Université de Montréal

Neurobiologically-Inspired Models: Exploring Behaviour Prediction, Learning Algorithms, and Reinforcement Learning

par

Sean Spinney

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

January 10, 2024

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Neurobiologically-Inspired Models: Exploring Behaviour Prediction, Learning Algorithms, and Reinforcement Learning

présenté par

Sean Spinney

a été évalué par un jury composé des personnes suivantes :

Guillaume Rabusseau
(président-rapporteur)

Irina Rish
(directeur de recherche)

Patricia Conrod
(codirecteur)

Dominique Beaini
(membre du jury)

Résumé

Le développement du domaine de l'apprentissage profond doit une grande part de son avancée aux idées inspirées par la neuroscience et aux études sur l'apprentissage humain. De la découverte de l'algorithme de rétropropagation à la conception d'architectures neuronales comme les Convolutional Neural Networks, ces idées ont été couplées à l'ingénierie et aux améliorations technologiques pour engendrer des algorithmes performants en utilisation aujourd'hui. Cette thèse se compose de trois articles, chacun éclairant des aspects distincts du thème central de ce domaine interdisciplinaire. Le premier article explore la modélisation prédictive avec des données d'imagerie du cerveau de haute dimension en utilisant une nouvelle approche de régularisation hybride. Dans de nombreuses applications pratiques (comme l'imagerie médicale), l'attention se porte non seulement sur la précision, mais également sur l'interprétabilité d'un modèle prédictif formé sur des données haute dimension. Cette étude s'attache à combiner la régularisation l1 et l2, qui régularisent la norme des gradients, avec l'approche récemment proposée pour la modélisation prédictive robuste, l'Invariant Learning Consistency, qui impose l'alignement entre les gradients de la même classe lors de l'entraînement. Nous examinons ici la capacité de cette approche combinée à identifier des prédicteurs robustes et épars, et nous présentons des résultats prometteurs sur plusieurs ensembles de données. Cette approche tend à améliorer la robustesse des modèles épars dans presque tous les cas, bien que les résultats varient en fonction des conditions. Le deuxième article se penche sur les algorithmes d'apprentissage inspirés de la biologie, en se concentrant particulièrement sur la méthode Difference Target Propagation (DTP) tout en l'intégrant à l'optimisation Gauss-Newton. Le développement de tels algorithmes biologiquement plausibles possède une grande importance pour comprendre les processus d'apprentissage neuronale, cependant leur extensibilité pratique à des tâches réelles est souvent limitée, ce qui entrave leur potentiel explicatif pour l'apprentissage cérébral réel. Ainsi, l'exploration d'algorithmes d'apprentissage qui offrent des fondements théoriques solides et peuvent rivaliser avec la rétropropagation dans des tâches complexes gagne en importance. La méthode Difference Target Propagation (DTP) se présente comme une candidate prometteuse, caractérisée par son étroite relation avec les principes de l'optimisation Gauss-Newton. Néanmoins, la rigueur de cette relation impose des limites, notamment en ce qui concerne la formation couche par

couche des poids synaptiques du chemin de rétroaction, une configuration considérée comme plus biologiquement plausible. De plus, l'alignement entre les mises à jour des poids DTP et les gradients de perte est conditionnel et dépend des scénarios d'architecture spécifiques. Cet article relève ces défis en introduisant un schéma innovant d'entraînement des poids de rétroaction. Ce schéma harmonise la DTP avec la BP, rétablissant la viabilité de la formation des poids de rétroaction couche par couche sans compromettre l'intégrité théorique. La validation empirique souligne l'efficacité de ce schéma, aboutissant à des performances exceptionnelles de la DTP sur CIFAR-10 et ImageNet 32×32. Enfin, le troisième article explore la planification efficace dans la prise de décision séquentielle en intégrant le calcul adaptatif à des architectures d'apprentissage profond existantes, dans le but de résoudre des casse-tête complexes. L'étude introduit des principes de calcul adaptatif inspirés des processus cognitifs humains, ainsi que des avancées récentes dans le domaine du calcul adaptatif. En explorant en profondeur les comportements émergents du modèle de mémoire adaptatif entraîné, nous identifions plusieurs comportements reconnaissables similaires aux processus cognitifs humains. Ce travail élargit la discussion sur le calcul adaptatif au-delà des gains évidents en efficacité, en explorant les comportements émergents en raison des contraintes variables généralement attribuées aux processus de la prise de décision chez les humains.

Mots-clés: apprentissage profond, neuroscience, rétropropagation, régularisation, prise de décision séquentielle, calcul adaptatif

Abstract

The development of the field of deep learning has benefited greatly from biologically inspired insights from neuroscience and the study of human learning more generally, from the discovery of backpropagation to neural architectures such as the Convolutional Neural Network. Coupled with engineering and technological improvements, the distillation of good strategies and algorithms for learning inspired from biological observation is at the heart of these advances. Although it would be difficult to enumerate all useful biases that can be learned by observing humans, they can serve as a blueprint for intelligent systems. The following thesis is composed of three research articles, each shedding light on distinct facets of the overarching theme. The first article delves into the realm of predictive modeling on high-dimensional fMRI data, a landscape where not only accuracy but also interpretability are crucial. Employing a hybrid approach blending l_1 and l_2 regularization with Invariant Learning Consistency, this study unveils the potential of identifying robust, sparse predictors capable of transmuting noise-laden datasets into coherent observations useful for pushing the field forward. Conversely, the second article delves into the domain of biologically-plausible learning algorithms, a pivotal endeavor in the comprehension of neural learning processes. In this context, the investigation centers upon Difference Target Propagation (DTP), a prospective framework closely related to Gauss-Newton optimization principles. This exploration delves into the intricate interplay between DTP and the tenets of biologically-inspired learning mechanisms, revealing an innovative schema for training feedback weights. This schema reinstates the feasibility of layer-wise feedback weight training within the DTP framework, while concurrently upholding its theoretical integrity. Lastly, the third article explores the role of memory in sequential decision-making, and proposes a model with adaptive memory. This domain entails navigating complex decision sequences within discrete state spaces, where the pursuit of efficiency encounters difficult scenarios such as the risk of critical irreversibility. The study introduces adaptive computation principles inspired by human cognitive processes, as well as recent advances in adaptive computing. By studying in-depth the emergent behaviours exhibited by the trained adaptive memory model, we identify several recognizable behaviours akin to human cognitive processes. This work expands the discussion of adaptive computing

beyond the obvious gains in efficiency, but to behaviours emerging due to varying constraints usually attributable to dynamic response times in humans.

Keywords: deep learning, neuroscience, neurobiologically plausible learning, regularization, sequential decision-making, adaptive computation

Contents

List of Tables

List of Figures

Liste des sigles et des abréviations

AMM: Adaptive Memory Model. 24

BP: Backpropagation. 24, 35
BPTT: Backpropagation Through Time. 31

CIFAR-10: Canadian Institute For Advanced Research. 24, 25
CNN: Convolutional Neural Network. 28
ConvLSTM: Convolutional LSTM. 31

DTP: Difference Target Propagation. 24, 36

F-MNIST: Fashion Modified National Institute of Standards and Technology. 25
FA: Feedback Alignment. 35
FC: Fully Connected. 75
FFN: Feedforward Neural Network. 27, 30
fMRI: Functional Magnetic Resonance Imaging. 23, 24

GMP: Gradient Matching Property. 64
GN: Gauss-Newton. 24

JMC: Jacobian Matching Condition. 64

KL: Kullback-Leibler. 34

L-DRL: Local Difference Reconstruction Loss. 63
LSTM: Long Short-Term Memory. 31

MDP: Markov Decision Process. 39
MLP: Multi-Layer Perceptron. 15, 27, 28
MNIST: Modified National Institute of Standards and Technology. 23
MRI: Magnetic Resonance Imaging. 23

NN: Neural Network. 32, 33

p-DDTP: Parallel Direct Difference Target Propagation. 25

RL: Reinforcement Learning. 32
RNN: Recurrent Neural Network. 30

s-DDTP: Simple Direct Difference Target Propagation. 25
SL: Supervised Learning. 32

TP: Target Propagation. 36

UL: Unsupervised Learning. 32

Remerciements

I would like to extend my heartfelt gratitude to my supervisors, Patricia Conrod and Irina Rish, for their invaluable support and patience. Their guidance has made this research journey both rewarding and enlightening.

Introduction

The evolution of deep learning owes a substantial debt to the synergistic collaboration between neuroscience and machine learning. This longstanding partnership has continued to enrich the design of new technologies and yielded profound insights in neuroscience and medicine in general. It is within this context that deep learning has flourished, drawing inspiration from the intricate workings of biology to pave the way for transformative algorithms. Notable examples range from the inception of backpropagation to the development of attention mechanisms and Convolutional Neural Networks. These seminal breakthroughs, rooted in both neuroscience and machine learning, have catalyzed the creation of algorithms that have left a significant impact on the field. This thesis encompasses three articles that exemplify the profound impacts of this interdisciplinary fusion.

The first delves into predictive modeling with high-dimensional functional magnetic resonance imaging (Functional Magnetic Resonance Imaging (fMRI)) data using hybrid regularization. In many practical applications (e.g., medical imaging), we are often concerned not only with the accuracy but also with the interpretability of a predictive model trained on high-dimensional data, such as its ability to identify a sparse subset of invariant (robust) predictors, generalizing across a wide variety of datasets affected by spurious noise (e.g., key factors related to a disease, across different patients and hospitals). Towards this goal, we explore here a combination of the sparsity-inducing l_1 and l_2 regularization, which focus on regularizing the norm of the gradients during training, with the recently proposed approach for robust predictive modeling, Invariant Learning Consistency, which forces alignment between gradients of the same class during training [78]. We investigate the ability of the combined approach to identify robust sparse predictors and demonstrate promising results on several datasets, including synthetic data, the Modified National Institute of Standards and Technology (MNIST) benchmark, and a functional Magnetic Resonance Imaging (MRI) dataset. Our approach tends to improve the robustness of sparse models in practically all cases, albeit with varying degrees of success and under certain conditions.

The second article centers on biologically-inspired learning algorithms, specifically concentrating on Difference Target Propagation (Difference Target Propagation (DTP)), while integrating it with Gauss-Newton optimization. The development of such biologically-plausible algorithms holds significance in comprehending neural learning processes; however, their practical scalability to real-world tasks is often constrained. This limitation impedes their explanatory potential for real brain learning. Thus, the exploration of learning algorithms that offer robust theoretical foundations and can rival backpropagation (Backpropagation (BP)) in complex tasks gains prominence. Difference Target Propagation (DTP) emerges as a promising candidate, characterized by its close association with Gauss-Newton (GN) optimization. Nonetheless, the rigor of this relationship imposes limitations, particularly regarding layer-wise training of feedback pathway synaptic weights, a configuration deemed more biologically plausible. Furthermore, the alignment between DTP weight updates and loss gradients is conditional and contingent upon specific architecture scenarios. This paper addresses these challenges by introducing an innovative feedback weight training scheme. This scheme harmonizes DTP with BP, restoring the viability of layer-wise feedback weight training without compromising theoretical assurances. Empirical validation underscores the efficacy of this scheme, culminating in DTP's unparalleled performance on Canadian Institute For Advanced Research (CIFAR-10) and ImageNet 32×32.

The third article explores efficient planning in sequential decision-making by integrating adaptive computation to existing machine learning architectures with the objective of solving complex puzzles. The study introduces adaptive computation principles inspired by human cognitive processes, and recent work investigating similar mechanism but applied to different domains [28, 8]. Our experiment demonstrate how our (Adaptive Memory Model (AMM)) can learn to vary its depth of analysis to become more efficient than its static counterpart, without any loss in performance. We have also observed a notable negative correlation between thinking time and task completion progress. Additionally, we observe that a trained AMM exhibits patterns of thinking akin to human cognitive processes. We identify three such mechanisms which we explore using various visualizations: increasing certainty, exploring alternatives, and reconsideration. We find certain trademark characteristics to each of these thinking patterns, such as distinct patterns in the evolution of the entropy of the agent's policy across pondering steps.

My contributions:

- The contributions of Sean Spinney to the first article titled: *Identifying Invariant and Sparse Predictors in High-dimensional Data*, were in the conception, programming, fMRI data preprocessing and quality control, as well as the analysis and writing of the article. The paper was accepted at the ICML 2021 conference on Uncertainty &

Robustness in Deep Learning.

- For the second article, *Towards Scaling Difference Target Propagation by Learning Backprop Targets*, Sean Spinney was tasked with the programming of the baseline alternative models used as a benchmark for the proposed algorithm (Simple Direct Difference Target Propagation (s-DDTP), Parallel Direct Difference Target Propagation (p-DDTP), and [63]), running and logging the results, hyperparameters, and architecture details for MNIST, Fashion Modified National Institute of Standards and Technology (F-MNIST), and CIFAR-10 experiments. He was also responsible for creating the figures for those experiments, with the guidance of Maxence Ernoult. This work was accepted at ICML in 2022.

- Finally, Sean Spinney was involved in every step of the third article titled *On the Role of Memory in Planning and Reasoning within Sequential Decision-Making Domains*, that is, the conception, programming, analysis, and writing. At the time of writing, this project is in the process of being submitted to ICLR 2024.

Chapter 1

Deep learning basics

With traditional machine learning failing to produce algorithms that can solve central problems in AI, such as recognizing speech or recognizing objects [26], the advancement of deep learning has opened up new learning capabilities at much larger scale and for a wide array of data types [43]. Deep learning is at the core of the work presented in this thesis, and this section will review the architectures used in our models.

1.1. Feedforward neural networks

A Feedforward Neural Network (FFN) is the most basic type of neural network, where the goal is to approximate some function f^* [26]. Whether the problem to be solved is classification or regression, the neural network maps the input $\mathbf{x}$ to a category or real number $\mathbf{y}$, and computes a loss function which allows a learning signal to update the parameters θ of the neural network to improve performance. The most basic architecture for a FFN is a multi-layer perceptrons neural network.

1.1.1. Multi-layer perceptrons

MLP have no feedback information or internal states (*e.g.* memory) and are typically composed of units called *neurons* which are connected by way of how information is propogated. Deep MLPs are built from many layers which improve its ability to solve the problem by increasing capacity, at the risk of overfitting to the training data. A layer is typically composed of a linear followed by a nonlinear transformation of the input, which can come from the data or from a previous layer. In fact the power of feedforward neural nets lies in stacking layers to represent more complex functions, making them "deeper". For example, if the model $y = f(\mathbf{x})$ has 3 layers then we can decompose the mapping to $y = f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$. The previous example implies that information flows in a forward way through the layers.

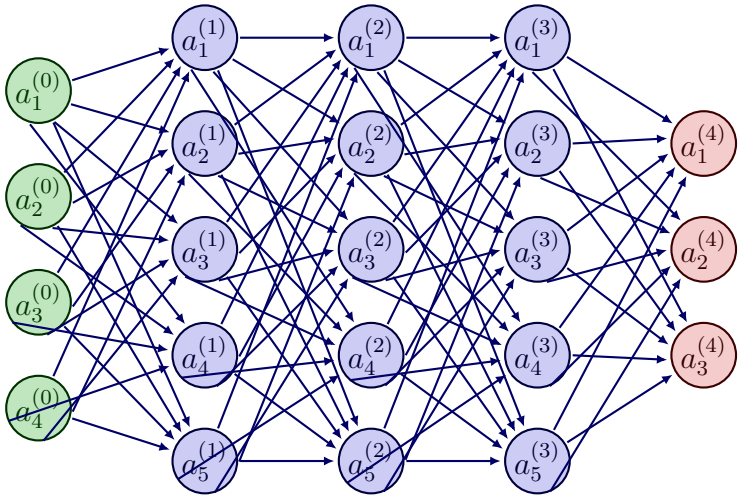


Figure 1.1. An example of a fully connected MLP with the green layer representing the input and red the output layer.

1.1.2. Convolutional neural network

This section introduces Convolutional Neural Network (CNN) which are useful for vision models primarily because they encode translation equivariance for small translations [48]. This allows the model to idenfity an object no matter its position or orientation in an image. For example, suppose K is a convolution kernel of dimension 3x3 which is applied on an input I of dimension 7x7. Computing the convolution operation is done by multiplying element-wise a subset of I and K, adding up the terms, and then moving the convolution kernel over to another subset. The distance the kernel is moved is called the stride. The highlighted subset of I in Figure1.2 shows which elements of K they are multiplied with to give the output of 4. Moving K by one step over to the right will give 1 found to the right of 4 in the I*K matrix. Let's further assume that the kernel has been trained to recognize a dog's ear during training for classifying whether or not a dog is in the input image. The output of the convolution $I * K$ is of dimension 5x5 and is shown in Figure1.2.

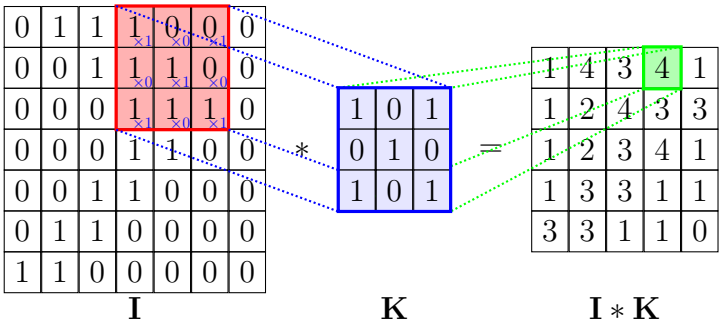


Figure 1.2. I is the input, K the convolution kernel and I*K is the output of the convolution operation of K on I.

A convolutional neural network will generally be composed of layers with multiple kernels (*e.g.* K in Fig 1.1). Such an example is given below:

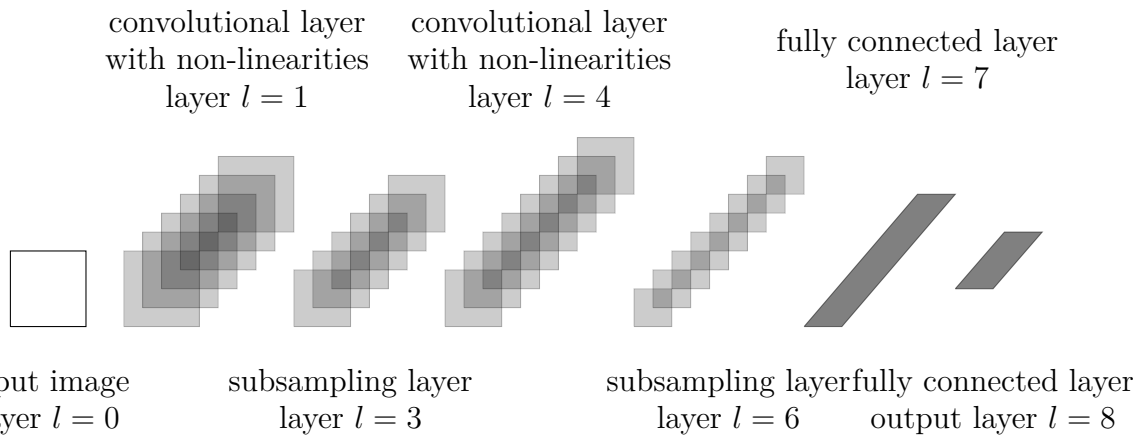


Figure 1.3. The architecture of the original convolutional neural network, as introduced by LeCun et al. (1989), alternates between convolutional layers including hyperbolic tangent non-linearities and subsampling layers. In this illustration, the convolutional layers already include non-linearities and, thus, a convolutional layer actually represents two layers. The feature maps of the final subsampling layer are then fed into the actual classifier consisting of an arbitrary number of fully connected layers. The output layer usually uses softmax activation functions.

1.1.3. Autoencoders

Autoencoders, a class of feedforward neural network architectures, have gained prominence in both machine learning and neuroscience due to their potential to capture informative data representations through unsupervised learning. An autoencoder comprises two main components: an encoder and a decoder. The encoder takes input data and transforms it into a lower-dimensional latent space, often referred to as an information bottleneck [32]. This bottleneck forces the network to capture the most salient and essential features of the input data, acting as an efficient representation that encapsulates the core information. The decoder then reconstructs the original input data from this compressed representation, resulting in a reconstruction that is ideally as close as possible to the original input.

The training process of autoencoders involves optimizing the network's parameters to minimize the difference between the input data and the reconstructed output. This is typically achieved by defining a loss function that quantifies the dissimilarity between the original input and the generated output. The most common loss function for this purpose is the Mean Squared Error (MSE), which measures the average squared difference between corresponding elements of the input and output vectors:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{1.1.1}$$

here x_i represents the corresponding element of the reconstructed output, and N is the total number of elements in the vectors.

The objective of training is to adjust the weights and biases of the encoder and decoder networks to minimize the loss. This optimization is typically performed using gradient-based methods like stochastic gradient descent (SGD) or its variants. The gradients of the loss with respect to the network parameters are computed through backpropagation, and the parameters are updated in the direction that reduces the loss.

During training, the encoder learns to extract informative features from the input data, while the decoder learns to generate accurate reconstructions from these features. As the training progresses, the network adapts its parameters to minimize the reconstruction error, effectively learning a compact representation that encapsulates the essential features of the input data.

1.2. Recurrent neural networks

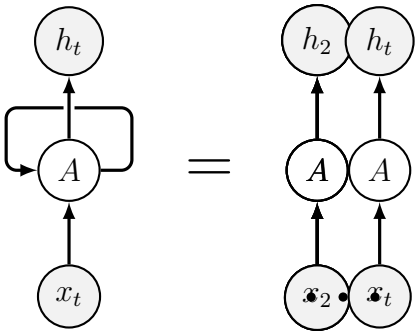


Figure 1.4. Recurrent neural network. To the right of the equality is a representation of the recurrence by "unrolling" the RNN.

Previously, we only considered single inputs which the FFN processes and an output is given by the model. Suppose instead that the input is composed of a sequence of ordered inputs such as an english sentence or a video as a series of images. It would be possible to combine the sequence of inputs into one large X that is fed to a FFN, but there is still no concept of memory. This in part motivated the creation of Recurrent Neural Network (RNN); a neural network capable of taking a sequence of inputs and retaining long-term dependencies by encoding them in a hidden state vector and making predictions at any point [26]. By carefully specifying how the hidden state is updated, h_t in Figure 1.4, RNNs can theoretically model dependencies across infinite time horizons. The major problem with

RNNs are exploding or vanishing gradients which are caused by backpropogating through very long time horizons, which is a bottleneck on how far back RNNs can model dependencies.

1.2.1. Long short-term memory

This prompted the development of the Long Short-Term Memory (LSTM) module, which allows long-term dependencies to flow through the model by using gates. The LSTM is a type of recurrent neural network that is designed to handle long-term dependencies in sequential data [37]. Unlike traditional RNNs, which can have difficulty learning and remembering long-term dependencies [13], LSTMs are able to capture and retain information from long sequences of data, and to use this information to make predictions and generate outputs.

LSTMs are able to handle long-term dependencies by using gates, which are neural network units that control the flow of information through the network. Each LSTM cell contains several gates, including input gates, output gates, and forget gates, which are used to regulate the flow of information into, out of, and within the LSTM cell. The gates are trained using a variant of Backpropagation Through Time (BPTT) [68, 90], which allows the LSTM to learn and adapt its behavior based on the input data and the desired output.

One advantage of LSTMs is that they can learn to make predictions and generate outputs based on long sequences of data, without losing track of the long-term dependencies and patterns in the data. This allows LSTMs to perform well on tasks that require the ability to remember and use long-term information, such as natural language processing, speech recognition, and machine translation. Additionally, LSTMs are able to handle complex and variable-length sequences, which makes them versatile and applicable to a wide range of tasks.

1.2.2. Deep repeated convolutional LSTM

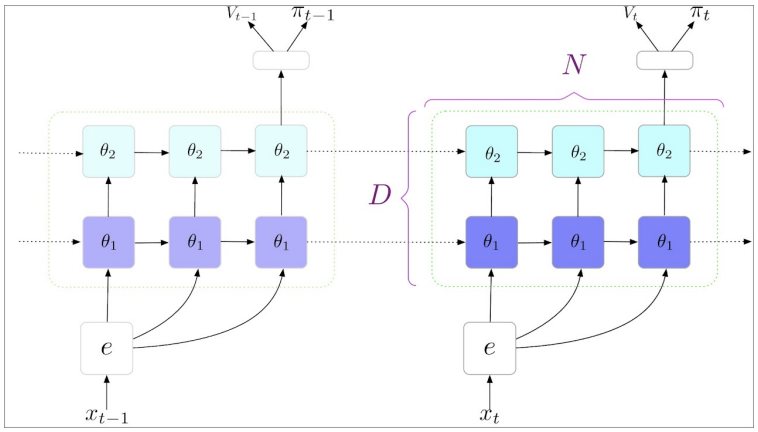


Figure 1.5. Deep Repeated ConvLSTM (DRC) architecture. Source: [31]

Combining the concepts of CNNs and LSTMs gave rise to Convolutional LSTMs or Convolutional LSTM (ConvLSTM) which allow a richer spatial structure to be modelled in

time [101]. The main idea is that instead of flattening the input to the ConvLSTM, the three dimensional structure is conserved using a 3-D hidden state representation throughout the rollout. There is good evidence to support that for vision tasks, this module performs better. One advantage of using ConvLSTMs for vision tasks is that they can capture and retain spatial information from the input data, and use this information to make more accurate and informative predictions. For example, in a task such as image classification, a ConvLSTM can learn to recognize and classify objects in an image by using the spatial relationships between the pixels in the image, and the spatial relationships between the objects in the scene. This ability to capture and use spatial information can improve the performance of ConvLSTMs on vision tasks, and make them more effective and versatile than traditional LSTMs.

1.3. Training and evaluation

The optimization involved in training a Neural Network (NN) relies on making changes to the weights θ such that performance is improved. How those changes are implemented is often referred to as the credit assignment problem, where the objective is to obtain a certain desired behaviour from the model by finding an optimal set of weights [64, 97]. When training a deep NN, the problem we aimed to solve will generally determine whether the NN learns in a supervised fashion where input and output pairs are fed to the model, or unsupervised which means the model only requires input and learns without a corresponding output example [96]. Both of these approaches can be combined, where Unsupervised Learning (UL) facilitates Supervised Learning (SL) and Reinforcement Learning (RL) through pretraining more compact representations using UL before feeding these to a task [7, 98, 99]. The focus of the current work is primarily on the supervised learning setting, as the RL algorithms under study require receiving rewards from the environment. This setting involves predicting output labels from input data to enable the training of models that can perform specific tasks with the aid of target labels.

Most deep learning models are trained using *backpropogation* [60, 111, 90]; an algorithm which updates the weights of a model after a certain number of examples have been evaluated in a way that lowers the current loss. Backpropogation uses gradient descent to find weights that minimize a loss function which is defined by the user *e.g.* cross-entropy loss. Let $\mathbf{X}$ be a batch of inputs, $\mathbf{y}$ the corresponding correct outputs, $\hat{\mathbf{y}}$ the model's output, and $\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = \mathbf{e}$ be the loss function which outputs a scalar value we will call the error. Then the update for the weights located at layer i following backpropogation is:

$$\Delta\theta_i = -\alpha \cdot \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \theta_i}$$

where α is the learning rate, a hyperparameter which determines the size of the weight updates. Once the model has been trained, it can be evaluated on a separate dataset called the validation set to assess its generalization performance. The model's performance on the validation set is used to choose the best performing model, or to determine if the model is overfitting to the training data and needs to be regularized or have more data added.

In Supervised Learning, the NN is trained using input-output pairs, where the inputs are fed to the model, and the corresponding desired outputs are provided as targets. The goal is for the model to learn a mapping from inputs to outputs that approximates the underlying relationship within the data. This process involves minimizing a loss function that quantifies the difference between the predicted outputs and the true targets.

Mathematically, given a dataset $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i)$ consisting of N input-output pairs, the loss function L can be defined as the average loss over the dataset:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\mathbf{y}_i, f(\mathbf{x}_i; \theta)), \tag{1.3.1}$$

where $\mathcal{L}$ is a suitable loss function, such as the mean squared error or cross-entropy, and $f(\mathbf{x}; \theta)$ is the output of the NN with weights θ for input $\mathbf{x}$. The weights are updated iteratively using gradient descent:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t), \tag{1.3.2}$$

where α is the learning rate controlling the step size of the updates.

Conversely, Unsupervised Learning encompasses various techniques aimed at extracting valuable insights from input data without the presence of corresponding output labels. This approach serves to uncover inherent patterns, structures, or relationships within the data itself. Several common types of unsupervised learning tasks include: clustering, learning by association, anomaly detection, and autoencoders [69].

1.3.1. Regularization

When training deep learning models, it is often required to improve the robustness of deep learning models and prevent overfitting, which occurs when a model learns to perform well on the training data but struggles to generalize to new, unseen data. Regularization techniques are employed to mitigate this issue by adding constraints or penalties to the model's learning process. These constraints encourage the model to prioritize simpler solutions and avoid fitting noise in the data, leading to improved generalization performance. Regularization methods play a vital role in promoting model

stability and preventing the model from becoming overly complex, ultimately enhancing its ability to generalize to new data points. In the following sections, we delve into some commonly used regularization techniques and their implications for training deep learning models.Two commonly used forms of regularization in deep learning are L1 and L2 regularization. These techniques add a penalty term to the loss function during training, discouraging the model from assigning excessively large weights to certain features or parameters.

L1 Regularization adds a penalty proportional to the absolute value of the weights to the loss function, mediated by a hyperparameter λ. This encourages the model to drive some of the weights to exactly zero, effectively selecting a subset of important features while disregarding others. L1 regularization is particularly useful for creating sparse models, where only a small subset of features are actively contributing to the model's predictions. Sparse models are not only computationally efficient but can also provide interpretability and insight into the underlying patterns in the data.

$$\mathcal{L}_{\text{reg}} = \lambda \sum_i |w_i|$$

L2 Regularization, on the other hand, adds a penalty proportional to the square of the weights to the loss function. This encourages the model to distribute the weights more evenly across all features, avoiding extreme values. L2 regularization is effective in preventing individual features from dominating the model's predictions and can lead to smoother weight distributions.

$$\mathcal{L}_{\text{reg}} = \lambda \sum_i |w_i|$$

Another effective approach to regularization involves the use of Kullback-Leibler (KL) divergence, a concept from information theory [83, 109]. KL divergence measures the difference between two probability distributions, indicating how one distribution diverges from another. In the context of regularization, KL divergence can be used to enforce certain properties on the learned parameters of a deep learning model. In other words, it is the expectation of the logarithmic difference between the probabilities P and Q:

$$\mathcal{L}_{\text{reg}} = \lambda \sum_i D_{\text{KL}}(P \parallel Q) = \lambda \sum_i \sum_j P_{ij} \log \left(\frac{P_{ij}}{Q_{ij}} \right)$$

where i and j are indices denoting the elements of the probability distributions P and Q. P_{ij} and Q_{ij} represent the values of the probability distributions P and Q at indices i and j, respectively, and is the regularization strength (hyperparameter). In this formulation, the Kullback-Leibler divergence $D_{\text{KL}}(P \parallel Q)$ is calculated for each pair of corresponding elements in the probability distributions P and Q, and the results are summed up to create the overall regularization term. By applying KL divergence regularization to a parameter

which describes a probability distribution as seen above, we can steer its values towards a predefined distribution, effectively constraining its variability. This regularization approach is particularly useful when we have prior knowledge about the desired distribution of a parameter, or when we want to ensure that a certain property is maintained during training. KL divergence regularization allows us to incorporate domain-specific information into the learning process, leading to models that adhere to specific constraints and exhibit desired behaviors.

1.4. Biologically plausible learning algorithms

BP is just one algorithm developed for training deep learning models, but its biological plausibility has been questioned upon considering how the brain learns by modifying synaptic connections between neurons [62, 25]. For instance, BP relies on linear transformations during propagation from output to weights, whereas biological neural networks involve a combination of linear and non-linear operations. Moreover, for the brain to accurately assign credit along feedback pathways like BP, it would need precise knowledge of derivatives for the non-linearities in the feedforward path. BP uses the same weights for both the forward pass and backward error propagation, which necessitates imposing symmetry and transposition of feedforward connections, which is unlikely happening in the brain given the uniqueness of each neuron [14, 57]. This is commonly referred to as the weight transport problem [58]. More discrepancies exist such as the fact that biological neurons use discrete signals (spiking) to communicate, rather than real-valued numbers [40], the timing of the propagation of the learning signal [114], and the observation of local connectivity patterns in real neural circuits, suggesting that learning happens locally, at the synaptic level, instead of relying on error signals being backpropogated down feedback pathways [39]. This is more plausible than supervised learning with examples and labels, since the origin of such labels in the brain is not known. There are several alternative algorithms for more biologically plausible learning, some of which are briefly mentioned below followed by a more detailed look at *Target Propagation* since it is the focus of the second article of this thesis.

Feedback Alignment (FA). : Feedback Alignment (FA) is a biologically plausible learning rule that decouples feedforward and feedback weights, making it more consistent with the biological structure of neural networks by relaxing BP's weight symmetry requirements [59]. Another concurrent work, focused on aligning the signs of the forward and feedback weights [56]

Spike Timing Dependent Plasticity (SDTP). : STDP approaches learning by adjusting the synaptic weights based on the relative timing of pre- and post-synaptic spikes [114, 14]

Target-Propagation (TP, DTP). : Target Propagation (TP) and its variant, DTP [52], represent a distinct line of research exploring alternatives to backpropagation. Unlike BP, TP and DTP propagate target activations instead of error signals to the hidden layers. This involves updating the weights of each layer to align with the propagated target activation [12, 14, 75, 49]. In essence, rather than computing gradients, these methods calculate *targets* by utilizing autoencoders at each layer. DTP refines TP by introducing a linear correction to compensate for autoencoders' imperfections [52, 63]. TP and DTP effectively address two key concerns regarding the biological plausibility of BP: the weight transport problem and the weight symmetry requirement. The following section provides a more comprehensive insight into how TP and DTP generate learning signals.

Consider a fully connected feedforward network characterized by a sequence of forward mappings:

$$h_i = f_i(h_{i-1}) = s_i(W_i h_{i-1}) = s_i(a_i), \quad i = 1, \ldots, L$$

Here, h_i denotes the vector containing the post-activation values of layer i, a_i signifies the pre-activation values, s_i represents a smooth nonlinear activation function, W_i stands for the layer weights, and h_0 denotes the network input. Given the network output h_L and the training sample label l, a loss $L(l, h_L)$ is calculated.

While BP computes gradients for this loss function, TP calculates an output target and backpropagates it. The adapted output target $\hat{h}_L$ is defined as the output activation adjusted in the opposite direction of the gradient:

$$\hat{h}_L = h_L - \hat{\eta}\frac{\partial L}{\partial h_L}$$

Here, $\hat{\eta}$ denotes the output target step size. This modified target $\hat{h}_L$ is then backpropagated to obtain hidden layer targets $\hat{h}_i$:

$$\hat{h}i = g_i(\hat{h}i + 1) = t_i(Q_i \hat{h}_{i+1}), \quad i = L - 1, \ldots, 1$$

Here, g_i approximates the inverse of f_{i+1}, Q_i represents the feedback weights, and t_i signifies a smooth nonlinear activation function. Different parameterizations for g_i are also possible. Local layer losses $L_i(\hat{h}_i, h_i) = |\hat{h}_i - h_i|_2^2$ are defined using $\hat{h}_i$. The forward weights W_i are updated through gradient descent steps with respect to this local loss, assuming $\hat{h}_i$ remains constant. Finally, the training of feedback parameters Q_i, f_{i+1}, and g_i is considered as training a shallow auto-encoder pair. The feedback parameter Q_i can be trained using a gradient step based on a *local* reconstruction loss:

$$L^i_{\text{rec}} = \frac{1}{2}|g_i(f_{i+1}(\hat{h}_i)) - \hat{h}_i|_2^2$$

The original variant of DTP includes a linear correction, called *difference correction* [53], which subtracts the reconstruction error from the propagated targets: $h_i = g_i(\hat{h}_{i+1}) - g_i(h_{i+1}) - h_i$.

The developments and insights derived from methodologies such as TP, DTP, and related approaches contribute to a more nuanced comprehension of neural computation. Through a reevaluation of the learning paradigm and by finding inspiration in cognitive systems, a direction has emerged that leads toward the enhancement of learning frameworks with more refined principles. Nevertheless, while these methods address specific limitations of BP in developing biologically plausible learning algorithms, none has comprehensively resolved all aspects. As the effort to bridge the gap between artificial and biological intelligence persists, it would seem wise to maintain an awareness of the development of both domains of research.

Chapter 2

Reinforcement learning

2.1. Introduction

The field of reinforcement learning has a long history of effective methods in solving control problems and even simple maze-like environments which remain fairly simplistic [103, 93, 15]. The introduction of deep learning methods to the field of reinforcement learning has lead to significant improvements, notably the first Deep Reinforcement Learning algorithm to successfully reach superhuman performances on the set of Atari 2600 games [65]. This was mainly attributed to the powerful function approximation and representation learning properties of deep learning methods [5]. One could suggest that the ability of deep neural networks in finding compact low-dimensional representations (features) in high-dimensional data (*e.g.* images) has been the major contribution of deep learning to RL.

Learning agents are presented with states which contain information about the environment and their position therein. Agents can take actions to transition to another state which is accessible from the previous one, and receive a reward after transitioning. More formally, let S represent the set of states, A the set of actions and $R(s)$ the reward function returning a scalar for each $s \in R$. Additionally, we have the transition dynamics $\mathcal{T}(s_{t+1}|s_t,a_t)$ which maps to which state at time point t+1 the agent is sent from its current state after having interacted with the environment. We will review the main ideas in the field of reinforcement learning that will be used throughout this thesis.

2.2. Learning from returns

The modelling framework most used in reinforcement learning is the stochastic Markov Decision Process (MDP), which makes the simplifying assumption that the current input contains all the necessary information to select the next best action. The state itself contains information pertaining to the environment and the agent, and can be

represented as images or vectors which represent the necessary information to navigate the environment. A successful RL agent will associate which actions lead to the highest reward with certain states, and ideally finds the optimal sequence of actions such that the cumulative reward is maximized. We will refer the optimal sequence of actions as the optimal policy which maximises the expected return. Through trial and error, the agent's goal is to discover the optimal policy. The learning process often involves interacting with the environment and learning from one's mistakes, but the usefulness of the information gleaned depends on the exploration strategy employed by the agent as well as its ability to use the information to its advantage. Exploration is akin to searching for states in the environment that will reveal useful information when interacted with and is a challenging facet to designing efficient artificial intelligence in traditional RL.

The reward signal is often assumed to be sufficient to learn goals. In this case, the agent's goal is to maximize the total amount of reward it receives called the return. Therefore the agent aims to maximize the cumulative reward as opposed to the immediate reward. This has been informally referred to as the *reward hypothesis* [103]:

> «*That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)* »

The reward helps the agent identify which actions have the highest return from a given state. In other words, the agent learns a *value function* which can be used to select actions with higher returns. This function must be estimated continuously as experience helps to cement which sequence of actions are optimal, and is one of the most used quantities in RL.

2.2.1. Value and Policy Functions

As previously stated, the return is the sum of *discounted* rewards defined as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

For purposes of this work, the sum will be finite as the number of steps permitted in each attempt at solving the puzzles is limited. This gives us:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

The value function maps a state to a real number representing the expected future reward from that state. If an agent were to encounter this state many times, then the average return from that state would define its value. In other words it is an estimate of how good it is for

the agent to be in that state. More formally we can define the *state-value function for policy* π using the future rewards obtained from that state as follows:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s\right]$$

where E_π is the expected value obtained by the agent given it follows the policy π at time step t. The value function serves as a metric of how good a given state is, independent of the subsequent action taken from that state. When we consider the state-action value function $q_\pi(s,a)$, then we can begin to measure the value of taking a certain action at any state. This function is often referred to as the *action-value function for policy* π [103].

At each step in the environment, the model outputs a policy $\pi_t = \pi_\theta(\cdot|h_t)$ which considers the past observations $h_t = (x_0,...,x_t)$ to form a probability distribution over actions, and an estimate of that policy's value $v^\pi(h_t) = \mathbf{E}[G_t|h_t]$ as a real numeric value can be formulated from the value function (or directly, see policy iteration). Finally, the agent may or may not have access to or learns a *model* of the the environment. We have previously defined this as the transition dynamics $\mathcal{T}(s_{t+1}|s_t,a_t)$. The model is used to do *planning* by accessing the next states from the current optimal policy, and is either explicitly define for *model-based* agent or abstracted away in *model-free* agents.

Many algorithms exist to learn value or policy functions from rewards, such as dynamic programming, Monte Carlo methods, temporal-difference learning. We call these tabular solutions since they track a value or policy for each state and action combination. Alternatively, there are approximate methods which learn state representations and optimal policies simultaneously. We will review policy gradient solutions before introducing the actor-critic method which is the core algorithm presented in this work.

2.2.2. Policy gradient methods

Policy gradient methods are a class of reinforcement learning algorithms that attempt to directly optimize the policy function π_θ by updating the parameters θ in the direction of improving performance of the actor. The objective function for policy gradients is defined as:

$$J(\theta) = \mathbf{E}_\tau[\sum_{t=0}^{T-1} r_{t+1}]$$

where r_{t+1} is the reward at time $t+1$ returned for using action a_t at state s_t. Since we want to maximize this function using gradient ascent, we take the gradient and obtain the following update to the weights of the policy function (a neural network) θ:

$$\theta = \theta + \frac{d}{d\theta} J(\theta)$$

We derive the policy gradient update equation $\frac{d}{d\theta} J(\theta)$ by expanding the expectation:

$$J(\theta) = \mathbf{E}_\tau[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta] \tag{2.2.1}$$

$$= \sum_{t=0}^{T-1} P(s_t, a_t | \tau) r_{t+1} \tag{2.2.2}$$

where τ is some trajectory and $P(s_t, a_t|\tau)$ is the probability of state and action s_t,a_t occuring given the trajectory τ.

This is done by using gradient descent to maximize the expected return of the policy, $J(\theta) = \mathbf{E}[h_t] \sim \pi\theta[v^\pi(h_t)]$. At each step, the gradient of the expected return is computed using the policy gradient theorem, which states that the gradient of the expected return with respect to the policy's parameters is equal to the expected value of the gradient of the log of the policy with respect to the same parameters:

$$\nabla_\theta J(\theta) = \mathbf{E}[h_t] \sim \pi\theta[\nabla_\theta \log \pi_\theta(a_t|h_t) \cdot v^\pi(h_t)]$$

This expected gradient can be approximated using sample trajectories generated by the policy, and the policy's parameters can then be updated using the gradient and a suitable optimization algorithm, such as stochastic gradient descent. By repeating this process over multiple steps, the policy is gradually improved and converges to the optimal policy.

One advantage of policy gradient methods is that they can directly optimize the policy without the need for an explicit value function, which can be difficult to learn in some environments. Additionally, the use of gradient descent allows for the use of techniques such as backpropagation, which can make it easier to optimize complex policies with many parameters. However, policy gradient methods can also suffer from high variance in the estimates of the gradient, which can make convergence slow and unstable. To address this issue, many variations of the policy gradient algorithm have been proposed, including actor-critic methods and trust region methods.

2.3. Offline RL

One of the primary advantages of offline RL methods is their superior sample efficiency compared to online RL approaches. Online methods require active exploration in the environment to collect data, which can be time-consuming and inefficient. Agents often

need to perform numerous interactions before acquiring enough experience to learn effective policies. In contrast, offline RL methods utilize pre-collected datasets, enabling agents to learn from a large amount of offline data without the need for additional exploration. By efficiently reusing data, offline RL methods can achieve higher sample efficiency, reducing the number of interactions required to reach desirable performance levels.

2.3.1. Behavioral Cloning

Behavioral cloning is a technique used in machine learning, specifically in the field of imitation learning, to train an agent to imitate an expert's behavior simply by observing its actions [105]. In behavioral cloning, the goal is to learn a policy that can replicate the expert's actions given observed states. The common approach to behavioral cloning involves using target expert actions and minimizing a loss L that characterises a measure of difference between the predicted actions and the expert actions:

$$L = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{a}_i, \hat{\mathbf{a}}_i)^2$$

where f computes some distance metric. To train a behavioral cloning model, a dataset is collected by observing the expert's behavior. The dataset consists of state-action pairs, where each state is associated with the action taken by the expert. These state-action pairs serve as the training examples for the behavioral cloning model. In behavioral cloning, the expert's actions are considered as the target actions that the model aims to imitate. These target expert actions are used during training to guide the learning process. For each state in the dataset, the expert's corresponding action is provided as the target output for the model. To train the behavioral cloning model, a common choice of loss function is the L2 loss (also known as mean squared error). The L2 loss measures the average squared difference between the predicted actions and the target expert actions. The loss function encourages the model to produce actions that are as close as possible to the expert's actions.

During training, the behavioral cloning model takes the observed states as input and generates predicted actions. The loss is then computed between the predicted actions and the target expert actions. The model's parameters are adjusted through backpropagation and gradient descent to minimize this loss. A main limitation of this method is when the expert trajectories are suboptimal, passing the error on to the model that is learning from these observations.

Chapter 3

First article: Identifying Invariant and Sparse Predictors in High-dimensional Data

Abstract:

In many practical applications (e.g., medical imaging), we are often concerned not only with the accuracy but also with the interpretability of a predictive model trained on high-dimensional data, such as its ability to identify a sparse subset of invariant (robust) predictors, generalizing across a wide variety of datasets affected by spurious noise (e.g., key factors related to a disease, across different patients and hospitals). Towards this goal, we explore here a combination of the sparsity-inducing l_1 and l_2 regularization with the recently proposed approach for robust predictive modeling, Invariant Learning Consistency. We investigate the ability of the combined approach to identify robust sparse predictors and demonstrate promising results on several datasets, including synthetic data, the MNIST benchmark, and a functional MRI dataset. Our approach tends to improve the robustness of sparse models in practically all cases, albeit with varying degrees of success and under certain conditions.

Main contributions:

The main contributions of Sean Spinney for this article are the following:

- ideation and conceptualization of the project
- processing and quality control of the fMRI data
- programming of the proposed approach
- running and logging of the experiments
- analysis and writing of the article

The work was split among the first three authors, except for the fMRI data preprocessing and quality control. Co-authors (in order of appearance) are: Amin Mansouri, Sean Spinney, Amin Memarian, Patricia Conrod, and Irina Rish. This article was submitted and accepted to ICML 2021 workshop for Uncertainty & Robustness in Deep Learning.

3.1. Introduction

In recent years, there has been a surge of interest in the deep learning community towards better understanding which aspects of deep network models allow them to generalize better to test data drawn from distributions different from the one on which these models were trained - the problem commonly referred to as out-of-distribution (OoD) generalization. A specific focus is on methods that aim to disentangle the true *causal* predictors from the possibly many spuriously correlated ones, often present in high-dimensional data [3, 67, 70, 1, 4].

In this work, we build upon the recently proposed *Invariant Learning Consistency (ILC)* approach [77] that imposes higher levels of consistency (stability) on the parameters of a neural network (or other predictive models), and is shown to improve out-of-distribution generalization, sometimes outperforming the popular IRM approach [4] for invariant/robust representation learning,

However, none of the previous approaches to robust predictive modeling seemed to focus directly on the consistency, or invariance, of the learned network structure, which is the novel contribution of this approach. Herein, we apply ILC to the problem of learning robust sparse predictive models which are *structurally consistent* across a range of data distributions, in terms of high overlap across the (sparse) network structure. More specifically, we combine ILC with sparsity-inducing regularization methods based on l_1-norm (e.g., Lasso [104] and Elastic Net [117])).

As it is well-known, l_1 regularization promotes learning *sparse* solutions, i.e., weight vectors with a relatively small number of non-zero entries, which translates into selecting a sparse subnetwork of an original (dense) neural network model. Consequently, a sparse set of links at a given network layer connects to a subset of features that tend to be "most relevant" (most predictive) about the nodes in the next layer. Note that a neural network can be viewed as a *deterministic* Bayes net, and imposing sparsity via an l_1 regularizer, we can find "parents" of a given node in this graphical model; however, in general, we cannot yet view this network as a *causal graph*, since l_1 regularization will not necessarily return the *same* sparse representation for each round of training. In other words, imposing sparsity alone is unlikely to yield a true causal model; rather, we get a different set of sparse selectors each time. This is the advantage of ILC: by forcing agreement between gradient updates, we suspect it will result in *consistent and causal sparse representations*.

3.2. Methodology

3.2.1. Invariant Learning Consistency

Let $\{\mathcal{D}^e\}_{e\in\mathcal{E}}$, where $|\mathcal{E}| = d$ and $\mathcal{D}^e = \{(x_i^e, y_i^e) | \, i = 1, \ldots, n_e\}$, be a collection of datasets sampled from different data distributions, or *environments* $e \in \mathcal{E}$. The empirical loss (risk) of

a model with parameters θ is defined for each environment e as follows:

$$\mathcal{L}_e(\theta) := \frac{1}{|\mathcal{D}^e|} \sum_{(x_i^e, y_i^e) \in \mathcal{D}^e} \ell(f(x_i^e; \theta), y_i^e); \tag{3.2.1}$$

Using the new loss with respect to the environment definition, we can define the new ILC-regularized loss function proposed in [77]:

$$\mathcal{L}^{\text{ILC}}(\theta) := \mathbb{E}_{\theta \sim p(\theta)} \left[\frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \mathcal{L}_e(\theta) \right] - \lambda \cdot \text{ILC}(\theta) \tag{3.2.2}$$

Here the regularization factor ILC(θ) reflects the consistency score of a given gradient across all environments. This means that if the neighborhood of the minimum is not consistent across environments, it is probably not a consistent minimum, and it does not relate to an invariant mechanism. Notice that λ adjusts the desire of predictive power (ERM [107]) and invariance (ILC) if $\lambda = 0$ we get back to the classic gradient descent (gradients are averaged and agreement among them will not be considered.) and $\lambda > 0$ means we care about finding invariant mechanisms.

AND-Mask. On top of the new ILC-regularized loss, the paper introduces the *AND-mask*. The regularization term needs not be explicitly added to the loss function. We can update the model by evaluating if the gradient directions (sign) are consistent across all environments and take the optimization step in those directions. If a component of the gradient has a majority of the same sign above a certain threshold $\tau \in [0,1]$ the component is left as is, if not, then the component is zeroed-out. Mathematically, the mask m_τ for a given component j is given by:

$$[m_\tau]_j = 1 \left[\tau d \leq \left| \sum_e \text{sign}([\nabla \mathcal{L}_e]_j) \right| \right] \tag{3.2.3}$$

where $d = |\mathcal{E}|$ is the number of environments in the batch. Finally we have the final definition of the masked-ILC-regularized gradient:

$$\nabla_\theta \mathcal{L}^{\text{m-ILC}}(\theta) = m_\tau \odot \nabla_\theta \mathcal{L}(\theta) \tag{3.2.4}$$

Note that the notion of *environments* is very general and could be interpreted differently in various settings. In [77] they treat every single sample as its own environment, and we follow the same convention.

3.2.2. Evaluation

In order to assess the quality of the improvement in recovering consistent estimators, we use the following metrics and motivate their relevance in evaluating the ILC algorithm: test accuracy and an overlap score for the MNIST experiment, which measures the consistency of learned sparse representations across environments. Therefore, a high overlap score means a greater agreement between which features of the data are useful for prediction across

environments. Together we can compare the predictive power (test accuracy) with robustness (overlap score).

3.3. Experiments

3.3.1. Synthetic Dataset

Using a procedure inspired by the original paper [77], we simulate a binary classification dataset with weak invariant predictors across samples (environments). Let $X \in \mathbf{R}^{N \times D}$ and $y \in \{0,1\}^N$ be a Bernoulli random variable, where N is the number of samples and D the number of predictors. We assume the data is generated according to the following:

$$\epsilon_n \sim N(0, \sigma^2), \forall n = 1,...,N$$

$$y_n = \begin{cases} 1 & \beta_R X_{n,\Gamma_R} + \beta_S X_{n,\Gamma_S} + \beta_I X_{n,\Gamma_I} + \epsilon_n > 0 \\ 0 & 0 \end{cases}$$

where we consider that $\beta = \{\beta_R; \beta_S; \beta_I : \beta_S \gg \beta_I\}$ are the coefficients defining the relationship between X and y. Γ_R refers to completely random valued columns (features), Γ_S refers to columns that contain spurious features but are *strong* (have high values), and Γ_I refers to columns that are invariant, *but* are weak (have lower values). β_R,β_S,β_I are the coefficients determining the values for each of these column sets. That is, for every n we have that $\beta_{I,n} \neq 0$ reflecting the fact that the invariant features of X_n have non-zero coefficients across environments by definition. The set $R = \{k \in D; k \notin \{\Gamma_S, \Gamma_I\}\}$ represents the set of all the random predictors in X (no association to y; *i.e.* β_R is random noise).

Note that here $\Gamma_R := \{1,2,3,4\}$ and $\Gamma_I := \{5\}$, and there is no added noise to X in (1). The error is drawn from a normal distribution for simplicity (*i.e.* probit). An example of such a dataset is given in the appendix.

3.3.2. MNIST

We divide the training set of MNIST into two environments and keep the test set for evaluating the predictive power of learned sparse representations from training environments. For this experiment, the environments are created by splitting the training set into two evenly shuffled subsets with images and labels corresponding to the 10 digit classification problem. Then we train a *sparse multiclass logistic regression* on the two environments on a range of values for the tuple ($l1$,$l2$,τ), where $l1$,$l2$ denote the coefficient for $l1$,$l2$ regularization terms, and τ denotes the agreement threshold among gradients (see section 3.2.1). According to the AND-mask, this model is typically referred to as an Elastic Net regularized network with the added ILC mechanism of computing gradients. Evaluation metrics were presented in 3.2.2. Overlap score is calculated by normalizing the number of weights that are non-zero and have

the same index (after rounding to 0.001) in the two sparse representations (obtained from each split) by the number of non-zero weights.

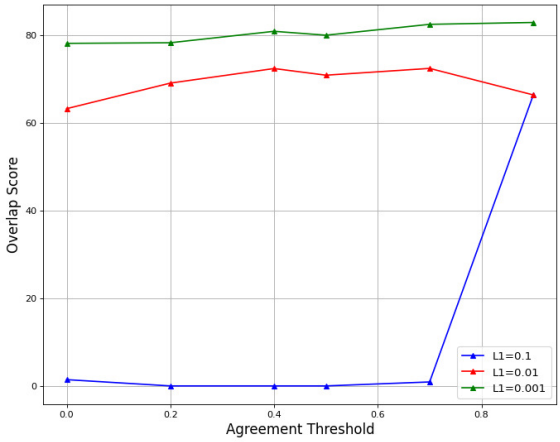


Figure 3.1. ILC is applied after regularization. Each color corresponds to a value of l1 coefficient and fixed l2, and shows the behavior of sparse representation's consistency (overlap score) averaged over all digits.

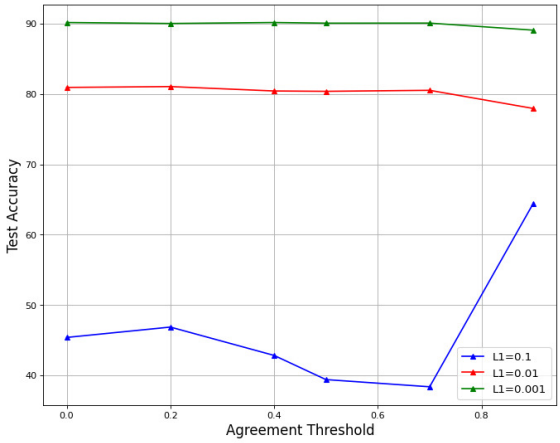


Figure 3.2. ILC is applied after regularization. Each color shows the trend of test accuracy for varying $l1$ values, with fixed $l2$, over increasing τ.

3.3.3. fMRI

The dataset comprises 38,700 3D MRI brain images, each of dimension 53x63x52, taken during a task where subjects must either go or stop following a stimulus that is viewed on a screen directly in front of them. Each subject is scanned three times over five years.

This experiment aims to train a model that can predict the event, which is a combination of what the subject sees and the action taken while being invariant to the inherent noise involved in this high dimensional data (53x63x52 = 173,628 variables). We propose to use a fully connected single hidden layer network with both $l1$ and $l2$ regularization such that the recovered sparse model represents the causal graph of brain activation for this 5-way classification problem (Table 3.3). We compare this model with the same setup but with ILC applied to enforce invariance across different sampled scans. Data from subjects and the first two time points are shuffled together, and we test generalization by evaluating the model on the third timepoint from the dataset.

3.4. Results

3.4.1. Synthetic

For the sake of brevity, we review the main results and mention interesting auxiliary findings observed through experimentation. First, we note that the agreement threshold has a significant impact on the test set performance of ILC, Figure 3.3. There is considerable improvement in out-of-distribution accuracy when using ILC (agreement $\in$ 0.2,0.4), and we note the sharp rate of increase in test accuracy in the first epochs following the onset of ILC.

Setting the agreement threshold to 0.4, we run additional experiments (Appendix A.2) to compare performance under different experimental setups (high-dimensionality and a large number of environments, high-dimensionality, and a small number of environments). When ILC outperforms traditional regularized Adam, the number of environments is greater than or equal to the dimensionality of the feature space. For the details of hyperparameters refer to appendix 3.1.

3.4.2. MNIST

Results for MNIST using sparse multiclass logistic regression are shown in Figures 3.1-3.2. Table 3.2 in the appendix contains the hyperparameters that we have trained the environments on. It should be noted that the lack of distributional shift in MNIST is suboptimal for using invariant representations (see appendix fig. 3.6 for the measurement of distribution shift.). Thus OoD performance in this setting is not very meaningful, and this experiment only demonstrates the effect of ILC on robustness, not OoD generalization. With that in mind, the findings are as follows:

In Figure 3.1,3.2 for large l1 coefficient and increasing agreement threshold, we see that more consistent sparse representations have been achieved at the expense of accuracy, i.e., very sparse representations yet with *good* predictive power on the test set. This suggests that a larger agreement threshold is beneficial in improving the robustness of the learned model

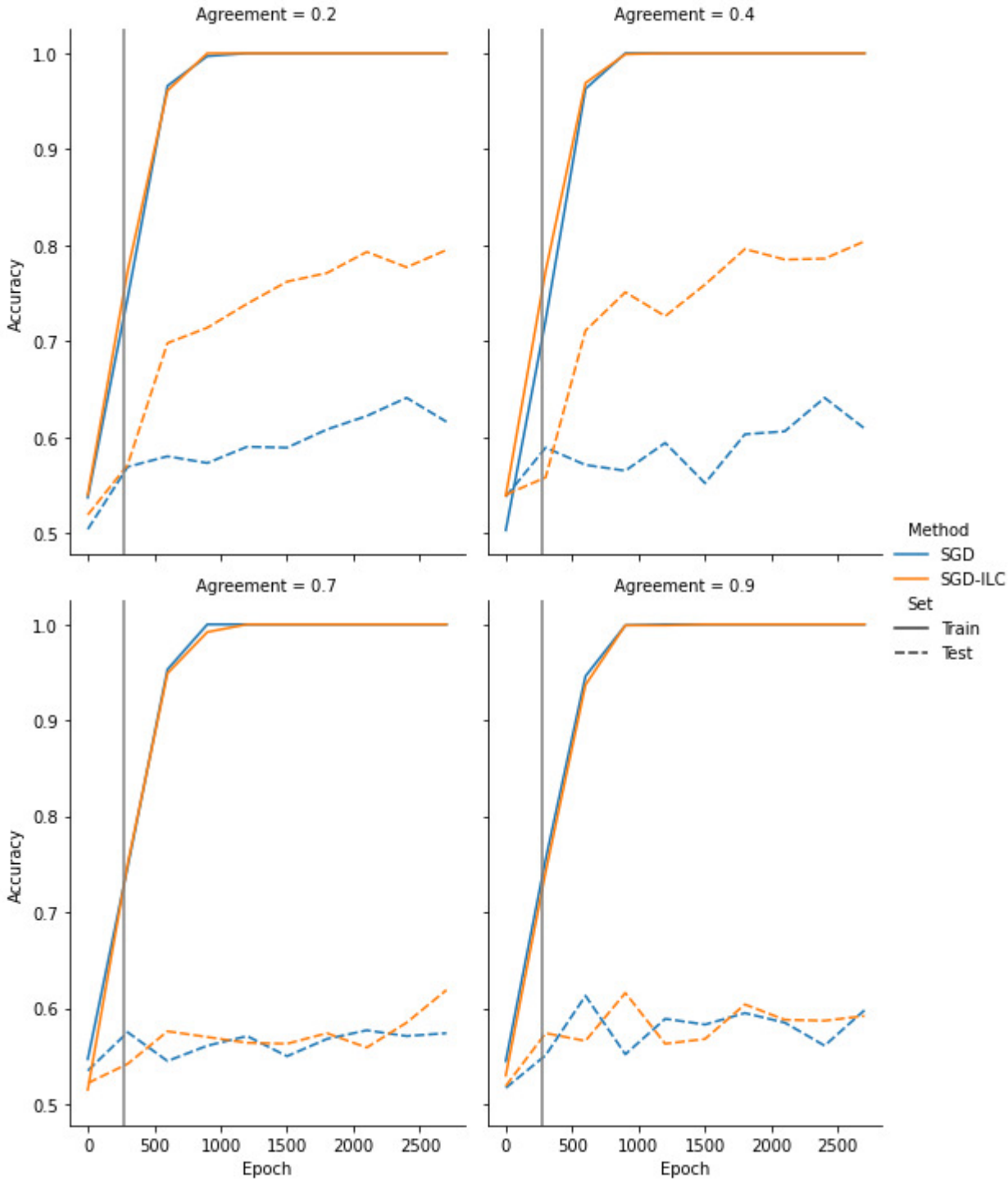


Figure 3.3. Grey vertical line indicates when ILC is turned on.

to varying strengths of the $l1$ regularization. This is especially noticeable for large values of $l1 > 0.1$. The correlation between a high overlap score (Figure 3.1) and a greater test accuracy (Figure 3.2) for larger agreement threshold ($\tau > 0.6$) suggests that we have achieved

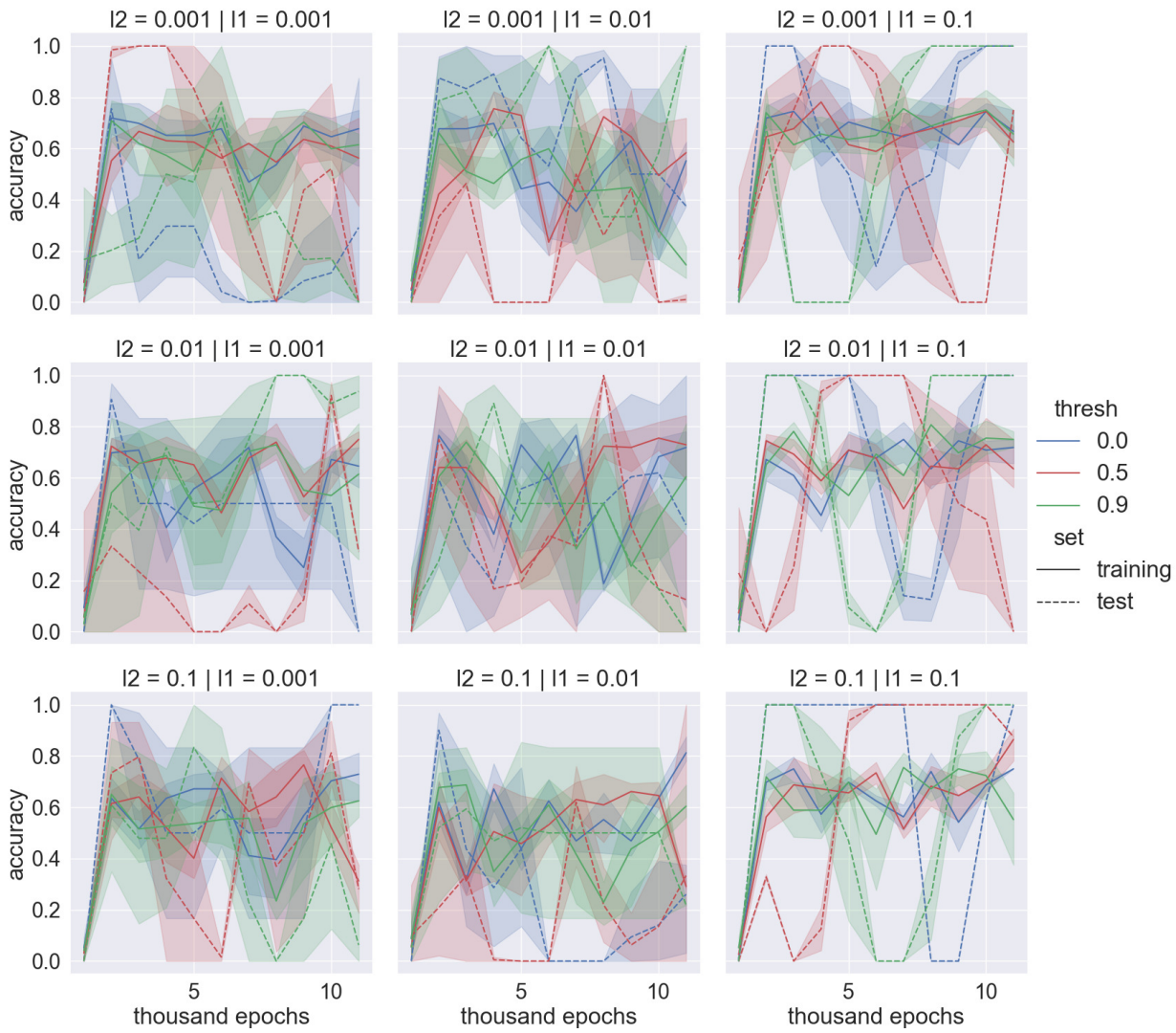


Figure 3.4. ILC was turned on halfway through training. Note that an agreement threshold of 0 means only Adam optimization is used.

sparse and invariant predictors in the case of MNIST that has a very small distribution shift between both environments. Thus, we suspect there are improvements to be made in cases where meaningful distribution shifts occur *e.g.* Coloured MNIST.

Late start of ILC: We observed that applying this kind of gradient alignment (AND-masking) should not be from the very first epochs. The reason is that masking all gradients in the initial stages where no meaningful features have been found would result in no learning, so we start masking gradients close to the middle of the training.

3.4.3. fMRI

The results for the fMRI experiment can be found in Figure 3.4. There is no obvious set of hyperparameters that achieves good classification scores across the five classes, and we

note that the volatile nature of the accuracy is due to the imbalanced distribution of classes across batches, which was not controlled for in this case (see Table 3.3). Considering that we find poor results even with just ElasticNet without ILC (thresh=0), we should not expect ILC to perform better since we cannot enforce invariance without any useful learned features. The complexity of brain fMRI data is very high (the number of voxels is on the order of 100000), and we have approached this challenging problem with only a single fully connected layer network (with the motivation of finding a deterministic causal Bayes net). These results do not rule out the possibility for ILC in improving the recovered sparse predictors for brain activity, but exploring more complex architectures may yield better results and is the focus of future work.

3.5. Conclusion

In this work, we extended the original ILC experiments to a more extensive set of synthetic experiments, outlining failure and success modes. Beyond this, we applied the method to the MNIST dataset and a much more difficult task of predicting events using voxel activations for an fMRI task. More specifically, we showed that for the case of MNIST, ILC had a meaningful impact in finding sparse predictors that also contribute to better OoD performance. The synthetic experiments showed that when the number of environments is large compared to a sparse and noisy feature space, ILC outperforms ElasticNet. Finally, we found that ILC did not perform significantly better for fMRI by a noticeable margin; however, several considerations may improve these results. For example, given the insights gleaned from the synthetic dataset, using more environments may benefit ILC especially given the very high dimensionality involved in fMRI. Additionally, there may be more optimal configurations of environments for ILC, such as considering the hierarchical nature of the data (*i.e.* subjects within years). As such, we conclude that ILC should not be used blindly with the expectation of improving OoD performance and sparse variable selection for all datasets, but these experiments suggest an important relationship between the number of environments and the dimensionality of the feature space, as well as the strength of $l1$ regularization.

3.6. Appendix

3.7. Synthetic task

3.7.1. Example of synthetic dataset

Below is an example of X where $N = 4$, $D = 5$:

$$X_{4,5} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}; y_4 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.7.1}$$

3.7.2. Hyperparameters for training sparse logistic regression on the synthetic dataset

	Elastic Net	Elastic Net + ILC
log L1 Regularization	1e-4	1e-4
log L2 Regularization	1e-4	1e-4
Agreement Threshold	0	[0.2,0.4,0.7,0.9]

Table 3.1. Hyperparameters used for training. Optimizer in all cases is Adam, and parameters for optimizers are default: lr=0.001, b1=0.9, b2=0.999. For each experiment, the number of true causal factors was set to 5.

3.7.3. Additional results:

Effect of late starting ILC 3.5.

3.8. MNIST task

3.8.1. Hyperparameters for training sparse logistic regression on the two environments on MNIST

	Elastic Net	Elastic Net + ILC
log L1 Regularization	[-1,-2,-3]	[-1,-2,-3]
log L2 Regularization	[-3,-4,-5]	[-3,-4,-5]
Agreement Threshold	0	[0.2,0.4,0.5,0.7,0.9]

Table 3.2. Hyperparameters used for training, overall there are 54 settings. Optimizer in all cases is Adam, and parameters for optimizers are default: lr=0.001, b1=0.9, b2=0.999, see below for samples of the sparse representations learned.

3.8.2. Sparse representations achieved for each digit in several sets of hyperparameters

Each row corresponds to an environment or MNIST train split.

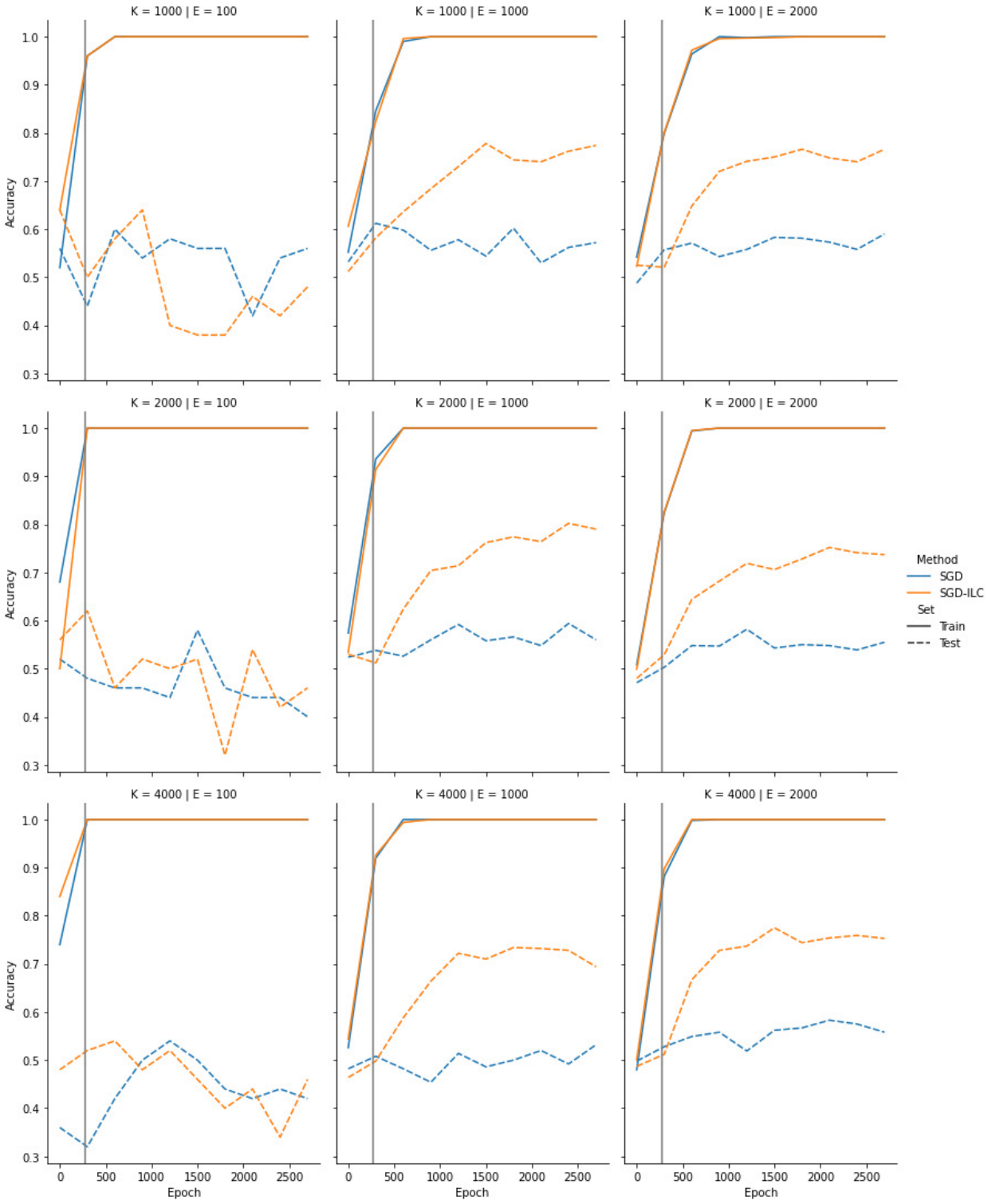
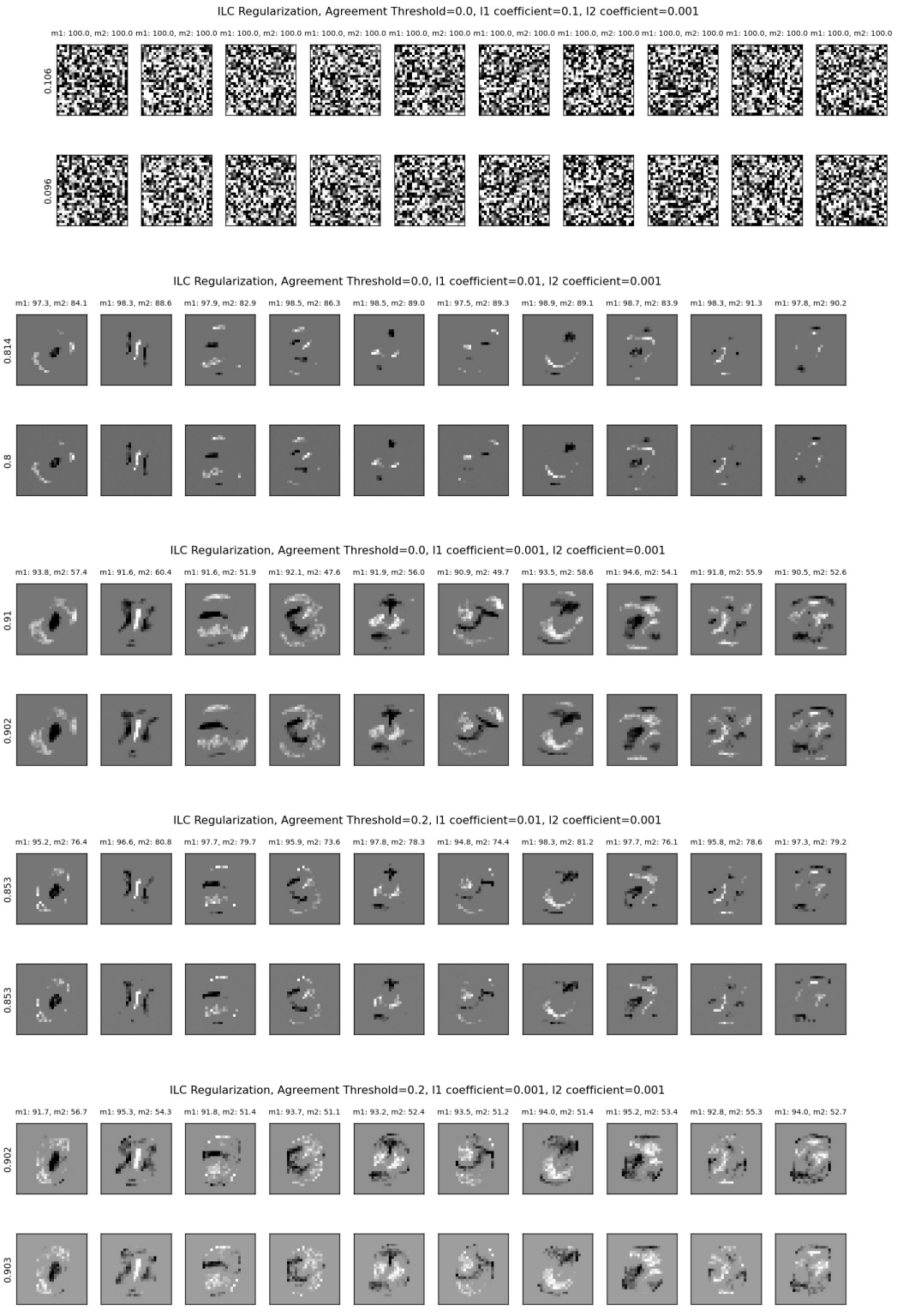


Figure 3.5. Grey vertical line indicates when ILC is turned on. ILC performs best when the number of predictors is close to or greater than the number of environments.


ILC Regularization, Agreement Threshold=0.0, l1 coefficient=0.1, l2 coefficient=0.001
m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0 m1: 100.0, m2: 100.0
0.106
0.096
ILC Regularization, Agreement Threshold=0.0, l1 coefficient=0.01, l2 coefficient=0.001
m1: 97.3, m2: 84.1 m1: 98.3, m2: 88.6 m1: 97.9, m2: 82.9 m1: 98.5, m2: 86.3 m1: 98.5, m2: 89.0 m1: 97.5, m2: 89.3 m1: 98.9, m2: 89.1 m1: 98.7, m2: 83.9 m1: 98.3, m2: 91.3 m1: 97.8, m2: 90.2
0.814
0.8
ILC Regularization, Agreement Threshold=0.0, l1 coefficient=0.001, l2 coefficient=0.001
m1: 93.8, m2: 57.4 m1: 91.6, m2: 60.4 m1: 91.6, m2: 51.9 m1: 92.1, m2: 47.6 m1: 91.9, m2: 56.0 m1: 90.9, m2: 49.7 m1: 93.5, m2: 58.6 m1: 94.6, m2: 54.1 m1: 91.8, m2: 55.9 m1: 90.5, m2: 52.6
0.91
0.902
ILC Regularization, Agreement Threshold=0.2, l1 coefficient=0.01, l2 coefficient=0.001
m1: 95.2, m2: 76.4 m1: 96.6, m2: 80.8 m1: 97.7, m2: 79.7 m1: 95.9, m2: 73.6 m1: 97.8, m2: 78.3 m1: 94.8, m2: 74.4 m1: 98.3, m2: 81.2 m1: 97.7, m2: 76.1 m1: 95.8, m2: 78.6 m1: 97.3, m2: 79.2
0.853
0.853
ILC Regularization, Agreement Threshold=0.2, l1 coefficient=0.001, l2 coefficient=0.001
m1: 91.7, m2: 56.7 m1: 95.3, m2: 54.3 m1: 91.8, m2: 51.4 m1: 93.7, m2: 51.1 m1: 93.2, m2: 52.4 m1: 93.5, m2: 51.2 m1: 94.0, m2: 51.4 m1: 95.2, m2: 53.4 m1: 92.8, m2: 55.3 m1: 94.0, m2: 52.7
0.902
0.903

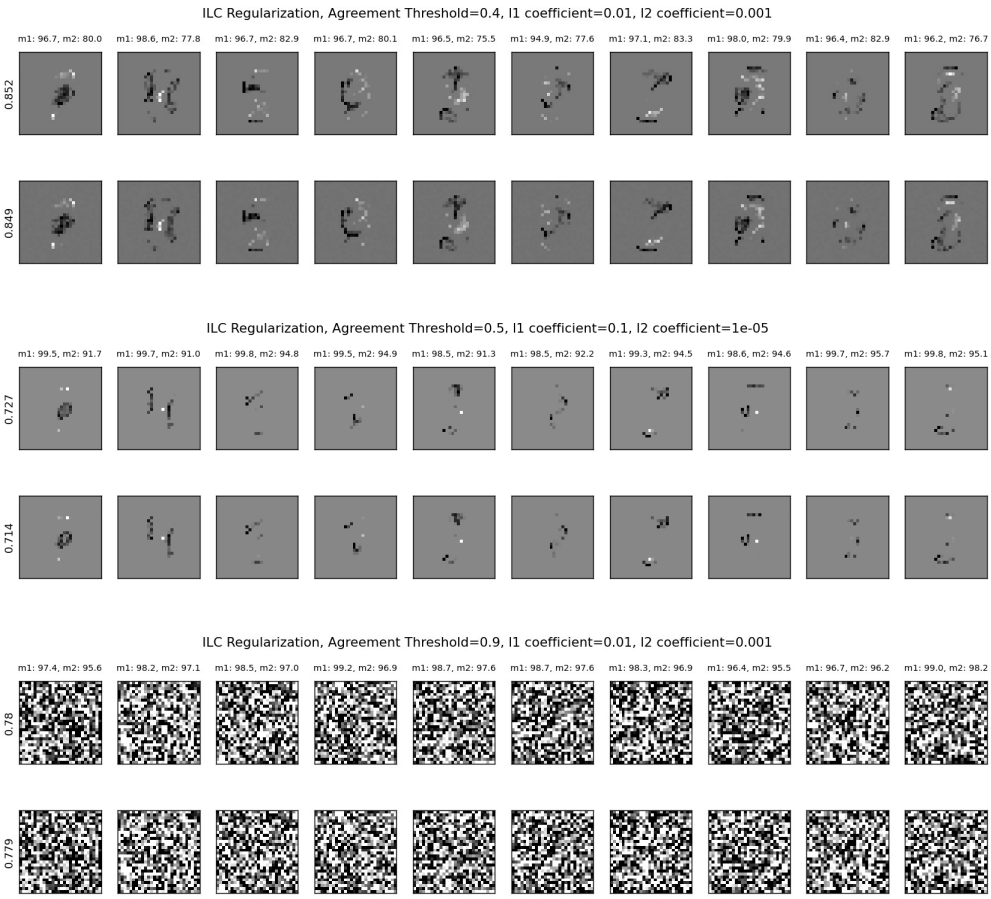


Figure 3.6 shows that the entropy of the predicted class on the test set is higher than the training set in each environment. However, there is no significant distribution shift across environments.

3.9. fMRI task

The task analyzed in this work is the stop-signal task, and we refer the reader to previous work, which details the same methodology used for the dataset (Li et al., 2006). The scans were gathered from three different time points for each subject. There are five distinct events used for the classification task: go-success, go-toolate, go-wrong, stop-failure, and stop-success. The data were normalized using z-normalization. The data of subjects from the first two time points were concatenated and shuffled for the training set. The data of the last time point was used for the test set. Each data sample was considered an environment in the experiments presented in this work. Other curations of environments could yield more meaningful results, e.g., considering data from each time point as an environment.

3.9.1. Distribution of classes

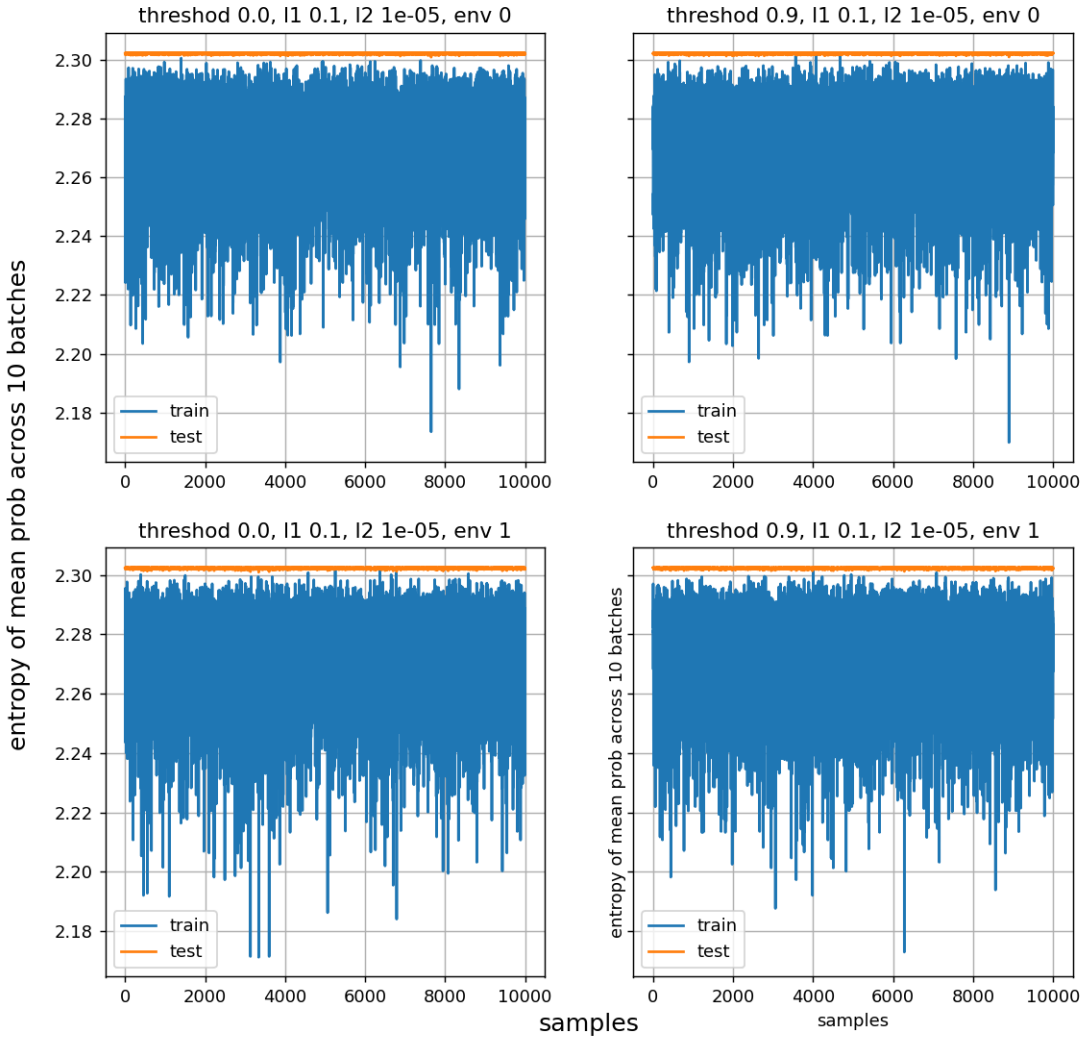


Figure 3.6. Entropy of of the predicted class for each sample in the test and train batches for two environments after 100 epochs with ILC turned on after epoch 50. The probabilities used for computation were averaged over all batches of size 10000.

	Training Set	Test Set
go-success	4619	2559
go-toolate	293	68
go-wrong	363	73
stop-failure	851	447
stop-success	864	453

Table 3.3. The number of samples used in training and test set belonging to each class. As we can see the classes are imbalanced.

Chapter 4

Second article: Towards Scaling Difference Target Propagation by Learning Backprop Targets

Abstract:

The development of biologically-plausible learning algorithms is important for understanding learning in the brain, but most of them fail to scale-up to real-world tasks, limiting their potential as explanations for learning by real brains. As such, it is important to explore learning algorithms that come with strong theoretical guarantees and can match the performance of backpropagation (BP) on complex tasks. One such algorithm is Difference Target Propagation (DTP), a biologically-plausible learning algorithm whose close relation with Gauss-Newton (GN) optimization has been recently established. However, the conditions under which this connection rigorously holds preclude layer-wise training of the feedback pathway synaptic weights (which is more biologically plausible). Moreover, good alignment between DTP weight updates and loss gradients is only loosely guaranteed and under very specific conditions for the architecture being trained. In this paper, we propose a novel feedback weight training scheme that ensures both that DTP approximates BP and that layer-wise feedback weight training can be restored without sacrificing any theoretical guarantees. This theory is corroborated by experimental results and we report the best performance ever achieved by DTP on CIFAR-10 and ImageNet 32×32.

Main contributions: The contributions of Sean Spinney to this project were:

- programming of the baseline alternative models used as a benchmark for the proposed algorithm (s-DDTP, p-DDTP, and [63])
- running the experiments for these models and logging the results, hyperparameters, and architecture details for MNIST, F-MNIST, and CIFAR10 experiments
- creating the figures for these experiments, with the guidance of Maxence Ernoult

Co-authors (in order of appearance) are: Maxence Ernoult, Fabrice Normandin, Abhinav Moudgil, Sean Spinney, Eugene Belilovsky, Irina Rish, Blake Richards, and Yoshua Bengio.

4.1. Introduction

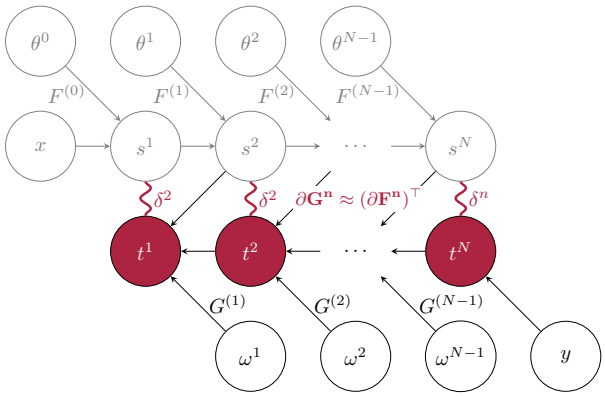


Figure 4.1. Computational graph of the feedforward pathway $\mathcal{F}$ (on the left, shaded) with input x and associated DTP feedback pathway with ground-truth label y (right). The *targets* t^n (purple nodes) are forward-propagated through the G^{n+1} operator whose Jacobian has been made to approximately match that of the transpose of F^n. This way, the resulting local activation differences $\delta^n \propto t^n - s^n$ encode backprop error signals. We thus learn to estimate layer-wise *backprop* targets.

Although artificial neural networks were originally inspired by the brain, the strict implementation of the backpropagation algorithm (BP) violates biological constraints, and no known biologically plausible candidate algorithm can match its performance on challenging tasks. Conversely, bridging this gap could bring a better understanding of biological learning [88]. Recent efforts towards this goal suggest that it could be achieved by developing learning algorithms that relax the requirements of BP while preserving strong theoretical guarantees.

Target Propagation (TP) [50] and its *Difference Target Propagation* (DTP) variants [12, 53, 10, 75, 11, 63] constitute a family of such algorithms which, from the biological prospective, sidesteps two issues of BP. Most importantly, TP computes error signals in feedforward architectures by propagating *target values* for the neurons rather than error gradients, thereby aligning better with the current understanding of what feedback pathways in the brain communicate [57]. A major consequence of handling neural activation targets across all layers is that feedforward weights can be updated in a fully local fashion to push neural activations closer to their target values. Second, TP routes those targets through a distinct set of feedback weights rather than transporting the weights from the feedforward pathway [59]. Rather than being fixed throughout learning, though, these feedback weights learn to *invert* the feedforward pathway. But what would it take to learn to

backprop the feedforward pathway? This is the central question addressed by the present work.

Nevertheless, TP algorithms have yet to scale to complex tasks, and as such, they do not yet stand as a compelling biological learning model. Recent work highlighting the connection between TP and Gauss-Newton (GN) optimization [11, 63] has incentivized to revisit the scalability of TP algorithms [10]. More precisely, meulemans2020theoretical demonstrate that while TP neural activations updates emulates GN optimization in invertible neural networks, this connection can be maintained with DTP on non-invertible networks if the feedback weights training scheme is changed accordingly. Indeed, to emulate GN optimization, as the pseudo-inverse of the whole feedforward pathway does not factorize as the product of each feedforward module's pseudo-inverse, each feedback module should capture the pseudo-inverse of the *whole* downstream feedforward pathway: when computing the resulting Difference Reconstruction Loss (DRL), noisy perturbations subsequently need to be propagated all the way up to the output layer. However, their approach still has limitations from a biological learning perspective. First, enforcing GN optimization in DTP like this precludes *layer-wise* feedback weights training and instead calls for the use of direct connections in the feedback pathway: this topological restriction seriously compromises biological plausibility. Second, the resulting optimization algorithm used to update the feedforward weights is a hybrid between between gradient descent and GN optimization. Therefore, only loose alignment between backprop and DTP updates can be accounted for by their theory and with restrictive assumptions on the architecture being trained. Finally, although that theory offers a principled way to design the architectures trained by these variants of DTP, the CIFAR-10 training experiments they report are limited to relatively shallow architectures with poor performance.

In this paper, we propose to revisit the GN interpretation of DTP by having the feedback pathway synaptic weights compute layer-wise *BP targets* rather than GN targets. To this end, we propose a novel feedback weights training scheme which, by construction, pushes the Jacobian of the feedback operator towards the transpose of its feedforward counterpart, in a layer-wise fashion and without having to use direct feedback connections in the feedback pathway. Therefore, assuming this condition holds for all the layers and keeping everything else unchanged in the DTP algorithm, the DTP feedforward weight updates closely approach those of BP. This leads us to a scalable biologically plausible approximation of BP.

More specifically, the contributions of this work are as follows:

- We propose a novel *Local Difference Reconstruction Loss (L-DRL)* along with an algorithm to train the feedback weights which ensures that the Jacobian of the

feedback pathway matches the transpose of the Jacobian of the associated feedforward pathway (Section 4.4.1, Theorem 4.4.2, Alg. 3). We call this condition the *Jacobian Matching Condition (JMC)* (Definition 4.4.1).

- Assuming the JMC holds for a given architecture and using the standard DTP equations to propagate targets, we demonstrate that DTP feedforward weight updates approximate BP gradients (Section 4.4.2, Theorem 4.4.3). We say that such an architecture satisfies the *Gradient Matching Property (GMP)*.
- We numerically demonstrate the GMP and JMC, showing that L-DRL is more efficient than DRL to align feedforward and feedback weights and that the GMP is subsequently significantly better satisfied (Section 4.5.1-4.5.2).
- Finally, we validate this novel implementation of DTP on training experiments on MNIST, Fashion MNIST, CIFAR-10 and ImageNet 32×32 [74] (Section 4.5.3). In particular, we achieve a 89.38 % accuracy on CIFAR-10 and 60.6 % top-5 accuracy on ImageNet 32×32, which are the best performances ever reported in the DTP literature on these datasets and nearly match the performance of BP on the same architectures.

4.2. Related Work

DTP borrows several key concepts from the biologically plausible deep learning literature. First, resorting to a distinct set of weights to route error signals in the feedback pathway as done in DTP solves a problem known as *weight transport* [59]. While having randomly initialized *fixed* feedback weights is sufficient to carry useful error signals on MNIST [59, 71], subsequent studies demonstrated it was insufficient to scale to harder tasks [66, 10, 47, 20]. The main approach undertaken to overcome this issue is to add extra mechanisms to promote alignment between feedforward and feedback weights [113, 46, 29, 2, 44]. More specifically, many of these mechanisms are based on the idea of perturbing the feedforward activations with noise, and communicating the resulting noisy activations in the feedback pathway to coordinate feedback and feedforward weight updates consistently [2, 46, 44], which constitutes a second important feature of DTP. However, a limitation of many of these algorithms is that they require gradient computation of the operations carried out in the feedforward pathway. One solution to mitigate this issue, which is the third important ingredient of DTP, is to propagate neural activation differences as implicit error signals rather than error gradients [57]. These error signals may typically arise from a mismatch between feedforward (bottom-up) predictions and (top-down) actual feedback [112, 92, 19], as it is the case for DTP, or from a perturbation from equilibrium [95]. Recent works have explored the application of DTP to recurrent neural networks [61, 89], albeit with the implausible requirement of processing inputs backward in time during target computation, a challenge that we do not aim to address in the present paper. As emphasized in the introduction, the

closest work to ours is that of [63], and we show the theoretical and experimental advantages of the proposed approach.

4.3. Background

We first introduce the key notations and assumptions used throughout this paper.

Definition 4.3.1. We define a *feedforward* architecture as:

$$\mathcal{F}(x) = F^{N-1} \circ F^{N-1} \circ \cdots \circ F^0(x), \tag{4.3.1}$$

where each feedforward module $F^n(\cdot;\theta^n)$ is parametrized by its feedforward weights θ^n. Each F^n is paired with a feedback module $G^n(\cdot;\omega^n)$ with distinct weights ω^n.

Definition 4.3.2. We recursively define the *layers* $s^1, \ldots, s^N$ of an architecture $\mathcal{F}$ defined by Definition 4.3.1 as:

$$\begin{cases} s^0 & = x \\ s^{n+1} & = F^n(s^n;\theta^n) \quad \forall n = 0 \cdots N-1 \end{cases} \tag{4.3.2}$$

G^n can either take as input the feedforward path activations $s^{n+1} = F^n(s^n)$ (with G^n then forming the decoder part of a kind of auto-encoder with F^n as encoder) or the backward path targets t^{n+1} produced by G^{n+1} from t^{n+2} and representing targets for s^{n+1}.

Learning setting. We study the supervised context where, given a target y, the goal is to find the forward weights θ^n which minimize a predictive loss $\mathcal{L}_{\text{pred}}(s^N, y)$.

Notations. We denote the Frobenius dot product between two matrices A and B as $\langle A, B\rangle_F \triangleq \text{Tr}\left(A \cdot B^\top\right)$. Also, we denote $\partial_x F(x_\star) = \frac{\partial F}{\partial x}(x_\star)$ the Jacobian of F with respect to x evaluated at $x_\star$. For notational simplicity, we may omit to write $x_\star$, in which case the Jacobians are implicitly evaluated on the feedforward activations.

4.3.1. Difference Target Propagation (DTP)

Instead of transporting the transpose Jacobian of the feedforward operators $\partial_{s^n} F^{n^\top}$ to the feedback pathway, TP and variants use a separate set of parameters through the feedback operator G^n to carry targets across layers. The G^n operators are subsequently trained, layer-wise, to approximately invert their associated feedforward counterpart: $G^n \approx (F^n)^{-1}$. TP learning thus entangles feedforward *and* feedback weights training.

Forward weights training. Target values for the neurons should be such that they decrease the predictive loss $\mathcal{L}_{\text{pred}}$. For most TP algorithms, the first target is computed as:

$$t^N_\beta = s^N - \beta \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{4.3.3}$$

where β is a small nudging parameter. In TP, the subsequent upstream targets are *propagated* the feedback operators as $t_\beta^n = G^n(t_\beta^{n+1}; \omega^n)$. However in non-invertible feedforward networks, this results in a significant reconstruction error $s^n - G^n(s^n; \omega^n)$, which was shown to compromise learning. *Difference* Target Propagation [53] aims to solve this issue by removing this reconstruction term from the target computation:

$$t_\beta^n = G^n(t_\beta^{n+1}; \omega^n) + s^n - G^n(s^{n+1}; \omega^n). \tag{4.3.4}$$

For later convenience, we denote

$$\tilde{G}(t^{n+1}, s^{n+1}; \omega^n) \triangleq G^n(t_\beta^{n+1}; \omega^n) + s^n - G^n(s^{n+1}; \omega^n) \tag{4.3.5}$$

the feedback operation used to propagate the targets in Eq. (4.3.4). Finally, the parameters θ^n are updated by the local loss $\mathcal{L}_\theta^n$, defined as:

$$\mathcal{L}_\theta^n = \frac{1}{2\beta}\|t_\beta^n - s^n\|^2, \tag{4.3.6}$$

where t_β^n is treated as a constant and the gradients blocked at s^{n-1}. For example, if F^n was linear, we would have the weight update $\Delta\theta^n \propto s^{n-1} \cdot (t_\beta^n - s^n)^\top$.
Feedback weights training. Both TP and DTP employ the same mechanism to train the G^n operators. First, the feedforward activations s^n get a noisy perturbation ϵ. The resulting noisy activations s_ϵ^n perturb the next layer s_ϵ^{n+1} through F^n, which in return yields a noisy reconstruction r_ϵ^n through G^n. The feedback weights are then updated to minimize the local loss $\mathcal{L}_\omega^n$ defined as:

$$\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2, \tag{4.3.7}$$

where s^n is treated as a constant and the gradient are blocked at s^{n+1}. Assuming again a linear G^n, the resulting feedback weight update also reads in a local fashion: $\Delta\omega^n \propto s^{n+1} \cdot (r_\epsilon^n - s_\epsilon^n)^\top$.

Algorithm 1 Standard DTP feedback weight training [53]

1: $\epsilon \sim \mathcal{N}(0, \sigma^2)$, $s_\epsilon^n = s^n + \epsilon$
2: $s_\epsilon^{n+1} = F^n(s_\epsilon^n; \theta^n)$
3: $r_\epsilon^n = G^n(s_\epsilon^{n+1}; \omega^n)$
4: Update ω^n with $\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2$.

4.3.2. Connection between DTP and Gauss-Newton Optimization

Using Eq. (4.3.3)-(4.3.4) and sending $\beta \to 0$, note that the DTP activation updates can be conveniently defined as:

$$\delta^n_{\text{DTP}} \triangleq \lim_{\beta \to 0} \frac{t^n_\beta - s^n}{\beta} = -\left[\prod_{k=n}^{N-1} \partial_{s^{k+1}} G^k\right] \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N} \tag{4.3.8}$$

It was suggested that under some conditions, δ^n_{DTP} encoded Gauss-Newton updates [24] of the layer activations with respect to the output loss function [11, 63]. In *invertible* networks, i.e. assuming $(F^n)^{-1}$ exists, the Gauss-Newton update of layer s^n with respect to $\mathcal{L}_{\text{pred}}$ is:

$$\delta^n_{\text{GN}} = -\left[\partial_{s^n} \overline{F}^n\right]^{-1} \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{4.3.9}$$

where $\overline{F}^n = F^N \circ \cdots \circ F^n$ denotes the forward mapping from s^n to s^N. Furthermore, assuming $G^n = F^{n-1^{-1}}$ for all $n = 1 \cdots N-1$, from Eq. (4.3.8), Eq. (4.3.9) and the inverse function theorem, it can be seen that $\delta^n_{\text{GN}} = \delta^n_{\text{DTP}}$.

In *non-invertible* networks, meulemans2020theoretical show that with a block-diagonal approximation of the Gauss-Newton curvature matrix, the Gauss-Newton update of s^n with respect to $\mathcal{L}_{\text{pred}}$ reads:

$$\delta^n_{\text{GN}} = -\left[\partial_{s^n} \overline{F}^n\right]^{\dagger} \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{4.3.10}$$

where $A^\dagger = \lim_{\lambda \to 0} A^\top \cdot \left(A \cdot A^\top - \lambda\right)^{-1}$ denotes the *Moore-Penrose* pseudo-inverse. However, there are are two reasons why in this case we may not have $\delta^n_{\text{GN}} = \delta^n_{\text{DTP}}$. First, using Eq. (4.3.7) as a reconstruction loss, it may not hold in general that $\partial_{s^{n+1}} G^n = (\partial_{s^n} F^n)^\dagger$. Second, even assuming this condition holds, $\left[\partial_{s^n} \overline{F}^n\right]^\dagger$ generally does not factorize as $\prod_{k=n}^N \left[\partial_{s^k} F^k\right]^\dagger$. A direct consequence of this is that DTP standard layer-wise feedback weights training leads to mostly inefficient feedforward weight updates that fail to move the output layer towards its target.

[63] show that by adapting DTP standard feedback training scheme, $\delta^n_{\text{GN}} = \delta^n_{\text{DTP}}$ can be recovered for non-invertible networks. Instead of propagating perturbated activations s^n_ϵ back and forth through F^n and G^n into the reconstructed activation r^n_ϵ to train ω^n, they prescribe sending s^n_ϵ up to $\hat{y}$ through $\overline{F}^n$, back into r^n_ϵ through $\tilde{G}^n \circ \cdots \circ \tilde{G}^N$ where $\tilde{G}^n$ (Eq. (4.3.5)) stands for the operator used for the target computation in Eq. (4.3.4). Finally, an extra noisy perturbation in the output layer $s^N + \eta$ needs to be propagated back into r^n_η. The resulting *Difference Reconstruction Loss* (DRL) to be optimized is defined as:

$$\hat{\mathcal{L}}^n_\omega = \frac{1}{2}\|r^n_\epsilon - s^n_\epsilon\|^2 + \lambda \|r^n_\eta - s^n\|^2. \tag{4.3.11}$$

In practice though, they replace the second term by weight decay. Taking expectation of Eq. (4.3.11) and sending the noise amplitude to 0, it can be shown that minimizing $\hat{\mathcal{L}}^n_\omega$ yields the desired property: $\prod_{k=n}^{N-1} \partial_{s^{k+1}} G^k = \left(\partial_{s^n} \overline{F}^n\right)^\dagger$.

Algorithm 2 Difference Reconstruction Loss (DRL) feedback weight training [63]

$\epsilon \sim \mathcal{N}(0, \sigma^2)$, $s_\epsilon^n = s^n + \epsilon$
for $k = n \cdots N-1$ **do**
  $s_\epsilon^{k+1} = F^k(s_\epsilon^k; \theta^k)$
end for
$r_\epsilon^N = s_\epsilon^N$
for $k = N-1 \cdots n$ **do**
  $r_\epsilon^k = G^k(r_\epsilon^N, r_\epsilon^{k+1}; \omega^k) + s^k - G^k(s^N, s^{k+1}; \omega^k)$
end for
Update ω^n with $\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2 + \lambda\omega^n$.

4.4. Learning Backprop Targets rather than Gauss-Newton Targets

In the spirit of meulemans2020theoretical, we propose to adapt the feedback weight training and the reconstruction loss, but we make it so that G^n learns the *transpose Jacobian* of its associated feedforward module F^n rather than its pseudo-inverse. This way, by construction, the DTP weight updates are made to match *BP* weight updates rather than a hybrid between BP and Gauss-Newton updates. We can also avoid the requirement of direct connections and restore layer-wise feedback weights training while preserving theoretical guarantees with respect to BP.

4.4.1. Feedback weights training

Definition 4.4.1. For a given architecture $\mathcal{F}$ defined by Definition 4.3.1, we say that a feedforward module F^n and associated feedback module G^n satisfy the *Jacobian-Matching Condition* (JMC) if:

$$\left(\partial_{s^n} F^n(s^n)\right)^\top = \partial_{s^{n+1}} G^n(s^{n+1}) \tag{4.4.1}$$

We say that an architecture $\mathcal{F}$ satisfies the JMC if for $n = 1 \cdots N$, (F^n, G^n) satisfy the JMC.

To illustrate the proposed algorithm to train the feedback weights, let us consider the feedforward module F^n and associated feedback module G^n. Let $\epsilon \sim \mathcal{N}(0, \sigma^2)$ be a perturbation to input feature s^n so that the resulting noisy activations s_ϵ^n triggers a noisy perturbation in the next layer s_ϵ^{n+1} through F^n. Then, we assume s_ϵ^{n+1} yields in turn a noisy reconstruction r_ϵ^n through $\tilde{G}^n$ from Eq. (4.3.5) (rather than G^n). Furthermore, we let $\eta \sim \mathcal{N}(0, \sigma^2)$ be a second source of noise in layer s^{n+1}. The resulting noisy activations s_η^{n+1} create the noisy reconstructions r_η^n again through $\tilde{G}^n$. We then prescribe updating the feedback weights with the *Local Difference Reconstruction Loss* (L-DRL) which we define as:

$$\hat{\mathcal{L}}_{\omega}^{n} = -\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) + \frac{1}{2}\|r_{\eta}^{n} - s^{n}\|^{2}. \tag{4.4.2}$$

Algorithm 3 Local Difference Reconstruction Loss (L-DRL)

for i = 1 to K **do**
  $s^{n+1} = F^{n}(s^{n}; \theta^{n})$
  $\epsilon \sim \mathcal{N}(0, \sigma^{2})$, $s_{\epsilon}^{n} = s^{n} + \epsilon$
  $s_{\epsilon}^{n+1} = F^{n}(s_{\epsilon}^{n}; \theta^{n})$
  $r_{\epsilon}^{n} = G^{n}(s_{\epsilon}^{n+1}; \omega^{n}) - G^{n}(s^{n+1}; \omega^{n}) + s^{n}$
  $\eta \sim \mathcal{N}(0, \sigma^{2})$, $r_{\eta}^{n+1} = s^{n+1} + \eta$
  $r_{\eta}^{n} = G^{n}(r_{\eta}^{n+1}; \omega^{n}) - G^{n}(s^{n+1}; \omega^{n}) + s^{n}$
  Update ω_n to descend layer-wise loss $\mathcal{L}_{\omega}^{n}$:
    $\hat{\mathcal{L}}_{\omega}^{n} = -\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) + \frac{1}{2}\|r_{\eta}^{n} - s^{n}\|^{2}$
end for

//

Contrary to existing DTP approaches, the above procedure is repeated K times per training batch, so that feedback weights can quickly and locally (per-layer) adapt on the fly to the feedforward activations and recent feedforward weight updates. This avoids interleaving phases of *pure* feedback weight training with frozen feedforward weights and instead makes it possible to train feedback and feedforward weights together from the beginning.

We now state Theorem 4.4.2 which guarantees that minimizing $\mathcal{L}_{\omega}^{n}$ as defined in Eq. (4.4.2) yields the JMC for layer n.

Theorem 4.4.2. *Let:*

$$\widehat{\mathcal{L}}_{\omega}^{n} = -\frac{1}{\sigma^{2}}\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) + \frac{1}{2\sigma^{2}}\left\|r_{\eta}^{n} - s^{n}\right\|^{2}, \tag{4.4.3}$$

$$\mathcal{L}_{\omega}^{n} = \frac{1}{2}\left\|\partial_{s^{n}} F^{n^{\top}} - \partial_{s^{n+1}} G^{n+1}\right\|_{F}^{2}. \tag{4.4.4}$$

Then:

$$\lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = -\left\langle \partial_{s^{n}} F^{n^{\top}}, \partial_{s^{n+1}} G^{n} \right\rangle_{F} + \frac{1}{2}\left\|\partial_{s^{n+1}} G^{n}\right\|_{F}^{2} \tag{4.4.5}$$

$$\frac{\partial}{\partial \omega} \lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = \frac{\partial \mathcal{L}_{\omega}^{n}}{\partial \omega}, \tag{4.4.6}$$

This means that training the feedback weights with respect to the local layer loss of Eq. (4.4.2) makes the feedback path compute the Jacobian of the feedforward path in the limit of small noise and in expectation over the noisy samples.

4.4.2. Feedforward weight training

Although this new implementation of DTP uses the exact same equations as standard DTP to propagate the targets (Eq. (4.3.3)-Eq. (4.3.4)) and update the forward weights (Eq. (4.3.6)), they acquire a very different meaning with the proposed novel feedback weights training scheme. If we assume that an architecture $\mathcal{F}$ satisfies the JMC upon applying Alg. (3) with fixed feedforward weights, then combining Eq. (4.3.8) and Eq. (4.4.1) yields:

$$\delta^n_{\text{DTP}} = -\left[\prod_{k=n}^{N-1} \partial_{s^k} F^{k^\top}\right] \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N} = \delta^n_{\text{BP}}, \tag{4.4.7}$$

where δ^n_{BP} denotes the activation updates computed by BP. Subsequently, given that the feedforward loss $\mathcal{L}^n_\theta$ defined in Eq. (4.3.6) is updated by gradient descent, the whole DTP gradient computing scheme exactly implements BP rather than a hybrid between gradient descent and Gauss-Newton optimization. We now formally state the result.

Theorem 4.4.3 (Gradient Matching Property)**.** *Let a feedforward architecture $\mathcal{F}$ defined per Definition 4.3.1 which satisfies the JMC. Then the following holds:*

$$\forall n \in [1, N], \qquad \frac{\partial \mathcal{L}_{\text{pred}}}{\partial \theta^n} = \lim_{\beta \to 0} \frac{1}{2\beta} \frac{\partial}{\partial \theta^n} \|t^n_\beta - s^n\|^2, \tag{4.4.8}$$

where the targets $(t^n_\beta)_{n \geq 1}$ obey the following recursive equations, $\forall n = N-1 \dots 1$:

$$\begin{cases} t^N_\beta = s^N - \beta \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s} \\ t^n_\beta = s^n + G^n(t^{n+1}_\beta; \omega^n) - G^n(s^{n+1}; \omega^n) \end{cases} \tag{4.4.9}$$

4.5. Experiments

In this section, we present several experimental results supporting the above theory. We first numerically demonstrate the claims stated by Theorem 4.4.2 and Theorem 4.4.3, thereby showing the efficiency of the proposed approach to align feedforward and feedback weights (JMC) and subsequently compute DTP feedforward weight updates well aligned with BP gradients (GMP). Next, we present training simulation results on MNIST, F-MNIST and CIFAR-10, where the proposed approach significantly outperforms meulemans2020theoretical's DTP. Finally, we report the best results ever obtained on ImageNet 32×32 by a DTP algorithm.

4.5.1. Demonstrating the JMC

Experimental set-up. The goal of the following experiment is to compare meulemans2020theoretical's DRL algorithm with the L-DRL approach in terms of their ability to align the (transposed) feedforward weights and their associated feedback weights for the last fully connected layer, and thereby realize the JMC in the output layer. We perform this test

with randomly initialized and *fixed* feedforward weights and on a *single* randomly selected input batch x (for a given seed). The choice of focusing only on the output layer is justified below.

Architecture. We consider a random batch of CIFAR-10 data along with a LeNet [51] architecture consisting of two convolutional layers and two fully connected (FC) layers. For both algorithms, we use the same feedforward pathway for the model. However since regular DRL prescribes by construction direct connections in the feedback pathway, the form of the G^n functions used depends on the feedback algorithm used. *For DRL*, we use the DDTP-linear architecture as per meulemans2020theoretical, where the output layer are directly connected to each upstream layer via linear connections. Therefore, the parameters of the resulting G^n functions have dimension $\dim(\omega^n) = s^n \times s^N$ for $n = N-1, \cdots, 1$. However, since the associated feedforward parameters θ^n have dimension $\dim(\theta^n) = s^{n+1} \times s^n$, we can only readily compare $\theta^{N^\top}$ and ω^N in the last FC layer. *For L-DRL*, we use layer-wise G^n functions such that $\dim(\omega^n) = s^n \times s^{n+1}$ for $n = N-1, \cdots, 1$. Full architecture details are included in the Appendix.

Results. We illustrate in Fig. 4.2 the results obtained. We show the angle (in degrees) and the relative distance between the last layer feedforward ($\theta^{N^\top}$) and feedback weights (ω^N) throughout pure feedback training on a single input batch. Therefore, each feedback training iteration here corresponds to a feedback weight update on the *same* input batch (for a given seed). However, we do use different input batches across different seeds. For each algorithm, the amount of noise and learning rates have been carefully tuned to achieve the minimal angles and distances after 5000 iterations, which we empirically found to be large enough to reach convergence for both algorithms. We observe that L-DRL achieves an angle of $\approx 3°$ and a relative distance of ≈ 0, while DRL can only reduce these quantities to $18.7°$ and ≈ 1.8 respectively. These results confirm that the L-DRL is more suited than DRL to achieve the JMC in the output layer.

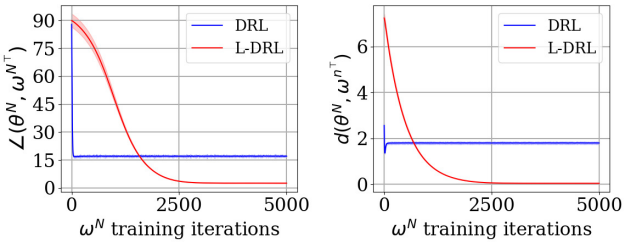


Figure 4.2. Angle in degrees ($\angle(\theta^N, \omega^{N^\top})$) and relative distance ($d(\theta^N, \omega^{N^\top})$) between θ^N and $\omega^{N^\top}$ throughout feedback weight learning with L-DRL (presented here) and DRL [63] with *fixed* feedforward weights.

4.5.2. Demonstrating the GMP

Experimental set-up. In this experiment, we want to demonstrate the ability of the proposed DTP to compute *feedforward* weight updates closely matching those prescribed by BP (therefore achieving the GMP), assuming that the JMC is initially satisfied, as hypothesized by Theorem 4.4.3. Again here, we assume a *single* randomly selected input batch x (also with different input across different seeds). In contrast with the previous experiment though, we carry out this analysis across *all* the layers. Indeed, regardless of the form of the G^n functions (whether we use direct connections or not), the DTP feedforward weight updates can always be compared against those of BP. Given randomly sampled feedforward parameters θ^n, we study five different feedback weight initialization schemes and associated targets computation: (a) ω^n are random and targets are computed through the DDTP-linear feedback pathway ($\text{DRL}_{\text{random}}$); (b) same as (a) with targets computed through the layer-wise feedback pathway ($\text{L-DRL}_{\text{random}}$); (c) ω^n are trained with DRL (DRL); (d) ω^n are trained with L-DRL (L-DRL); (e) Finally, $\omega^n = {\theta^n}^\top$ with targets propagated through the layer-wise feedback pathway ($\text{L-DRL}_{\text{sym}}$). For each of these situations, the feedforward DTP weight updates are thereafter obtained with Eq. (4.3.6) on the one hand. On the other hand, we compute BP gradients via standard BP through the feedforward pathway.

Architecture. The architecture used for this experiment is the same LeNet architecture than the one used for the previous experiment, with two convolutional layers and two fully connected layers.

Results. We show on Fig. 4.3 the results obtained. The blue, red, green and purple bars correspond to the angle between DTP feedforward weight updates and those of BP ($\angle(\Delta\theta^n_{\text{DTP}},\Delta\theta^n_{\text{BP}})$) for the first Conv, second Conv, first FC and second FC layers respectively: the lower $\angle(\Delta\theta^n_{\text{DTP}},\Delta\theta^n_{\text{BP}})$, the more the GMP is satisfied. We show these quantities for each of the five feedback weight initialization mentioned above. We observe that upon training the feedback weights with L-DRL (compared to a random configuration), the GMP is significantly better satisfied ($\angle(\Delta\theta^n_{\text{DTP}},\Delta\theta^n_{\text{BP}})$ going from $\approx 90°$ to $\lessapprox 35°$) than when trained with DRL ($\lessapprox 79°$)), and almost as well as in the ideal situation with symmetrically initialized weights. Overall, these results confirm the prediction of Theorem 4.4.3.

4.5.3. DTP learning dynamics

Experimental set-up. We present here the training experiments obtained on MNIST, F-MNIST and CIFAR-10 with the implementation of DTP (refered to as "DTP" or "Ours" below) and that of meulemans2020theoretical which we will refer to as "DDTP". While the previous DRL/L-DRL terminology concerns feedback weights training specifically, the

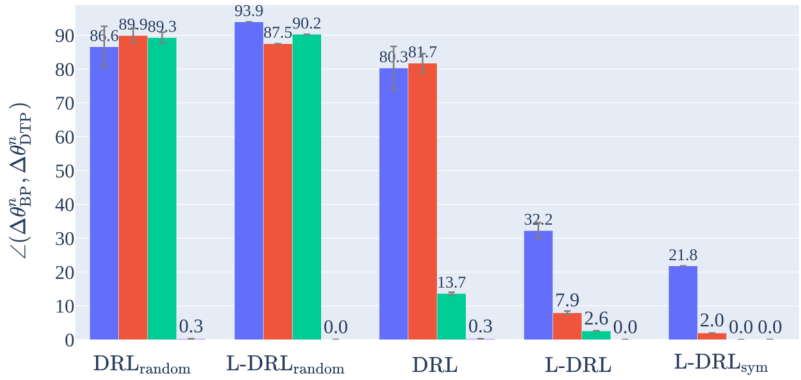


Figure 4.3. Angle between the forward weight updates obtained through L-DRL (proposed here) or DRL [63] and those obtained through BP, for each layer, under various initial conditions. Each color corresponds to a specific layer, ordered from input to output: Blue (1st Convolution), Red (2nd Convolution), Green (3rd Convolution), Light Purple (4th Convolution), Orange (5th Convolution), and Turquoise (Fully Connected Output).

term "DDTP" is used here to refer to the resulting feedforward weights training algorithm when GN targets are being computed, rather than the architecture itself. Also, we want to emphasize that two features of DDTP fundamentally differs from the proposed DTP. First, DTP is made to emulate BP while DDTP is a hybrid between GN optimization and BP as highlighted previously. Second, while DDTP employs feedback weights pre-training and subsequent interleaved epochs of pure feedback weights training, the presented DTP method trains together *at all times* feedforward weights and feedback weights and allowing for *multiple* feedback weight updates per mini-batch. To disentangle these two aspects and ensure a fair comparison between the proposed DTP and DDTP, we propose two different implementations of DDTP.

Simple DDTP ("s-DDTP") is the standard DDTP implementation of meulemans2020theoretical that yields their best training results. For s-DDTP, training starts with $N_{\omega,\text{i}}$ epochs of pure feedback weights training, then at each subsequent epoch feedback weights and feedforward weights are both updated once per batch, and each of these epoch is followed by $N_{\omega,\text{e}}$ epoch of pure feedback training. Therefore, denoting N_θ the number of epochs where the feedforward weights are trained, there are $N_\omega = N_{\omega,\text{i}} + N_\theta \times (1 + N_{\omega,\text{e}})$ epochs where the feedback weights are trained, therefore $\mathcal{O}(N_{\omega,\text{i}} + N_\theta \times (1 + N_{\omega,\text{e}}))$ feedback weight updates.

We define *Parallel* DDTP ("p-DDTP") as a variant of DDTP where there is no initial feedback pre-training ($N_{\omega,\text{i}} = 0$), nor interleaved epochs of pure feedback training ($N_{\omega,\text{e}} = 0$), but where feedback weights and feedforward weights are always trained altogether, with K

feedback weight updates per batch, yielding $\mathcal{O}(N_\theta \times K)$ feedback weight updates. Therefore, p-DDTP has the same complexity cost for feedback weights training as in the proposed DTP. We use the same architecture in this study as in Section 4.5.1-4.5.2.

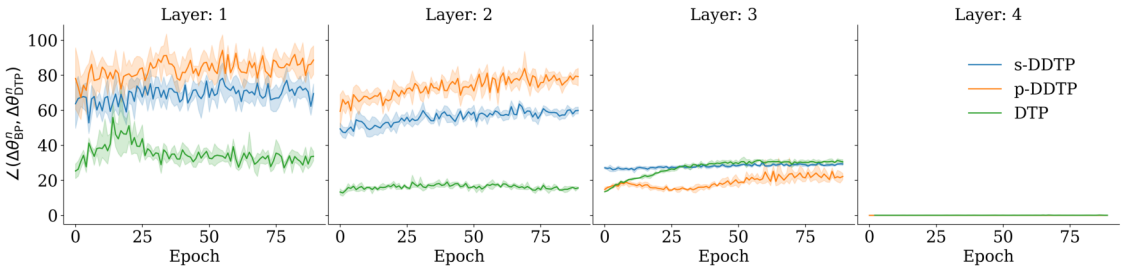


Figure 4.4. Angle between forward weight updates obtained through DTP (ours), s-DDTP, p-DDTP and those obtained through back-propogation for each layer throughout training on CIFAR-10 with LeNet architecture. Each epoch represents a feedforward training epoch, pure feedback training epochs for s-DDTP are not displayed.

Results. We display in Table 4.1 the accuracies obtained with the proposed DTP, s-DDTP and p-DDTP on MNIST, Fashion MNIST ("F-MNIST") and CIFAR-10. The DTP method outlined in this work outperforms s-DDTP and p-DDTP on all tasks, by $\approx 0.3\%$ on MNIST and F-MNIST, by at least $\approx 9\%$ on CIFAR-10 and is within $\approx 1\%$ of the BP baseline performance. While p-DDTP slightly outperforms s-DDTP on MNIST and F-MNIST, it performs worse than s-DDTP on CIFAR-10, suggesting that DDTP does not benefit much from multiple feedback weight updates per batch. An important conclusion to be drawn here is that the gap in performance between DTP and DDTP is not due to updating the feedback weights multiple times per training batch, but more fundamentally to the feedback training scheme at use (L-DRL for the proposed DTP, or DRL for DDTP), yielding better feedforward error signals with the proposed DTP. This conclusion is also confirmed by Fig. 4.4 where we plot the angle between the DTP feedforward weight updates and those of BP ($\angle(\Delta\theta^n_{\text{BP}}, \Delta\theta^n_{\text{DTP}})$) throughout learning CIFAR-10, for each layer and each algorithm. While the angles obtained by DTP, s-DDTP and p-DDTP are comparable for the last two (FC) layers ($\approx 0°$), they are at least twice as smaller for DTP compared to s-DDTP and p-DDTP in the first two (Conv) layers. Finally, these angles are $\approx 15°$ smaller for s-DDTP compared to p-DDTP. Consequently, these curves directly account for the discrepancies in results on CIFAR-10 reported in Table 4.1.

4.5.4. Towards scaling up DTP

Since we have demonstrated that the proposed DTP method learns better error signals to update the feedforward weights than DDTP with consistently better performance, we focus in this section on learning a slightly deeper and wider architecture on CIFAR-10 and ImageNet 32×32 [74], a downsampled version of the full ImageNet data. For these experiments, we

Table 4.1. Accuracies (%) obtained with BP, DDTP and the proposed DTP on the LeNet architecture for MNIST, F-MNIST and CIFAR-10 test set. Each result is in terms of the mean and standard deviation obtained over five different seeds.

	MNIST	F-MNIST	CIFAR-10
s-DDTP	$98.59_{\pm 0.16}$	$88.86_{\pm 0.44}$	$76.33_{\pm 0.27}$
p-DDTP	$98.58_{\pm 0.13}$	$89.42_{\pm 0.69}$	$72.15_{\pm 0.29}$
Ours	$\mathbf{98.93_{\pm 0.04}}$	$\mathbf{90.35_{\pm 0.11}}$	$\mathbf{85.33_{\pm 0.32}}$
BP	$98.92_{\pm 0.04}$	$91.94_{\pm 0.33}$	$86.34_{\pm 0.35}$

employ a 6-layers VGG-like architecture, consisting of 5 Conv layers and 1 Fully Connected (FC) layer (see Appendix for architecture details).

Results. We report the results in Tables 4.2–4.3. With this choice of architecture, the proposed DTP method achieves 89.38% accuracy on CIFAR-10 and 60.55% top-5 accuracy on ImageNet 32×32, which is both cases within $< 1\%$ of the BP baseline.

Table 4.2. Accuracies (%) obtained on CIFAR-10 with BP and the proposed DTP model on a VGG-like architecture. Each result is in terms of the mean and standard deviation obtained over five different seeds. We also report below the current best CIFAR-10 accuracies obtained by DTP in the literature on any architecture.

	Accuracy
BP	$89.07_{\pm 0.22}$
Ours	$\mathbf{89.38_{\pm 0.20}}$
meulemans2020theoretical	76.01
bartunov2018assessing	60.53

Table 4.3. Top-1 and Top-5 Accuracies for ImageNet 32×32 validation set obtained with BP and the proposed DTP on a VGG-like architecture across five seeds.

	Top-1	Top-5
BP	37.29 ± 0.14	61.28 ± 0.11
Ours	36.79 ± 0.05	60.55 ± 0.06

4.6. Discussion

Training feedforward weights with Gauss-Newton targets results in optimal updates to move the feedforward activations towards their associated target, yet they appear sub-optimal to decrease the prediction loss [63], which calls for the design of a principled way to build backprop-like targets. In this work, we have demonstrated the benefits of such an approach, with mathematically and experimentally grounded arguments. We showed the efficiency

of the presented L-DRL algorithm to align feedforward and (transposed) feedback weights and therefore achieve the Jacobian matching condition (JMC). We also showed that the resulting feedforward weight updates prescribed by Difference Target Propagation (DTP) closely match those of BP, a property we called the gradient-matching property (GMP). The proposed DTP implementation subsequently outperforms DDTP [63] on all training tasks and approaches the BP baseline performance. We also consistently showed that the more the GMP is satisfied throughout learning, the better the resulting performance. The best CIFAR-10 performance obtained by DTP is $\approx 13\%$ higher than the existing DTP performances reported in the literature [10, 63] and to our knowledge this is the first report of a DTP performance closely matching that of BP on such a complex task as ImageNet 32×32.

Limitations and Future Work. The prescription to run several feedback weight updates per training batch entails longer simulation times but may be biologically plausible since local recurrent paths will have shorter axons that should have much shorter delays than long-range paths with complex, long axons [21, 22]. As it can be seen from Appendix 4.13, the proposed DTP implementation can be up to 30 times slower than BP. Future work could be done to leverage the parallelism allowed by the layer-wise feedback weight training strategy presented in this work, to accelerate training and subsequently scale up this DTP implementation to ImageNet on deeper architectures. However we emphasize again that DTP is not meant as a new practical optimization method but rather an abstract but plausible biological implementation of a BP-like update mechanism.

The code is available at `https://github.com/ernoult/scalingDTP`.

4.7. Acknowledgements.

BR was supported by NSERC (Discovery Grant: RGPIN-2020-05105; Discovery Accelerator Supplement: RGPAS-2020-00031) and CIFAR (Canada CIFAR AI Chair and Learning in Machines and Brains Program). EB and AM are supported by NSERC Discovery Grant RGPIN-2021-04104. We acknowledge resources provided by Compute Canada. YB was funded by NSERC and CIFAR. Many thanks to Anirudh Goyal and Guillaume Bellec for their useful feedback.

4.8. Appendix

The appendix is structured as follows:

- Section 4.9 provides the demonstrations of Theorem 4.4.2 and Theorem 4.4.3
- Section 4.10 shows the explicit equations used for inference and weight updates on a simple example.

- Section 4.11 precisely describes the architectures used for the experiments.
- Section 4.12 lists all the hyperparameters used across all the experiments.
- Finally, Section 4.13 gives the simulation run times of all the training experiments.

4.9. Theoretical results

4.9.1. Feedback weights training

We re-state the main theorem for feedback weights training (Theorem (4.4.2) in the main).

Theorem 4.9.1. *Let:*

$$\widehat{\mathcal{L}}_{\omega}^{n} = -\frac{1}{\sigma^2}\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) + \frac{1}{2\sigma^2}\left\| r_{\eta}^{n} - s^{n} \right\|^2, \tag{4.9.1}$$

$$\mathcal{L}_{\omega}^{n} = \frac{1}{2}\left\| \partial_{s^n} F^{n^{\top}} - \partial_{s^{n+1}} G^{n+1} \right\|_F^2. \tag{4.9.2}$$

Then:

$$\lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = -\left\langle \partial_{s^n} F^{n^{\top}}, \partial_{s^{n+1}} G^{n} \right\rangle_F + \frac{1}{2}\left\| \partial_{s^{n+1}} G^{n} \right\|_F^2 \tag{4.9.3}$$

$$\frac{\partial}{\partial \omega^n} \lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = \frac{\partial \mathcal{L}_{\omega}^{n}}{\partial \omega^n}, \tag{4.9.4}$$

Démonstration. We have:

$$\begin{aligned} \epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) &= \epsilon^{\top} \cdot (G^{n} \circ F^{n}(s^{n} + \epsilon) - G^{n} \circ F^{n}(s^{n})) \\ &= \epsilon^{\top} \cdot (\partial_{s^{n+1}} G^{n} \cdot \partial_{s^n} F^{n} \cdot \epsilon + o(\|\epsilon\|)) \\ &= \mathrm{Tr}(\epsilon \cdot \epsilon^{\top} \cdot \partial_{s^{n+1}} G^{n} \cdot \partial_{s^n} F^{n}) + o(\|\epsilon\|^2). \end{aligned} \tag{4.9.5}$$

Likewise:

$$\left\| r_{\eta}^{n} - s^{n} \right\|^2 = \mathrm{Tr}\left(\eta \cdot \eta^{\top} \cdot \partial_{s^{n+1}} G^{n} \cdot \partial_{s^{n+1}} G^{n^{\top}}\right) + o(\|\eta\|^2). \tag{4.9.6}$$

Moreover, since $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\eta \sim \mathcal{N}(0, \sigma^2)$, $\mathbb{E}_{\epsilon}\left[o(\|\epsilon\|^2)\right] = \mathbb{E}_{\eta}\left[o(\|\eta\|^2)\right] = o(\sigma^2)$. Altogether with Eq. (4.9.5) and Eq. (4.9.6), we finally obtain:

$$\mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_\omega^n\right] = -\frac{1}{\sigma^2}\mathrm{Tr}(\underbrace{\mathbb{E}_\epsilon\left[\epsilon\cdot\epsilon^\top\right]}_{=\sigma^2\times\mathbf{1}}\cdot\partial_{s^{n+1}}G^n\cdot\partial_{s^n}F^n) + \frac{1}{2\sigma^2} + \mathrm{Tr}\left(\underbrace{\mathbb{E}_\eta\left[\eta\cdot\eta^\top\right]}_{=\sigma^2\times\mathbf{1}}\cdot\partial_{s^{n+1}}G^n\cdot\partial_{s^{n+1}}G^{n^\top}\right) + o(1)$$

$$= -\left\langle\partial_{s^n}F^{n^\top},\partial_{s^{n+1}}G^n\right\rangle_F + \frac{1}{2}\left\|\partial_{s^{n+1}}G^n\right\|_F^2 + o(1). \tag{4.9.7}$$

Therefore, sending $\sigma\to 0$ yields the desired result Eq. (4.9.3). Finally, noticing that:

$$\lim_{\sigma\to 0}\mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_\omega^n\right] = \mathcal{L}_\omega^n - \frac{1}{2}\left\|\partial_{s^n}F^n\right\|_F^2, \tag{4.9.8}$$

with the second term of Eq. (4.9.8) not depending on the feedback weights ω^n, Eq. (4.9.4) is straightforward. □

4.9.2. Feedforward weights training

We re-state here the main theorem for feedforward weights training (Theorem 4.4.3) in the main).

Theorem 4.9.2 (Gradient Matching Property)**.** *Let a feedforward architecture $\mathcal{F}$ defined per Definition 4.3.1 which satisfies the JMC. Then the following holds:*

$$\forall n\in[1,N],\qquad \frac{\partial\mathcal{L}_{\text{pred}}}{\partial\theta^n} = \lim_{\beta\to 0}\frac{1}{2\beta}\frac{\partial}{\partial\theta^n}\|t_\beta^n - s^n\|^2, \tag{4.9.9}$$

where the targets $(t_\beta^n)_{n\geq 1}$ obey the following recursive equations, $\forall n = N\dots 2$:

$$\begin{cases} t_\beta^N = s^n - \beta\frac{\partial\mathcal{L}_{\text{pred}}}{\partial s} \\ t_\beta^n = s^n + G^n(t_\beta^{n+1};\omega^n) - G^n(s^{n+1};\omega^n) \end{cases} \tag{4.9.10}$$

Démonstration. First, note we have:

$$\begin{cases} t_\beta^N - s^n = -\beta\frac{\partial\mathcal{L}_{\text{pred}}}{\partial s}, \\ t_\beta^n - s^n = \partial_{s^{n+1}}G^n\cdot\left(t_\beta^{n+1} - s_\beta^{n+1}\right) + o\left(\left\|t_\beta^{n+1} - s_\beta^{n+1}\right\|\right). \end{cases} \tag{4.9.11}$$

Since $t_\beta^N - s^N = o(\beta)$, by immediate induction $t_\beta^n - s^n = o(\beta)\quad\forall n = N-1\cdots 1$. Denoting $\hat{\delta}_{\text{DTP}}^n(\beta)\triangleq\frac{t^n-s^n}{\beta}$, we therefore obtain:

$$\begin{cases} \hat{\delta}_{\text{DTP}}^N(\beta) = -\frac{\partial\mathcal{L}_{\text{pred}}}{\partial s} + o(1), \\ \hat{\delta}_{\text{DTP}}^n(\beta) = \partial_{s^{n+1}}G^n\cdot\hat{\delta}_{\text{DTP}}^{n+1}(\beta) + o\left(1\right). \end{cases} \tag{4.9.12}$$

Furthermore note that, treating t_β^n as a constant, we also have:

$$\frac{1}{2\beta}\frac{\partial}{\partial\theta^n}\|t^n_\beta - s^n\|^2 = -\frac{1}{\beta}\partial_{\theta^n}{F^n}^\top \cdot (t^n_\beta - s^n)$$
$$= -\partial_{\theta^n}{F^n}^\top \cdot \hat{\delta}^n_{\mathrm{DTP}}(\beta). \tag{4.9.13}$$

Finally, sending $\beta \to 0$, defining:

$$\delta^n_{\mathrm{DTP}} \triangleq \lim_{\beta\to 0} \hat{\delta}^n(\beta) \tag{4.9.14}$$

$$\Delta^n_{\theta,\mathrm{DTP}} \triangleq \lim_{\beta\to 0} \frac{1}{2\beta}\frac{\partial}{\partial\theta^n}\|t^n_\beta - s^n\|^2 \tag{4.9.15}$$

along with the JMC property $\partial_{s^{n+1}} G^n = \partial_{s^n}{F^n}^\top$, we obtain:

$$\begin{cases} \delta^N_{\mathrm{DTP}} = -\frac{\partial \mathcal{L}_{\mathrm{pred}}}{\partial s}, \\ \delta^n_{\mathrm{DTP}} = \partial_{s^{n+1}}{F^n}^\top \cdot \hat{\delta}^{n+1}_{\mathrm{DTP}} \\ \Delta^n_{\theta,\mathrm{DTP}} = -\partial_{\theta^n}{F^n}^\top \cdot \hat{\delta}^n_{\mathrm{DTP}} \end{cases} \tag{4.9.16}$$

Note that Eq. (4.9.16) is equivalent to computing $\frac{\partial \mathcal{L}_{\mathrm{pred}}}{\partial \theta^n}$ by backprop, yielding the desired result Eq. (4.9.9). □

4.10. A concrete example with explicit equations

We detail for completeness and clarity all the equations for the neural dynamics and the learning rules of the forward and of the backward weights for a LeNet-like architecture with two Conv layers and one fully connected layer for the sake of simplicity.
Forward operations.

$$s^1 = F^0(x;\theta^0) = \sigma(\theta^0 \star x),$$
$$s^2 = F^1(s^1;\theta^1) = \sigma(\theta^1 \star s_1),$$
$$s^3 = F^2(s_2;\theta^2) = \theta^2 \cdot s^2,$$
$$\hat{y} = \mathrm{softmax}(s_3),$$

where we implicitly assume the flattening operation between s^2 and s^3 for notational convenience.
Backward operations. We assume here the following feedback operators G^1 and G^2 associated with F^1 and F^2 respectively:

$$G^2(s^3;\omega^2) = \omega^2 \cdot s^3$$
$$G^1(s^2;\omega^1) = \omega^1 \star \sigma(s^2).$$

Again, note that there is no G^0 feedback operator paired to F^0 since we do not need to propagate error signals down to the input layer.
Feedback weights training. Given input noises ϵ^2 and η^3 in layers s^2 and s^3 respectively, we update ω^2 so as to minimize the loss $\mathcal{L}_{\omega^2}$. More precisely, ϵ^2, η^3 and $\mathcal{L}_{\omega^2}$ are defined as:

$$\begin{aligned}
\epsilon^2 &\sim \mathcal{N}(0, \sigma^2), \\
\eta^3 &\sim \mathcal{N}(0, \sigma^2), \\
\mathcal{L}_{\omega^2} &= -\left(\epsilon^2\right)^\top \cdot (G^2(F^2(s^2 + \epsilon^2)) - G^2(s^3)) + \frac{1}{2}\left\|G^2(s^3 + \eta^3) - G^2(s^3)\right\|^2,
\end{aligned}$$

which results in the weight update for ω^2:

$$\Delta\omega^2 = \epsilon^2 \cdot (\theta^2 \cdot \Delta x^2)^\top - (\omega^2 \cdot \eta^3) \cdot \Delta y^{3^\top}. \tag{4.10.1}$$

Similarly, we train the feedback convolutional filters ω^1 by injecting the input noise ϵ^1 and η^2 in s^1 and s^2 respectively and minimizing the loss $\mathcal{L}_{\omega^1}$ defined as:

$$\begin{aligned}
\epsilon^1 &\sim \mathcal{N}(0, \sigma^2), \\
\eta^2 &\sim \mathcal{N}(0, \sigma^2), \\
\mathcal{L}_{\omega^1} &= -\left(\epsilon^1\right)^\top \cdot (G^1(F^1(s^1 + \epsilon^1)) - G^1(s^2)) + \frac{1}{2}\left\|G^1(s^2 + \eta^2) - G^1(s^2)\right\|^2,
\end{aligned}$$

which results in the weight update for the convolutional feedback filters ω^1:

$$\Delta\omega^1 = \epsilon^1 \star (\sigma(F^1(s^1 + \epsilon^1)) - \sigma(s^2)) - (G^1(s^2 + \eta^2) - G^1(s^2)) \star (\sigma(s^2 + \eta^2) - \sigma(s^2)). \tag{4.10.2}$$

Forward weights training. We compute the first target t^3_β and associated weight update $\Delta\theta^2$ as:

$$t^3_\beta = s^3 + \beta(\hat{y} - y), \tag{4.10.3}$$

$$\Delta\theta^2 = \frac{1}{\beta}(t^3_\beta - s^3) \cdot s^{2^\top}. \tag{4.10.4}$$

Then, the target t^3_β passes through G^2, yielding the target t^2_β and associated weight update $\Delta\theta^1$:

$$t^2_\beta = s^2 + G^2(t^3; \omega^2) - G^2(s^3; \omega^2), \tag{4.10.5}$$

$$\Delta\theta^1 = \frac{1}{\beta}((t^2_\beta - s^2) \odot \sigma'(s^2)) \star s^1. \tag{4.10.6}$$

Similarly, we compute t^1_β and $\Delta\theta^0$ as:

$$t^1_\beta = s^1 + G^1(t^2; \omega_2) - G^1(s^2; \omega_2) \tag{4.10.7}$$

$$\Delta\theta^0 = \frac{1}{\beta}((t^1_\beta - s^1) \odot \sigma'(s^1)) \star x \tag{4.10.8}$$

4.11. Architecture Details

In Table 4.4, we give the details of the two representative architectures studied in this work.

LeNet	VGGNet
Conv 5x5x32 (stride=1, pad=2)	Conv 3x3x128 (stride=1, pad=1)
Maxpool 3x3 (stride=2, pad=1)	Maxpool 2x2 (stride=2, pad=0)
Conv 5x5x64 (stride=1, pad=2)	Conv 3x3x128 (stride=1, pad=1)
Maxpool 3x3 (stride=2, pad=1)	Maxpool 2x2 (stride=2, pad=0)
FC 512	Conv 3x3x256 (stride=1, pad=1)
FC+Softmax 10	Maxpool 2x2 (stride=2, pad=0)
-	Conv 3x3x256 (stride=1, pad=1)
-	Maxpool 2x2 (stride=2, pad=0)
-	Conv 3x3x512 (stride=1, pad=1)
-	Maxpool 2x2 (stride=2, pad=0)
-	FC+Softmax 10

Table 4.4. Architectures described by layer

4.12. Hyperparameters

In Tables 4.5,4.6,4.8,4.9, we report the hyperparameters for each method and dataset studied. In both CIFAR-10 and Imagenet 32×32 experiments we use the same data augmentation consisting of random horizontal flipping with 0.5 probability and random cropping with padding 4.

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
activation	ELU	ELU	ELU
lr_f	0.007938	0.01374	0.03
forward optimizer	SGD	SGD	SGD
forward momentum	0.9	0.9	0.9
wd_f	0.0001	0.0001	0.0001
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform	kaiming uniform
batch size	166	140	193
epochs	40	40	90

Table 4.5. Tuned BP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
β	0.47	0.47	0.46
σ	[0.4, 0.4, 0.2]	[0.4, 0.4, 0.2]	[0.41, 0.28, 0.19]
activation	ELU	ELU	ELU
lr_f	0.02	0.005	0.0.01
forward optimizer	SGD	SGD	SGD
forward momentum	0.9	0.9	0.9
wd_f	0.0001	0.0001	0.0001
lr_fb	[0.06, 0.006, 0.018]	[0.068, 0.006, 0.018]	[0.001,0.005,0.045]
feedback training iterations	[18, 23, 12]	[18, 23, 12]	[41, 51, 24]
backward optimizer	SGD	SGD	SGD
backward momentum	0.9	0.9	0.9
wd_fb	None	None	None
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform	kaiming uniform
batch size	107	107	100
epochs	40	40	90

Table 4.6. Tuned DTP LeNet hyperparameters

Hyperparameter	Dataset	
	CIFAR-10	ImageNet 32×32
channels	[128, 128, 256, 256, 512]	[128, 128, 256, 256, 512]
β	0.7	0.7
σ	[0.4, 0.4, 0.2, 0.2, 0.08]	[0.4, 0.4, 0.2, 0.2, 0.08]
activation	ELU	ELU
$\mathrm{lr_f}$	0.05	0.01
forward optimizer	SGD	SGD
forward momentum	0.9	0.9
$\mathrm{wd_f}$	0.0001	0.0001
$\mathrm{lr_{fb}}$	[1e-4, 3.5e-4, 8e-3, 8e-3, 0.18]	[1e-4, 3.5e-4, 8e-3, 8e-3, 0.18]
feedback training iterations	[20, 30, 35, 55, 20]	[25, 35, 40, 60, 25]
backward optimizer	SGD	SGD
backward momentum	0.9	0.9
$\mathrm{wd_{fb}}$	None	None
scheduler	cosine	cosine
scheduler eta min	0.00001	0.00001
scheduler Tmax	85	85
scheduler inter-val/frequency	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform
batch size	128	256
epochs	90	90

Table 4.7. Tuned DTP VGGNet hyperparameters

Table 4.8. Tuned s-DDTP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
target stepsize η	0.044	0.044	0.016
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	[6.5e-05, 1.1e-05, 6.8e-05, 2.5e-05]	[6.5e-05, 1.1e-05, 6.8e-05, 2.5e-05]	[2.7e-08, 1.9e-08, 4.5e-06, 4.0e-05]
ϵ_{fb}	9.5e-08	9.5e-08	7.5e-07
$\beta_{1,\text{fb}}$	0.999	0.999	0.999
$\beta_{2,\text{fb}}$	0.999	0.999	0.999
σ	[4.7e-05, 8.7e-4, 2.4e-4, 3.9e-05]	[4.7e-05, 8.7e-4, 2.4e-4, 3.9e-05]	[9.2e-3, 9.2e-3, 9.2e-3, 9.2e-3]
activation	tanh	tanh	tanh
lr_f	[1.7e-3, 0.09, 0.02, 0.002]	[0.0016, 0.097, 0.025, 1.9e-3]	[2.6e-4, 8.9e-4, 1.4e-4, 3.4e-06]
forward optimizer	Adam	Adam	Adam
wd_f	0	0	0
lr_fb	1.6e-4	1.6e-4	4.5e-3
feedback training iterations	[1, 1, 1, 1]	[1, 1, 1, 1]	[1, 1, 1, 1]
feedback activation	linear	linear	linear
backward optimizer	Adam	Adam	Adam
wd_fb	3.993e-05	3.993e-05	6.169295107849636e-05
feedback pre-training epochs	10	10	10
feedback extra training epochs	1	1	1
scheduler	cosine	cosine	cosine
scheduler eta min	1e-5	1e-5	1e-5
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	xavier normal	xavier normal	xavier normal
batch size	143	143	128
epochs	40	40	90

Table 4.9. Tuned p-DDTP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
target stepsize η	0.0428	0.01308	0.09983
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	[4.4e-06, 9e-07, 2.2e-05, 1.4e-05]	[2.4e-08, 4.8e-06, 4.7e-06, 1.6e-07]	[3.8e-05, 1e-07, 4.7e-07, 2.4e-06]
ϵ_{fb}	1.2e-08	2.5e-07	2e-07
$\beta_{1,\text{fb}}$	0.999	0.999	0.999
$\beta_{2,\text{fb}}$	0.999	0.999	0.999
σ	[1.3e-05, 2.4e-4, 2.9e-05, 3.3e-4]	[3.8e-3, 2.2e-3, 5.6e-4, 4.2e-3]	[3e-4, 3e-05, 7.9e-05, 6.6e-4]
activation	tanh	tanh	tanh
lr_f	[0.004, 1.4e-3, 6.1e-4, 1.3e-4]	[3.9e-4, 1.8e-3, 1.5e-3, 1.5e-4]	[2.2e-4, 6.2e-3, 1.5e-4, 5.5e-05]
forward optimizer	Adam	Adam	Adam
wd_f	0	0	0
lr_fb	6.4e-4	1.8e-4	1.1e-3
feedback training iterations	[49, 32, 54, 11]	[48, 42, 52, 22]	[24, 35, 36, 19]
feedback activation	linear	linear	linear
backward optimizer	Adam	Adam	Adam
wd_fb	2.236e-05	0.000128	3.564e-06
feedback pre-training epochs	0	0	0
feedback extra training epochs	0	0	0
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	xavier normal	xavier normal	xavier normal
batch size	138	141	190
epochs	40	40	90

4.13. Simulation run times

Table 4.10. Run time and accuracy of the training experiments (*hours:minutes*) for ImageNet 32×32 obtained with BP and the proposed DTP on a VGG-like architecture across 5 seeds.

	Top-1	Top-5	Run time
BP	37.29 ± 0.14	61.28 ± 0.11	2:22 ± 0:2
Ours	36.79 ± 0.05	60.55 ± 0.06	61:41 ± 0:54

Table 4.11. Run time of the training experiments (*hours:minutes*) obtained with BP, DDTP and the proposed DTP model on the LeNet architecture for MNIST, F-MNIST and CIFAR-10 across 5 seeds.

	MNIST	F-MNIST	CIFAR-10
s-DDTP	0:36 ± 0:06	0:32 ± 0:05	1:11 ± 0:04
p-DDTP	3:03 ± 0:04	3:01 ± 0:03	3:42 ± 0:05
Ours	1:17 ± 0:10	1:05 ± 0:07	4:13 ± 0:33
BP	0:9 ± 0:02	0:10 ± 0:02	0:22 ± 0:03

Table 4.12. Run time of the training experiments (*hours:minutes*) obtained on CIFAR-10 with BP and the proposed DTP model on a VGG-like architecture across 5 seeds.

	Run time
BP	0:17 ± 0:03
Ours	6:03 ± 0:39

$\prod_{k=n}^{N-1} \partial_{s^{k+1}} G^k = \left(\partial_{s^n}(F^N \circ \cdots \circ F^n)\right)^\dagger$

Theorem 4.13.1 (Informal)**.** *If JMC is satisfied then:*

$$\forall n \in [\![1, N]\!], \qquad \frac{\partial \mathcal{L}_{\text{pred}}}{\partial \theta^n} = \lim_{\beta \to 0} \frac{1}{2\beta} \frac{\partial}{\partial \theta^n} \|t_\beta^n - s^n\|^2, \tag{4.13.1}$$

Theorem 4.13.2 (Informal)**.**

$$\frac{\partial}{\partial \omega} \lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_\omega^n\right] = \frac{\partial}{\partial \omega}\left[\frac{1}{2}\left\|\partial_{s^n} F^{n^\top} - \partial_{s^{n+1}} G^{n+1}\right\|_F^2\right], \tag{4.13.2}$$

Chapter 5

Third article: On the Role of Memory in Planning and Reasoning within Sequential Decision-Making Domains

Abstract: Efficient planning in sequential decision-making tasks remains a core challenge for machine learning models. These tasks often involve intricate decision sequences within discrete state spaces, leading to combinatorial complexity that hampers traditional planning methods. In this study, we investigate adaptive computation, a strategy inspired by human cognitive processes, to enhance planning efficiency and resource allocation. The presented approach borrows concepts from adaptive computation, incorporating memory and reusability mechanisms into models that must solve tasks where planning is required. In summary, this paper presents an exploratory study on integrating adaptive computation for efficient planning, demonstrating the potential to achieve model efficiency while preserving performance. We find that models that can perform more computation on the same input have better performance, and are more efficient during training. Additionally, we present an architecture with an adaptive memory module that can learn to dynamically adjust its computation, and we show that it performs slightly better than its static counterpart. Finally, this study of the adaptive memory module reveal patterns comparable to human decision-making mechanisms such as reconsideration or exploring alternatives. These findings pave the way for future research in resource-aware planning architectures, promising broader applications across AI and machine learning domains. This work contributes to the evolving understanding of harnessing adaptive computation to enhance machine learning models' capabilities in complex decision-making tasks.

My contributions:

The main contributions of Sean Spinney for this project are:

- ideation and conceptualization of the project

- programming of every respect of the codebase (models, environments, training, figures, etc.)
- running and logging all experiments
- analysis and writing of the article

At the time of writing this thesis, the article is in preparation to be submitted to ICLR 2024. Co-authors (in order of appearance): Sean Spinney, Kshitij Gupta, Xutong Zhao, Janarthanan Rajendran, Irina Rish, Patricia Conrod, and Sarath Chandar.

5.1. Introduction

Planning and reasoning are integral to human decision making and have yet to be adequately incorportated into AI models. These components are required in building highly capable general AI agents that assist in scientific discovery and pushing the frontiers of human knowledge. Furthermore, reasoning in sequential decision making domains is characterized by an interaction between past and future, where information and decisions from previous steps significantly informs the present decisions while these decisions, in turn, influence future inputs and observations.

The study of decision-making and planning extends to human psychology, particularly in scenarios where task complexity requires varying degrees of cognitive effort [80].Throughout human development, engagement with the environment undergoes a series of changes, shaping one's own ability to navigate challenges and solve tasks. Early on, basic functions emerge in response to simple, repetitive tasks, forming the foundation for later complexities. As we mature, higher-order cognitive processes emerge in the brain, allowing us to handle more intricate situations and integrate conflicting information through complex deliberation. This progressive development aligns with the evolving demands of human surroundings, reflecting the adaptable nature of human cognition. Furthermore, it is clear from empirical data that human reaction times vary considerably based on task complexity [81, 16], where results have been shown that correct response times to a stimulus increase either with a greater number of stimuli or possible planning alternatives to be considered. For example, increased fixation time on a set of increasingly more complex, or contradictory, instructions to complete a task was observed using eye tracking [106]. There is a clear pattern relating task complexity in the form of information processing load and possible sequence of actions or interactions to complete a task. Notably, previous research has indicated that temporal pressure and the risk of critical irreversibility can hinder decision-making, even in situations demanding swift actions [42, 76, 82]. Human decision-making exhibits a remarkable flexibility and responsiveness to environmental cues, often prompted by a sense of urgency [87]. These influences of urgency, whether biologically driven by factors like aging or

environmentally-induced pressures, create a natural push for more efficient decision-making mechanisms, such as reflexes. Such cognitive mechanisms would be extremely useful in modern machine learning methods which are becoming larger due to scaling laws.

Prior research has addressed this context and attempted to emulate features of urgency of the human decision-making process within machine learning models. A common approach for tackling sequential decision-making challenges involves model-based reinforcement learning. In this approach, the decision-maker constructs mental simulations of potential future states, actions, or outcomes within a decision tree, using a learned transition dynamics model. However, such methods exhibit limitations in terms of adaptability and scalability. The requirement to manage state transition dynamics can lead to computational challenges, particularly in environments with a large range of states. A recent approach [31], demonstrates that an agent can meta-learn flexible planning routines by utilizing recurrent neural networks, which serve as a memory mechanism, within an end-to-end training process. This also helps agents adapt and learn their planning strategy, and avoiding tree search planning algorithms, or any algorithms reliant on discrete state planning.

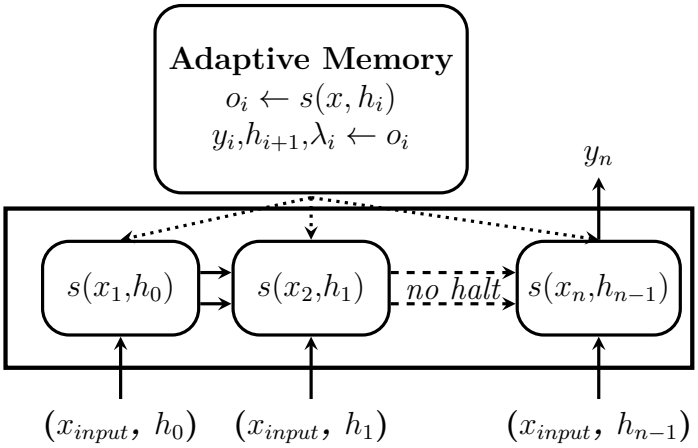


Figure 5.1. The embedded input is combined with an initialized core state h_0, which is updated with the same input at each iteration by the step function $s(x,h)$, until the halting probability (λ) forces termination. The outcome of the memory rollout ,y, is then passed to a linear layer which outputs the action logits.

Leveraging these prior advancements, we extend prior work to suit the requirements of sequential decision-making scenarios, thereby enabling us to probe the effects of adaptive computation on the planning process. Earlier research by [8, 28] has not only demonstrated the performance benefits of adaptive computation but has also highlighted instances where higher computation is needed, indicating increased task complexity. Understanding the role of these

features in task performance grants us insight into the underlying mechanisms of these models.

The aim of this work is to enhance decision time efficiency in artificial neural networks (ANNs) to emulate reflexive response intervals, while retaining the capacity to opt for additional computational deliberation when warranted. The motivation underlying the utilization of adaptive computation originates from empirical observations of human decision-makers, who consistently demonstrate context-driven biases aimed at mitigating the computational load associated with evaluating diverse alternatives in an efficient manner [16, 73]. 5.1 describes the main halting mechanism for terminating the adaptive computation in the suggested architecture.

With this goal in mind we extensively study the interplay between adaptive computation and memory and the contributions of this work are as follows:

(1) **Analysis of Memory Modules**: This investigation centers on agents equipped with single and multiple-step memory modules in the realm of sequential decision-making. This comprehensive analysis sheds light on the potential influence of memory integration on agents' planning efficiency and their capacity to make informed decisions.
(2) **Demonstration of Adaptive Computation**: We present a compelling demonstration of the adaptive computation capabilities inherent in the proposed architecture. By applying it to intricate puzzle-solving scenarios, we seek to highlight the potential of the proposed approach to amplify planning efficiency while concurrently upholding performance standards.
(3) **Exploration of Complexity-Pondering Relationship**: This research studies the interplay between information complexity within a state and the subsequent pondering steps carried out by an adaptive agent. We observe three distinct learned behaviors, namely increasing certainty, exploring alternatives, and reconsideration. These behaviors are interpreted through a psychological lens by analyzing the evolution of internal transformations within a trained AMM agent. The central hypothesis of this work posits that the adaptive agent strategically allocates computation resources based on the perceived or actual complexity of input data, as well as the remaining difficulty of the task.

In essence, the contributions of this work collectively advance the understanding of memory modules, adaptive computation, and their interplay in the context of sequential decision-making. This paper fosters a deeper appreciation for the intricate dynamics at play within adaptive agents and their implications for efficient planning.

5.2. Related Works

Recent trends have shown how simply scaling up model size, and dataset size can result in emergent capabilities like reasoning [110, 115, 72, 116, 45, 38, 54]. However, its been shown [94], these large language models are still weak at more complex reasoning tasks. Factors contributing to these limitations include low quality step-by-step reasoning data available on the internet and difficulty in collecting high quality reasoning data for complex reasoning problems. Collecting such data becomes very difficult for problems that humans themselves cannot solve, and for visual reasoning problems.

Another common approach taken towards building AI agents capable of planning has been studied under the umbrella of model-based reinforcement learning. These methods rely on learning world models, and leveraging explicit search strategies like Monte Carlo Tree Search to build AI agents that can solve long horizon planning problems [100, 102, 85, 84, 33]. These works also leverage additional decision/inference time to achieve super-human level performance in various complex planning games like Chess and Go. However, these approaches lack flexibility and adaptability, and are often difficult to scale in more complex environments. Human's on the other hand adapt their planning strategies and planning time based on the complexity of the problem and urgency around the need to arrive at a decision. Works like [23], show the benefit of other planning routines involving hierarchical planning and reverse directional planning routines.

Works like [86, 79, 30, 27] have taken a step toward flexible and adaptable planning routines by allowing agents to learn to plan. However, these still rely on having an explicit transition dynamics model. Recently [31] proposed to learn an end-end agent that meta-learns planning routines via its memory. Another recent work [9], also shows similar effects in other planning domains. However, the computation time in above approaches is fixed. These works have also shown how leveraging more computation at inference time for more difficult problems [6], can help improve the performance of the model. Other works have also shown how additional computation time can help improve performance in complex planning domains [31, 9, 41, 34]. All these works however fix the computation time and do not allow the agent to adapt the number of repetitions taken by the memory module *e.g.* LSTM with a fixed number of iterations.

5.3. Methodology

In this section we formally introduce the proposed architecture, *Adaptive Memory Model* (AMM; Figure 5.1), and the training procedure. Many of the same parts of AMM are also used for the alternative models found in the experiments section, mainly the encoder and size of memory module and policy.

5.3.1. Architecture

The models are composed of an encoder which takes an input (state or observation), followed by the recurrent memory module which processes the encoded representation and is further transformed through a linear layer (policy linear layer). A softmax transformation is applied to the output of the policy layer which probabilistically selects an action from the policy's logits. The following section reviews the specifics of each part of the model.

5.3.1.1. Encoder. The encoder is a composed of layers of CNN's with MaxPooling and Dropout between each layer. Once the input is processed by the encoder, it is either flattened for memory modules which take flat inputs *RNN, LSTM* or the three dimensional structure is retained for the DRC module. The input is a one-hot encoded transformation of the entire grid of the puzzle *i.e.* a 7x7 grid with 7 different elements (*e.g.* walls, boxes, etc.) becomes a 7x7x7 boolean matrix.

5.3.1.2. Dynamic memory module. The dynamic memory module introduces adaptability in the recurrent network by altering the number of computational steps it undertakes based on input conditions, and its available memory which is not reset at each step. AMM attains this adaptability through an end-to-end gradient descent learning framework. The core of AMM's operation is encapsulated in a step function of the form:

$$\hat{y}_n, h_{n+1}, \lambda_n = s(x, h_n)$$

where the symbol x denotes the input, while h_n represents the state, and $\hat{y}_n$ characterizes the predictive output at the n-th computational step. It is important to note that x is the same original for each iteration of the step function, as can be seen in Figure5.1 as $x_i nput$. The parameter λ_n encapsulates the likelihood of halting or terminating the ongoing process at the current step. The function s can be instantiated as a diverse array of neural network architectures, ranging from Long Short-Term Memory (LSTM) [37] and Multi-Layer Perceptron (MLP) [91] to Gated Recurrent Unit (GRU) [18] and Attention-based layers [108], thereby imparting flexibility and adaptability to the model.

The unconditioned probability associated with the act of halting at the n-th step, denoted as p_n, is derived as follows:

$$p_n = \lambda_n \prod_{j=1}^{n-1} (1 - \lambda_j)$$

This represents the probability of avoiding termination in preceding computational stages and subsequently halting at the specific step, denoted as n. During inference, the halting process is implemented via either a stochastic sampling mechanism, where the halting probability λ_n guides the decision to terminate by drawing from the Bernoulli distribution

with parameter λ_n, or simple probability threshold *i.e.* halt if $\lambda_n > 0.5$. Finally, the predictive output $\hat{y}_n$ is obtained from the halting layer.

$$\text{halt}_n = \text{Bernoulli}(\lambda_n) \quad \text{or} \quad \text{halt}_n = \lambda_n > 0.5$$

During training, predictions are gathered from each layer, initiating the computation of distinct loss values for individual layers. Subsequently, a weighted averaging strategy is used to consolidate these losses, with the weighting scheme derived from the layer-specific halting probabilities (p_n). Simultaneously, the step function is bounded by a predetermined upper limit on computational steps, as characterized by the variable N. The aggregate loss function for AMM, denoted as L, is composed as follows:

$$L = L_{Rec} + \beta L_{Reg} \tag{5.3.1}$$

$$L_{Rec} = \sum_{n=1}^{N} p_n \mathcal{L}(y, \hat{y}_n) \tag{5.3.2}$$

$$L_{Reg} = KL\left(p_n \| p_G(\lambda_p)\right) \tag{5.3.3}$$

This aggregate loss is the summation of two constituent components: the recurrent segment of the loss and the regularization term in the form of Kullback–Leibler divergence. L_{Rec} captures the aggregate effect of losses across pondering steps, with $\mathcal{L}$ representing the loss function quantifying the dissimilarity between the target value y and the predictive output $\hat{y}_n$. The regularization loss, denoted as L_{Reg}, leverages the Kullback–Leibler divergence (KL), where p_G signifies the geometric distribution parameterized by the prior λ_p. We select the geometric distribution as the probability distribution for regularizing the λ parameter for halting, where λ_p is the prior probability of stopping at any given pondering step. The geometric distribution takes the following form:

$$Pr_{p_G(\lambda_p)}(X = k) = (1 - \lambda_p)^k \lambda_p$$

The regularization loss biases the network in performing around $\frac{1}{\lambda_p}$ steps and incentivizes non-zero probabilities for all pondering steps. In essence, it encourages the network to explore different possibilities and options.

5.3.2. Training

The training approach described here employs behavior cloning [105], wherein the agent learns from expert trajectories by predicting subsequent actions in the sequence. Specifically, we employ a cross-entropy loss function that measures the discrepancy between the expert's

actions and the model's predictions.

For the static model, the loss is calculated without pondering, following the equation:

$$\text{Loss without pondering} = -\sum_{i=1}^{N} \text{expert action}_i \cdot \log(\text{model prediction}_i)$$

where N is the number of examples (number of episodes and timepoints) in a batch. In the case of the adaptive model, the loss that considers a weighted average across the various pondering steps can be expressed using cross-entropy loss as:

$$\text{Loss with pondering} = -\sum_{j=1}^{K} w_j \cdot \sum_{i=1}^{N_j} \text{expert action}_i \cdot \log(\text{model prediction}_{ij})$$

where N_j is the number of expert actions at pondering step j, and w_j is the weight assigned to pondering step j.

To create the expert trajectories, we resort to utilizing the deterministic solver A* [35], which generates solutions in the form of action sequences. For instance, the sequence "lLuURrD" corresponds to the actions: move left, push left, move up, push up, push right, move right, and push down. Employing the initial state of the puzzle and the derived action sequence, we simulate the solution rollout in the environment. This process involves executing the actions on the initial state and recording the corresponding states, rewards, actions, and a "done" flag. Consequently, a trajectory is represented as a tensor of length equal to the number of steps needed to solve the puzzle. Each element within this tensor comprises a sub-tensor containing the state, reward, action, and "done" flag. The splitting of training and validation subsets of the datasets generated is explained for each experiment demonstrated in section 6.

For optimization, we use the AdamW optimizer with weight decay. The proposed training strategy involves a scheduler that initially executes the LRlambda scheme for a predefined number of warmup steps, followed by the cosine annealing algorithm for the remaining training iterations. The model undergoes training for a single epoch only. The details of the hyperparameter values can be found in the appendix (Tables 5.5,5.6).

5.3.3. Evaluation

We evaluate the following metrics by performing the experiments explained in section 5.5:

Performance : $The final mean return and fraction solved serve as key indicators to evaluate the efficacy of th$

Training efficiency: Additionally, we investigate the impact on the length of solutions achieved by varying the number of repetitions permitted in the adaptive memory module (Table 5.3).

Adaptability: Through an examination of the mean, standard deviation, and range of ponder steps taken by a trained model to solve levels against the number of repetitions allowed for adaptive computation, we ascertain the model's capacity to adapt its computation based on input. We also explore the extent to which increasing repetitions influence these statistical measures (Table 5.4).

Finally, we produce Figures (**??**) that highlight certain behaviors observed when investigating the relationship between input content, the number of pondering steps, and the evolution of the policy entropy across pondering steps. We observe certain distinct patterns which are described in detail in Section 5.6.3.

5.4. Environments

We have chosen the subsequent pair of environments due to the recognition that both necessitate a degree of planning to determine a sequence of actions that precludes future complications arising from a lack of foresight. For each game, we will outline the specific factors that determine its level of difficulty.

5.4.1. Sokoban

This challenging puzzle is presented as a two dimensional grid where an agent must push boxes over targets in under a fixed amount of steps [86, 31, 17]. When increasing the number of boxes, it becomes more important to become aware of potential irreversible moves that could be avoided with better planning. We use the same methodology as [17] to generate levels which are then split into train and test sets. The following represent the environments used in the experiment section:

There are a total of 8 possible actions: move or push in the 4 possible directions, and 7 possible elements in the state: walls, spaces, targets for boxes, boxes on target, and boxes not on target. We decided to omit the inclusion of the no-op action which has the effect that the agent performs no operation, while the environment may or may not change (if there are

Dataset Name	Number of Samples	Description
Sokoban-small-v0-train	2.1M	Training data; 7x7 grid, 2 boxes
Sokoban-small-v0-validation	1000	Validation in-distribution; 7x7 grid, 2 boxes
Sokoban-small-v1-validation	1000	Validation out-of-distribution; 7x7 grid, 3 boxes

Table 5.1. Summary of the datasets.

no other moving objects, it will remain the same input). Also referred to as "stay" actions, many variations on the same idea have been implemented with the goal being to include stochasticity which may help learn a better performing policy, with aperiodic sequence of actions [36]. It is important to note that the internal state of the memory module will not be reinitialized. This can be seen as a way to grant the ability to ponder for any agent by allowing it to do nothing and watch its environment, similar to the no-op action. The reasoning for this is that the AMM already includes the possibility of adaptive computation for an input, and does not entangle the concept of thinking and action by including no-op in the space of possible actions.

5.5. Experiments

The selected model is composed of an encoder which takes an input (state or observation), followed by the recurrent memory module which takes in the encoded representation and is further processed through a linear layer (policy linear layer). A softmax transformation is applied to the output of the policy layer which probabilistically selects an action from the policy's logits. It is important to note that the memory is not reset after each step in the environment (static and adaptive), enabling the possibility for the agent to formulate longer term planning.

Each experiment is engineered in a way to answer one or more hypotheses about dynamic planning's abilities. The following experiments examine the following:

(1) A static memory module with more repetitions obtains the greatest final mean return and fraction solved. We test this hypothesis in the single epoch setting.
(2) Similarly, more repetitions leads to more efficient training *i.e.* less steps/interactions with the environment are needed to obtain the same performance
(3) A dynamic memory module should perform at least as well as the same version but static
(4) Simpler (i.e. objective and/or subjective complexity) inputs require a smaller number of repetitions and is reflected by the adaptive model
(5) How does the memory module output affect the decision-making process? In other words, how is the policy's action probability transformed with more repetitions?

5.5.1. Experiment 1: Impact of Repetitions in Memory Module

In this section, we study the impact of augmenting the number of repetitions within the memory module on the average return observed across both training and validation datasets. The underlying hypothesis posits that an increased number of repetitions facilitate the encoding of long-term interactions which should improve planning capabilities. While this augmentation may not prove advantageous for every input scenario, the expectation is that it would, at the very least, not yield inferior performance compared to an identical model employing only a single repetition.

Furthermore, an additional dimension of interest pertains to sample efficiency during training. Specifically, we probe whether a higher count of repetitions expedites the learning process. To gauge this, we examine the average episode return after a fixed number of steps undertaken by the agent.

Four models with increasing number of repetitions are trained on a subset of Sokoban-small-v0 levels and validated on a different subset of Sokoban-small-v0 as well as Sokoban-small-v1 levels to test generalization to an environment with one more box. The models are compared on the basis of mean episode return and percent of fraction solved, as well as sample efficiency during training.

5.5.2. Experiment 2: Comparing Dynamic and Static Planning

This section tests the hypothesis that the dynamic planning model performs on par with its static counterpart. We now allow the model to learn when to stop pondering and compare the same metrics observed in Experiment 1. The hypothesis explored here is whether or not the dynamic pondering model can perform as well or better than a static model. If this can be shown, then a model that acts dynamically will have a shorter inference time making it more efficient after training. The application of the dynamic module to the training process is another question not explored in this work.

5.5.3. Experiment 3: Relationship Between Input Complexity and Pondering Steps

The concept of increased computation is inherently intertwined with the notions of objective and experiential complexity. The aim of this work is to explore the interplay between input complexity and the number of pondering steps.

Objective Complexity refers to the inherent intricacy and fundamental level of difficulty that are intrinsic to a given problem or the process of decision-making, which can be objectively measured by factors such as the number of variables involved, the depth of analysis required, and the computational resources demanded. In dynamic planning, ANNs strive to tackle objective complexity by adapting their computational efforts based on the intricacy of the task at hand. This adaptability allows the network to allocate more resources for complex scenarios, leading to more accurate and thoughtful decisions.

Experiential Complexity, on the other hand, takes into account the subjectivity of decision-making, considering the cognitive load and mental effort experienced by an individual or an artificial system when processing information and making choices. In dynamic planning, ANNs aim to replicate experiential complexity by adjusting their decision-making strategies to mirror the varying cognitive demands posed by different stimuli. This enables ANNs to simulate a more human-like decision process, where certain decisions may feel effortless while others require deeper contemplation.

The hypothesis put forward posits that inputs with higher complexity necessitate a greater amount of computational resources. To explore this notion, we will employ established definitions of both objective and subjective complexity. By doing so, we aim to establish a correlation between the difficulty of inputs and the corresponding number of pondering steps employed by the adaptive agent. Furthermore, we will selectively analyze specific levels, focusing on visualizing the count of pondering steps taken per input. This exploration aims to provide valuable insights into the intricate connection between acquired pondering behaviors and the complexity of inputs. Importantly, the approach described in this work leverages the information available to the model during the pondering process, contributing to a comprehensive understanding of the dynamics at play.

5.6. Results

5.6.1. Experiment 1

In the first experiment, we sought to gauge the effects of augmenting the number of repetitions within the memory module on both training and validation datasets. The primary focus was to ascertain if an increased number of repetitions aids in encoding long-term interactions, thereby enhancing planning capabilities. The main hypothesis posited that such an augmentation would, at the very least, not yield inferior performance compared to models utilizing a single repetition.

Validation Performance: The outcomes presented in Table 5.2 manifest a discernible enhancement in the fraction of puzzles solved by models with an augmented repetition count. Notably, as the repetition count increased from 1, both validation datasets displayed an upward trajectory in the fraction of puzzles successfully solved. This observed trend substantiates the underlying hypothesis that increased repetitions contribute to improved planning efficacy, particularly when addressing intricate decision sequences.

Static Models						
Number of Repetitions	Sokoban-small-v0			Sokoban-small-v1		
	6.4×10^4	1.28×10^6	2.4×10^6	6.4×10^4	1.28×10^6	2.4×10^6
1	60.69 (5.7)	86.1 (2.8)	90.01 (3.1)	32.00 (4.0)	57.76 (6.9)	62.3 (5.2)
3	67.59 (3.0)	90.06 (2.4)	**92.37** (2.9)	35.30 (2.5)	62.82 (4.6)	66.19 (4.3)
9	72.75 (5.0)	**91.77** (2.3)	92.36 (2.7)	42.71 (4.8)	63.97 (7.0)	67.30 (5.6)
25	**76.30** (4.5)	91.47 (3.4)	92.10 (2.4)	**47.97** (6.5)	65.83 (5.8)	**69.39** (6.0)
50	75.36 (4.6)	90.51 (2.8)	92.17 (0.1)	47.55 (5.9)	**66.73** (5.6)	66.31 (0.1)
Adaptive Models						
Number of Repetitions	Sokoban-small-v0			Sokoban-small-v1		
	6.4×10^4	1.28×10^6	2.4×10^6	6.4×10^4	1.28×10^6	2.4×10^6
3	63.86 (1.2)	88.67 (1.2)	92.07 (1.1)	28.51 (1.0)	61.90 (1.4)	65.74 (0.9)
9	**76.56** (2.2)	**91.07** (2.3)	**93.70** (2.2)	**45.05** (2.6)	**68.22** (1.9)	**71.03** (3.2)
25	61.89 (10.8)	90.33 (3.6)	92.86 (2.0)	27.20 (7.1)	65.21 (4.4)	68.89 (3.0)

Table 5.2. Percentage of mean fraction solved at different steps of training for static versus dynamic models. Both dataset tested are from a subset of unseen data from either in-distribution (Sokoban-small-v0), or out-of-distribution (Sokoban-small-v1).

Sample Efficiency: The assessment of sample efficiency during training revealed that an increased number of repetitions indeed expedited the learning process. Models with more repetitions displayed faster improvement, reaching comparable performance levels with fewer training steps, as evidenced by higher average fraction solved after a fixed number of steps in Table5.2. We also noted in Table 5.3 that models with a higher number of repetitions solved levels with a lower average number of steps, as well as a smaller maximal number of steps taken suggesting they may be more effective planners.

5.6.2. Experiment 2

The second experiment focused on comparing the performance of dynamic and static planning models. Here, we explored the hypothesis that a dynamic planning model could achieve similar or better results than its static counterpart. The dynamic model was equipped with the ability to learn when to halt pondering, potentially leading to a more efficient inference process.

Static Models						
Repetitions	Sokoban-small-v0			Sokoban-small-v1		
	Mean	Std	Range	Mean	Std	Range
1	12.14	6.91	61.00	17.65	9.76	89.00
3	11.99	6.39	37.00	17.84	9.57	77.00
9	11.95	6.30	39.00	16.84	8.29	46.00
25	12.16	6.75	44.00	16.89	8.46	62.00
50	12.34	7.12	41.00	17.15	9.01	70.00
Adaptive Models						
Repetitions	Sokoban-small-v0			Sokoban-small-v1		
	Mean	Std	Range	Mean	Std	Range
3	12.15	7.10	86.00	16.46	8.08	57.00
9	11.93	6.38	41.00	17.18	9.09	67.00
25	11.78	6.46	61.00	16.18	7.85	75.00

Table 5.3. Length of solved episodes statistics for both sets of validation (in-distribution and out-of-distribution). This represents the number of steps taken to complete a level.

Performance Parity: The results in Table5.2 reveal that the adaptive planning model exhibited improved performance in general, when compared with its static counterpart. This observation holds true for both in-distribution (Sokoban-small-v0) and out-of-distribution (Sokoban-small-v1) settings, suggesting that the adaptive model's flexible decision-making does not compromise its planning capabilities, and in most cases it scores higher.

Inference Efficiency: Moreover, the comparable performance of the adaptive planning model bears favorable implications for inference efficiency. Indeed, Table5.4 show the memory iteration statistics for an AMM. It is noteworthy that, across varying repetition numbers, the trained AMM adeptly capitalizes on its adaptive computational resources, often aligning closely with its maximum iteration count. For example, in Table5.4 we see that each AMM's mean halting step is off by its max value by about 1, but with a large relative standard deviation, and a range that implies each AMM can and does halt at all possible steps. This inclination towards comprehensive pondering underscores the model's capacity to thoroughly deliberate when necessary, yet also showcases its ability to swiftly reach decisions by terminating after a few initial steps, thus presenting a clear avenue for computational savings.

5.6.3. Experiment 3

In the third experiment, we delved into the relationship between input and the number of pondering steps taken. One of the hypothesis covered in this work is centered on the notion that more complex inputs would require a higher degree of computation. To this end, we create visualizations to observe how the adaptive memory module of a trained agent impacts

Repetitions	Sokoban-small-v0			Sokoban-small-v1		
	Mean	Std	Range	Mean	Std	Range
3	1.12	0.93	2.0 0	1.19	0.94	2.00
9	6.76	2.66	8.00	7.14	2.31	8.00
25	22.50	5.44	24.00	22.7	5.13	24.00

Table 5.4. Pondering statistics across validation datasets. Shown is the result of taking the mean, standard deviation, and range of the number of ponder steps taken by a trained AMM over 1000 levels for each environment.

its policy.

Complexity-Steps Relationship: Harder levels found in the Sokoban-small-v1 dataset, characterized by higher objective complexity due to having an extra box to place, exhibited a greater number of pondering steps on average, as can be seen in Table5.4. Moreover, we investigate the connection between early-episode pondering and the stage nearing level completion by quantifying the correlation between the current step within an episode and the agent's frequency of computational iterations (max repetition = 9). The subsequent analysis uncovers a statistically significant negative correlation ($r = -0.31, p = 6.14e^{-16}$) between the step number within an episode and the duration of contemplation, when the agent chose to adapt its computation (number of repetition smaller than the max). This indicates that, as the episode unfolds, the corresponding halting times exhibit a marginal reduction. This trend suggests that there is a perceived decrease in the overall complexity level by the agent, which decides to deliberate more quickly when nearing the end of an episode.

Visualizing Pondering Steps: Visualizations of pondering steps taken per input are used as visual support for highlighting certain observed behaviors from a trained AMM agent with 9 maximum possible repetitions (Figures 5.35.45.5). By comparing the input with how this information is used throughout pondering, we find further evidence of the interplay between input complexity and adaptive planning in the form of learned behaviours. The observed behaviours are the following:

(1) **Increasing certainty**: the action with the highest probability of being selected does not change across pondering steps, however the probability increases with every step. This is the most commonly observed use case for pondering by the trained model.
(2) **Exploring alternatives**: one or more other actions are more seriously considered (increase in probability during deliberation), but ultimately the same action is chosen.
(3) **Reconsideration**: the action with the highest probability after the first pondering step is not the same one at the last step taken. This is the rarest observed behavior found

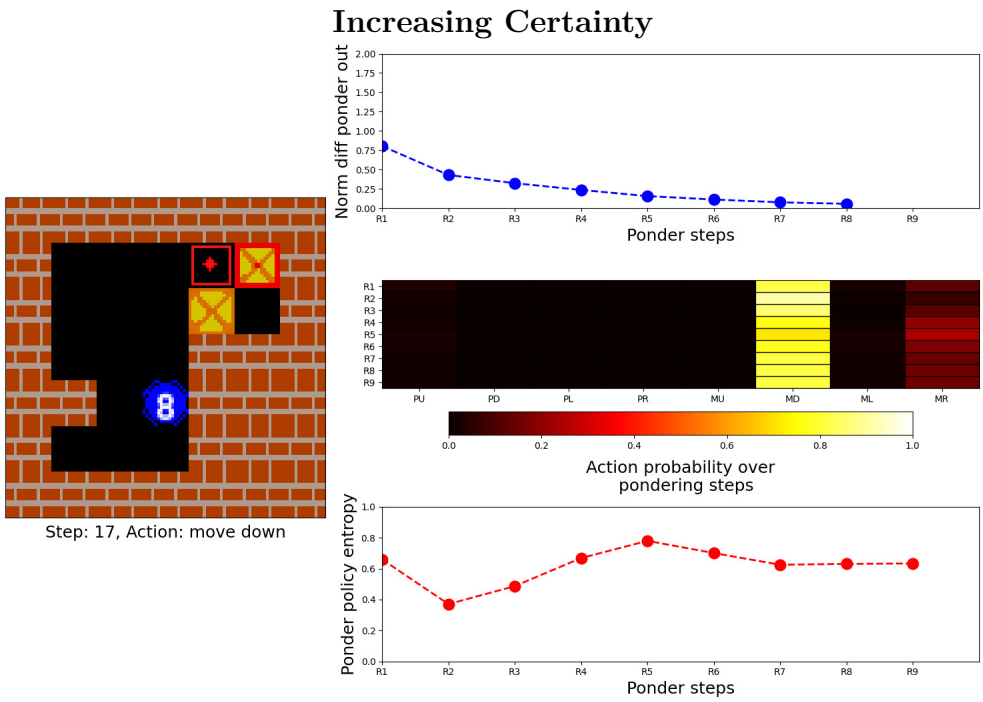


Figure 5.2. Visualization of a trained AMM agent. Left: the game state with the number of steps pondered at that state displayed on the agent (blue); the current step and the resulting action chosen after pondering is displayed below the image. Right top: the difference in norm of the output between pondering steps. Right middle: the action probability distribution across actions and pondering steps. Right bottom: the entropy of the policy across ponder steps.

The first observed behaviour is analogous to the case of thinking longer to confirm an initial plan. Indeed, we observe that a trained AMM agent will sometimes choose to ponder for longer with the effect of steadily increasing the probability of its initial plan (the action chosen). Figure 5.2 shows on such example, and more examples can be found with different repetition lengths in the appendix. We note that the policy's entropy follows a downward trend, reinforcing the hypothesis that the agent ponders for longer to decrease its uncertainty.

Another consistently observed behaviour is related to deliberation or the exploration of alternatives, Figure 5.3. This occurs when throughout pondering the agent's policy temporarily increases the probability of selecting one or other actions, but the final decision remains the same. This type of behaviour is easily compared to forms of exploration, where the policy's entropy is adjusted for exploration. In fact, we can observe this fluctuation in the agent's policy entropy, which shows a temporary increase when the agent begins to more seriously consider an alternative and then begins to fall again.

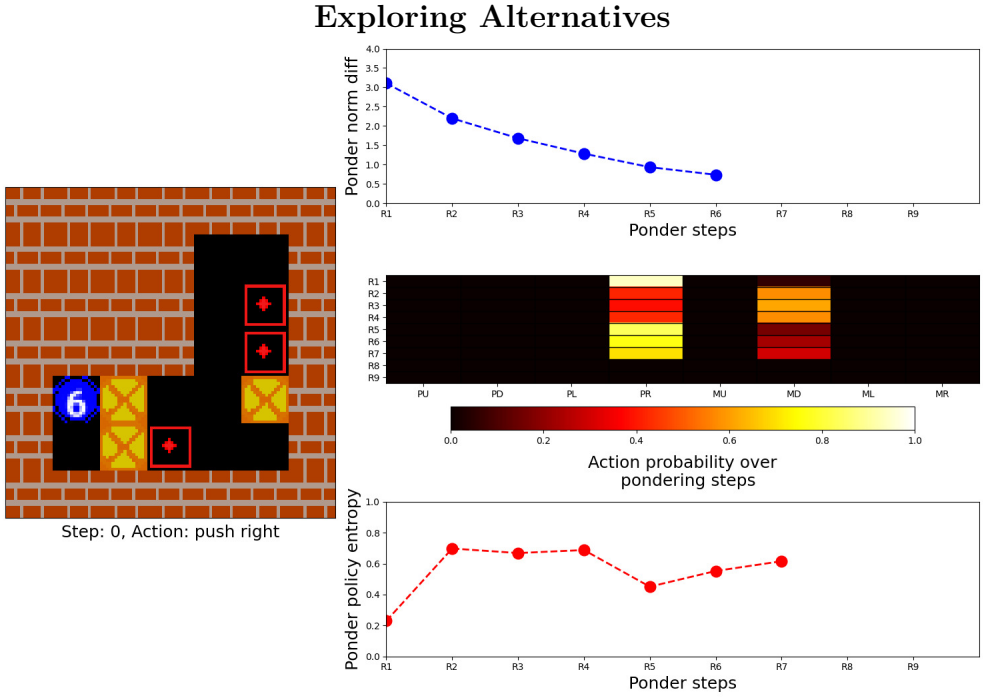


Figure 5.3. Visualization of a trained AMM agent. Left: the game state with the number of steps pondered at that state displayed on the agent (blue). Right top: the difference in norm of the output between pondering steps. Right middle: the action probability distribution across actions and pondering steps. Right bottom: the entropy of the policy across ponder steps.

Lastly, the phenomenon of reconsideration manifests when an agent's initial and final choices of action differ across the sequence of pondering steps (Figure 5.4; appendix for more examples). This behavior is characterized by fluctuating levels of certainty after the initial pondering step, followed by a gradual decrement in the probability assigned to the initial action. Subsequently, one or more alternative actions observe a surge in their probabilities, culminating in the selection of a single action among the candidates.

5.7. Discussion

The series of experiments detailed in the preceding section highlight the compelling potential of adaptive computation in bolstering planning efficiency. By learning to dynamically adjust the duration of deliberation according to the input, the proposed models simulate a cognitive adaptability characterized by learned behaviors that closely resemble the decision-making processes observed in humans. The integration of memory and reusability mechanisms further empowers models to dynamically allocate computational resources where required, ultimately resulting in more efficient planning strategies.

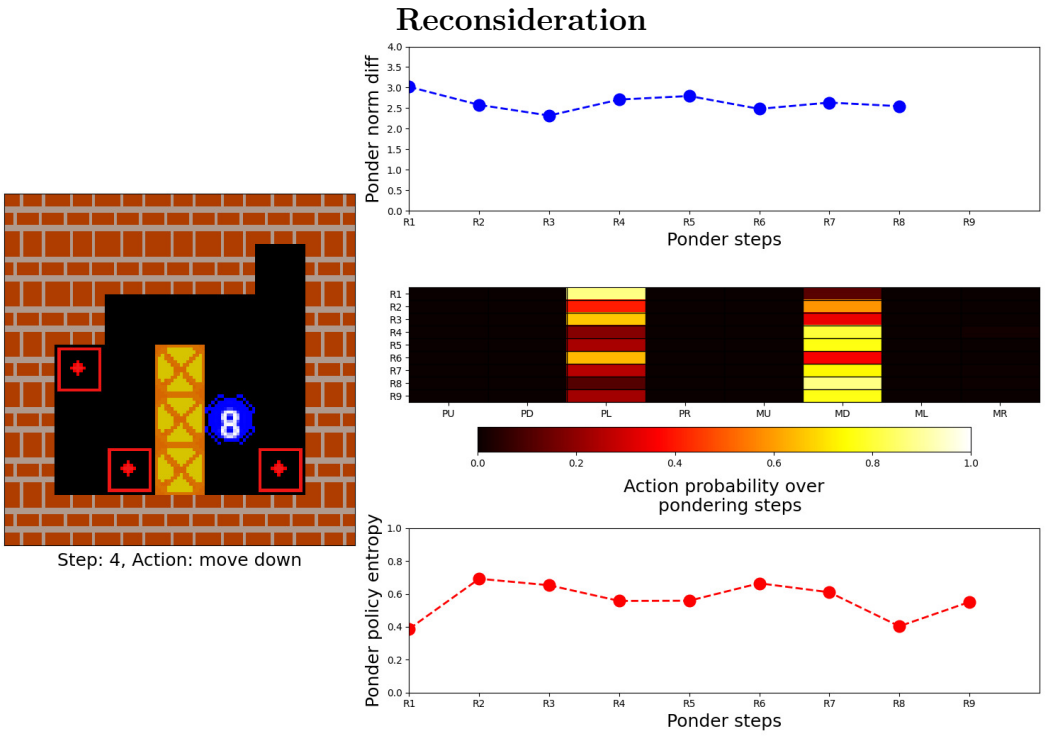


Figure 5.4. Visualization of a trained AMM agent. Left: the game state with the number of steps pondered at that state displayed on the agent (blue). Right top: the difference in norm of the output between pondering steps. Right middle: the action probability distribution across actions and pondering steps. Right bottom: the entropy of the policy across ponder steps.

The AMM's ability to make quicker decisions by deliberating for less iterations (Table 5.4), combined with the empirical observation that more repetitions lead to shorter solutions on average (Table 5.3), translates into a more efficient inference process, potentially leading to shorter decision-making times in practical applications. Moreover, the adaptive memory module (AMM) exhibits an elevation in the average pondering steps when subjected to the more challenging dataset (Sokoban-small-v1; Table 5.4). This observation hints at a potential connection between effectively solving more intricate problems and engaging in prolonged contemplation. Furthermore, this investigation reveals a pattern where trained AMMs tend to deliberate for shorter intervals as they approach the conclusion of an episode, characterized by a negative correlation between step number in an episode and thinking time. This behavior may be an indication that the model is actively assessing the remaining complexity of the episode and adjusting its computational effort accordingly.

In the course of the second experiment, we noted a superior performance of the adaptive version compared to its static counterpart for most of the models under investigation. This discrepancy in performance could potentially stem from differences in their respective loss

functions. Unlike static models, AMMs produce an output at each iteration of the memory module, contributing to the optimization of the loss function. The adaptive module's loss, characterized by the weighted sum across pondering steps, could introduce a stabilizing effect during the optimization process.

The manifestation of the identified learned behaviors (increasing certainty, exploration of alternatives, and reconsideration) in the third experiment, prompts us to inquire whether these behaviors are intrinsic to the task on which the Adaptive Memory Module is trained, or if they arise as emergent capabilities in parallel with scaling laws. Moreover, a fundamental question arises: can we deliberately induce the emergence of additional, potentially novel planning-oriented behaviors by scaling the models, expanding the dataset, and augmenting computational resources?

Several inherent limitations warrant acknowledgment in this study. Primarily, this investigation has been confined to a specific range of environments, prompting the need for expanded explorations across a wider gamut of scenarios to ensure the generalizability of these findings. Moreover, while this study makes a notable contribution to the expanding body of research supporting the effectiveness of adaptive computation, a more comprehensive evaluation could result from intricate comparisons involving alternative architectures that incorporate adaptive memory modules.The analysis of the relationship between input complexity and pondering time was primarily grounded in objective measures of difficulty drawn from the dataset, rather than a metric that establishes a connection between current input information and deliberation time. Future endeavors delving deeper into this relationship would be instrumental in designing new algorithms with a heightened difficulty threshold. Lastly, we did not study the effects of varying the prior on the halting probability distribution, an aspect that holds potential for influencing training dynamics towards either swift decision-making or more thorough contemplation.

We hope that the insights gleaned from these results will serve as a catalyst for more profound and expansive analyses of adaptive computation's potential. Furthermore, the implications of this research extend beyond academic curiosity, potentially influencing the evolution of the next generation of artificial intelligence. Particularly pertinent is the role that efficient adaptive computation could play in the training and deployment of progressively larger models. Not only could this significantly mitigate the operational costs associated with these models, but it could also pave the way for more resource-efficient AI systems that continue to deliver comparable performance through refined computational adaptation strategies.

5.8. Appendix

5.8.1. Additional visualizations

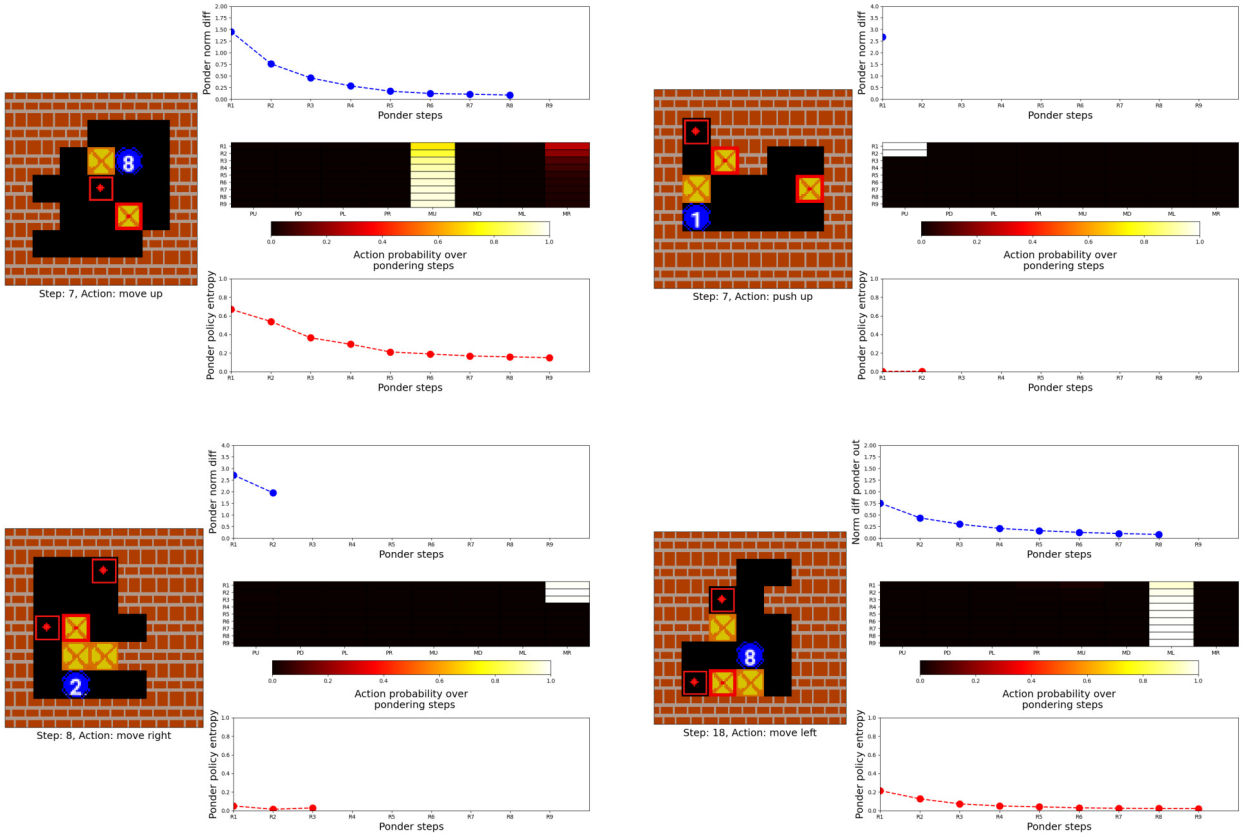


Figure 5.5. Increasing certainty

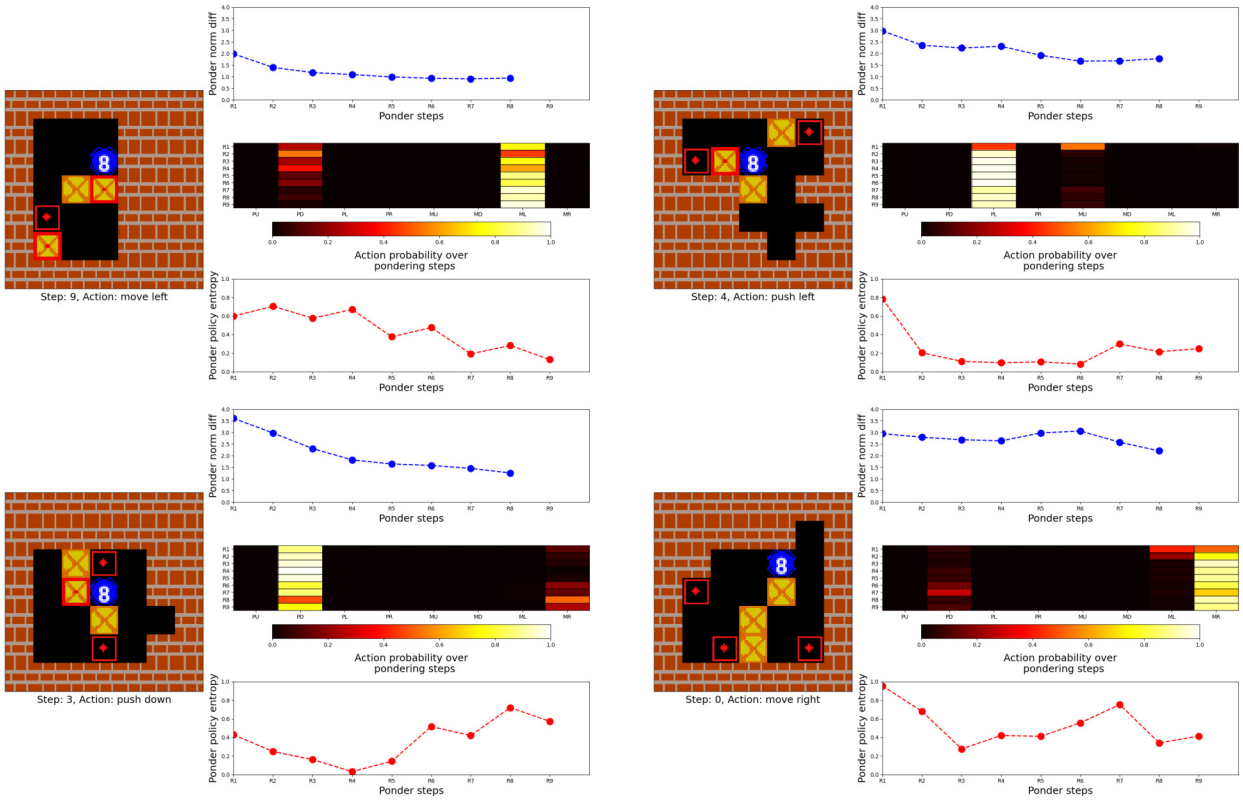


Figure 5.6. Exploring alternatives

Hyperparameter	Value
Memory Architecture	DRC(X,Y)
Learning Rate	0.01
Batch Size (number of episodes)	64
Epochs	1
Optimizer	AdamW
Weight Decay	1×10^{-4}
Hidden Units	128
Dropout Rate	0.1
Activation Function	ReLU
Learning Rate Schedule	LRlambda for 6.4×10^4 steps then cosine annealing
Gradient Clipping	1.0
Embedding Size	50
Recurrent Units	64
Initialization	He Normal
Regularization	None
Total Number of Parameters	1297097

Table 5.5. Hyperparameters for static models.

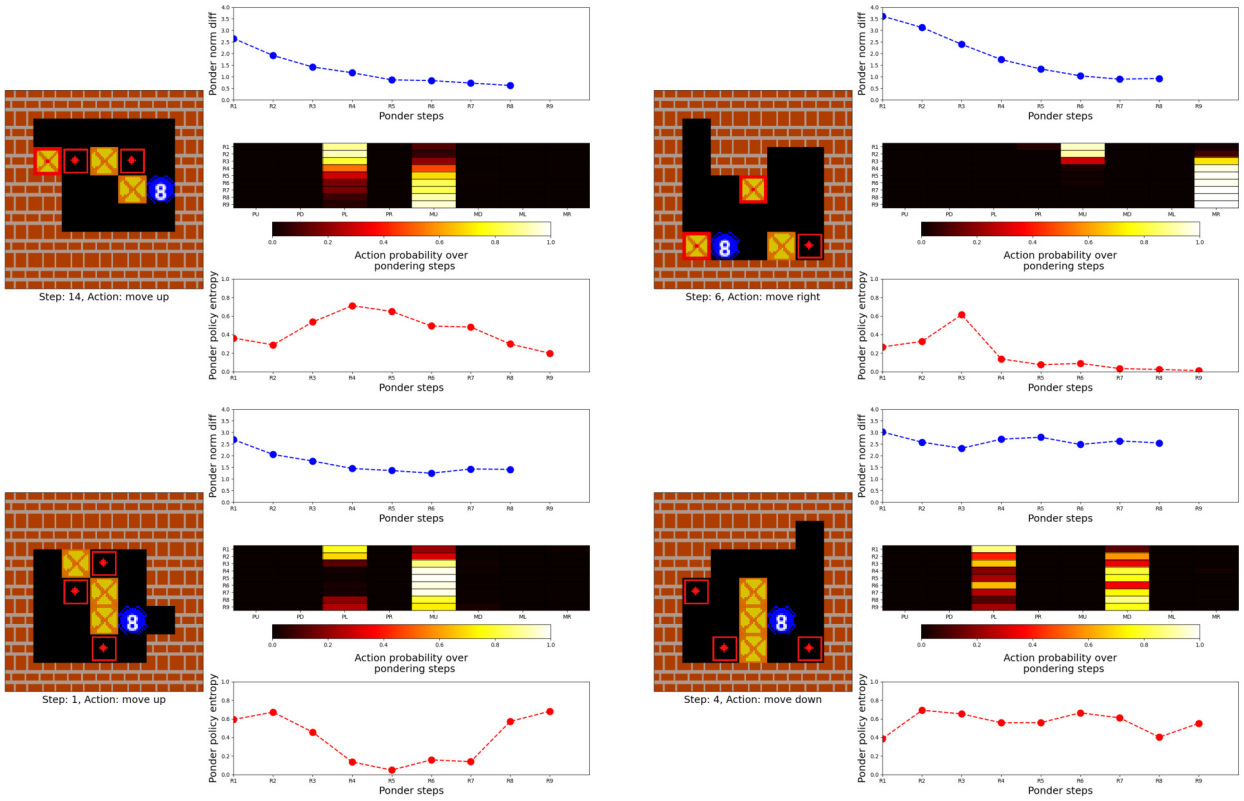


Figure 5.7. Reconsideration

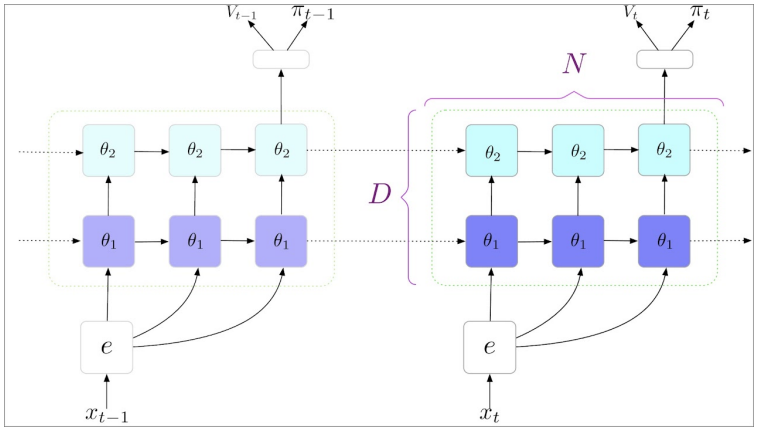


Figure 5.8. Deep Repeated ConvLSTM (DRC) architecture. Source [31]

Hyperparameter	**Value**
Memory Architecture	DRC(X,Y)
Learning Rate	0.01
Batch Size (number of episodes)	64
Epochs	1
Optimizer	AdamW
Weight Decay	1×10^{-4}
Hidden Units	128
Dropout Rate	0.1
Activation Function	ReLU
Learning Rate Schedule	LRlambda for 6.4×10^4 steps then cosine annealing
Gradient Clipping	1.0
Embedding Size	50
Recurrent Units	64
Initialization	He Normal
Regularization	Kullback-Liebler for λ
Total Number of Parameters	1306314

Table 5.6. Hyperparameters for adaptive models.

Chapter 6

Conclusion

In the pursuit of advancing our knowledge of complex systems like humans brains or large artificial models, this thesis sought out to explore how the tools of neuroscience and psychology could help in discovering new applications in deep learning or developing a greater understanding of the challenges involved in applying deep learning to complex brain data. Across the trio of articles presented within this thesis, the convergence of deep learning, neuroscience, and cognitive psychology is evident. The first article experimented with mixing traditional and more recent work in regularization. This is a significant challenge in studying the brain, because the data is high-dimensional (*e.g.* one brain scan has more than 100 000 voxels), and noisy due to variations across many variables which are difficult to control for, such as variations in the magnetic field during scanning, in the pre-processing of the raw data and many other elements of the methodology that accompanies MRI studies. It was found that the proposed model performed better under certain conditions *i.e.* a large number of groups compared to the dimensionality of the data on synthetic data, which suggests how to improve its performance on brain MRI data; by increasing the examples of brain scans from different groups of interests *e.g.* alzheimer patients, depression, anxiety, and other neuropsychopathologies which show promise in being diagnosed from brain scans.

Improving models to study the brain has clear implications, as previously mentioned, but in turn implementing the knowledge gleaned from studying the brain is a promising direction in developing artificial intelligence. The brain shows incredible efficiency and learning capabilities, and the biological neural networks studied have demonstrated high levels of complex interactions and mechanisms to promote learning. In this view, trying to recreate the underlying learning process brings us closer to a deeper understanding of our own biology, but possibly more efficient and smarter AI. The purpose of the second article seeks to add to the growing field of biologically plausible learning algorithms, another advancement which shows state-of-the-art improvements on benchmarks. This work demonstrates how

the proposed method *L-DRL* closes the gap the traditional based on backpropagation by carefully designing a principled way to build backprop-like targets. The alignement of the feedforward and feedback weights is shown to be crucial, and *L-DRL* demonstrates greater efficiency in ensuring that the Jacobian of the feedback pathway matches the transpose of the Jacobian of the associated feedforward pathway. This improvement closes the gap with traditional backprogation, and highlights the mapping from theory to experimental observation. Unfortunately, our method is up to 30 times slower than backpropagation, due to the amount of iterations per layer in order to properly align the weights. The good news is this issue could be resolved by better parallelism, or optimized hardware and would be a promising direction for future work. Many algorithms today are prevalent because of technological advancements on the hardware side.

The efficiency with which humans learn to explore the world, and achieve amazing feats in new situations remains unmatched by biology and artificial intelligence. One remarkable ability we have is how long we can ponder over difficult problems, using writing to more easily return to a previous line of thinking even, thereby greatly enhancing the window of reflection on any given topic. The last article explores the integration of such adaptive computation to artificial models applied to sequential decision-making tasks, in the form of puzzles. Three experiments routed in the sequential decision-making domain seek to establish that first, extended deliberation *longer thinking time* improves performance. We find that allowing the models to use a larger deliberation window (more repetitions in the memory module), directly contributed to more efficient learning. This was observed in the form of higher percentage of fraction solved for less data trained on. Next, we showed that using a simple mechanism to enable adaptive computation, you can train a model which performs on par with its static counterpart, but learns to deliberate for shorter times. Finally, through a set of detailed visualizations, we established three forms of patterns observed in the evolution of the output of the memory module over pondering steps: increasing certainty, exploring alternatives, and reconsideration. By integrating adaptive computation principles in investigating memory's role in planning and reasoning within sequential decision-making domains, these experiments showcase the value and potential cost benefits to training models with adaptive capabilities. Future work which commits to testing adaptive computation over a very large number of domains, and across architectures, would help cement the validity of the empirical observations made in this work. Furthermore, studying different formulations of the architecture responsible for the halting mechanism could lead to the discovery of new and improved models.

While each article tackled distinct facets of the convergence between neuroscience, cognitive psychology, and deep learning, they collectively underscore the promise and complexities inherent in this interdisciplinary field. As we navigate this path, further investigations across

diverse scenarios and rigorous comparisons with alternative architectures are essential to comprehensively assess the efficacy of the proposed approaches. By delving into the biological foundations of learning, integrating adaptive computation and other cognitive mechanisms, and unveiling novel learning strategies, this thesis has contributed valuable insights to the ongoing exploration of deep learning's potential. These findings raise thought-provoking questions and open avenues for future research, fostering a deeper understanding of the intricate interplay between neuroscience, cognitive psychology, and artificial intelligence.

Références bibliographiques

[1] Kartik Ahuja et al. “Empirical or Invariant Risk Minimization? A Sample Complexity Perspective”. In: (2010), pp. 1–10.

[2] Mohamed Akrout et al. “Deep Learning without Weight Transport”. In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 976–984.

[3] Martin Arjovsky. “Out of Distribution Generalization in Machine Learning”. PhD thesis. 2020, pp. 1–120. ISBN: 9788578110796. arXiv: `arXiv:1011.1669v3`.

[4] Martin Arjovsky et al. *Invariant Risk Minimization*. 2020. arXiv: `1907.02893 [stat.ML]`.

[5] Kai Arulkumaran et al. “A brief survey of deep reinforcement learning”. In: *arXiv preprint arXiv:1708.05866* (2017).

[6] Robert J. N. Baldock, Hartmut Maennel, and Behnam Neyshabur. *Deep Learning Through the Lens of Example Difficulty*. 2021. arXiv: `2106.09647 [cs.LG]`.

[7] Dana H Ballard. “Modular learning in neural networks.” In: *Aaai*. Vol. 647. 1987, pp. 279–284.

[8] Andrea Banino, Jan Balaguer, and Charles Blundell. “PonderNet: Learning to Ponder”. In: *ICML* (2021).

[9] Arpit Bansal et al. *End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking*. 2022. arXiv: `2202.05826 [cs.LG]`.

[10] Sergey Bartunov et al. “Assessing the scalability of biologically-motivated deep learning algorithms and architectures”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 9390–9400.

[11] Yoshua Bengio. “Deriving differential target propagation from iterating approximate inverses”. In: *arXiv preprint arXiv:2007.15139* (2020).

[12] Yoshua Bengio. “How auto-encoders could provide credit assignment in deep networks via target propagation”. In: *arXiv preprint arXiv:1407.7906* (2014).

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[14] Yoshua Bengio et al. “Towards biologically plausible deep learning”. In: *arXiv preprint arXiv:1502.04156* (2015).

[15] Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[16] Dominic Bläsing and Manfred Bornewasser. “Influence of increasing task complexity and use of informational assistance systems on mental workload”. In: *Brain Sciences* 11.1 (2021), p. 102.

[17] Adi Botea. “Using Abstraction for Heuristic Search and Planning”. In: *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 326–327. ISBN: 3540439412.

[18] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Conference on Empirical Methods in Natural Language Processing* (2014). DOI: 10.3115/v1/D14-1179.

[19] Anna Choromanska et al. “Beyond backprop: Online alternating minimization with auxiliary variables”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1193–1202.

[20] Brian Crafton et al. “Direct feedback alignment with sparse connections for local learning”. In: *Frontiers in neuroscience* 13 (2019), p. 525.

[21] Dominique Debanne. “Information processing in the axon”. In: *Nature Reviews Neuroscience* 5.4 (2004), pp. 304–316.

[22] Dominique Debanne et al. “Axon physiology”. In: *Physiological reviews* 91.2 (2011), pp. 555–602.

[23] Ashley D. Edwards, Laura Downs, and James C. Davidson. “Forward-Backward Reinforcement Learning”. In: *arXiv preprint arXiv: 1803.10227* (2018).

[24] Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Vol. 7. FA Perthes, 1877.

[25] Wulfram Gerstner et al. “A neuronal learning rule for sub-millisecond temporal coding”. In: *Nature* 383 (1996), pp. 76–78. DOI: 10.1038/383076a0.

[26] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT Press, 2016.

[27] Sven Gowal et al. “On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models”. In: *arXiv preprint arXiv: 1810.12715* (2018).

[28] Alex Graves. “Adaptive Computation Time for Recurrent Neural Networks”. In: *arXiv preprint arXiv: 1603.08983* (2016).

[29] Jordan Guerguiev, Konrad Kording, and Blake Richards. “Spike-based causal inference for weight alignment”. In: *International Conference on Learning Representations*. 2019.

[30] A. Guez et al. “Learning to Search with MCTSnets”. In: *International Conference On Machine Learning* (2018).

[31] Arthur Guez et al. “An Investigation of Model-Free Planning”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2464–2473. URL: http://proceedings.mlr.press/v97/guez19a.html.

[32] Hassan Hafez-Kolahi and Shohreh Kasaei. “Information bottleneck and its applications in deep learning”. In: *arXiv preprint arXiv:1904.03743* (2019).

[33] Danijar Hafner et al. “Mastering Diverse Domains through World Models”. In: *arXiv preprint arXiv: 2301.04104* (2023).

[34] Jessica B. Hamrick et al. *On the role of planning in model-based deep reinforcement learning*. 2021. arXiv: 2011.04021 [cs.AI].

[35] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[36] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *Aaai Conference On Artificial Intelligence* (2017). DOI: 10.1609/aaai.v32i1.11796.

[37] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[38] Shengran Hu and Jeff Clune. “Thought Cloning: Learning to Think while Acting by Imitating Human Thinking”. In: *arXiv preprint arXiv: 2306.00323* (2023).

[39] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), p. 106.

[40] Bernd Illing, Wulfram Gerstner, and Johanni Brea. “Biologically plausible deep learning — But how far can we go with shallow networks?” In: *Neural Networks* 118 (2019), pp. 90–101. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0893608019301741.

[41] Andy L. Jones. *Scaling Scaling Laws with Board Games*. 2021. arXiv: 2104.03113 [cs.LG].

[42] Gary Klein. “Chapter 4 - Applied Decision Making”. In: *Human Performance and Ergonomics*. Ed. by P.A. Hancock. Handbook of Perception and Cognition (Second Edition). San Diego: Academic Press, 1999, pp. 87–107. ISBN: 978-0-12-322735-5. DOI: https://doi.org/10.1016/B978-012322735-5/50005-1. URL: https://www.sciencedirect.com/science/article/pii/B9780123227355500051.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[44] Daniel Kunin et al. "Two routes to scalable credit assignment without weight symmetry". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5511–5521.

[45] Jack Lanchantin et al. "Learning to Reason and Memorize with Self-Notes". In: *arXiv preprint arXiv: 2305.00833* (2023).

[46] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. "Learning to solve the credit assignment problem". In: *International Conference on Learning Representations*. 2019.

[47] Julien Launay, Iacopo Poli, and Florent Krzakala. "Principled training of neural networks with direct feedback alignment". In: *arXiv preprint arXiv:1906.04554* (2019).

[48] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[49] Yann LeCun. "Learning processes in an asymmetric threshold network". In: *Disordered Systems and Biological Organization*. Ed. by Francois Fogelman-Soulié, Eliot Bienenstock, and George Weisbuch. Les Houches, France: Springer-Verlag, 1986, pp. 233–240.

[50] Yann LeCun. "Modeles connexionnistes de lapprentissage". PhD thesis. These de Doctorat, Universite Paris, 1987.

[51] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[52] Dong-Hyun Lee et al. "Difference Target Propagation". In: *ECML/PKDD* (2014). DOI: 10.1007/978-3-319-23528-8_31.

[53] Dong-Hyun Lee et al. "Difference target propagation". In: *Joint european conference on machine learning and knowledge discovery in databases*. Springer. 2015, pp. 498–515.

[54] Aitor Lewkowycz et al. "Solving Quantitative Reasoning Problems with Language Models". In: *Neural Information Processing Systems* (2022). DOI: 10.48550/arXiv.2206.14858.

[55] Chiang-Shan Ray Li et al. "Imaging response inhibition in a stop-signal task: neural correlates independent of signal monitoring and post-response processing". In: *Journal of Neuroscience* 26.1 (2006), pp. 186–192.

[56] Q. Liao, Joel Z. Leibo, and T. Poggio. "How Important is Weight Symmetry in Backpropagation?" In: *AAAI Conference on Artificial Intelligence* (2015). DOI: 10.1609/aaai.v30i1.10279.

[57] Timothy P Lillicrap et al. “Backpropagation and the brain”. In: *Nature Reviews Neuroscience* 21.6 (2020), pp. 335–346.

[58] Timothy P Lillicrap et al. “Random feedback weights support learning in deep neural networks”. In: *arXiv preprint arXiv:1411.0247* (2014).

[59] Timothy P Lillicrap et al. “Random synaptic feedback weights support error backpropagation for deep learning”. In: *Nature communications* 7.1 (2016), pp. 1–10.

[60] Seppo Linnainmaa. “Taylor expansion of the accumulated rounding error”. In: *BIT Numerical Mathematics* 16.2 (June 1976), pp. 146–160. ISSN: 1572-9125. DOI: 10.1007/BF01931367. URL: https://doi.org/10.1007/BF01931367.

[61] Nikolay Manchev and Michael W Spratling. “Target Propagation in Recurrent Neural Networks.” In: *J. Mach. Learn. Res.* 21 (2020), pp. 7–1.

[62] Henry Markram et al. “Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs”. In: *Science* 275.5297 (Jan. 1997), pp. 213–215. DOI: 10.1126/science.275.5297.213.

[63] Alexander Meulemans et al. “A Theoretical Framework for Target Propagation”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020.

[64] Marvin Minsky. “Steps toward artificial intelligence”. In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30.

[65] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[66] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. “Feedback alignment in deep convolutional networks”. In: *arXiv preprint arXiv:1812.06488* (2018).

[67] S Chandra Mouli and Bruno Ribeiro. “Neural Network Extrapolations With G-Invariances From A Single Environment”. In: (2021), pp. 1–20.

[68] Michael C. Mozer. “A Focused Backpropagation Algorithm for Temporal Pattern Recognition”. In: *Complex Syst.* 3 (1989). URL: https://api.semanticscholar.org/CorpusID:18036435.

[69] Samreen Naeem et al. “An Unsupervised Machine Learning Algorithms: Comprehensive Review”. In: *IJCDS Journal* 13 (Apr. 2023), pp. 911–921. DOI: 10.12785/ijcds/130172.

[70] Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. “Understanding The Failure Modes of Out-of-Distribution Generalization”. In: (2021), pp. 1–25.

[71] Arild Nøkland. “Direct Feedback Alignment Provides Learning in Deep Neural Networks”. In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 1037–1045.

[72] Maxwell Nye et al. “Show Your Work: Scratchpads for Intermediate Computation with Language Models”. In: *arXiv preprint arXiv: 2112.00114* (2021).

[73] Richard W. Olshavsky. “Task complexity and contingent processing in decision making: A replication and extension”. In: *Organizational Behavior and Human Performance* 24.3 (1979), pp. 300–316. ISSN: 0030-5073. DOI: https://doi.org/10.1016/0030-5073(79)90032-1. URL: https://www.sciencedirect.com/science/article/pii/0030507379900321.

[74] Aaron van den Oord et al. “Conditional Image Generation with PixelCNN Decoders”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016.

[75] Alexander G Ororbia and Ankur Mali. “Biologically motivated algorithms for propagating local target representations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4651–4658.

[76] Julius Pahlke, Martin G. Kocher, and Stefan T. Trautmann. “Tempus Fugit: Time Pressure in Risky Decisions”. In: (Apr. 2011). URL: https://ssrn.com/abstract=1809617%20or%20http://dx.doi.org/10.2139/ssrn.1809617.

[77] Giambattista Parascandolo et al. *Learning explanations that are hard to vary*. 2020. arXiv: 2009.00329 [cs.LG].

[78] Giambattista Parascandolo et al. *Learning Independent Causal Mechanisms*. 2018. arXiv: 1712.00961 [cs.LG].

[79] Razvan Pascanu et al. “Learning model-based planning from scratch”. In: *arXiv preprint arXiv: 1707.06170* (2017).

[80] John Payne, James Bettman, and Eric Johnson. “Adaptive Strategy Selection in Decision Making”. In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 14 (July 1988), pp. 534–552. DOI: 10.1037/0278-7393.14.3.534.

[81] John W Payne, James R Bettman, and Eric J Johnson. *The adaptive decision maker*. Cambridge university press, 1993.

[82] John W. Payne. “Task complexity and contingent processing in decision making: An information search and protocol analysis”. In: *Organizational Behavior and Human Performance* 16.2 (1976), pp. 366–387. ISSN: 0030-5073. DOI: https://doi.org/10.1016/0030-5073(76)90022-2. URL: https://www.sciencedirect.com/science/article/pii/0030507376900222.

[83] Fernando Pérez-Cruz. “Kullback-Leibler divergence estimation of continuous distributions”. In: *2008 IEEE international symposium on information theory*. IEEE. 2008, pp. 1666–1670.

[84] Thomas Pierrot et al. “Learning Compositional Neural Programs for Continuous Control”. In: *arXiv preprint arXiv: 2007.13363* (2020).

[85] Thomas Pierrot et al. “Learning Compositional Neural Programs with Recursive Tree Search and Planning”. In: *Neural Information Processing Systems* (2019).

[86] S. Racanière et al. “Imagination-Augmented Agents for Deep Reinforcement Learning”. In: *NIPS* (2017).

[87] B. Reddi and R. Carpenter. “The influence of urgency on decision time”. In: *Nature Neuroscience* 3 (2000), pp. 827–830. DOI: 10.1038/77739.

[88] Blake A Richards et al. “A deep learning framework for neuroscience”. In: *Nature neuroscience* 22.11 (2019), pp. 1761–1770.

[89] Vincent Roulet and Zaid Harchaoui. “Target Propagation via Regularized Inversion”. In: *arXiv preprint arXiv:2112.01453* (2021).

[90] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[91] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536. URL: https://api.semanticscholar.org/CorpusID:205001834.

[92] João Sacramento et al. “Dendritic cortical microcircuits approximate the backpropagation algorithm”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[93] Arthur L Samuel. “Some studies in machine learning using the game of checkers. II—Recent progress”. In: *IBM Journal of research and development* 11.6 (1967), pp. 601–617.

[94] Tomohiro Sawada et al. “ARB: Advanced Reasoning Benchmark for Large Language Models”. In: *arXiv preprint arXiv: 2307.13692* (2023).

[95] Benjamin Scellier and Yoshua Bengio. “Equilibrium propagation: Bridging the gap between energy-based models and backpropagation”. In: *Frontiers in computational neuroscience* 11 (2017), p. 24.

[96] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks : the official journal of the International Neural Network Society* 61 (2014), pp. 85–117. URL: https://api.semanticscholar.org/CorpusID:11715509.

[97] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[98] Jürgen Schmidhuber. “Learning complex, extended sequences using the principle of history compression”. In: *Neural Computation* 4.2 (1992), pp. 234–242.

[99] Jürgen Schmidhuber. “My first deep learning system of 1991+ deep learning timeline 1962-2013”. In: *arXiv preprint arXiv:1312.5548* (2013).

[100] Julian Schrittwieser et al. “Mastering atari, go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (2020), pp. 604–609.

[101] Xingjian Shi et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in neural information processing systems* 28 (2015).

[102] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[103] Richard S Sutton. “Introduction: The challenge of reinforcement learning”. In: *Reinforcement learning*. Springer, 1992, pp. 1–3.

[104] Robert Tibshirani. “Regression shrinkage and selection via the lasso: a retrospective”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.3 (2011), pp. 273–282.

[105] F. Torabi, Garrett Warnell, and P. Stone. “Behavioral Cloning from Observation”. In: *International Joint Conference on Artificial Intelligence* (2018). DOI: 10.24963/ijcai.2018/687.

[106] Shimon Ullman. “Rigidity and misperceived motion.” In: *Perception* (1984).

[107] V. Vapnik. “Principles of Risk Minimization for Learning Theory”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R. P. Lippmann. Vol. 4. Morgan-Kaufmann, 1992, pp. 831–838. URL: https://proceedings.neurips.cc/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

[108] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[109] Nino Vieillard et al. “Leverage the Average: an Analysis of KL Regularization in RL”. In: *arXiv preprint arXiv: 2003.14089* (2020).

[110] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=_VjQlMeSB_J.

[111] Paul J. Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural Networks* 1.4 (1988), pp. 339–356. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90007-X. URL: https://www.sciencedirect.com/science/article/pii/089360808890007X.

[112] James CR Whittington and Rafal Bogacz. “An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity”. In: *Neural computation* 29.5 (2017), pp. 1229–1262.

[113] Will Xiao et al. “Biologically-Plausible Learning Algorithms Can Scale to Large Datasets”. In: *International Conference on Learning Representations*. 2018.

[114] Xiaohui Xie and H. Sebastian Seung. "Spike-based Learning Rules and Stabilization of Persistent Neural Activity". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/c0560792e4a3c79e62f76cbf9fb277dd-Paper.pdf.

[115] Shunyu Yao et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *arXiv preprint arXiv: 2305.10601* (2023).

[116] E. Zelikman, Yuhuai Wu, and Noah D. Goodman. "STaR: Bootstrapping Reasoning With Reasoning". In: *Neural Information Processing Systems* (2022). DOI: 10.48550/arXiv.2203.14465.

[117] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320.