2787. URL https://doi.org/10.1038/nrn2787.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[12] Alex Graves. Generating sequences with recurrent neural networks, 2013. URL https://arxiv.org/abs/1308.0850.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[15] Haoliang Li, Yufei Wang, Renjie Wan, Shiqi Wang, Tie-Qiang Li, and Alex Kot. Domain generalization for medical imaging classification with linear-dependency regularization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3118–3129. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/201d7288b4c18a679e48b31c72c30ded-Paper.pdf.

[16] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

[17] Poorya Mianjy and Raman Arora. On convergence and generalization of dropout training. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21151–21161. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/f1de5100906f31712aaa5166689bfdf4-Paper.pdf.

[18] Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *CoRR*, abs/2006.04182, 2020. URL https://arxiv.org/abs/2006.04182.

[19] Beren Millidge, Tommaso Salvatori, Yuhang Song, Rafal Bogacz, and Thomas Lukasiewicz. Predictive coding: Towards a future of deep learning beyond backpropagation?, 2022. URL https://arxiv.org/abs/2202.09467.

[20] Wenlong Mou, Yuchen Zhou, Jun Gao, and Liwei Wang. Dropout training, data-dependent regularization, and generalization bounds. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3645–3653. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/mou18a.html.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[22] Luca Pinchetti, Tommaso Salvatori, Yordan Yordanov, Beren Millidge, Yuhang Song, and Thomas Lukasiewicz. Predictive coding beyond gaussian distributions, 2022. URL https://arxiv.org/abs/2211.03481.

[23] Rajesh Rao and Dana Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2:79–87, 02 1999. doi: 10.1038/4580.

[24] Russell D. Reed and Robert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1999.

[25] Tommaso Salvatori, Luca Pinchetti, Beren Millidge, Yuhang Song, Tianyi Bao, Rafal Bogacz, and Thomas Lukasiewicz. Learning on arbitrary graph topologies via predictive coding, 2022. URL https://arxiv.org/abs/2201.13180.

[26] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

[27] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? -exact implementation of backpropagation in predictive coding networks. *Advances in neural information processing systems*, 33: 22566–22579, 01 2020.

[28] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc., 2020.

[29] Michael Spratling. A review of predictive coding algorithms. *Brain and Cognition*, 112, 01 2016. doi: 10.1016/j.bandc.2015.11.003.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[31] James Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29:1–34, 03 2017. doi: 10.1162/NECO_a_00949.

[32] James Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23, 03 2019. doi: 10.1016/j.tics.2018.12.005.

A PC GRADIENTS DERIVATION

In this Appendix section, we include our derivation of the gradients of the total energy function $\mathcal{L}$ used in training, for both activations $\hat{x}_l$ and weights θ_l. Firstly, we define the total energy function to be the sum over the energies ϵ_l (error signals) of all layers as follows

$$\mathcal{L} = \frac{1}{2}\sum_{l=1}^{L}\epsilon_l = \frac{1}{2}\sum_{l=1}^{L}(\hat{x}_l - f(\theta_l\hat{x}_{l-1}))^2$$

where f is the activation function, θ_l the weight of layer l and $\hat{x}_{l-1}$ the activation guessed in the previous layer. The gradients of the loss $\mathcal{L}$ with respect to the activations $\hat{x}_i$ can be then derived as follows

$$\begin{aligned}
\frac{d\mathcal{L}}{d\hat{x}_i} &= \frac{d}{d\hat{x}_i}\left[\frac{1}{2}\sum_{l=1}^{L}(\hat{x}_l - f(\theta_l\hat{x}_{l-1}))^2\right] \\
&= \frac{1}{2}\frac{d}{d\hat{x}_i}\left[(\hat{x}_i - f(\theta_i\hat{x}_{i-1}))^2 + (\hat{x}_{i+1} - f(\theta_{i+1}\hat{x}_i))^2\right] \\
&= \frac{1}{2}\frac{d}{d\hat{x}_i}[\hat{x}_i^2 + f(\theta_i\hat{x}_{i-1})^2 - 2\hat{x}_i f(\theta_i\hat{x}_{i-1}) \\
&\qquad + \hat{x}_{i+1}^2 + f(\theta_{i+1}\hat{x}_i)^2 - 2\hat{x}_{i+1}f(\theta_{i+1}\hat{x}_i)] \\
&= \hat{x}_i - f(\theta_i\hat{x}_{i-1}) + f(\theta_{i+1}\hat{x}_i)f'(\theta_{i+1}\hat{x}_i)\theta_{i+1} \\
&\qquad - \hat{x}_{i+1}f'(\theta_{i+1}\hat{x}_i)\theta_{i+1} \\
&= (\hat{x}_i - f(\theta_i\hat{x}_{i-1})) + (f(\theta_{i+1}\hat{x}_i) - \hat{x}_{i+1})f'(\theta_{i+1}\hat{x}_i)\theta_{i+1} \\
&= \epsilon_i - \epsilon_{i+1}f'(\theta_{i+1}\hat{x}_i)\theta_{i+1}
\end{aligned}$$

which only depends on previous and successive layers parameters. In the same way, we can derive the gradients of the loss $\mathcal{L}$ with respect to the weights θ_i as

$$\begin{aligned}
\frac{d\mathcal{L}}{d\theta_i} &= \frac{d}{d\theta_i}\left[\frac{1}{2}\sum_{l=1}^{L}(\hat{x}_l - f(\theta_l\hat{x}_{l-1}))^2\right] \\
&= \frac{1}{2}\frac{d}{d\theta_i}\left[(\hat{x}_i - f(\theta_i\hat{x}_{i-1}))^2\right] \\
&= \frac{1}{2}\frac{d}{d\theta_i}\left[\hat{x}_i^2 + f(\theta_i\hat{x}_{i-1})^2 - 2\hat{x}_i f(\theta_i\hat{x}_{i-1})\right] \\
&= f(\theta_i\hat{x}_{i-1})f'(\theta_i\hat{x}_{i-1})\hat{x}_{i-1} - \hat{x}_i f'(\theta_i\hat{x}_{i-1})\hat{x}_{i-1} \\
&= (f(\theta_i\hat{x}_{i-1}) - \hat{x}_i)f'(\theta_i\hat{x}_{i-1})\hat{x}_{i-1} \\
&= -\epsilon_i f'(\theta_i\hat{x}_{i-1})\hat{x}_{i-1}
\end{aligned}$$

which as well only depends on previous and successive layers parameters, hence definitely proving the locality of the updates.

B PCLAYER VISUALIZATION

In this Appendix section, we include a visualization of our PCLayer component that we introduced in this project, depicted in Figure 7.

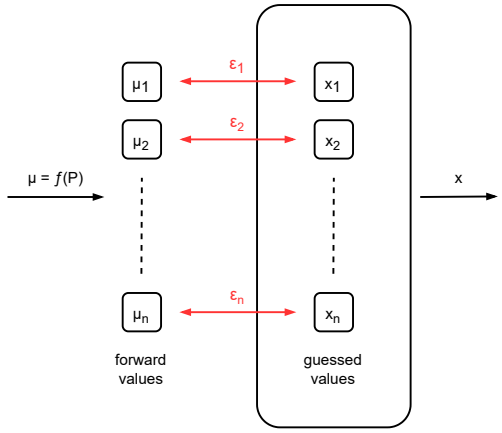

Figure 7: PC layer visualization.

Here, $\mu = f(P)$ are the forward values from the previous layer in the computational graph, for which the PC layer is guessing the activation functions, while x are the guessed activations (propagated forward to the following layers of the network) and the ϵ are the error signals (differences between the true and guessed activations, used to train the network).

C SINE BP VS. PC TRAINING DYNAMICS

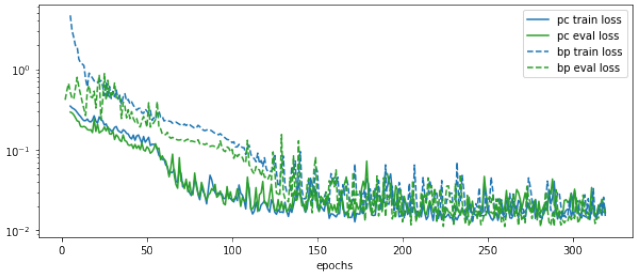

Figure 8: Sine BP vs. PC training dynamics
Sample of a learning curve for sine BP and PC respectively, to show that our PC implementation converges to BP. In this case the weights of the two samples you see have been set with RMSProp and for PC the optimization was performed with momentum.

D AUGMENTED OUT-OF-DISTRIBUTION MNIST

This section describes a method we used in our experiments, but left out from our results due to convergence problems and limited time to fine-tune the models, likely due to limited network parameters and training data.

In order to measure the generalization power of the classification models on a similar but different domain, we tested them on out-of-distribution data, which is less likely to resemble anything already seen during training. The images from the test dataset are augmented by randomly adding Gaussian noise, swirling the center of the image by a radius of five pixels, adding sinusoidal oscillation to the image row coordinates, and/or reducing the resolution.

Visual examples of the augmentations can be found in Figure 9. The first row depicts ten samples of the original MNIST data. In the second row, these samples are randomly augmented by adding Gaussian noise, swirling the center of the image by a radius of five pixels, adding sinusoidal oscillation to the image row coordinates, and/or reducing the resolution. In future experiments, this approach can be helpful to test the generalization capabilities of the network cross-domain, additionally to the test dataset, which tests the generalization capabilities within the same data domain.

Figure 9: MNIST Augmentation for Out-of-Distribution data.