Backpropagation and Predictive Coding: an experimental comparison

Abstract

Predictive coding is a biologically inspired learning framework that is becoming increasingly popular in the artificial intelligence community. Despite some initial results about its relation with backpropagation, most aspects are yet to be investigated. In this work, I propose an empirical comparison between the two techniques, showing significantly different learning outcomes based on the choice of different hyperparameters.

1 Introduction

Neural networks are an efficient and high-performance solution to any optimization problem and are widely researched and used in industry. The transition from shallow to deep networks began around 2012 when convolutional deep models started to achieve human-like results in image recognition tasks [5], overcoming their shallow counterparts. Nowadays deep architectures are used everywhere, from transformers [12] to GANs [3], obtaining results before considered impossible in many fields [10, 8, 11]. However, the increase in the number of layers comes at the price of explainability of the models themselves. Analysing the behaviour of this multi-dimensional objects has proven incredibly hard. As a consequence, empirical results are considered one of the best tools to understand what is happening behind the hood. In particular, the study of the behaviour of the loss function can provide insights on the effects that different hyper-parameters can produce. The simple projection of it on a one or two-dimensional space already allows for an immediate and effective visualization of some of the model's properties [6].

Despite the possibility of constructing various models with different topologies to suit the most exotic tasks, the used learning paradigm is always the same: the problem is formalised by defining a loss function $l(\mathcal{D}, \theta)$ (that represents the performance on the data $\mathcal{D}$ given the model's parameters θ) and the training process consists in finding the value $\hat{\theta}$ that minimizes it. This optimization problem is always solved by computing the gradient $\nabla = \frac{\partial l(\mathcal{D},\theta)}{\partial \theta}$ and by updating θ accordingly to the gradient descent equation

$$\theta' = \theta - \eta \nabla \tag{1}$$

where η is a small positive constant, until a local optimum is reached. However, the computation of ∇ is not trivial, considering that state-of-the-art models are composed by millions, if not billions, of parameters. For years, to solve this problem, the solution has always been the same: the backpropagation algorithm, developed in 1986 [9]. Consequently, although we are able to experiments with different hyper-parameters to tune the performance of a model, we are limited by the training technique that might affect the quality of the reached local minima.

1.1 Backpropagation

Throughout this work, I will focus on a supervised setting, where the data $\mathcal{D}$ consist of $(\boldsymbol{x}, y)$ pairs and the network M has to approximate the underlying conditional distribution $p(y|\boldsymbol{x})$. In a standard fully connected network, neurons are organised in N layers $L_1, ..., L_N$, each of which transforms its input $\boldsymbol{x_{i-1}}$ according to the formulas

$$\boldsymbol{a_i} = \theta_i \boldsymbol{x_{i-1}} \tag{2}$$

$$\boldsymbol{x_i} = f(\boldsymbol{a_i}),\ y = \boldsymbol{x_L} \tag{3}$$

where f is a non linear activation function. In order to obtain ∇, the error between the computed y and the true $\hat{y}$ is computed and backpropagated through the net according to the chain rule.

Backpropagation has proven successful throughout the years, however, multiple concerns have been raised regarding its actual feasibility as a biological learning system that can occur in our brain. In fact, it is difficult to explain how the global error signal would flow back through the network, as it requires complex and structured relationships within neurons, that have not yet been observed. It is justified, therefore, to question whether different mechanisms, more naturally inspired, would perform better.

1.2 Predictive coding

Predictive coding is a framework that only relies on local synaptic connections to perform learning [2]. Instead of trying to globally minimize the loss function, the networks optimizes a layer-local free energy function by balancing the value μ_i computed by the previous layer with the value x_i predicted to be correct for the next layer. Once this inference step reaches convergence, the network's weights can be updated to further decrease the free energy. As each computation is local (i.e., it does not involve the flowing of the error value through multiple layers), the learning is completely Hebbian [4] and, therefore, more biologically plausible. Formally, predictive coding can be described through variational inference lens, obtaining the following. As we want to model the posterior $p(y|\boldsymbol{x};\theta)$, assuming a single layer scenario, we can define a family of distributions q_θ and try to approximate p by minimizing the KL divergence

$$\mathcal{E} = D_{KL}[q_\theta(\boldsymbol{x}|y)||p(y,\boldsymbol{x};\theta)] \tag{4}$$

which is an upper bound to the divergence between $q_\theta(\boldsymbol{x}|y)$ and the posterior $p(y|\boldsymbol{x};\theta)$. Further assuming a Gaussian form for the joint distribution $p(y,\boldsymbol{x};\theta) = p(y|\boldsymbol{x};\theta)p(\boldsymbol{x};\theta) = \mathcal{N}(y; f(\theta_1 x), \Sigma_2)\mathcal{N}(x; f(\theta_2\hat{\mu}), \Sigma_1)$ and a point mass distribution for q, we obtain that

$$\mathcal{E} = -\mathbb{E}_{q_\theta}[\ln p(y,\boldsymbol{x};\theta)] = -\frac{1}{2}(\Sigma_2^{-1}\epsilon_0^2 + \Sigma_1^{-1}\epsilon_x^2 + \ln 4\pi^2\Sigma_1\Sigma_2) \tag{5}$$

where $\epsilon_0 = y - f(\boldsymbol{x},\theta_1)$ and $e_x = \boldsymbol{x} - f(\hat{\mu},\theta_2)$. In a multi layer network $\hat{\mu}$ is the $\boldsymbol{x}$ in the layer before and we have that

$$\mathcal{E} = \sum_{l=1}^{N}\mathcal{E}_l = \sum_{l=1}^{N}\Sigma_l^{-1}\epsilon_l^2 + \ln 2\pi\Sigma_l \tag{6}$$

where $\epsilon_l = \boldsymbol{x_l} - f(\boldsymbol{x_{l-1}},\theta_{l-1})$. The inference step, therefore, consists in updating the values x_l to minimize $\mathcal{E}$ until convergence, while in the learning step we update each θ_l to further minimize $\mathcal{E}$. The update dynamics are defined by the gradient descent algorithm, according to the following derivatives:

$$-\frac{\partial\mathcal{E}}{\partial x_l} = \Sigma_{l-1}^{-1}\epsilon_{l-1}f'\theta_l^T - \Sigma_l^{-1}\epsilon_l \tag{7}$$

$$-\frac{\partial\mathcal{E}}{\partial\theta_l} = \Sigma_l^{-1}\epsilon_{l-1}f'\boldsymbol{x_l} \tag{8}$$

It is easy to see that each rule requires only post and pre-synaptic values within each layer l, instead of requiring the loss value to be transmitted throughout the layer hierarchy. A visualization of the architecture is provided in figure 1. For a complete derivation of the formulas refer to [7][1] and [1].

1.3 Relations between the two learning techniques

Backpropagation and predictive coding share the same learning procedure: given an input $\boldsymbol{x}$ an output y is computed through an inference step and, successively, an error between y and the given $\hat{y}$ is computed and used to train the network. However, the nature of these steps is profoundly different between the two algorithms: backpropagation requires a single forward and backward pass involving a hierarchical computational structure among the layers, while predictive coding relies on multiple iterative steps executed independently by each layer. It has been shown, however, that under precise theoretical conditions the two learning algorithm converge to the same update rule [13]. These conditions, however, are not easily maintainable in a real-world scenario or, even, seem to be a limitation for the expressiveness of predictive coding. Throughout this work, I will try to explore the similarities and differences that arise between two identical architectures (i.e., same parameters and layout), one trained with backpropagation and the other with predictive coding. The three condition highlighted by [13] are:

[1] The paper is still an early version and contains some inconsistencies in the mathematical derivations.

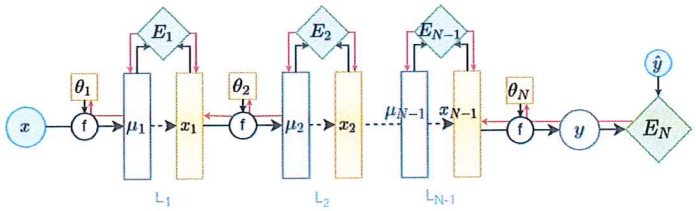


Figure 1: Learning through predictive coding. The red arrows show the errors' propagation, which affects only nearby weights via Hebbian rules.

1. $|\hat{\mathrm{y}} - \mathrm{y}| < \delta$: the predicted output should be close enough to the given label. This condition is met once the model has achieved a sufficiently high accuracy, but is not true at the beginning of the training process. Consequently it is not guaranteed that the minima found through backpropagation and predictive coding will be the same.
2. **Convergence of the inference step**: before performing the learning step and update the network's weights, it is necessary for each x_l to have converged to the minimum of $\mathcal{E}(x_1, ..., x_l, ..., x_N)$. However, it could require hundreds of inference steps to reach this state, making predictive coding several orders of magnitude slower than backpropagation and, as a consequence, not implementable for even the simplest tasks.
3. **Fixed Σ_l**: it is required that $\Sigma_l = 1$. However Σ_l could be a trainable parameter of the network (among with θ_l), allowing to optimize the energy function even further. As backpropagation does not include these extra parameters, predictive coding models could be seen as a super-set of it.

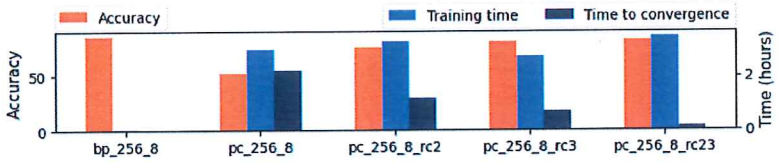
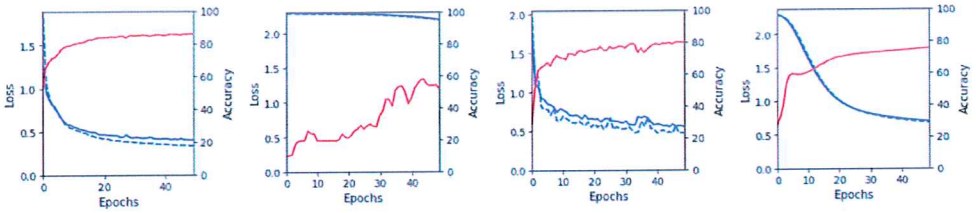


Figure 2: Benchmarks of the different models, showing accuracy and required training time. Time to convergence is the time necessary to reach 90% of the final accuracy.

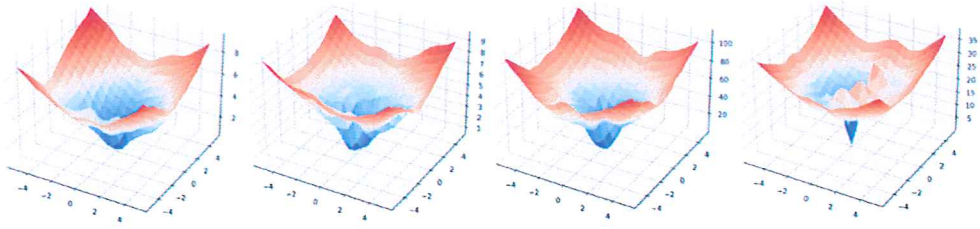
2 Experiments and results

In order to compare the different learning settings, I trained several models with different hyper-parameters using both backpropagation and predictive coding. The datasets used were MNIST and FashionMNIST; the displayed results were obtained on FashionMNIST. A predictive learning framework was developed using PyTorch and the models trained using a Nvidia Titan RTX. As condition 1 cannot be enforced, I focus on the effects of relaxing conditions 2 and 3. Consequently, for each architecture layout there will be five models: one trained with backpropagation and four trained with predictive coding, with or without conditions 2 and 3. For convenience, I will use the following naming scheme to refer to a particular network: algorithm_width_depth_relaxedConditions. For example, bp_1024_8 is a network trained with backpropagation, with hidden dimension of 1024 and 8 hidden layers, while pc_256_8_rc2 is a 256x8 network trained using predictive coding and by relaxing condition 2. To relax condition 3, it is simply necessary to add each Σ_l to the model's trainable parameters. On the other hand, to enforce condition 2, I ran the inference step on each training batch for 256 steps before performing the weights update. This was observed to be enough to guarantee the desired convergence. Instead, when removing condition 2, I updated the weights every 8 of those 256 steps, using the current values stored in the x_l variables. As shown in figure 2, this allowed for significantly reduced training times, despite backpropagation still being faster. All the results provided refer to networks composed by eight fully connected hidden layers, as, due to the computational requirements of predictive coding and the limited time available, it was not possible to experiment with more complex architectures.



(a) bp_256_8 (b) pc_256_8 (c) pc_256_8_rc3 (d) pc_256_8_rc2

Figure 3: Comparison of the performance of different models. Predictive coding with condition 3 relaxed is the most similar to backpropagation. (The accuray is shown in red, the train (dotted) and test loss in blue.)



(a) bp_256_8 (b) pc_256_8_rc3 (c) pc_256_8_rc2 (d) pc_256_8_rc23

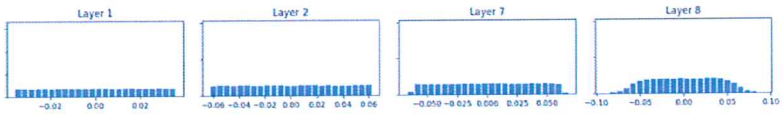
Figure 4: Loss landscape of different models. The loss function is mostly convex for every model, but the pc_256_8_rc23. This can explain the training instability experienced.

2.1 Analysis of the loss landscape

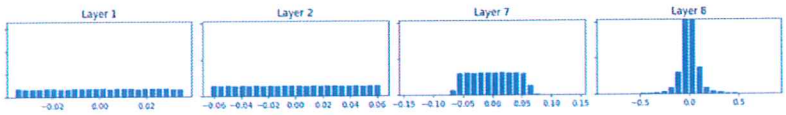
By comparing the loss value and the accuracy obtained by the same network trained via backpropagation and predictive coding, we may conclude that the two techniques are actually equivalent, as they generally produce similar results. However, it can happen that the standard predictive coding model is not able to correctly approximate and reach condition 1 (especially in narrow networks) and this results in an early convergence to a significantly sub-optimal minimum (fig. 3b). Relaxing condition 3, on the other hand, seems to bring huge improvements to these scenarios, with the network performance always reaching the backpropagation one (fig. 3c). This is partially true also for condition 2, but it may be due to the extra weight training steps available when training for a fixed amount of epochs (relaxing condition 2 allows the network to updates its weight every 8 steps, instead of 256, fig. 3d). The results are consistent among both datasets and with different hyper-parameters.

Using the code provided by Li et al. [6] adapted to work with predictive coding networks, I plotted the two-dimensional loss landscape of the various models in order to verify that, given their similar final loss and accuracy, their behaviour was comparable even in the neighborhood of the trained parameters. This was not the case, with significant differences between them. Observing figure 4 it can be seen that the landscape defined by the predictive coding training is similar to what is obtained through backpropagation when relaxing condition 3. This could be explained by considering the role of the variance Σ in equations (7) and (8): the error ϵ is scaled accordingly to the inverse of the variance, consequently a variable and trainable Σ could allow for a smoother flow of the error throughout the network and, therefore, for a faster and more stable convergence. In fact, the ability to rescale the error at a neuron level could allow the network to satisfy condition 1 more easily.

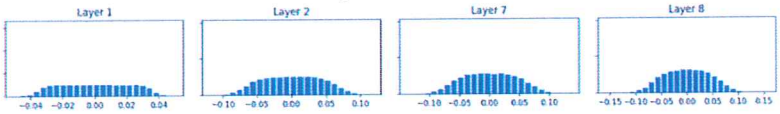
On the contrary, the effect of relaxing condition 2 is incredibly impacting and profoundly changes the dynamics of the network (fig. 4c). The loss function remains convex in most scenarios, but it rapidly grows to excessive values, showing that even a slight change in the network's weights would result in significant loss in the prediction accuracy. Relaxing both condition 2 and 3 seems to bring improvements, however the loss landscape appears chaotic and convexity is lost everywhere around each minimum but in the immediate vicinity of it (fig. 4d). This would explain why, in these settings, it was necessary to use a learning rate as low as 1e-5 in order to perform any training



(a) pc_256_8



(b) pc_256_8_rc2



(c) pc_256_8_rc3

Figure 5: Weight distributions of the first two and last two layers of different models.

at all (as it is very easy to escape from the small convex area). Appendix A shows more detailed reconstructions. Despite these difficulties, relaxing both condition 2 and 3 is, using the currently available hardware, the only feasible, though unstable, way to train scalable architectures. Future studies, should, therefore aim at understanding the causes of such different behaviours.

2.2 Analysis of the weights distributions

Similar differences can be found when analyzing the weight distributions across the networks' layers. Relaxing condition 3 results in the most similar model to the backpropagation one, in which the weights in each layer are updated and follow approximately a Gaussian distribution, as expected (fig. 5c). On the contrary, the networks trained according to the remaining settings, fail to update the deepest layers, as if the error was not flowing smoothly through the network. At the same time the last layers' weight distributions present a less regular shape, confirming that relaxing condition 3 seems to be the most performing solution (fig. 5a, 5b).

2.3 Effect of weight decay

Using the variational inference interpretation of predictive coding, we can consider the process of minimizing the free energy $\mathcal{E}$ as trying to maximize the probability

$$p(\boldsymbol{x}, \boldsymbol{x_1}, ..., \boldsymbol{x_N}) = p(\boldsymbol{x}) \prod_{l=0}^{L-1} p(\boldsymbol{x_l}|\boldsymbol{x_{l-1}}) \tag{9}$$

where $p(\boldsymbol{x_l}|\boldsymbol{x_{l+1}}) = \mathcal{N}(\boldsymbol{x_l}; f(\theta_{l-1}, \boldsymbol{x_{l-1}}), \Sigma_l)$. Consequently, adding weight decay on the trainable parameters θ and Σ could be seen as implementing some sort of Gaussian unit prior for each normal distribution $p(\boldsymbol{x_l}|\boldsymbol{x_{l+1}})$. In fact, to avoid negative variances, the network effectively stores the values $\ln \Sigma_l$. Therefore, the weight decay makes the values $\Sigma \to 1$ and $f(\theta_{l-1}, \boldsymbol{x_{l-1}}) \to 0$, and, thus, $p(\boldsymbol{x_l}) \to \mathcal{N}(0, 1)$. By applying a mild weight decay (i.e. 1e-6), we can see that the resulting loss landscape is smoother and the training process more stable (figure 6). Furthermore, the weights appear to follow a Gaussian distribution in every layer, showing that the entirety of the network is learning. However, it seems very sensitive in the choice of this hyperparameter as even slightly different values prevent any meaningful training.

3 Conclusion

In this work, I have shown how different implementations of predictive coding results in different learning outcomes. Predictive coding with trainable variance seems to lead to the results that are most similar to backpropagation. Furthermore, adding a mild weight decay allows for a fast and stable training, even when updating the model's weight before reaching convergence on the $\boldsymbol{x_i}$.

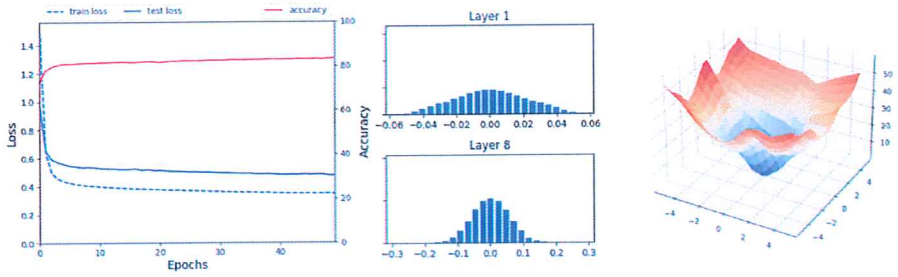


Figure 6: Performance of predictive coding when weight decay is applied. Both condition 2 and 3 are relaxed.

References

[1] Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017. Model-based Cognitive Neuroscience.

[2] Karl Friston and Stefan Kiebel. Predictive coding under the free-energy principle. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 364:1211–21, 06 2009.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[4] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[6] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, 2018.

[7] Beren Millidge, Anil Seth, and Christopher L Buckley. Predictive coding: a theoretical and experimental review, 2021.

[8] Martin Popel, Markéta Tomková, Jakub Tomek, Łukasz Kaiser, Jakob Uszkoreit, Ondrej Bojar, and Z. Žabokrtský. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 2020.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, (6088):533–536, 1986.

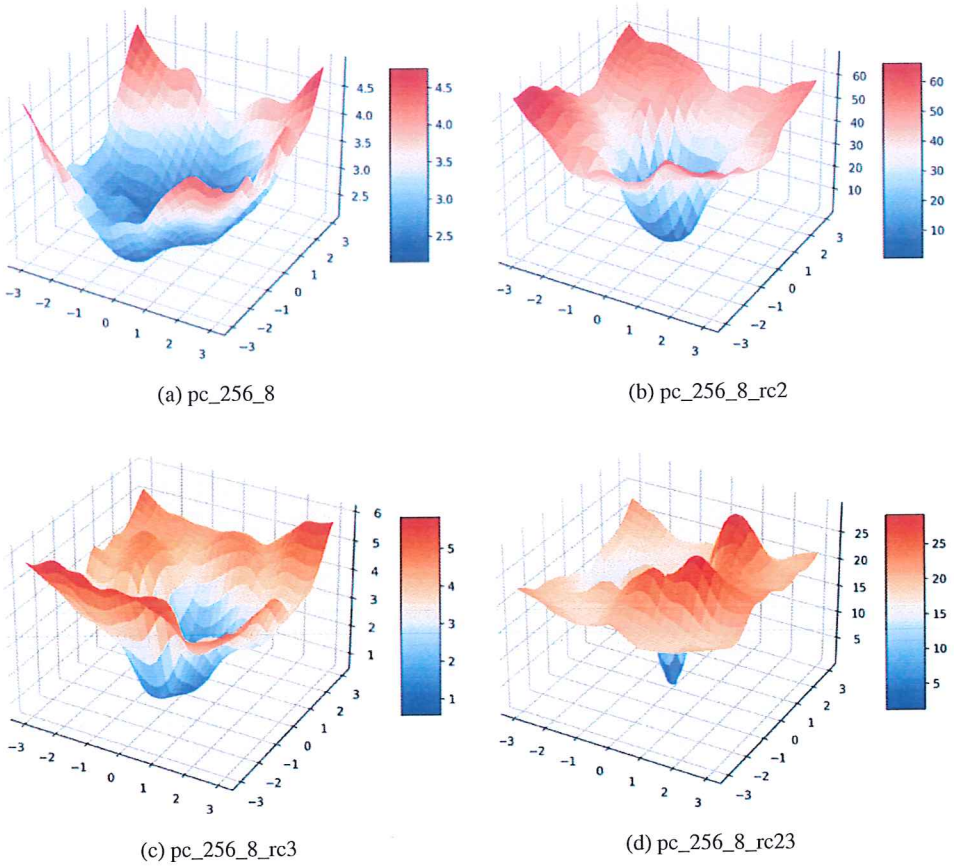
[10] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[11] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, Sameer Velankar, Gerard Kleywegt, Alex Bateman, Richard Evans, Alexander Pritzel, Michael Figurnov, Olaf Ronneberger, Russ Bates, Simon Kohl, and Demis Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596:1–9, 08 2021.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[13] 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.

A Detail of the loss landscape.



(a) pc_256_8

(b) pc_256_8_rc2

(c) pc_256_8_rc3

(d) pc_256_8_rc23

Figure 7: Zoomed version of the visualization of the loss landscape. It can be seen that pc_256_8_rc23 presents clear irregularities, while the other graphs are mostly convex.