Universiteit Leiden

Bachelor Data Science and Artificial Intelligence

Uncertainty Quantification in Deep Learning via Predictive Coding

Piotr Perski

Anna Dawid-Łękowska & Björn van Zwol

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

24/08/2026

Abstract

Deep learning needs uncertainty quantification (UQ) to be useful in science and other crucial high-stakes settings. Standard neural networks do not provide this important feature, so methods like Monte Carlo (MC) dropout and deep ensembles bolt it on. Predictive coding (PC) instead offers a view of neural networks that is natively probabilistic and treats each hidden node as a distribution rather than a point. In practice, on the other hand, the structure is discarded: both training and inference rely on deterministic estimates. In this paper we check if keeping it gives UQ for free. Sampling the hidden distributions turns out to be additive Gaussian noise on the activations, known as Gaussian noise injection, which helps calibration and robustness, but has, to our knowledge, not been tested for UQ. It is also the additive counterpart to MC dropout, which is multiplicative. We compare both noise types under PC and backpropagation (BP) on a toy regression task, measuring whether uncertainty grows away from the training data. MC dropout does almost equally well under PC and BP, though PC holds up better as depth grows, which we trace to its local learning rule. Additive noise does not seem to provide meaningful UQ. We explain this by observing that its output variance depends weakly on the input, whereas multiplicative noise scales directly with the activations. We also take a step towards using PC with heteroscedastic regression for aleatoric uncertainty. PC's native probabilistic structure is not enough for usable UQ.

1 Introduction

Deep learning has become a standard way of solving regression and classification tasks, but there is one problem with those algorithms. They are typically point-estimate predictors that return a single output with no measure on how confident they are about the result [4]. This is a significant drawback in high stakes environments such as medical diagnosis and autonomous driving, where basing actions on wrong information can lead to dangerous results. Thus, knowing when a model is guessing can be more valuable than the prediction itself [6]. Uncertainty Quantification (UQ) solves the lack of confidence score by attaching a certainty score to each of the predictions.

The most popular method to assess uncertainty is the Bayesian approach. Instead of fixing the weights of the network we should treat them as distributions and average the prediction scores over all of their possible settings. The issue with this is that nowadays computers do not have enough computational power to do this approach exactly, so we should use approximations. One of the most common ways is Monte Carlo (MC) dropout [4]. This methodology has dropout turned on at test and training time, runs the network many times and then calculates how much the outputs differ from each other. This way we can compute the level of uncertainty. There are other methodologies that do the same thing, for example adding Gaussian noise to the activations during training (Gaussian Noise Injection, GNI) [3], which Camuto et al. propose as a regulariser rather than as an uncertainty method or adding noise to the parameter updates to sample the posterior over weights (Stochastic Gradient Langevin Dynamics, SGLD) [12] - a more general approach to approximate Bayesian inference.

These methodologies were developed mostly for networks that are trained using backpropagation (BP). However, there is an alternative approach called predictive coding (PC), which is fundamentally a probabilistic model: every layer is a random variable rather than a deterministic function, and the network is trained by minimising local prediction errors between each layer and the prediction made by that layer above it [11, 10]. Because PC already describes the network as a distribution rather than a point estimate, it is a natural candidate for uncertainty quantification. Yet it is unknown whether the noise based UQ methods that work for BP can be also used in PC, and if the different structure of PC offers any advantages.

Aim This thesis analyses whether sampling from the probabilistic model that predictive coding already optimizes yields useful uncertainty estimates. We treat a neural network as probabilistic graphical model and show that feed-forward inference noise-based UQ and PC learning are different approximations of inference and learning in the same model. Specifically, that ancestral sampling of the model is exactly Gaussian noise injection. The question is therefore not how to add uncertainty quantification to predictive coding, but whether sampling the model it already describes produces a calibrated predictive distribution. We test this for GNI, and compare it against MC dropout and SGLD on the same benchmark.

Research Question We will answer the three following questions:

1. Does sampling the probabilistic model that predictive coding optimises, i.e. Gaussian noise injection, yield a useful uncertainty estimate - in particular one that captures out-of-distribution (OOD) uncertainty - and how does this compare with networks trained by backpropagation?
2. Under the only-test (OT) design, in which noise is injected at test time only, how do GNI and MC dropout compare between PC and BP in accuracy and calibration?
3. Does a heteroscedastic output head - a direct extension of the PC network - provide calibrated uncertainty without test-time sampling on the same benchmark?

Thesis Structure Section 2 describes the probabilistic graphical model view of neural networks and investigates feed-forward inference, PC, noise based UQ, and only test design approach. Later, Section 3 describes the experimental methodology used in this paper. Section 4 reports the experiments that answer our research questions on a one-dimensional regression benchmark, Section 5 discusses the limitations of our research, Section 6 outlines directions for future work and Section 7 answers the research questions.

2 Theory

We begin with Gaussian noise injection (Section 2.1), the method whose usefulness for uncertainty quantification this thesis sets out to test. Then we introduce the probabilistic view (Section 2.2) and show how predictive coding implements inference and learning on such a model (Section 2.3). Section 2.4 explains what uncertainty quantification is and surveys the noise-based methods used here, and Section 2.5 combines the two approaches. Section 2.6 describes the metrics used in the thesis.

2.1 Gaussian noise injection (GNI)

Gaussian noise injection adds Gaussian noise to the hidden activations of a network during training [3]. Writing $\boldsymbol{h}_l$ for the activation of hidden layer l in the noiseless network, GNI replaces it with

$$\widetilde{\boldsymbol{h}}_l = \boldsymbol{h}_l + \boldsymbol{\epsilon}_l, \qquad \boldsymbol{\epsilon}_l \sim \mathcal{N}(0, \sigma^2 I), \tag{1}$$

so that both the forward pass and the resulting gradients become stochastic.

Camuto et al. [3] study the effect of injecting Gaussian noise after the activations of a neural network during training. The main result of the experiment is that Gaussian noise injection has the same impact on the neural network performance as training the noiseless network with an additional regulariser. That regulariser is proportional to the squared Frobenius norm of the network Jacobian, which measures how sensitive the output is to small changes in the input. This means that GNI is more than just a stochastic forward pass - it works as a penalty that biases the model towards locally insensitive predictions. The researchers conclude that GNI improves generalisation and calibration on standard benchmarks, also making the model more robust to adversarial perturbations.

It is important to underline that the paper of Camuto et al. [3] uses GNI as regularisation, but not as a way of quantifying uncertainty. The question if the GNI's stochastic forward pass returns a meaningful estimate of uncertainty is not part of the researchers' paper. Testing this hypothesis is one of the aims of this thesis.

As we show in Section 2.3.3, this noisy forward pass is exactly ancestral sampling from the probabilistic model, with the injected noise corresponding to the latent noise $\boldsymbol{\epsilon}_z$ of (8).

2.2 Probabilistic machine learning

2.2.1 Directed graphical models

A directed graphical model represents a joint distribution as a graph in which nodes are random variables and a directed edge $\boldsymbol{a} \to \boldsymbol{b}$ means $\boldsymbol{b}$ depends directly on $\boldsymbol{a}$. The joint then factorises into one

conditional per node given its direct parents, $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \prod_i p(\boldsymbol{x}_i \mid \mathrm{pa}(\boldsymbol{x}_i))$. Shaded nodes are observed, while unshaded nodes are treated as latent [5].

For example, the directed acyclic graph in Figure 1 over the variables A, B, C, D has the joint factorisation

$$p(A, B, C, D) = p(A)\, p(B \mid A)\, p(C \mid A)\, p(D \mid A, C), \tag{2}$$

where each node is conditioned only on its parents.

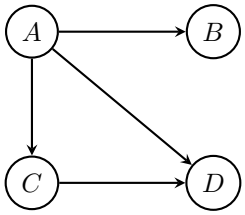

Figure 1: A directed acyclic graph over four variables, with the factorisation in (2).

In Section 2.3.2 we apply this framework to a neural network, treating its hidden activations as latent variables.

2.2.2 Stochastic gradient Langevin dynamics (SGLD)

SGLD [12] is a way of turning ordinary gradient descent into a sampler. Every time it makes a downhill step, it also adds a small amount of Gaussian noise. Because of this, the updates keep moving around and visit a significant range of possible values, instead of settling on a single best point value. After many steps, this set of quantities can be used as samples from the target distribution. Recent work applies this idea directly within predictive coding: Zahid et al. [17] use Langevin dynamics on the latent activations to sample from the posterior during inference, and Oliviers et al. [18] use Monte Carlo predictive coding to learn full probability distributions of the inputs rather than point estimates. Both find that noise lets the network represent a whole distribution over its hidden states instead of a single best guess, and that models trained this way learn the data much better.

2.3 Predictive coding

This paper compares two training algorithms based on the PGM introduced below. Backpropagation (BP) trains the noiseless MAP version of the chain $\bar{\boldsymbol{y}} = f(W_2 f(W_1 \boldsymbol{x}))$, which output is a single point prediction, by computing the gradient of a loss on that prediction (mean squared error in our regression setting). It then propagates this gradient through the autograd graph and updates all weights with a single global signal. Predictive coding (PC), on the other hand, treats the hidden activations as latent variables and infers them during training. The weight update is local to each layer. We first state predictive coding in its standard form (Section 2.3.1), and then we set up the probabilistic model that both algorithms share and show that this model is exactly what the standard formulation optimises (Sections 2.3.2-2.3.3).

2.3.1 Standard PC definition

Predictive coding trains the model using the expectation-maximisation approach (EM). Its objective is an energy built from the prediction errors that are between each layers and the prediction made by the layer above it [11, 10]. In the original formulations each error is weighted by its variance. Because of the fact that we fix all covariances to the identity (Section 2.3.2), the weights are constant and the energy gets reduced to a plain sum of squared layer-wise prediction errors,

$$E = \tfrac{1}{2} \sum_{l=1}^{L} \left\| \boldsymbol{x}_l - f(W_l \boldsymbol{x}_{l-1}) \right\|^2, \tag{3}$$

In case when a network has two layers, the calculation is: $E = \frac{1}{2}\left(\|\boldsymbol{y} - f(W_2 \boldsymbol{z})\|^2 + \|\boldsymbol{z} - f(W_1 \boldsymbol{x})\|^2 \right)$. Training then switches between these two steps [10]:

- **E-step (inference).** The weights are fixed and the latent activations are inferred by minimising the energy, $\hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} E$. This is a MAP estimate of the latents after seeing $(\boldsymbol{x}, \boldsymbol{y})$, obtained by gradient descent on E. It is the PC inference loop.
- **M-step (learning).** With the latents fixed at $\hat{\boldsymbol{z}}$, the weights are updated by gradient descent on the same energy, $\Delta W_l \propto -\partial E / \partial W_l\big|_{\hat{\boldsymbol{z}}}$. Writing the layer prediction as $\hat{\boldsymbol{x}}_l = f(W_l \boldsymbol{x}_{l-1})$ and the local prediction error as $\boldsymbol{e}_l = \boldsymbol{x}_l - \hat{\boldsymbol{x}}_l$, this gives the local, Hebbian-style PC weight rule [11]

$$\Delta W_l \propto \big(\boldsymbol{e}_l \odot f'(W_l \boldsymbol{x}_{l-1})\big)\, \boldsymbol{x}_{l-1}^{\top}. \tag{4}$$

  Each weight is updated using only its own local error and the activation that goes into it. There is no global signal propagated back from the output.

This energy is mentioned here as definition. However, Section 2.3.2 shows that it is exactly the negative log-joint of the Gaussian chain in (5)–(6) (up to a fixed constant). This makes predictive coding a probabilistic graphical model, not only a learning rule.

2.3.2 Neural networks as a probabilistic graphical model

Consider this very simple non-trivial network. It has only one input $\boldsymbol{x}$, one hidden node $\boldsymbol{z}$, one output $\boldsymbol{y}$, with weights $\theta = \{W_1, W_2\}$. We associate it with the directed graphical model in Figure 2, with joint factorization

$$p_\theta(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) = p_\theta(\boldsymbol{y} \mid \boldsymbol{z})\, p_\theta(\boldsymbol{z} \mid \boldsymbol{x}). \tag{5}$$

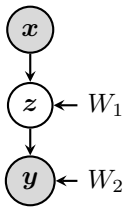

Figure 2: Graphical model for a one-hidden-node network. Shaded nodes are observed. $\boldsymbol{z}$ is a latent activation.

In PC, we model the conditionals as Gaussians, in which the mean of each layer is predicted by the layer above and identity covariance:

$$\begin{aligned} p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) &= \mathcal{N}\big(\boldsymbol{z};\, f(W_1 \boldsymbol{x}),\, I\big), \\ p_\theta(\boldsymbol{y} \mid \boldsymbol{z}) &= \mathcal{N}\big(\boldsymbol{y};\, f(W_2 \boldsymbol{z}),\, I\big). \end{aligned} \tag{6}$$

Generalising to L layers with activations $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ and weights $W_1, \ldots, W_L$ gives a chain $p_\theta(\boldsymbol{x}_{0:L}) = p(\boldsymbol{x}_0) \prod_{l=1}^{L} \mathcal{N}\big(\boldsymbol{x}_l;\, f(W_l \boldsymbol{x}_{l-1}),\, I\big)$. This PGM is the fundamental element of the theory discussed in this chapter. The models that are studied in the experimental sections are examples of it, but with different approximations of inference and learning.

2.3.3 Inference: the predictive mean

Predictive mean. For a given input $\boldsymbol{x}$, we are interested in the expected output

$$\begin{aligned} \bar{\boldsymbol{y}} = \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y} \mid \boldsymbol{x})}[\boldsymbol{y}] &= \int d\boldsymbol{y}\; p(\boldsymbol{y} \mid \boldsymbol{x})\, \boldsymbol{y} = \int d\boldsymbol{y}\, d\boldsymbol{z}\; p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})\, \boldsymbol{y} \\ &= \int d\boldsymbol{y}\, d\boldsymbol{z}\; p(\boldsymbol{y} \mid \boldsymbol{z})\, p(\boldsymbol{z} \mid \boldsymbol{x})\, \boldsymbol{y}, \end{aligned} \tag{7}$$

where we first treat $p(\boldsymbol{y} \mid \boldsymbol{x})$ as the joint $p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$ integrated over $\boldsymbol{z}$. Then we factorise it using (5) and the Markov property $\boldsymbol{y} \perp \boldsymbol{x} \mid \boldsymbol{z}$ [5]. This integral is difficult for general activations f, but (6) gives us a natural ancestral sampling scheme - sample $\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z} \mid \boldsymbol{x})$, then $\boldsymbol{y}^{(i)} \sim p(\boldsymbol{y} \mid \boldsymbol{z}^{(i)})$. This means

$$\boldsymbol{z}^{(i)} = f(W_1 \boldsymbol{x}) + \boldsymbol{\epsilon}_z, \qquad \boldsymbol{y}^{(i)} = f(W_2 \boldsymbol{z}^{(i)}) + \boldsymbol{\epsilon}_y, \qquad \boldsymbol{\epsilon}_z, \boldsymbol{\epsilon}_y \sim \mathcal{N}(0, I), \tag{8}$$

and then the predictive mean is approximated by $\bar{\boldsymbol{y}} \approx \frac{1}{T}\sum_i \boldsymbol{y}^{(i)}$. We repeat this sampling method T times and average the results. Thanks to that procedure we can receive accurate approximations - this way of estimation is called Monte Carlo estimation.

The approach in (8) is exactly the Gaussian noise injection of Section 2.1. A noiseless single forward pass corresponds to the maximum a posteriori (MAP) estimate $\bar{\boldsymbol{y}} \approx f(W_2 f(W_1 \boldsymbol{x}))$, which is the standard feed-forward neural network.

2.4 Uncertainty quantification

Predictive variance. Once the model is treated as a PGM, it is possible to obtain more information for this research. The same Monte Carlo samples used to estimate the predictive mean also allow access to the predictive variance, which we can use to later count the uncertainty of the model. The marginal predictive variance can be defined as

$$\mathrm{Var}(\vec{\boldsymbol{y}}) = \mathbb{E}_{\boldsymbol{y}\sim p(\boldsymbol{y}|\boldsymbol{x})}\left[(\vec{\boldsymbol{y}} - \bar{\boldsymbol{y}})^\top (\vec{\boldsymbol{y}} - \bar{\boldsymbol{y}})\right]. \tag{9}$$

For the scalar output used here ($d = 1$), the transpose product $(\vec{\boldsymbol{y}} - \bar{\boldsymbol{y}})^\top(\vec{\boldsymbol{y}} - \bar{\boldsymbol{y}})$ can be reduced to $(y - \bar{y})^2$, so

$$\begin{aligned}
\sigma_y^2 &= \mathbb{E}\big[(y-\bar{y})^2\big] \\
&= \mathbb{E}\big[y^2 - 2\bar{y}\,y + \bar{y}^2\big] && \text{(expand the square)} \\
&= \mathbb{E}[y^2] - 2\bar{y}\,\mathbb{E}[y] + \bar{y}^2 && \text{(linearity of } \mathbb{E};\ \bar{y} \text{ is constant)} \\
&= \mathbb{E}[y^2] - 2\bar{y}\cdot\bar{y} + \bar{y}^2 && \text{(since } \mathbb{E}[y] = \bar{y}) \\
&= \mathbb{E}[y^2] - \bar{y}^2. && \text{(combine } -2\bar{y}^2 + \bar{y}^2)
\end{aligned} \tag{10}$$

which can be estimated from the same T Monte Carlo samples used for the mean. Have in mind that σ_y^2 measures spread about the model's own mean $\bar{\boldsymbol{y}}$, not about the target. It captures how uncertain the model is rather than how wrong. The full predictive distribution is approximated by $\mathcal{N}(\bar{\boldsymbol{y}}, \sigma_y^2)$, which can be used to compute the negative log-likelihood (NLL). Starting from the Gaussian probability density function, we have:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(\boldsymbol{y}-\bar{\boldsymbol{y}})^2}{2\sigma_y^2}\right)$$

$$\begin{aligned}
\log p(\boldsymbol{y} \mid \boldsymbol{x}) &= -\tfrac{1}{2}\log\big(2\pi\sigma_y^2\big) - \frac{(\boldsymbol{y}-\bar{\boldsymbol{y}})^2}{2\sigma_y^2} \\
&= -\tfrac{1}{2}\log 2\pi - \tfrac{1}{2}\log\sigma_y^2 - \frac{(\boldsymbol{y}-\bar{\boldsymbol{y}})^2}{2\sigma_y^2}
\end{aligned} \tag{11}$$

The constant part of the equation $\frac{1}{2}\log 2\pi$ is independent of $\bar{\boldsymbol{y}}$ and σ_y^2 therefore it is irrelevant. Thus, we can negate and drop it, which gives us

$$\mathrm{NLL} = \tfrac{1}{2}\log\sigma_y^2 + \frac{(\boldsymbol{y}-\bar{\boldsymbol{y}})^2}{2\sigma_y^2} \tag{12}$$

This is also reported in the experimental chapters of Lakshminarayanan et al. [2] and Hernández-Lobato & Adams [1].

Here it is useful to mention two standard sources of predictive uncertainty: *aleatoric* and *epistemic* [6, 7]. Aleatoric is the permanent noise that is received from the data itself, and depends on the input. Collecting more data does not remove the noise. On the other hand, *epistemic* uncertainty comes from the fact that the model does not know enough from the data that it was trained on. It is the largest in regions where there was little or no training data. This source of predictive uncertainty shrinks when more data is collected.

σ_y^2 captures the sensitivity of the network to small changes to its latent activations. It is large when the function is unstable and small when its determined well. In fact, it is not an aleatoric term. The reason for this is that the conditional covariances in (6) are fixed to the identity I and are unable to capture input-dependent data noise. That kind of term would require a learned input-dependent $\sigma^2(\boldsymbol{x})$ [19, 6], which we introduce later in this section.

Monte Carlo dropout. Dropout is a type of multiplicative noise. Instead of adding Gaussian noise to the activations, it multiplies them by a random mask before interacting with the next layer. The standard MC dropout network training has the mask turned on both during training and testing. This way the T stochastic forward pass is reached, and the variance of the outputs is used to estimate the model's uncertainty.

From the perspective of PGM in section 2.3.2 the construction is exactly the same for GNI and dropout - the only feature that changes is the noise channel. The additive Gaussian in (8) is replaced either by a Bernoulli mask, which randomly zeros out activations or by multiplicative Gaussian mask, which rescales them. Lakshminarayanan et al. [2] use MC dropout as a baseline on standard regression tasks, including the $y = x^3$ toy used here, and find that it produces useful uncertainty, but is outperformed by deep ensembles on NLL and calibration.

The only-test (OT) design Recently, Ledda et al. [8] demonstrates that if we inject dropout only at test time to a network which was trained without it, we can still receive post hoc uncertainty. This suggest a design that we apply throughout the rest of the thesis. The predictive distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$ in (7) is not affected by whether the network was trained with noise turned on or off, thus the two phases can be separated. In OT design, the netowrk is trained with all noise turned off (recovering the basic PC and BP) and the noise is turned back on only during testing. We call this the only-test (OT) design and we apply it to the noise schemes used in the thesis. For BP, the analysis of Section 2.3 suggests that because of the weight gradient carrying signal from the loss, adding noise during training does not corrupt learning. Thus, the OT design should perform much like training with noise. Nonetheless, training time noise may act as a regulariser [3], so the two dont have to return identical results. Regarding PC, the design matters more - Section 2.3 predicts that training time noise destroys the local prediction errors that drive the weight update, so turning it off during training should solve the problem. The two design differ only in the learned weights - the test-time sampling used to estimate the uncertainty is unchanged. Whether the OT design improves or worsens performance is answered in later sections.

Heteroscedastic output head All noise-based methods above are about estimating σ_y^2 indirectly, by measuring how much the output varies across stochastic forward passes. As noted earlier, this variance captures sensitivity to perturbation rather than true data noise, because the conditional covariances in (6) are fixed to identity I. A heteroscedastic model removes this constraint by replacing the fixed output covariance with a learned, input-dependent one [19, 6]:

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{z}, \boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{y};\ f(W_L \boldsymbol{z}),\ \sigma^2(\boldsymbol{x})\big), \tag{13}$$

where $\sigma^2(\boldsymbol{x}) > 0$ is predicted by a separate *variance head* that reads the last hidden activation $\boldsymbol{x}_{L-1}$:

$$\sigma^2(\boldsymbol{x}) = \mathrm{softplus}(W_{\mathrm{var}}\, \boldsymbol{x}_{L-1} + b_{\mathrm{var}}) + \delta, \tag{14}$$

where $\mathrm{softplus}(a) = \log(1 + e^a)$ is a smooth, strictly positive function that guarantees $\sigma^2(\boldsymbol{x}) > 0$, and $\delta > 0$ is a small floor added for numerical stability [15, 16]. The network is trained to minimise the Gaussian NLL of (12) directly - thanks to this, both the mean and the variance are learned end-to-end from data. This is the output parametrization introduced by Nix and Weigend [19] and later used as the building block of deep ensembles [2]. Unlike noise-based methods, a stochastic forward pass is not needed at test time - a single deterministic forward pass returns both $\bar{\boldsymbol{y}}$ and $\sigma^2(\boldsymbol{x})$. Because $\sigma^2(\boldsymbol{x})$ is learned from the residuals $\boldsymbol{y} - \bar{\boldsymbol{y}}(\boldsymbol{x})$ between the observed targets and the predicted mean, it estimates input-dependent data noise (aleatoric uncertainty) [6]. As noted above, the noise is irreducible. A better fit makes the estimate more accurate, but the noise does not disappear by itself with more data.

Deep ensembles Deep ensembles [2] take a more direct way to predictive uncertainty - instead of perturbing a single network, they train M independent networks from different random initializations. Each network is a heteroscedastic model with its own mean and variance head as in (13). The networks form a mixture of Gaussians, which is summarised by a single Gaussian whose mean and variance are:

$$\mu_*(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M}\mu_m(\boldsymbol{x}), \qquad \sigma_*^2(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M}\big(\sigma_m^2(\boldsymbol{x}) + \mu_m^2(\boldsymbol{x})\big) - \mu_*^2(\boldsymbol{x}). \tag{15}$$

The two contributions to σ_*^2 separate cleanly: the averaged per-network variances $\frac{1}{M}\sum_m \sigma_m^2$ capture the aleatoric noise, while the remaining terms combine into the variance of the means, $\mathrm{Var}(\mu_m) = \frac{1}{M}\sum_m \mu_m^2 - \mu_*^2$, which captures the epistemic uncertainty. The networks agree where training data is dense and disagree where it is scarce or out of distribution [6]. This fact makes deep ensembles very well calibrated methods in practice and a strong baseline for predictive uncertainty. On the other hand, the downside is that it requires M full training runs and M stored models, in contrast to the single model methods that are the focus of this thesis.

2.5 Predictive Coding with uncertainty quantification

SGLD in predictive coding In PC, SGLD influences the E-step. Instead of taking the single best latent ($\arg\min_{\boldsymbol{z}} E$), it updates the latent with a noisy step

$$\boldsymbol{z}^{(i)} = \boldsymbol{z}^{(i-1)} - \eta\,\frac{\partial E}{\partial \boldsymbol{z}} + \sqrt{2\eta}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I), \tag{16}$$

in which the first part is the normal downhill step and the second part is the random alteration. As stated before, repeating this process returns many possible values of the hidden activations for each data point, instead of only one. SGLD is between a basic PC which keeps only the single best latent (MAP) and a full posterior estimation that uses the entire distribution.

Heteroscedastic output head of PC The heteroscedastic head of (14) can be attached to a PC network without modifying its training algorithm. The mean path - the E-step that infers the latent activations and the M-step that updates the weights layer-by-layer - is left entirely unchanged. The variance head is trained with a separate gradient step on the NLL of (12), which is taken after the regular PC weight update. During the first half of the training (*warmup*) the variance head is switched off - only the mean is trained. After that part of the process ends, the variance head turns on and learns how uncertain the network should be at each point. This approach prevents the variance from collapsing before the mean is stable [15, 16]. Because the variance head reads $\boldsymbol{x}_{L-1}$ - the penultimate activation of the feedforward pass - it uses the already existing network representation and adds no new inference cost. The result is a PC network that outputs a calibrated $(\bar{\boldsymbol{y}}, \sigma^2(\boldsymbol{x}))$ pair from a single forward pass, without any noise injection.

2.6 Evaluation metrics

The experimental sections report two quantities for every model.

Mean squared error (MSE). Point prediction accuracy is measured by

$$\mathrm{MSE} = \frac{1}{N}\sum_{n=1}^{N}\big(\boldsymbol{y}_n - \bar{\boldsymbol{y}}_n\big)^2, \tag{17}$$

where $\bar{\boldsymbol{y}}_n$ is the predictive mean for test point n. It is computed by averaging $T = 50$ Monte Carlo samples for the stochastic models, or a single forward pass for the deterministic baselines. MSE is not sensitive for the predicted variance and thus it measures only how close the mean is to the target.

Gaussian negative log-likelihood (NLL). A model is well calibrated when its uncertainty matches the errors it makes [13]. The true target should lie in the predicted range as often as the model claims it will. If a model is very confident (small σ_y) but often wrong, or not sure (large σ_y), but usually right,

is wrongly calibrated. Calibration is measured by the Gaussian NLL of (12), averaged over the test set. NLL combines accuracy and calibration. Small σ_y^2 together with a large error destroys the second term, while a large σ_y^2 inflates the first. A model that is both accurate and honestly uncertain wins on both terms in the same time. Negative NLL values are possible and indicate slightly conservative uncertainty estimates (predicted spread wider than the realized error).

3 Methodology

Dataset. All experiments are based on the one-dimensional toy regression $y = x^3 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 9)$ and $x \sim \mathcal{U}(-4, 4)$, introduced by Lakshminarayanan et al. [2]. We sample $N = 29$ points and split them 70/30 into train and test sets. This setup matches the original regression benchmark of [2]. Inputs and outputs are standardised to zero mean and unit variance using the training set statistics. Predictions are shown over $x \in [-6, 6]$, so that $x \in [-4, 4]$ is the in-distribution region and $x \in [-6, -4] \cup [4, 6]$ provides an out of distribution (OOD) region in which a well-calibrated model should report significant uncertainty.

Architecture. A single hidden layer of 100 units with ReLU activations, input $(1) \to$ Linear $\to$ ReLU $\to$ Linear $\to$ output (1). Both BP and PC use the same architecture and the same initialisation, $W \sim \mathcal{N}(0, 0.05^2)$, $b = 0$, so that any observed difference in performance is attributable to the algorithm rather than to the starting point.

Evaluation protocol. Stochastic models draw $T = 50$ Monte Carlo samples per test input. Each configuration is run with 5 independent random seeds, and then we report mean $\pm$ standard deviation of MSE and NLL across seeds. All reported MSE and NLL values are computed on the held-out test set. Training error is never reported in this paper.

4 Experiments and Results

This section evaluates the methods of Section 2 on the toy regression task described in Section 3 ($y = x^3 + \epsilon$, $\epsilon \sim \mathcal{N}(0, 9)$). We use $N = 20$ training points, matching the toy benchmark of Hernández-Lobato & Adams [1] and Lakshminarayanan et al. [2]. In contrast with the researchers, we evaluate this task also quantitatively. We hold out a further 9 points so that MSE and NLL can be reported - no validation set is used. The learning rates are fixed ($\eta_W = 0.05$, $\eta_z = 0.01$). Each configuration is run with 5 seeds, where both the data split and the initialisation vary per seed.

Before we continue, it has to be mentioned that the dataset is small, so the seed to seed variance is significant [14]. Thus, we should read the results as trends across methods instead of precise point estimates. In addition, the Gaussian NLL used here has no variance floor (Section 2.6). A model that is over-confident returns a predictive variance almost equal to zero, which together with being even slightly wrong makes the NLL explode [15, 16]. Therefore, the very large NLL values that are captured at levels where a tiny amount of noise was added are results of the over-confident uncertainty.

4.1 Deterministic baselines

In order to begin our experiments, we first need to start with establishing our baselines (Table 1, Figure 3). With a fixed learning rate, predictive coding performs very similar to the standard Backpropagation. The mean squared error of PC (0.084) is close to that of BP (0.073), with overlapping standard deviations across seeds. The fact that none of the algorithms produce an uncertainty estimation serves as motivation to perform noise based methods described below.

4.2 Dropout transfers to predictive coding

We begin with Monte Carlo dropout, which has not been previously applied to predictive coding. Two multiplicative noise channels are compared: Bernoulli masks, which randomly zero activations with probability p, and multiplicative Gaussian noise, which rescales each activation by a draw from

Model	MSE	NLL
BP (deterministic)	0.073 ± 0.056	-
PC (deterministic)	0.084 ± 0.057	-

Table 1: Deterministic baselines ($N = 29$). BP and PC return very similar results, but neither produces an uncertainty estimate.

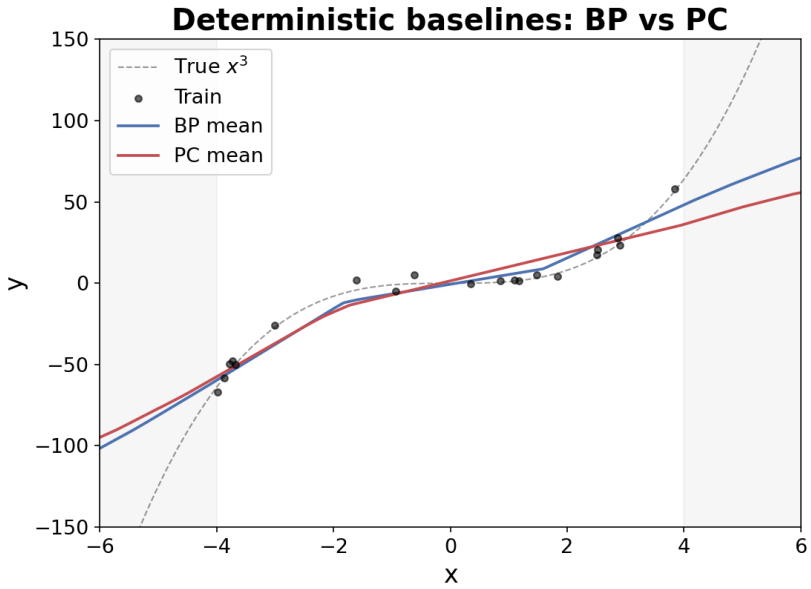

Figure 3: Deterministic baselines: BP (blue) and PC (red) fitted to $y = x^3$ (grey = OOD regions, dots = training points). Both algorithms fit the data region comparably.

$\mathcal{N}(1, \sigma^2)$. Each is run both in the standard design (noise during training and test) and under the OT design of Section 2.4. Table 2 reports all four combinations for BP and PC.

On accuracy PC with training-time dropout achieves the lowest MSE of any noise-based method in this study and both channels agree - 0.063 for Bernoulli at p=0.1 and 0.065 for Gaussian at σ^2=0.1, in both cases below the deterministic PC baseline of 0.084. This is very much consistent with dropout acting as a regulariser that improves the mean, exactly as it does for BP.

On calibration the OT design is clearly better than training with the noise turned on. For PC the best train + test NLL is 0.78 (Bernoulli, p=0.2), while the OT design reaches 0.06 for both channels. Specifically, Bernoulli at p=0.5 and Gaussian at σ^2=1 - better than the best GNI result (PC-OT-GNI, 0.24, Section 4.4). The two channels differ in how forgiving they are - Bernoulli degrades gracefully across the whole grid, while multiplicative Gaussian noise is calibrated only near $\sigma^2 = 1$ and is over-confident below it. However, at their respective best settings, the choice of multiplicative channel makes no measurable difference.

Figure 4 shows why multiplicative noise behaves well here - the bands are input dependent, widening where the activations are larger, because the perturbation scales with the quantity it multiplies. In short, dropout transfers to predictive coding for both noise channels. In the OT design its NLL is at least as good as the best additive noise result. Its uncertainty is input-dependent where GNI's as we show next is constant in the input.

The metrics above are computed in-distribution. In order to fully understand the behavior of the models, we should also inspect the out of distribution area. We do that by plotting the predictive mean and the $\pm 2\sigma$ band over $x \in [-6, 6]$, where $[-4, 4]$ is the training region and the shaded bands $[-6, -4] \cup [4, 6]$ are OOD. In each figure the columns increase the noise level and the different figures use different network depths and numbers of PC inference steps T. Noise is turned on during training and testing. In order to have one learning rate that works across all depths, these plots use $\eta_W = \eta_z = 10^{-2}$.

		BP		PC	
Noise channel	p / σ^2	MSE	NLL	MSE	NLL
Bernoulli, train + test	0.05	0.087	7.37	0.065	4.15
	0.1	0.087	1.98	**0.063**	2.19
	0.2	0.090	0.56	0.067	0.78
	0.5	0.120	0.08	0.077	0.92
Bernoulli, test only	0.05	0.074	4.50	0.087	2.94
	0.1	0.076	1.42	0.084	1.13
	0.2	0.080	0.17	0.086	0.12
	0.5	0.086	**−0.05**	0.087	**0.06**
Gaussian, train + test	0.01	0.075	3432	0.089	2414
	0.1	0.074	28.9	0.065	16.8
	1.0	0.132	0.39	0.114	2.91
	10	1.015	8826	1.823	2.09
Gaussian, test only	0.01	0.073	3271	0.084	2327
	0.1	0.074	33.7	0.084	20.8
	1.0	0.082	**0.01**	0.088	**0.06**
	10	0.489	1.95	0.355	2.13

Table 2: Monte Carlo dropout for both training algorithms: Bernoulli masks and multiplicative Gaussian noise, each with noise during training and test or at test time only. Means over 5 seeds. Both noise channels reach comparable calibration at their best setting, and BP and PC behave alike throughout: the very large NLL values at small noise are the over-confidence effect described at the start of this chapter.

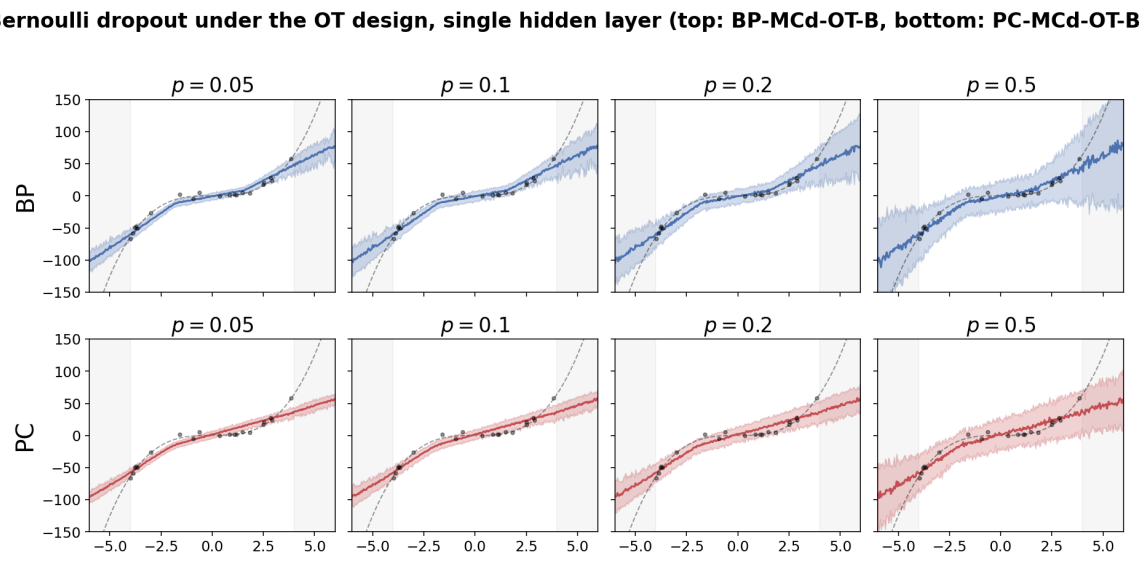

Figure 4: Bernoulli dropout under the OT design, single hidden layer (top: BP-MCd-OT-B, bottom: PC-MCd-OT-B; columns increase p). The PC bands behave like the BP bands, widening with the dropout rate: dropout transfers to predictive coding.

Unlike the single-layer experiments, the deep networks here use fan-in initialisation, $W \sim \mathcal{N}(0, 1/d_{\text{in}})$: with the fixed $\mathcal{N}(0, 0.05^2)$ initialisation of Section 3, the ReLU signal shrinks by roughly half per layer, and at ten layers both BP and PC collapse to a constant prediction regardless of the noise scheme. Fan-in scaling keeps the signal magnitude constant across depth, so the figures isolate the effect of the noise rather than of vanishing activations. We apply this first to dropout; the GNI depth figures in Section 4.4 follow the same conventions.

Dropout at depth breaks BP, not PC Using Bernoulli dropout in a deep, ten layer network (Figure 5), the two methods are similar in accuracy at low noise. Nonetheless, BP collapses over the whole input range to an almost flat prediction and at $p = 0.5$ its predictive band explodes to cover the entire plot - the uncertainty becomes uninformative. PC keeps a sloped fit and a sensible band at all dropout rates. We believe that the reason is where the noise enters each algorithm's learning signal. The BP gradient is a product of terms across all of the ten layers, so the random masks multiply through the chain. The variance of the gradient compounds layer by layer, and with only 20 training points the network retreats to predicting the mean. In PC, the weight update at layer l (eq. (4)) depends only on that layer's own prediction error $\boldsymbol{e}_l$ and its input activation. A mask at one layer perturbs one local error - there is no ten-layer product for the noise to compound through.

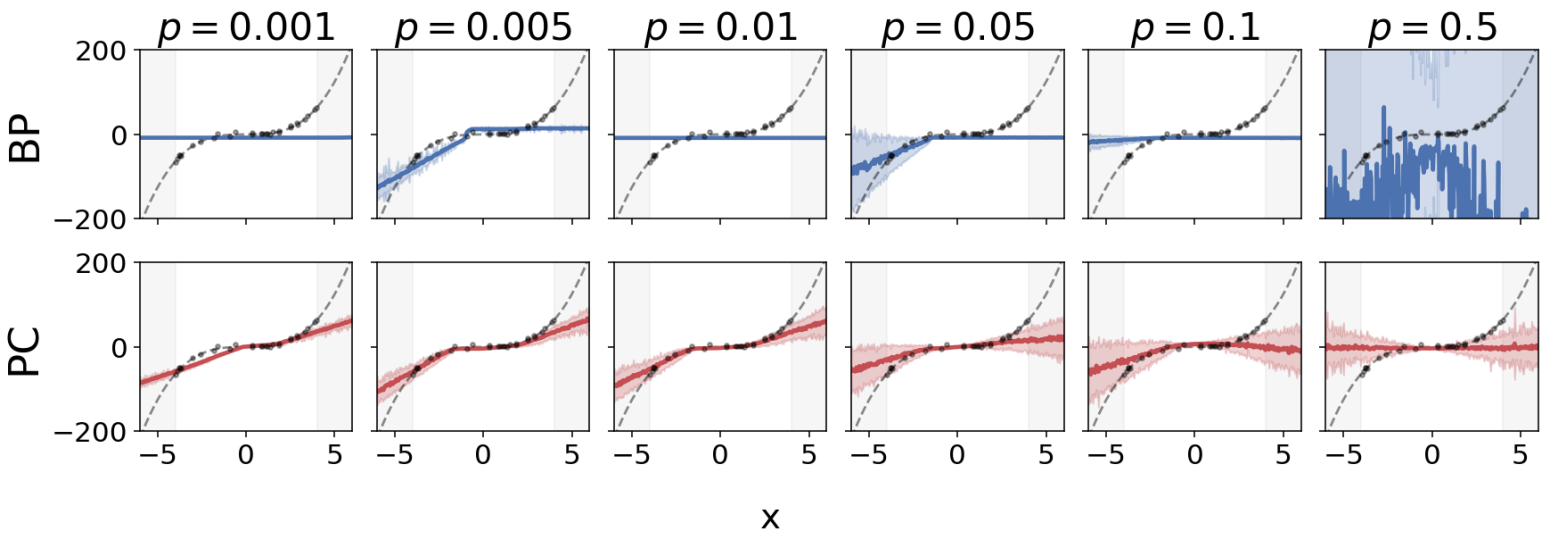

Figure 5: Bernoulli dropout, ten-layer network ($D = 10$, $T = 1$), noise during training and test (top: BP, bottom: PC; columns increase p). Deep BP collapses to a flat prediction and its band explodes at $p = 0.5$; PC keeps a reasonable fit and uncertainty band at all dropout rates.

4.3 GNI during training breaks predictive coding

We now turn to additive noise injected during PC training, using the schemes of Section 2.5: additive GNI on the activations, SGLD on the latent updates, and their combination. Table 3 reports the results.

Model	σ^2	MSE	NLL
PC-GNI	0.01	0.237 ± 0.077	1.78 ± 0.69
	0.1	0.267 ± 0.066	0.93 ± 0.26
	1.0	0.456 ± 0.303	1.06 ± 0.22
	10	0.690 ± 0.444	1.67 ± 0.15
PC-SGLD	0.01	0.249 ± 0.065	3.43 ± 1.32
	0.1	0.261 ± 0.083	0.77 ± 0.13
	1.0	0.332 ± 0.163	1.53 ± 0.06
	10	0.884 ± 0.566	2.64 ± 0.06
PC-GNI-SGLD	0.01	0.214 ± 0.082	3.42 ± 1.71
	0.1	0.213 ± 0.080	0.80 ± 0.44
	1.0	0.397 ± 0.177	1.00 ± 0.16
	10	0.841 ± 0.479	1.64 ± 0.10

Table 3: Predictive coding with noise injected during training.

	BP-OT-GNI		PC-OT-GNI	
σ^2	MSE	NLL	MSE	NLL
0.01	$\mathbf{0.072 \pm 0.054}$	$\mathbf{0.24 \pm 0.50}$	$\mathbf{0.088 \pm 0.056}$	$\mathbf{0.24 \pm 0.20}$
0.1	0.081 ± 0.046	0.77 ± 0.06	0.113 ± 0.064	1.05 ± 0.07
1.0	0.205 ± 0.036	1.86 ± 0.10	0.319 ± 0.217	2.17 ± 0.08
10	1.548 ± 0.602	3.00 ± 0.10	2.266 ± 1.577	3.32 ± 0.08

Table 4: GNI applied only at test time, for both training algorithms. At $\sigma^2 = 0.01$ both keep near-baseline accuracy and reach the same NLL of 0.24. Larger noise widens the band but degrades both metrics for both algorithms.

Every scheme is worse than the deterministic PC baseline (MSE 0.084) at every noise level. Even the smallest injected noise ($\sigma^2 = 0.01$) more than doubles the MSE, and the degradation grows with the noise scale, 0.69–0.88 at $\sigma^2 = 10$.

For GNI the mechanism is visible in the update rule. In backpropagation, $\partial\mathcal{L}/\partial W$ is driven by the output error and propagates through every hidden state. The noise injected into a hidden state perturbs the forward pass but the gradient still carries signal from the loss. In PC, on the other hand, the weight gradient at layer l is local, $\Delta W_l \propto \big(\boldsymbol{e}_l \odot f'(W_l \boldsymbol{x}_{l-1})\big)\boldsymbol{x}_{l-1}^\top$ (eq. (4)), in which $\boldsymbol{e}_l = \boldsymbol{x}_l - f(W_l \boldsymbol{x}_{l-1})$ is the local prediction error. Additive noise injected into $\boldsymbol{x}_l$ therefore corrupts $\boldsymbol{e}_l$ directly, and with enough noise $\boldsymbol{e}_l$ becomes pure noise and the weight update stops tracking the data. For backpropagation the same noise is far less damaging - the weight gradient carries the global loss signal, so training-time noise acts as a regulariser [3] rather than as corruption of the learning signal itself.

The case of SGLD is a bit different. Its noise is added to the latent update in the E-step, scaled so that the inference loop samples from the posterior instead of settling on the MAP estimate, and the weights are then updated at sampled latents rather than at the mode. This is principled sampling rather than corruption, nonetheless Table 3 shows it degrades accuracy just as much on this benchmark.

4.4 GNI at test time only

Switching off the noise during training and turning it back on during testing (PC-OT-GNI) gives very similar fit to the baseline (Table 4). For $\sigma^2 = 0.01$ the model reaches MSE equal to 0.088, almost the same as the deterministic PC baseline (0.084) and matching the best NLL of any GNI-based model in this study (0.24). Compared with Table 3, the OT design improves both metrics in the small-noise regime: at $\sigma^2 = 0.01$ it reduces the MSE of PC-GNI from 0.237 to 0.088 and the NLL from 1.78 to 0.24. BP behaves the same way under the OT design, reaching an NLL of 0.24 at the same noise level, so the two algorithms are indistinguishable on calibration here.

The noise level σ^2 controls the width of the predictive band, but it cannot be increased with no consequence. Table 4 shows that raising σ^2 widens the band, but it also worsens the mean prediction: the MSE rises from 0.088 at σ^2=0.01 to 2.27 at σ^2=10. The useful noise range is therefore small – $\sigma^2 \in [0.01, 0.1]$ gives near-baseline accuracy, while larger values degrade both MSE and NLL.

GNI captures little out-of-distribution uncertainty. Additive GNI in a shallow network (Figure 6) with one hidden layer worked well for both BP and PC. The models fit in a decent manner and their bands widen smoothly with the noise level in distribution, but they are not less confident in the OOD regions. This is the negative result of this thesis - the model is equally as confident far from the data as near it, thus the additive noise does not give useful and meaningful uncertainty.

For one hidden layer the explanation is straightforward - the noise is added after the last nonlinearity, so it only passes through the final layer, and the output variance is $\sigma^2\|W_2\|^2$ - a constant, independent of x. Input-dependent variance can only arise when the noise passes through nonlinearities after the injection point, which requires depth. This connects to section 2.3.3 - GNI variance measures local sensitivities to perturbation, which is not guaranteed to grow in unseen regions.

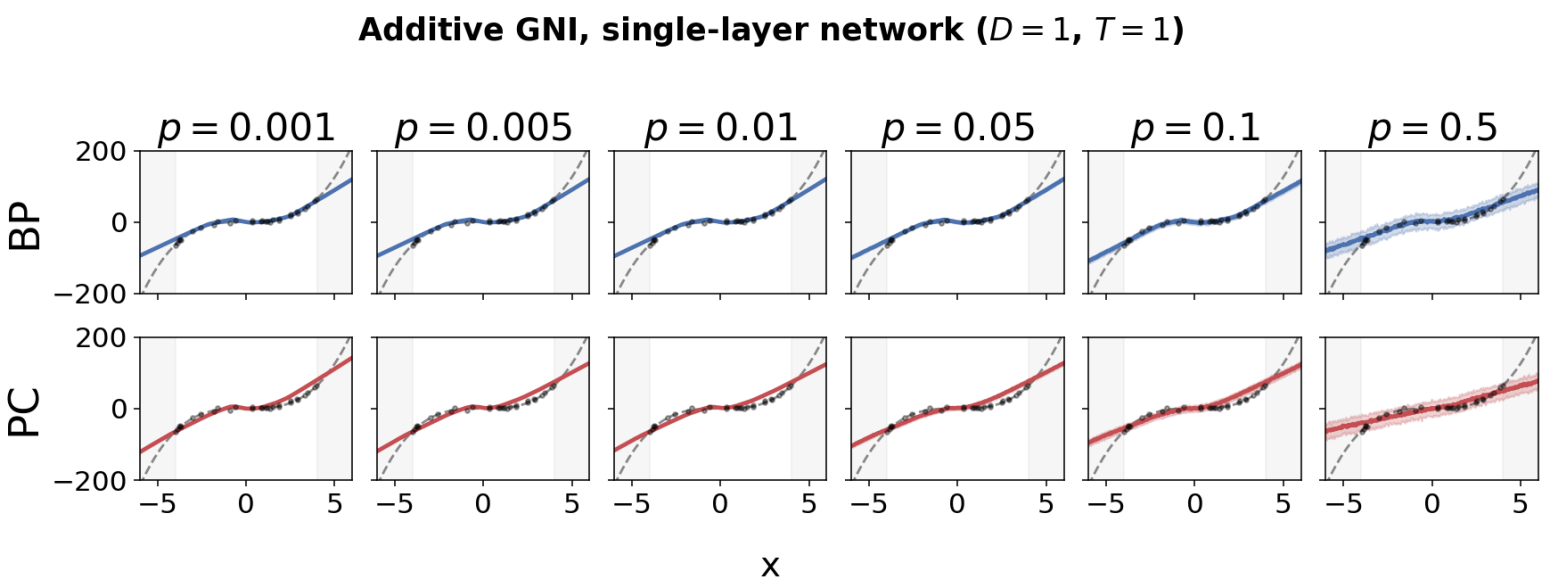

Figure 6: Additive GNI, single-layer network ($D = 1$, $T = 1$). Both BP (top) and PC (bottom) widen in-distribution with the noise level but not in the OOD regions (shaded): with one layer the output variance is constant in x, so GNI yields no out-of-distribution uncertainty.

GNI at depth breaks PC, not BP. Depth, however, does not rescue GNI - it breaks it, and specifically for PC (Figure 7). With three layers and $T = 10$ inference steps, BP remains stable at every noise level, while PC degrades progressively: at $\sigma = 0.05$ the fit visibly deteriorates, at $\sigma = 0.1$ the mean collapses into a tent shape unrelated to the data, and at $\sigma = 0.5$ the prediction is pure noise with a chaotic band covering the whole plot. This is the depth-and-noise version of the mechanism in Section 4.3 - additive noise is injected into the hidden states at initialisation, and because each layer is initialised from the layer below it that already has noise, the corruption compounds with depth. Thus, the local errors $\boldsymbol{e}_l$ are corrupted before the inference loop begins. We believe this is why three layers degrade so much more than one. BP is not affected by this, because its gradient carries the global loss signal, which averages over the injected noise.

4.5 The noise channel decides which algorithm breaks

Putting the dropout and GNI depth results together gives the most interesting result of this section. Depth plus noise breaks exactly one of the two algorithms, and which one depends on the type of noise. Multiplicative noise (dropout) compounds through BP's global gradient chain and kills BP at depth, but it scales with the activations it multiplies, so PC's local errors keep a constant signal-to-noise ratio and PC survives. *Additive* noise (GNI) contributes only a constant variance to BP's averaged gradient and leaves BP intact, but its magnitude is independent of the (small) activations it is added to, so it drowns PC's local prediction errors and kills PC at depth. Neither algorithm is more robust to noise - robustness is a property of the pairing between the learning rule and the noise channel. For PC and GNI this creates a trap. A shallow network trains fine but has no input-dependent uncertainty, and depth does not help - at three layers BP trains successfully at every noise level yet its predictive band is still flat across the input range, while PC collapses for $\sigma \geq 0.1$. Depth thus fails to deliver input-dependent variance even where training succeeds.

4.6 A heteroscedastic head: a first step towards aleatoric uncertainty

Finally, we test the heteroscedastic output head of Section 2.5 attached to a PC network. As a first sanity check, we test the model on the same dataset that we did in previous experiments. This dataset uses constant, i.e. homoscedastic (independent of the data) noise, so this experiment only partially tests the model's capacity for aleatoric uncertainty. Properly testing this would require a task with input-dependent noise, which is left as future work because of the time constraints of this project.

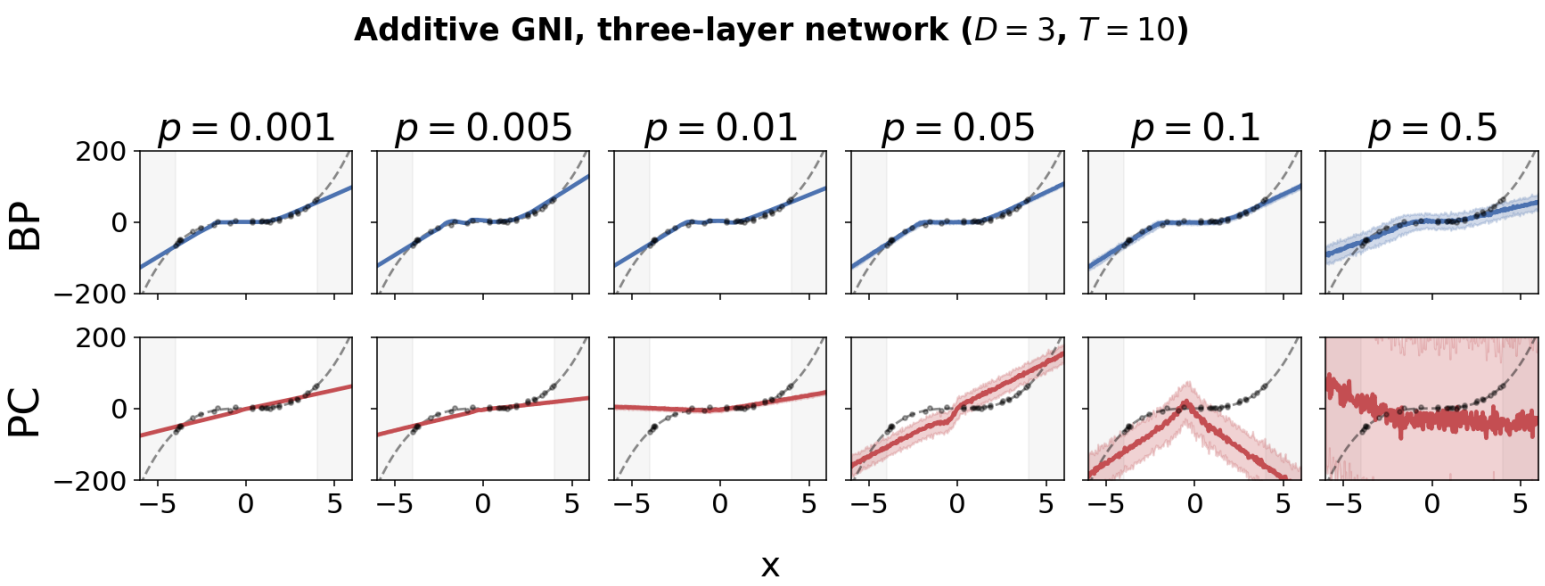

Figure 7: Additive GNI, three-layer network with $T = 10$ inference steps (top: BP, bottom: PC; columns increase σ). BP remains stable at all noise levels; PC degrades with the noise and collapses entirely at $\sigma \geq 0.1$: additive noise corrupts PC's local prediction errors, and depth compounds the corruption through the initialisation.

The head is trained under the same training circumstances as all previous experiments ($\eta_W = 0.05$, $\eta_z = 0.01$, $T = 1$, 40 epochs, same data splits and seeds).

Table 5 compares the result against the best noise-based configurations from the previous sections. The heteroscedastic head matches the deterministic PC baseline on accuracy (MSE 0.084 in both cases) and reaches an NLL of 0.26, comparable to PC-OT-GNI (0.24). It does not beat the best dropout configurations on calibration, but it obtains its uncertainty from a single deterministic forward pass, where every noise-based method requires $T = 50$ stochastic passes per prediction. In other words, the head delivers GNI-level calibration at baseline accuracy for a $1/50$ of the test-time cost, and with no noise injected in the network at any time.

Furthermore, the learned variance estimates the aleatoric uncertainty only: like GNI (Section 4.5), the band does not widen in the OOD regions, because nothing in the training objective actually rewards it for doing that. On the constant-noise task, a flat band of the correct width is in fact the right aleatoric answer. Capturing epistemic OOD uncertainty would require combining the head with an ensemble or a sampling scheme [2]. In addition, the NLL varies considerably across seeds (± 0.64) - according to the warmup design, the variance head is trained for only the final 20 epochs, and with 20 training points some seeds leave it undertrained.

Model	Best setting	MSE	NLL
PC heteroscedastic	—	0.084 ± 0.062	0.26 ± 0.64
PC-MCd-OT-B	p=0.5	0.087 ± 0.063	$\mathbf{0.06 \pm 0.26}$
PC-OT-GNI	σ^2=0.01	0.088 ± 0.056	0.24 ± 0.20
BP-MCd-OT-B	p=0.5	0.086 ± 0.062	$\mathbf{-0.05 \pm 0.38}$

Table 5: The heteroscedastic PC head against the best configuration of each noise-based family (Table 4 and Section 4.2), all under the identical training protocol. The head matches baseline accuracy and GNI-level calibration while being the only method that needs no test-time sampling.

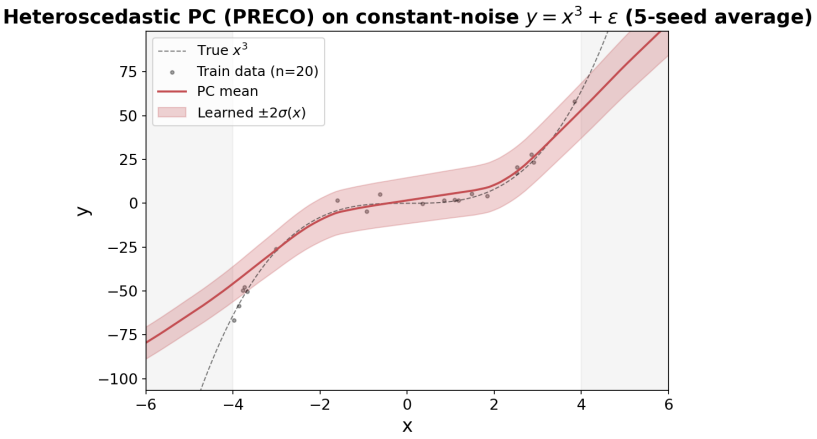

Figure 8: Heteroscedastic PC on $y = x^3 + \epsilon$ with constant noise (5-seed average).

5 Limitations

There are some limitations of this research.

Firstly, the benchmark is simple and small: a one-dimensional cubic with 20 training points and 9 test points. With so few points, individual MSE and especially NLL values vary considerably across seeds.

Secondly, NLL has no variance floor, so the NLL scale is very sensitive to near zero predicted variances. Thus, we use NLL as a comparative feature rather than a measure.

Thirdly, all conclusions are drawn from a single one-dimensional toy task. Whether they carry over to higher-dimensional regression or classification is left for future work.

Fourthly, the depth experiments of Section 4.5 required switching from the fixed $\mathcal{N}(0, 0.05^2)$ initialisation to fan-in scaling, because the former prevents any ten-layer network from training. Depth-related conclusions are therefore tied to that initialisation.

Finally, all methods are compared under one fixed set of learning rates and a coarse grid of noise levels, without per-method tuning (so the experiments stay comparable). It is possible that individual methods could reach better results under their own optimal hyperparameters, and the reported ranking should be read as a comparison under a shared protocol rather than a comparison of each method at its best performing configuration.

6 Future work

Three directions follow naturally.

First, an epistemic component: from the single-model methods studied here, only dropout's multiplicative noise widens with the input and therefore grows in the OOD region. GNI and the heteroscedastic head both report constant or near-constant uncertainty outside the training range, though for different reasons: GNI's additive noise does not scale with the activations, while the heteroscedastic head is input-dependent by construction but was only tested on homoscedastic data, where a constant variance is the correct answer. Combining the heteroscedastic head with test-time dropout or with a small ensemble [2] would add proper epistemic sensitivity while keeping the PC training algorithm unchanged.

Second, a benchmark with input-dependent noise: on the constant noise tasks used here, a learned $\sigma^2(\boldsymbol{x})$ is indistinguishable from a well-chosen constant, so a task where the noise level varies with x would test the defining ability of the heteroscedastic head.

Third, precision-weighted predictive coding. The output-level variance head is the first step towards a PC network in which every layer's covariance in (6) is learned and input-dependent rather than fixed

to the identity, connecting uncertainty estimation directly to the probabilistic model that PC already optimises.

7 Conclusions

We close by returning to the question this thesis set out to answer: whether sampling-based uncertainty methods developed for backpropagation transfer to predictive coding. The three research questions can now be answered.

RQ1: Does sampling the PC model (additive GNI) yield useful uncertainty, in particular out-of-distribution uncertainty, and how does it compare with multiplicative MC dropout and with BP? The answer to this question depends on the noise channel - for additive noise it is negative. Injected during training, additive noise corrupts PC's local prediction errors and degrades accuracy (Section 4.3), while the same noise merely regularises BP. Restricted to test time it recovers baseline accuracy, but the uncertainty bands are constant in the input. For a single hidden layer the output variance is exactly $\sigma^2\|W_2\|^2$, independent of x so GNI reports no extra uncertainty out of distribution. Multiplicative dropout (Bernoulli and Gaussian forms) on the other hand, transfers to PC directly - it improves the mean and, under the OT design, reaches an NLL of 0.06.

RQ2: under the only-test design, how do GNI and MC dropout compare between PC and BP? The uncertainty itself comes from the same test-time sampling in both designs, so the OT design does not add uncertainty. What it changes is the quality of the weights that sampling is applied to. Training without noise and sampling only at test time restores PC to baseline accuracy (PC-OT-GNI at $\sigma^2 = 0.01$: MSE 0.088 vs. baseline 0.084, NLL 0.24), whereas the same noise during training more than doubles the error (Section 4.4). Between the two algorithms the OT results are very similar - PC-OT-GNI reaches an NLL of 0.24 against 0.24 for BP-GNI-OT and PC-MCd-OT-B reaches 0.06 against -0.05 for BP-MCd-OT-B.

RQ3: does a heteroscedastic output head - a direct extension of the PC network - provide calibrated uncertainty without test-time sampling on the same benchmark? Yes. Under the identical training protocol, the head matches baseline accuracy and GNI level calibration from a single deterministic forward pass, replacing $T = 50$ stochastic passes (Section 4.6). Its uncertainty is aleatoric only, so like GNI it does not widen out of distribution. On this constant-noise benchmark a learned $\sigma^2(\boldsymbol{x})$ cannot be distinguished from a fitted constant, so the experiment tests the head's calibration rather than its ability to track an input-dependent noise level.

Beyond the research questions, the depth experiments revealed a symmetry we did not anticipate - depth plus noise always break exactly one of the two training algorithms, and which one depends on the noise type. Multiplicative noise compounds through BP's global gradient chain. Additive noise drowns PC's local errors. Robustness to noise is thus not a property of a training algorithm alone, but of its pairing with the noise channel. Taking it all together, these results suggest that predictive coding does not need a new uncertainty mechanism. It needs a compatible one.

References

[1] J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015. arXiv:1502.05336.

[2] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1612.01474.

[3] A. Camuto, M. Willetts, U. Şimşekli, S. Roberts, and C. Holmes. Explicit regularisation in Gaussian noise injections. arXiv:2007.07368, 2021.

[4] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. arXiv:1506.02142, 2016.

[5] C. M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[6] A. Kendall and Y. Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1703.04977.

[7] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021. arXiv:1910.09457.

[8] E. Ledda, G. Fumera, and F. Roli. Dropout injection at test time for post hoc uncertainty quantification in neural networks. *Information Sciences*, 645:119356, 2023. arXiv:2302.02924.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–22, 1977.

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

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

[12] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[13] V. Kuleshov, N. Fenner, and S. Ermon. Accurate uncertainties for deep learning using calibrated regression. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018. arXiv:1807.00263.

[14] X. Bouthillier, P. Delaunay, M. Bronzi, et al. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems (MLSys)*, 2021. arXiv:2103.03098.

[15] N. S. Detlefsen, M. Jørgensen, and S. Hauberg. Reliable training and estimation of variance networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1906.03260.

[16] M. Seitzer, A. Tavakoli, D. Antic, and G. Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2203.09168.

[17] U. Zahid, Q. Guo, and Z. Fountas. Sample as you infer: Predictive coding with Langevin dynamics. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pages 58105–58121, 2024.

[18] G. Oliviers, R. Bogacz, and A. Meulemans. Learning probability distributions of sensory inputs with Monte Carlo predictive coding. *PLOS Computational Biology*, 20(10):e1012532, 2024.

[19] D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, volume 1, pages 55–60, 1994.