JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Marius-Constantin Dinu, BSc

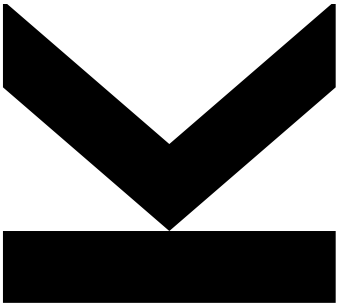
Submitted at
Institute for Machine Learning

Supervisor
Univ.-Prof. Dr. Sepp Hochreiter

Co-Supervisor
Mag. Dr. Günter Klambauer

July 2019

Overcoming Catastrophic Forgetting with Context-Dependent Activations and Synaptic Stabilization



Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references. This printed thesis is identical with the electronic version submitted.

Johannes Kepler University Linz, July 1, 2019

Marius-Constantin Dinu

Contents

Preface

Nowadays, the subject of Artificial Intelligence (AI) belongs to one of the most prominent topics of discussion. Its presence is ubiquitous, stretching from self-driving cars from Uber, to recommendation systems at Amazon and cutting edge research by DeepMind—solving complex combinatorial tasks, such as in popular computer game StarCraft II. AI has a huge impact on society and reshaped all industries. My passion to research this field emerged during my bachelor's and grew ever since. I continue my work and contribute as hard as I can to help realize what I believe will be humanity's greatest invention, Artificial General Intelligence (AGI).

We have a long way to go until we are able to create self-conscious machines. Some experts even argue that this is not possible, that consciousness is unique to humans. I disagree with these claims, and do not see any physical barriers hindering us from solving AGI. Maybe our ability to quantify intelligence and consciousness has to mature or humanity has to accept that consciousness is often used as an excuse to separate us from our novel evolutionary trails. If we redefine the behavioral formalism of intelligent beings, maybe we can also improve the objectives of AGI. The behavioral formalism of agents interacting with an environment was already addressed by Legg and Hutter, 2007, and dates further back to another popular machine learning sub-field, known as Reinforcement Learning (Sutton and Barto, 2017). Advances in Deep Reinforcement Learning yielded the development of AlphaGo (Silver et al., 2016), a software that is adapted in the ancient game of Go (Moyer, 2016). Go is believed to be one of the most difficult games available according to its huge state space which also rules out hand-crafted solutions. AlphaGo went as far as winning against Lee Sedol, a world renowned master of the game. The most interesting part of the software is that its success was not a result of brute forcing the problem, but rather through the ability to *learn and develop its own intuition* about the game while playing against itself.

Needless to say, such achievements are not only accountable to individuals. It is the collaborative work of many researchers, engineers and visionaries which dedicated many years of small incremental improvements. This urges me to offer my gratitude to all the people who inspired and supported me over the last years. First, I like to thank my parents and beloved partner, Laura Neumayer. They gave me the time and freedom I needed to delve into my research and encourage me whenever possible. Second, I thank my friends, who stood by my side although I was seldom available and helped me out when I needed them most. I am also grateful to my boss Rene Loitzenbauer at CELUM GmbH for supporting me with flexible work schedules and inspiring projects. Great thanks also to Daniel Glaser for his constructive feedback and helpful hints in the finishing phase. Special thanks also to my co-supervisor Günter Klambauer, who

supported me throughout this project. He allowed me to explore my craziest ideas and guided me to a successful outcome. I also offer my gratitude to all my professors and colleagues, who inspired me during my studies and for the amazing conversations we had after classes. Last but not least, great thanks to my teacher José Arjona Medina. The valuable talks we had ignited the thoughts to the solution presented in this thesis.

Abstract

Overcoming Catastrophic Forgetting in neural networks is crucial to solving continuous learning problems. Deep Reinforcement Learning uses neural networks to make predictions of actions according to the current state space of an environment. In a dynamic environment, robust and adaptive life-long learning algorithms mark the cornerstone of their success. In this thesis we will examine an elaborate subset of algorithms countering catastrophic forgetting in neural networks and reflect on their weaknesses and strengths. Furthermore, we present an enhanced alternative to promising synaptic stabilization methods, such as *Elastic Weight Consolidation* or *Synaptic Intelligence*. Our method uses context-based information to switch between different pathways throughout the neural network, reducing destructive activation interference during the forward pass and destructive weight updates during the backward pass. We call this method *Context-Dependent Activations* (XdA). We show that XdA enhanced methods outperform basic synaptic stabilization methods and are a better choice for long task sequences.

Kurzfassung

Als *Catastrophic Forgetting* im Kontext von Deep Learning bezeichnet man das Phänomen eines sequentiellen Optimierungsvorgangs, welches sich perfekt an die Spezifika des aktuellen Problems anpassen kann, jedoch vergangene Ereignisse *vergisst* bzw. überschreibt. Deep Reinforcement Learning verwendet neuronale Netze, um Vorhersagen von Aktionen entsprechend dem aktuellen Zustandsraum einer Umgebung zu treffen. In einem dynamischen Umfeld markieren robuste und adaptive Algorithmen für lebenslanges Lernen die Eckpfeiler ihres Erfolgs. In dieser Arbeit werden wir eine ausführliche Untergruppe von Algorithmen vorstellen, die dem katastrophalen Vergessen in neuronalen Netzen entgegenwirken und deren Schwächen und Stärken reflektieren. Darüber hinaus stellen wir eine verbesserte Alternative zu vielversprechenden Methoden zur synaptischen Stabilisierung wie Elastic Weight Consolidation oder Synaptic Intelligence vor. Unsere Methode verwendet kontextbasierte Informationen, um zwischen verschiedenen Pfaden im gesamten neuronalen Netzwerk zu wechseln. Dadurch werden destruktive Aktivierungsstörungen während des Vorwärtsdurchlaufs und destruktive Gewichtsaktualisierungen während des Rückwärtsdurchlaufs reduziert. Wir bezeichnen diese Methode Context-Dependent Activations (XdA). Wir zeigen, dass XdA-Erweiterte Methoden die grundlegenden synaptischen Stabilisierungsmethoden übertreffen und eine bessere Wahl für lange Tasksequenzen darstellen.

Chapter 1

Introduction

> "A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."
>
> *Alan Turing*

1.1 Background

The strive for creating intelligent machines dates back to the Renaissance, starting with the first designs of steam-based engines, followed by automation up until the early days of the first programmable digital computers in 1940. McCorduck's book "Machines Who Think" (McCorduck, 1979) first published in 1979 already offered some insights into the prospects of AI. It took a few more decades until the widespread adoption of modern machine learning to start to revolutionize the industry. Nowadays, researchers worldwide from different fields of research commit themselves to solving the puzzle of the human brain and have been trying to design our first artificial counterpart. Thanks to the algorithmic breakthroughs from backpropagation of the first multi-layered connectionist models by Werbos, 1975, to the advances in modern day's Deep Learning architectures, we gradually set the stepping stones for creating the first *Artificial General Intelligence* (AGI).

1.2 Motivation

There are three major steps we need to accomplish to advance AGI. First, we need to improve the ability of life-long learning. *Artificial Neural Networks* (ANN) need to be more dynamically adaptable and expandable while still remaining stable to previous information. The Google DeepMind team proposed the Neural Touring Machines (NTM) (Graves, Wayne, and Danihelka, 2014) to improve life-long learning by reading and writing activations to external memory. However in practice, the applicability is still very limited.

Secondly, we need to include temporal aspects to the core of our architectures, where the state dynamics remain stable over long time sequences. Long-Short Term Memory (LSTM) by Hochreiter and Schmidhuber, 1997 and Gaten Recurrent Units (GRU) by Chung et al., 2014 are advocates of such temporal approaches, yet the issues with these approaches rely within the stability of their hidden state. It has been shown by Merity, Keskar, and Socher, 2017 that simple regularization approaches, such as Dropout—which are effective in the context of *fully connected* or *convolutional* network structures—disrupt the ability of a long term memory, requiring more elaborate alternatives. We need to encode features in a continual and sequential manner, with stable internal state representations.

Third, we propose to shift the focus from encoding task-specific information into the weight space towards the activation space. Using this approach, the weight space only operates as an encoding and decoding entity of activation space. Currently, the most efficient way to train a well performing model is by iterating over a large amount of data multiple times until the model converges. The trained statistical representation then offers reasonable inference performance on all samples with the assumption that they are originating from the same distribution. However, the dynamics of the input space can drastically change by altering the task objective or extending the objective. Backpropagation lacks dynamic adaptiveness, requiring a more elaborate notion of persisting and retrieving activation encodings. The answer may rely within handling context-dependent activations, which are either allocated and retrieved from an external memory system, or switched during the training or testing phase. This slightly shifts the training objective from matching input to output classes, towards activation-dependent training units. We then try to avoid the encoding of domain specifics within the neural network weights but within a dynamically changing context.

The combination of all three ideas would result in a dynamically extensible, and adaptive system, capable of learning with few samples and preserving information over long periods of time. As an analogy, we can reflect on the architecture of general purpose computers. Computers use Random Access Memory (RAM) to execute program specific code without hardwiring data into its system, by dynamically operating on the memory state with no persistence. The hardware design offers only an operational set and does not restrict or embed any domain specific information during runtime. In case of neural networks, by manipulating the activation space we also saw promising results with *One-Shot Learning* approaches, as proposed by Kaiser et al. in *Learning to Remember Rare Events* (Kaiser et al., 2017). They combined LSTMs with a nearest neighbor based memory module, and showed that they can improve the performance on multiple tasks and learn from rare occurring samples, although this thesis shifts its focus on the aspect of improving life-long learning of neural networks with the usage of context-dependent activations.

The bottom line 1.2.1 *If we want to advance in AGI we need to improve training methods to store and recall learned patterns. By operating on the activation space we can train more flexible models that adapt dynamically and require less data. Solving the continuous learning problem will also reduce repetitive learning of tasks and advance the AI field. Improving the life-long learning capability of models extends to all fields of machine learning, such as reinforcement learning.*

1.3 Scope

Serrà et al., 2018, presented in their paper *Hard Attention to the Task* (HAT) a task-based attention algorithm that preserves previously learned information and minimally constrains the learning of the current task. Additionally, they compare multiple methods for avoiding catastrophic forgetting and analyzed the effectiveness of masking activations based on embeddings based hard attention. While learning new tasks in a sequential manner, they gradually freeze weights to preserve information of previous tasks. To avoid fast saturation of the network capacity they use an effective regularization for compression. Although this method works very effectively, it is restricted in usage by pre-requiring knowledge about the task context to select the correct hard attention mask. Also freezing weights is not practical for larger scales of tasks. In contrast, *Elastic Weight Consolidation* (EWC) (Kirkpatrick et al., 2017) and *Synaptic Intelligence* (SI) (Zenke, Poole, and Ganguli, 2017) consolidate weights to a common subspace, which allows continuous learning due to neural plasticity, but aggressively restricts the task adaptiveness and reduces the overall performance of the model. The scope of this thesis is to analyze existing methods of avoiding catastrophic forgetting and propose improvements for synaptic stabilization methods. Throughout this work, we will use synapses and weights interchangeably. The proposed method is operating on the activation space by defining the corresponding context and is practical for online and continuous learning problems. We also perform in-depth analysis and comparing the obtained results with the original HAT, which currently marks the state-of-the-art in overcoming catastrophic forgetting. This thesis concludes with insights and key aspects for continuous learning and proposes an outlook for future work.

1.4 Target

This thesis targets intermediate to experienced machine learning developers and researches. The reader should have some knowledge of machine learning and basic deep learning concepts.

1.5 Structure

The structure of this thesis follows a common approach. The first chapter (2) defines the theoretical foundation of modern neural networks and introduces related topics, such as multi-task learning, continuous learning, transfer learning, etc. and defines general training methods. It will also provide a theoretical discourse, where continuous learning plays an essential role, such as in supervised and reinforcement learning problems. Additionally, some of the most important concepts for modern deep learning will be introduced and which will be used throughout this work. The chapter closes with the introduction of the architectural design and used datasets for testing and evaluating the models. Chapter 3 presents the related work to catastrophic forgetting and summarizes the proposed solutions. In chapter 4 we then reflect on different aspects of the proposed methods and give some insights on the results obtained from the literature. The subsequent chapter 5 presents an improvement for synaptic stabilization methods,

denoted as Context-Dependent Activations (XdA), which is based on ideas from the Context-Dependent Gating (XdG) (Masse, Grant, and Freedman, 2018), Elastic Weights Consolidation and Synaptic Intelligence. Additionally, we also empirically evaluate the training behavior and provide some insights on converges. Chapter 6 summarizes the results reconstructed from the original HAT paper and compares the different methods with in depth focus on Context-Dependent Activations 5. The last chapter (7) reflects upon this work and offers some future outlook for improvements and steps to take to reduce catastrophic forgetting and improve continuous learning.

Chapter 2

Theoretical Foundations

> "Truth is ever to be found in simplicity, and not in the multiplicity and confusion of things."
>
> *Isaac Newton*

This chapter will focus on the terminology and related concepts of catastrophic forgetting, modern Deep Neural Networks (DNN) and the core building blocks requires for the subsequent chapters. All sections are self-contained and can be skipped, if the basic concepts are already understood.

2.1 Learning Definitions

The term catastrophic forgetting or catastrophic interference was first coined by McCloskey and Cohen, 1989, and refers to the issue that simplistic connectionist neural network models were not designed for learning tasks sequentially. A connectionist model refers to models with focus on the weighted connections between nodes, without including contextual meaning while learning a task. The lack of context, makes such models innate to learn new data without forgetting previously learned information. A majority of research focus on optimizing models to become more sensitive to but not disrupted by new data. The main approaches are to switch between context-dependent subsets of weights or to regularize the objective function to enable the consolidation of weights. The first approach mainly uses freezing of weights and compression techniques to stabilize them while the latter converges the weights to a common sub-space during a sequential training procedure.

To frame a more complete view of the training dynamics we differentiate between three concepts of overcoming catastrophic forgetting: rehearsal, pseudo-rehearsal, and reduced representational overlap (Robins, 1995). These concepts are independent to the machine learning method, such as supervised, unsupervised or reinforcement learning, and describe how a solution is obtained with respect to data handling or regularization of the training procedures. This thesis mainly focuses on the reduced representational overlap to define a continuous learning method.

2.1.1 Single Task Learning

To ensure a common terminology this work will refer to learning a task t, as solving a minimization problem by reducing an evaluated error between a predictive value $\hat{y}$ and the actual value y, where $\hat{y}$ is obtained by forwarding a set of training samples X to a neural network, where $(X, y) \in \mathcal{D}_t$ and the dataset $\mathcal{D}_t$ can be subject to one or more classes $C = 1 \dots N$. Unless stated differently, each class is refereed to as an one-hot encoded representation of mutually excluding class references.

2.1.2 Multi-Task Learning

Multi-task learning in neural networks aims to optimize multiple objectives or tasks at the same time, in contrast to optimizing a single task. The advantage of learning on multiple tasks is that the complementary tasks can improve together, meaning that solving task B can also improve or guide the solution for task A as demonstrated by Ruder in "An Overview of Multi-Task Learning in Deep Neural Networks" (Ruder, 2017). In the context of this thesis, the output layers are similar to a multi-task learning problem, assigning each task its own head or set of parameters. When optimizing we will switch between the tasks in a sequential manner and the corresponding output head. All other parameters are either shared or manipulated as in the papers described.

2.1.3 Transfer Learning

Transfer learning is the notion of reusing a pre-trained neural network model, which was optimized on some particular task, and is redefined to fit a new task. In this case, the weights of a pre-trained model are kept frozen and only the classification layer at the top of the network is exchanged and trained on the new task. This approach is very popular for image recognition problems based on convolutional neural networks (CNN) as described in section 2.3.3, because the CNN layers mainly function as translation invariant feature extractors (Wiatowski and Bölcskei, 2015) and the top layer is in many cases a fully connected classification layer. When switching the task-specific classifier, the weights are immune to catastrophic forgetting since they are never altered. The critical point is that not all tasks emerge from the same distribution and the reuse of pre-trained feature extractors shows limiting performance compared to re-training from scratch. But it still is intensively used, because training networks from scratch requires a very large amount of data, which is often not available.

2.1.4 Fine-Tuning

Fine-Tuning is similar to transfer learning, except that we do not freeze all feature extraction layers. We reuses already well-generalized feature extractors and allow minor weight alternations to gradually adjust the new task objective. This usually adds the cost of becoming incompatible with the original classifier or reducing its original performance. This method is widely used, because it is more efficient to alter pre-trained weights, instead of training the entire network from scratch, which may not only take days, but also requires a large amount of data to properly generalize. In the context of overcoming catastrophic forgetting this approach was often used with an additional L2 regularization

term to avoid large updates. L2 regularization does not assess the importance the task-specific weights and usually shows drastic performance drops in previously trained tasks. *Less-Forgetting Learning* 3.3 and *Learning Without Forgetting* 3.4 from the related work chapter 3 are based on fine-tuning techniques with L2 regularization of weights and the task specific objectives.

2.1.5 Continuous Learning

Continuous learning or continual life-long learning in the context of machine learning defines a set of algorithms that enable online sequential learning. After learning N tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$ on N datasets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N$, which can be of different *types* or *domains*, and are now confronted with task $\mathcal{T}_{N+1}$ of the dataset $\mathcal{D}_{N+1}$, the algorithm can optimize itself towards the new objective without compromising the performance on previously learned tasks. Such an algorithm must learn general features that are valid across all tasks and reuse them while training on new tasks (Chen and B. Liu, 2016). In Figure 2.1 we provide an illustration of the life-long learning cycle. The evaluation of LML

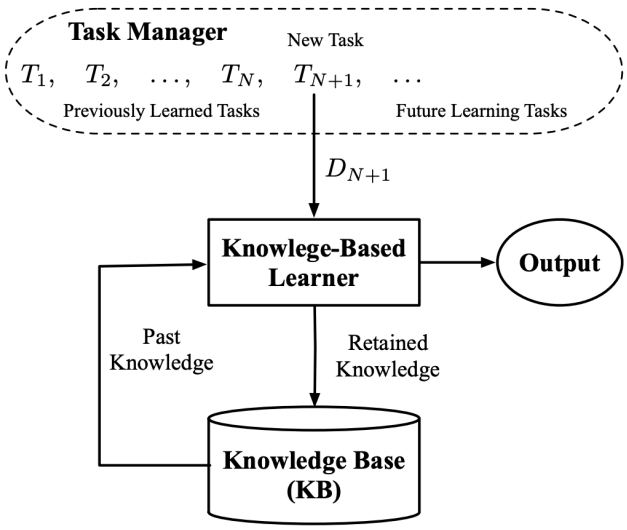


Figure 2.1: Life-long learning schematics. The knowledge-based learner has to retain its knowledge base over time and can query it for solving new tasks. If new information is provided it requires to store its newly gained knowledge to the knowledge base without disrupting previous entities (Chen and B. Liu, 2016).

algorithms is different compared to statically pre-defined task-based methods, since we need to evaluate not only on the current task, but also on all previously learned tasks. Chapter 6 will introduce a metric proposed by Serrà et al., 2018, to evaluate the performance of algorithms that are trained in a sequential manner, measuring the average precision drop on previous tasks and the capability to adopt new task-specific features.

2.1.6 Rehearsal

Learning approaches categorized as rehearsal refer to the accumulation of information and requires all requested tasks. If new information is provided, we need to train a new model reusing the entire collection of data. Such an approach is not only very inefficient, but also quickly exceeds capacity limitations due to restricted computational power and memory—but it remains the most performing way to train top notch neural networks. This method will be used to define our best achievable performance, when evaluating the models.

2.1.7 Pseudo-Rehearsal

Pseudo-rehearsal refers to a data processing approach that has no permanent access to the entire task set, but trains on a generative representation resembling the original data distribution. Transfer learning is an example of this category. Furthermore, fine-tuning a pre-trained model with small learning rates represents a relaxed form of a pseudo-rehearsal approach. In fact, such approaches can be stated as memory-free, because their training process can be applied in an online learning environment.

2.1.8 Reduced Representational Overlap

This approach focuses on applying structural regularization to the input, intermediate and/or output layers of neural networks (French, 1991). By controlling the weight updates we can control how much influence the new data has upon the existing model. The main objective is the efficient distribution of information across the network as well as usage of the full capacity of the network. The most difficult part is to steer the training process to retain important information and only readjust irrelevant weights. This approach is the main focus of this thesis.

2.1.9 Overcoming Catastrophic Forgetting

Alleviating catastrophic forgetting in neural networks can be practically tackled in three major ways:

- Architectural regularization
- Functional regularization
- Structural regularization

Architectural

An architectural method is defined as the reduction of interference without changing the objective function. Freezing weights, using low learning rates or injecting noise via *Dropout* 2.3.6 can be seen as such regulatory improvements (Zenke, Poole, and Ganguli, 2017).

Functional

Functional approaches try to maintain the input to output mapping by adding a regularization term to the objective function. This method requires the tracking of activations

before and after an update and corrects for the computed deviations.

Structural

On a structural level, we can add a penalization term at the parameter level and enforce them to stay close to representations of old tasks. By avoiding large updates and, therefore, restricting weight changes we try to find a representation that performs well on multiple tasks. *Elastic Weight Consolidation* 3.2 and *Synaptic Intelligence* 3.9 are examples of such approaches.

2.2 Learning Approaches

The learning approach defines how an algorithm is applied to solve a particular problem. The algorithm is either instructed with the correct labels, it builds its own internal clusters or is indirectly rewarded for positive behavior over time periods. The three approaches are known as supervised, unsupervised or reinforcement learning respectively. The concepts for controlling the learning process is denoted as rehearsal, pseudo-rehearsal or reduced representational overlap and can be applied to all three learning approaches. All these topics are equally important to solve AGI, but in this thesis, we will mainly focus on the supervised learning approach, and provide a brief referential description of reinforcement learning since it represents the approach which benefits mostly alleviating catastrophic forgetting and enabling continuous learning.

2.2.1 Supervised Learning

For each input data we have our target output value, whereas the goal is to identify the relationship between input and output. This is also known as *predictive modeling*, due to the goal to generalize towards the training samples and predict the output value of unseen samples. Given a set of N training samples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, whereas x_i defines the feature vector at the i-th position and y_i the corresponding label, we try to learn a function $g : X \rightarrow Y$, where X is the input space and Y the output space (Wikipedia, 2019b). Our goal is to find an unbiased estimator R_{emp} of the generalization error R, such that

$$R_{\text{emp}}(g(.; \boldsymbol{w}), \boldsymbol{Z}_m) = \frac{1}{m} \sum_{j=1}^{m} L(y^{l+j}, g(\boldsymbol{x}^{l+j}; \boldsymbol{w})), \tag{2.1}$$

whereas l defines the size of the training set, m the size of a i.i.d. test set $\boldsymbol{Z}_m = (\boldsymbol{z}^{l+1}, \ldots, \boldsymbol{z}^{l+m})$, $\boldsymbol{w}$ the parameters to optimize and L the loss function. By minimizing the loss L w.r.t. $\boldsymbol{w}$ we are able resemble the generalization error R and perform better on future unseen samples.

In the context of a fully connected ANN we can define g to be a (multi-layered) approximation function

$$g : f(\boldsymbol{X}; \boldsymbol{W}) = f_{j=H}(\ldots, f_{j=1}(\sum_{i=1,j}^{k_j} \sigma(x_{i,j} w_{i,j} + b_j))), \tag{2.2}$$

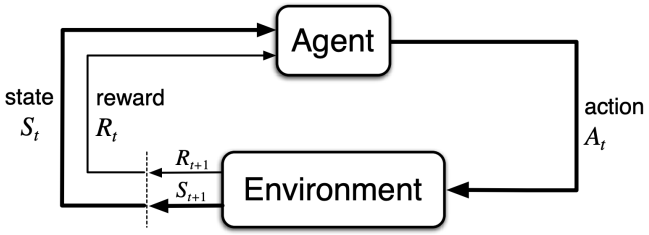


Figure 2.2: Reinforcement learning agent-environment interactions (Sutton and Barto, 2017). The agent can select actions according to the state of the environment and receives the next state and a reward for the state transition.

whereas σ denotes the applied activation function, f_j the current layer function, i the current neuron or unit at layer j, k the number of units at layer j, b the bias for each layer and H the maximum number of layers (Kojouharov, 2017). We also try to find an input to output mapping by adjusting the weight matrix $\boldsymbol{W}$ according to the backpropagated losses (see section 2.3.1), which are obtained by comparing the predictions with the actual labels, evaluated by the selected cost metric.

2.2.2 Reinforcement Learning

Richard Sutton's first publication in 1998 set of a revolution of how people thought about approaching machine learning problems (Sutton and Barto, 2017). Instead of actively instructing an algorithm with labeled information—requiring hours of hard work labeling of datasets—an agent is defined which explores its environment by performing different actions and is progressively rewarded when making proper decisions. The advantage of this approach is that we are mapping situations to actions, by maximizing the received reward, without having to tell the agent which exact actions it has to take. The advantage of not pre-labeling the data, helps to solve problems which were not approachable using supervised learning. This shifts the focus to a new issue, known as the exploration versus exploitation trade-off, which requires the algorithm to choose between reusing already promising solutions and finding new unknown pathways. The received reward can be delayed over longer time periods, which increases the difficulty of converges or causes the algorithm to get stuck in a local minimum. Although, with the right configurations and adjustments this machine learning approach opens a huge field of possibilities.

Reinforcement learning basically consists of four main elements, a *policy*, a *reward signal*, a *value function*, and optionally a *model* of the environment. In Figure 2.2 we provide an overview of the interactions of the main elements of reinforcement learning. In the subsequent sections we will explain the terminology and introduce some reinforcement learning methods.

Policy

Given an environmental state s_t, an agent can place actions a to move to a next state s_{t+1} of the environment, where t denotes the current timestep. The agent accomplishes

this by following its learned *policy* π, which basically maps perceived states to actions. It is also noticeable that states can be stochastic, meaning that the selected action does not lead to a deterministic next state. Reinforcement Learning methods basically split into two categories, denoted as on-policy and off-policy, where the off-policy methods let the agent learn from historical data and does not follow directly the current scheme. It is also important to state that the policies are usually approximated by universal function approximators, such as a neural networks, since computing exact solutions for large state and action spaces becomes unfeasible.

Reward Signal

The *reward signal* r_t is received from the environment and provides the agent with additional information about the quality of its chosen action a given state s. The main objective of an RL algorithm is to maximize the total possible reward G_t obtained on the long run and intermediate rewards act as a guidance throughout the decision process. Rewarding signals can also be stochastic functions based on the environmental state and the action taken.

Model

In some reinforcement learning methods, the behavior of the environment is learned and represented through a *model*. *Model-based* methods encode the environment to help a system to make predictions into the future. In contrast, *model-free* methods are simple trial-and-error learners, without tracking long term implications of their changing environment.

Value Function

The *value function* v_π can be defined as the most important element in reinforcement learning. The reward signal r_t defines the immediate positiveness or negativeness of an action taken in a state s_t, the value function v_π estimates how well an agent will perform on the long run according to its current policy π. By following the path of the highest immediate reward it is not guaranteed that the maximum achievable reward can be obtained. In fact, following temporarily the worst path can lead to a better total future reward or even to the optimal outcome. For non-trivial problems the value function cannot simply be computed and has to be estimated over the entire state-actions sequences. To enable a decomposed state-action update sequence the American mathematician Richard Bellman proposed the so-called "Bellman equation of optimality", which allows to approach the optimal achievable reward through through discounted intermediate rewards. More formally speaking, for a non-deterministic environment with a given Markov Decision Process (MDP) the state-value function equals the immediate reward plus the expected discounted total reward of future states:

$$V(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{2.3}$$

$$= \mathbb{E} \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right]. \tag{2.4}$$

And for an episodic evaluation of the state-value function starting at v_0 the agent selects the action a which maximizes the total discounted future reward

$$v_0 = \max_{a \in A} \mathbb{E}_{s \sim S} \left[r_{s,a} + \gamma v_s \right]. \tag{2.5}$$

Recursively repeating the state-value function and considering the Bellman optimality equation with an discrete action space according to a policy π, we receive

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \text{ for all } s \in S, \tag{2.6}$$

where s' denotes the next state, $p(s', r|s, a)$ the probability of transitioning to that state when taking the action a in state s and γ the discount factor. Allowing us to optimize our policy by selecting the actions which maximize the total discounted reward.

Q-Learning

Q-Learning shifts the focus from the state-value function towards the action-value function $q(s, a)$, which in practice defines a better quantity when assessing the discounted future reward of a state. It can be derived from the state-value function as $q_{s,a} = \mathbb{E}_{s' \in S} \left[r_{s,a} + \gamma v_{s'} \right]$. The main difference to the state-value function is that we use a state-action pair to obtain the discounted future reward and don't have to compute the state-value function for each state, since we can derive it from the action-value function as $v_s = \max_{a \in A} q_{s,a}$. For an finite Markov Decision Process (MDP) finding the optimal policy π_* means that we choose actions based on our policy given the current state s to maximize towards the optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$, and by transitioning this to the optimal action-value function q_*, we can define the expected reward of a state based on the action-value function analogously:

$$Q_*(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_*(s_{t+1}) \mid S_t = s, A_t = a \right], \tag{2.7}$$

where the Bellman optimality equation can be applied recursively and is defined as

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \tag{2.8}$$

This method works extremely well for smaller state and action spaces, but for very long episodes, where the reward is only received in an defined outcome at the end of a sequence, the following approaches are proposed.

Temporal-Difference Learning

One of the central aspects of reinforcement learning is temporal-difference (TD) learning, which combines concepts from Monte Carlo with dynamic programming. The core idea is that we need to find an algorithm that can update our action-value function $q_\pi(s, a)$, which determines the policy π, in an online manner. In contrast to Monte Carlo we do not compute an entire sequence until the end of an episode to correct for estimated values, because in practice we do not know the required sequence length and in theory it

Algorithm 2.1: Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$ (Sutton and Barto, 2017).

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, $Q(s_0, .) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
  Chose A from S using policy derived from Q
    Take action A, observe A, S'
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$
    $S \leftarrow S'$
  until S is terminal

might contain an infinite amount of episodes. The alternative is to make small steps in one direction and backtrack false expectations, which might result in random or noisy behavior on local scales but on global scales convergences towards the optimal action-value function q_*. The update for the action-value function focuses on the differences between time steps and can be described as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right], \tag{2.9}$$

where α denotes the learning rate in which we only make small correcting steps during the state transitions. One variation of TD-Learning is shown in equation 2.9 and is known as *Sarsa On-Policy TD Control*, since it uses every element from the state-action pair of the $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ events. In practice, we continuously estimate the action-value function $q_\pi(s, a)$ for the policy π and at the same time greedily alter π to follow q_π. One of the milestones in reinforcement learning was to find an formalism to directly approximate q_*, by using an off-policy method, which is not directly following the policy π. The policy still has an effect by defining which action-value pairs are visited and updated, however it simplifies equation 2.9 to

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right], \tag{2.10}$$

and still guarantees convergence. This is known as Q-Learning and the pseudo-code for the algorithm is shown in Listing 2.1. Several improvements have also emerged for Q-Learning methods, such as Double Q-Learning or by using bootstrapping techniques or *Actor-Critic* systems. The details go far beyond the scope of this work, but for completeness in terms of continuous learning, it is important to state, that policy and value functions can be learned by using neural networks. For instance, in an *Actor-Critic* setup the *actor* learns the policy for taking actions in a given state and the *critic* learns to evaluate the policy currently followed and criticizes the chosen actions of the *actor* if improperly selected.

2.3 Deep Learning Concepts

In the previous section, neural networks were mentioned to be universal function approximators capable of learning complex abstract concepts. In this section, we provide

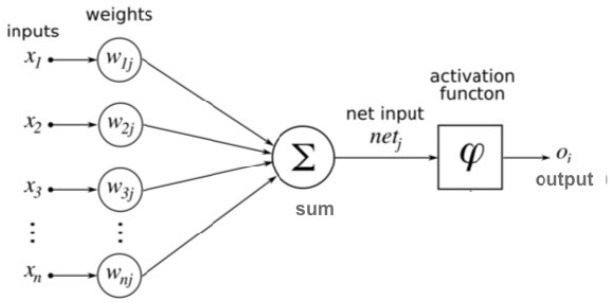


Figure 2.3: Weighted sum computation of a single neuron with n-input features $x_1, x_2 \ldots, x_n$ and one output feature o_i, where i denotes the i-th neuron of a hidden layer (Cintra and Velho, 2011)

.

a short overview of the current state-of-the-art design concepts related to Deep Neural Networks (DNN) which also overlap with the methods introduced in the related work chapter 3.

2.3.1 Fully Connected Layers

Fully connected layers or also known as dense layers compute an input $x_1, x_2, \ldots, x_n$ to output o mapping as a weighted sum between the network parameters and each unit of the input space. Optionally we can add a bias term b to the computation, and afterwards perform a non-linearity operation denoted as φ, as shown in equation 2.11.

$$a = \varphi\left(\sum_{j}^{n} w_j x_j + b\right) \tag{2.11}$$

An illustration of a single neuron computation is shown in Figure 2.3. Since we intend to train more elaborate feature extractors which can represent more complex problems, we can use multiple neurons per layer and also stack multiple layers into a so called multi-layer perceptron (Rosenblatt, 1961), as shown in Figure 2.4. Each neuron resembles a linear combination of the entire input space and finds correlated patterns according to all activations. The overall behavior of the subsequent layers shifts if only some neurons activate differently, which also restricts us to introduce additional weights into the system. This marks one of the most difficult parts in Deep Learning, since one of the requirements for continual learning is to have a dynamically extensible system. However, introducing additional weights in lower layers may disrupt the entire network. To handle this issue, some strategies will be introduced in the next chapter 3.

To train our multi-layer network we usually rely on automatic differentiation (AD) methods, first proposed in the 1970s by Linnainmaa, 1970, which recursively applies the chain rule to evaluate the error at each layer respectively. A special case of AD is known as the backpropagation algorithm, described in the following section.

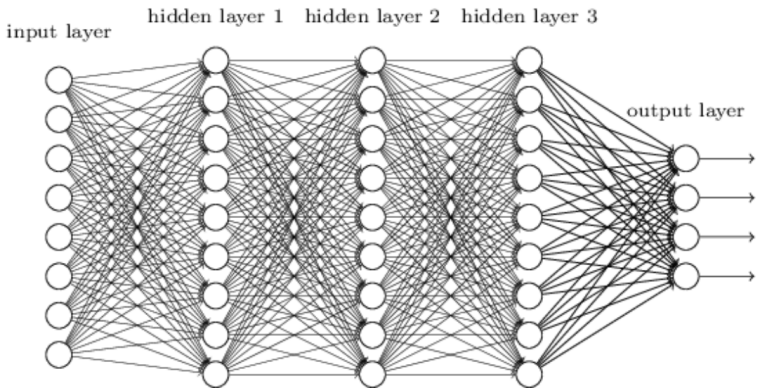


Figure 2.4: Fully connected multi layer perceptron (Nielsen, 2016).

Backpropagation

To optimize the function output towards the target classes, it is necessary to update the parameters of our network, such that we minimize the error between the predicted output and the expected output. In almost all cases of DNN we operate on a large amount of data, where we cannot process all available data at once to find the global minimum. Therefore we perform stochastic sampling and compute batch-wise gradients to minimize the error, known as mini-batch gradient descent (Ruder, 2016). Since stochastic gradient descent introduces noise into our optimization problem, we can average batch-wise evaluated errors C_x and define our cost function C as

$$C = \frac{1}{b} \sum_x C_x, \tag{2.12}$$

where b denotes the sampled batch size. Afterwards we backpropagate the error term δ_j^l evaluated from the cost function C according to the layer-wise weight w_j^l and bias b_j^l contributions, where l denotes the evaluated layer and j the neuron at each layer l. Equation 2.13 shows the partial gradients for the weights and biases, where a^{l-1} marks the activation values of the previous layer $l-1$.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.13}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.14}$$

The calculated gradient is used to determine the direction of the steepest descent for which we update our weights w^l and biases b^l respectively. Listing 2.2 summarizes the entire training steps (Nielsen, 2016):

Algorithm 2.2: Gradient Descent Algorithm

1. **Select a set of training samples X.**
2. **For each training sample x:** Set the corresponding input $a^{x,0}$, and perform the following steps:

Feedforward:	For each layer $l = 1, 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \varphi(z^{x,l})$.
Output error $\delta^{x,L}$:	Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \varphi'(z^{x,L})$.
Backpropagate the error:	For each layer $l = L-1, L-2, \ldots, 1$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \varphi'(z^{x,l})$.

3. **Gradient descent update step:** For each $l = L, L-1, \ldots, 1$ update the weights according to the rule $w^l \leftarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$ and the biases according to $b^l \leftarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

The above algorithm generalizes to different types of architectures, such as CNNs and LSTMs.

2.3.2 Activation Functions

The previous section gave an short introduction to the widely used backpropagation algorithm and introduced the term of activation functions. However, the choice of the proper activation function was left unanswered and represents an entire sub-field of research. This section briefly covers the most common activation functions and explains their main effects on the activation space.

The most important property of a non-linearity activation function is to remain differentiable, such that we can backpropagate the error to lower layers and enable a high gradient flow during training. For the next sub-sections we define z as the activation of our neuron output:

$$z = \sum_j w_j x_j + b. \tag{2.15}$$

Sigmoid

One of the first activation functions used, was the sigmoid function, illustrated in Figure 2.5 at the first row, first column. If φ is equivalent to:

$$\varphi(z) \equiv \frac{1}{1 + e^{-z}}. \tag{2.16}$$

Its active range is approximately within $[-6, 6]$ and values beyond these thresholds have in practice zero gradient information. In the activation space it provides either close to 0 values for the negative range or close to 1 values to the positive range. This property makes it useful from a probabilistic perspective, since in a binary case, we can interpret the sigmoid output probability as the decision boundary between two classes. The unit-wise application of sigmoid activations is often used to define a gating function, which

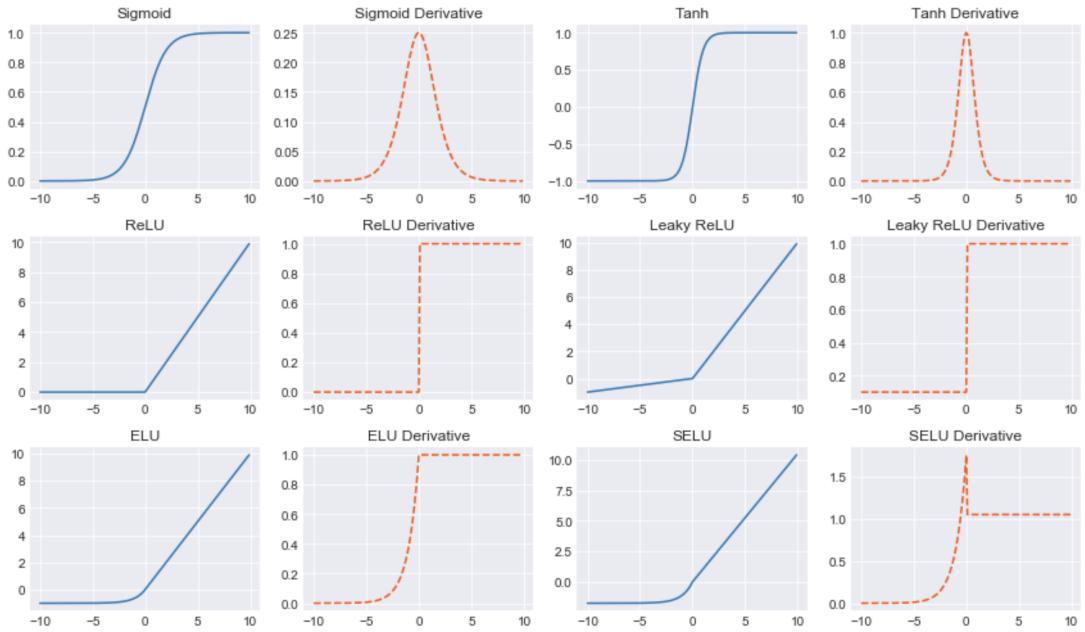


Figure 2.5: List of common activation functions in neural networks.

allows the effects of *forgetting* or *focusing* on specific regions of the input space. The usage of a sigmoid activation as the default non-linearity function for hidden layers in deep neural structures rather troublesome, since the computed gradient is at most $\frac{1}{4}$ at the apex and falls towards zero at both extrema. By analyzing the first order derivative of the sigmoid function 2.17 we can see how the values outside of the active range result in zero gradients.

$$\varphi'(z) = \varphi(z)(1 - \varphi(z)). \tag{2.17}$$

This gradient effect is known as the vanishing gradient problem, which was addressed in the early work of Hochreiter in 1991 (Hochreiter, 1991) and defines a fundamental problem in deep neural networks. Usually, if one stacks more than three layers together, the lower layers will start to suffer from diminishing gradient effects, since multiplications with near zero values will result in almost zero updates.

tanh

The tanh function behaves similar to the sigmoid function, but ranges from -1 to 1 and enables constant gradient at 0. For tanh we set φ equivalent to:

$$\varphi(z) \equiv \frac{e^{2z} - 1}{e^{2z} + 1}. \tag{2.18}$$

As shown in Figure 2.5 in the first row, second column, we get a wider range between $[-1, 1]$, which only slightly improves the vanishing gradient problem for values within the active range, but for extreme values, we suffer from the same effects as in the sigmoid case. The derivative of tanh is defined as

$$\varphi'(z) = 1 - \tanh^2(z). \tag{2.19}$$

ReLU

Rectified Linear Units (ReLUs) enabled the first *deep* neural structures (Glorot, Bordes, and Bengio, 2011), since the gradient flow of ReLUs is constant for all positive values of z, and offers a stabilizing saturations through its zero non-linearity on all negative values. For φ equivalent to:

$$\varphi(z) \equiv \max(0, z), \tag{2.20}$$

we get the first order derivative

$$\varphi'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0. \end{cases} \tag{2.21}$$

It is important to state that the function is not fully differentiable at exactly zero, but in practice, this does not matter. Although this method helps with the vanishing gradient, the opposite effect can be demonstrated with setting a too high initialization variance (G. Yang and Schoenholz, 2017), resulting in an exploding gradient. However, ReLUs with proper initialization have shown outstanding performance, enable very deep network structures, help finding sparse representations and are the de facto standard for modern deep neural networks. For fully connected layers another non-linearity was proposed by Klambauer et al., 2017, known as SELUs 2.3.2 and tackles the vanishing and exploding gradient problem by self-normalization. The ReLU activation function is shown in Figure 2.5 in the second row, first column.

Leaky ReLU

ReLUs help to find sparse representations, however, they can also suffer heavily from dying neurons. By performing wrong initialization of ReLUs, we can easily end up with a majority of the neurons never firing. Since it requires positive activation values to remain in active state, backpropagation will pass no signals through the network if the values are less then zero. To counter this issue, some empirical experiments (Xu et al., 2015) were performed based on leaky ReLUs, which are a softer variant compared to ReLUs, where we downscale z instead of setting it to zero, such that the equivalence for φ is expressed as:

$$\varphi(z) \equiv \max(\alpha z, z), \tag{2.22}$$

where α equals a very small scalar value, such as 0.1 as shown in Figure 2.5 second row, second column. Further alternatives are *Parametric Rectified Linear Unit* PReLU, where α is trainable and not fixed as in the leaky ReLU case. The derivative is stated as:

$$\varphi'(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z < 0. \end{cases} \tag{2.23}$$

This ensures that the neurons are never zero and hence cannot die.

ELU

Exponential Linear Units (ELUs) (Clevert, Unterthiner, and Hochreiter, 2015) are a special case of leaky ReLUs, where instead of having an unbound negative value, the activation function shows the ability to saturate, which not only introduces noise-robustness

but also speeds up training and helps with generalization. The negative values also help to keep the gradient closer to the unit gradient value. Setting the equivalence of the activation function to

$$\varphi(z) \equiv \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{otherwise,} \end{cases} \tag{2.24}$$

where the ELU saturation is controlled through the α term, similar to the leaky ReLU setting. In a simple case, a default value for $\alpha = 1.0$ as shown in Figure 2.5 third row, first column. The derivative of the ELU is defined as:

$$\varphi'(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha e^z(e^z - 1) & \text{if } z < 0. \end{cases} \tag{2.25}$$

SELU

Scaled Exponential Linear Units (SELUs) (Klambauer et al., 2017) are an improvement of ELUs. They incorporate the property of self-normalizing the layer activations by pushing them to zero mean and unit variance. This removes the need for batch normalization (explained in section 2.3.4) for fully connected feedforward neural networks and solves the vanishing and exploding gradient problem. This not only enables very deep architectures and introduces a self regularizing scheme, but also ensures a very robust learning procedure. Setting φ equivalence to

$$\varphi(z) \equiv \lambda \begin{cases} z & \text{if } z > 0 \\ \alpha e^z - \alpha & \text{otherwise,} \end{cases} \tag{2.26}$$

where $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$ are calculated analytically, we can observe the behavior of the SELU in Figure 2.5 third row, second column. The derivative of the SELU is defined as:

$$\varphi'(z) = \begin{cases} \lambda & \text{if } z > 0 \\ \lambda\alpha e^z & \text{if } z < 0. \end{cases} \tag{2.27}$$

Softmax

The softmax function can be established as a special case of the sigmoid function which takes a un-normalized feature vector and normalizes it to a probability distribution for all units. Input vectors can have negative or positive values and are re-normalized to an interval between $[0, 1]$. The probability distribution is often used to match an one-hot encoding setting when assigning prediction values to output classes. To specify the softmax function we set φ equivalent to:

$$\varphi(z) \equiv \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K, \tag{2.28}$$

where K denotes the number of elements of the evaluated feature vector. The first order derivative of the softmax function is defined as:

$$\varphi'_j(z_i) = \begin{cases} \varphi(z_i)(1 - \varphi(z_j)) & \text{if } i = j \\ -\varphi(z_j)\varphi(z_i) & \text{if } i \neq j, \end{cases} \tag{2.29}$$

with partial derivatives of the i-th output and j-th input, since the softmax is a $\mathbb{R}^K \rightarrow \mathbb{R}^K$ function.

2.3.3 Convolutional Layers

The first concepts of convolutional neural network (CNN) can be traced back to 1980, known by the name *Neocognitron* proposed by Fukushima, 1980. Yet, this discovery did not ignite the comeback of neural networks, because backpropagation was not used for optimization and it took another nine years until LeCun, Boser, et al., 1989, published the known *LeNet* architecture, that showed gradient-based optimization with shift invariant unit computations.

In contrast to fully connected units, convolutional units enable weight sharing properties, across the input space, known as filters. These shared filters not only drastically reduce the number of parameters, since we do not require the same number of weights per unit as there are input values, but also creates shift invariant activations, which enable the detection of objects or patches of objects independent of their spacial position. Figure 2.6 shows a typical convolutional neural network architecture, where in many architectures only the last layers are defined by dense layers, to map relations between activations. Such filters are mere matrices as illustrated in Figure 2.7 and convolve over

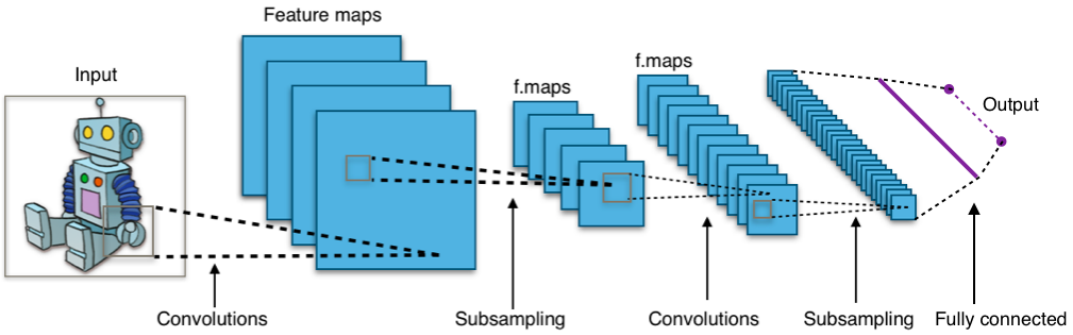


Figure 2.6: Simple CNN architecture scheme (Wikipedia, 2019a).

the input space, where some regions show higher activations, if the input matches with the filter structure, whereas others will be canceled out or damped. Filters can be N-dimensional tensors, although usually in image recognition tasks we define 2d kernel filters. The size of the filters defines the receptive field of a unit, which is moved across the input space. Because kernel filters have originated from the computer vision field, other elements that reduce the dimensionality of the input space are also applicable, such as average pooling and max-pooling layers. Max-pooling uses the maximum activation within the defined filter and average pooling averages for all activations within the filter. Although max-pooling is a non-reversible function, it is often used in practice as it reduces the dimensionality of the input space. The resulting activations from a 2d convolution is a volume of size $H \times W \times C$, denoting height, width and channel size respectively (Stanford Blog, 2019). To evaluate the dimensions for the width and height, we need to compute the values based on the kernel filter size, stride movement

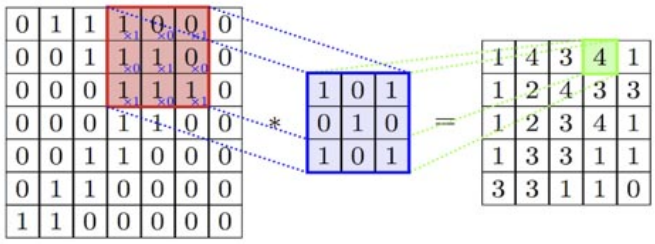


Figure 2.7: CNN kernel filter example (Chatterjee, 2017).

and padding around the image as shown in equation 2.30.

$$\text{output} = \left[\frac{\text{input} + 2 \times \text{padding} - \text{kernel}}{\text{stride}}\right] + 1 \tag{2.30}$$

An important statement is the notion of *effective receptive field* of neurons within the hidden layers, since they indirectly determine the representation capabilities at each layer (Luo et al., 2017). In contrast to the receptive field, which is the field of view for each neuron towards its input space, the effective receptive field can be computed for subsequent layers to establish how much a neuron perceives and hence contributes to the output. It has been analyzed that the units at the center of the input image contributed more to the overall outcome. The effective receptive field is asymptotically Gaussian distributed from the center and only takes a fraction of the theoretical available receptive field. One interesting analysis of Luo et al., 2017 is that skip connections in neural networks decrease the effective receptive field, and currently, subsampling and dilated convolutions are an effective way to increase the effective receptive field. Dilated convolutions are a special type of convolution, where the filter mask allows to skip input values, such that a sparse mask is convolved over the input space (Yu and Koltun, 2015).

2.3.4 Batch Normalization

The depth of modern artificial neural architectures has greatly grown over the past years as seen in deep Residual Networks (ResNet) with greater than 1000 layers (He et al., 2015). One key component which enabled this growth is batch normalization. Neural networks stack layer-wise operations to extract meaningful information from their input space. Since each layer's output depends on the predecessor's values and the distributions of these values shift continuously during training, we need small learning rates to stabilize the training process and are highly dependent on the initial weight conditions. The deeper we build the networks the harder they become to stabilize the distributions of activations during training. To counter this effect Ioffe et al. introduced batch-wise input value normalization, which is applied during training on each mini-batch (Ioffe and Szegedy, 2015). First they compute the mini-batch mean such that

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{2.31}$$

where $\mathcal{B} = \{x_{1...m}\}$ represents the range of mini-batch values. Then they compute the mini-batch variance as

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}}). \tag{2.32}$$

Then they normalize the values by computing

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{2.33}$$

where ϵ is a constant for numerical stability. Additionally, they introduce two new trainable parameters γ and β, which rescale and shift the normalized value such that

$$y_i \leftarrow \gamma\hat{x}_i + \beta. \tag{2.34}$$

This normalization allows higher learning rates, makes the network less sensible to parameter initialization and speeds up convergence, even if the values are not decorrelated (Montavon, Orr, and Müller, 2012). To combat inference, they simply perform a linear transformation and need to fix the mean and variance and they require a minimum batch dimension.

2.3.5 Layer Normalization

In contrast to batch normalization, this method is not dependent on the mini-batch size and was introduced to help to stabilize the hidden state of recurrent neural networks (Ba, Kiros, and Hinton, 2016). They, therefore, compute the layer normalization statistics for all hidden units such that the layer mean is obtained by computing

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{2.35}$$

and the layer standard deviation is obtained by computing

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}, \tag{2.36}$$

where H the number of hidden units in a layer. All hidden units share the same normalization μ and σ. This also lifts the constrain of batch normalization and allows a batch size of 1.

2.3.6 Dropout

According to Hinton et al. one of the causes of feedforward neural networks to overfit are neurons learning co-adaptive features, which are only helpful in the context of other neurons (Hinton, Srivastava, et al., 2012). When training, he proposed to randomly drop out a percentage of neurons, forcing each neuron to learn a feature detector that

is in general meaningful to the task and not to specifics according to other activations. Equation 2.37 shows the dropout scheme for the activations h.

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases} \tag{2.37}$$

When testing, all neurons are set active, but the activations are normalized by the dropout factor to compensate the greater activation values. This method has similarities between training multiple neural networks based on the same weight space and averaging their predictions.

2.3.7 Embedding

Embeddings are usually used in natural language processing where the similarity between words is computed according to some defined metric on the feature vectors, such as the cosine similarity (PyTorch Tutorials, 2017). Similar feature vectors result in the same embedding representation and therefore in the same region of the vector space. This allows functional transformations, were, for example, the word *king* minus the word *men* plus the word *women* results in the word *queen*. In deep learning the word vector mapping is usually modeled using a few preceding and following words as a context to predict the currently required word. This is known as the Continuous Bag-of-Words model. This relational mapping is not only helpful in cases of word embeddings, but was also often applied to create feature mappings projecting attention masks, or other encodings helpful for neural network modeling.

2.3.8 Attention

Attention is a technique that gained popularity with the proposal of the Neural Machine Translation paper (Luong, Pham, and Manning, 2015), which offered a gating mechanism for the encode-decoder state and enabled an automatic soft-search mechanism. This can be applied as a hard—either zero or one—or a soft attention mask with values continuously ranging from zero to one. It is often implemented using Softmax non-linearity to enforce hight regularization, since the values have to sum up to one. In the case of sigmoid-based attention with trainable parameters such as embeddings, we use L1 regularization to emphasize sparsity. Attention-based models not only showed outstanding results on language translation tasks (Bahdanau, Cho, and Bengio, 2015) but also by applying them on convolutional neural networks and lately in the convolutional self-attention model (B. Yang et al., 2019). Chapter 3 will introduce "Hard Attention to the Task" (Serrà et al., 2018), which uses hard attention masks to filter activations to overcome catastrophic forgetting.

2.4 Architectures

After the comeback of neural networks in 2005—based on LeNet developed by LeCun, Bottou, et al., 1998—many architectures were proposed. The most prominent ones are AlexNet from the University of Toronto (Krizhevsky, Sutskever, and Hinton, 2012), InceptionNet from Google (Szegedy et al., 2014), VGGNet from the University of Oxford

(Simonyan and Zisserman, 2014), ResNet from Microsoft Research (He et al., 2015) and DenseNet from Facebook AI Research and the Cornell University (Huang, Z. Liu, and Weinberger, 2016). These architectures mainly differ in three components: the amount of weights, the wiring of skip connections and the parallel computations paths. VGGNet-19—which is similar to AlexNet, but offers much deeper layer structure—has the largest amount of parameters, which makes it the most difficult to regularize. InceptionNet first used parallel pathways to compute not only one type of convolution, but a mixture of 1×1, 3×3 and 5×5 convolutions in parallel. Furthermore, it included max pooling and concatenated all outputs for the next layer computation. ResNet introduced residual identity mappings, allowing for one pathway to compute convolutions and the other to concatenate the original activations with the transformed activations. DenseNet brought multiple skip connects together which not only use the previous layer activations, but almost all previous layer activations. All these improvements allow to preserve the original information for higher layers but still enable meaningful transformations and also mainly improve the gradient flow during the backward pass. AlexNet offers no skip connections or parallel computation paths and is simply computing layer-wise transformations. This makes it ideal for evaluating improvements, such as vanishing or exploding gradients, without introducing unwanted side effects. Although it is a simplistic architecture it first showed a significant improvement at the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 and has been often used as a reference architecture to evaluate new neural network adaptions with state-of-the-art performance. In this section we will introduce the main architecture of AlexNet, which will be used in all subsequent chapters, since all related papers and experiments are based on AlexNet and we want to conclude on a comparable baseline.

2.4.1 AlexNet

AlexNet mainly stacks three convolutional layers which are altering between convolution, ReLU activation, Dropout, and Max Pooling and combines the spacial invariant features in the two top fully connected layers. For the tests conducted in this thesis, we will mainly use the AlexNet with Dropout architecture (Goodfellow, Mirza, et al., 2014) since all the related papers conducted experiments with the same architecture and we want to remain comparable to existing benchmarks. The datasets used are CIFAR-10 and CIFAR-100, the input space consists of $32 \times 32 \times 3$ pixels (3 corresponds to the RGB color channels). This results in the stacked convolutions and hyperparameters as shown in Listing 2.1. whereas *in* represents the channel input dimensions, *out* the channel output dimensions, k the kernel size, s the strides, p the paddings in the context of convolutional layers and in the context of dropout layers the dropout probability. The dimension of the last layer is dependent on the defined number of classes for each task t.

2.5 Datasets

This section introduces the visual datasets that are used to conduct the experiments and assessing the improvements. It was chosen according to the papers introduced in chapter 3 to compare and conduct the experiments which will be presented in chapter

1. Input: $x = 32 \times 32 \times 3$
2. Convolution: in $= 3$, out $= 64$, $k = 4 \times 4$, $s = 1$, $p = 0$
3. ReLU
4. Dropout: $p = 0.2$
5. Max Pooling: $k = 2 \times 2$
6. Convolution: in $= 64$, out $= 128$, $k = 3 \times 3$, $s = 1$, $p = 0$
7. ReLU
8. Dropout: $p = 0.2$
9. Max Pooling: $k = 2 \times 2$
10. Convolution: in $= 128$, out $= 256$, $k = 2 \times 2$, $s = 1$, $p = 0$
11. ReLU
12. Dropout: $p = 0.5$
13. Max Pooling: $k = 2 \times 2$
14. Fully Connected: in $= 1024$, out $= 2048$
15. ReLU
16. Dropout: $p = 0.5$
17. Fully Connected: in $= 2048$, out $= 2048$
18. ReLU
19. Dropout: $p = 0.5$
20. Fully Connected: in $= 2048$, out $=$ number of classes per task t,

Table 2.1: AlexNet layer operation stack.

6. In general, CIFAR-10 and CIFAR-100 are labeled as subsets of the 80M tiny images dataset (University of Toronto Website, 2009). The advantage of using a visual dataset is that the data is human understandable and methods such as attention can be visually analyzed by observing the resulting activation space.

2.5.1 CIFAR-10

CIFAR-10 consists of 60000 color images with a dimension $32 \times 32 \times 3$, where the first two dimensions denote the 2-D pixel plane and the last dimension the RGB channels. The 60000 images are divided into 10 classes of each 6000 samples. Furthermore, 50000 images are intended for training and 10000 for testing. The test batches contain 1000 randomly selected images from each class. Figure 2.8 gives an example of 10 random images from each class.

2.5.2 CIFAR-100

This dataset is similar to CIFAR-10, except that it consists of 100 different classes containing 600 images per class (University of Toronto Website, 2009). Each class consists of 500 training samples and 100 test samples. The 100 classes in CIFAR-100 are grouped into 20 main classes, such as that a main class fish contains sub-classes aquarium fish,

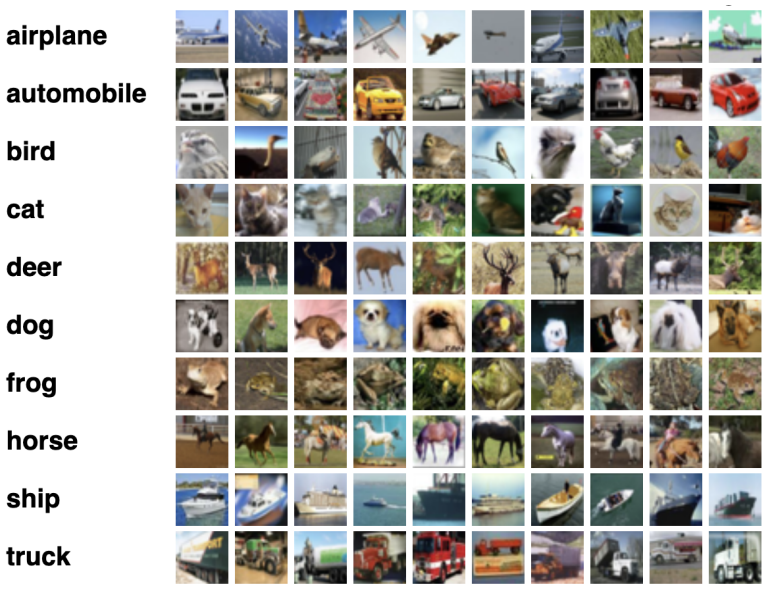


Figure 2.8: 10 random images per class from CIFAR-10 (University of Toronto Website, 2009).

flatfish, ray, shark, etc.

Chapter 3

Related Work

> "You don't understand anything until you learn it more than one way."
>
> *Marvin Minsky*

This chapter will introduce several related approaches targeting the issue of catastrophic forgetting. An overview of the obtained results and a comparison will be offered in the next chapter 4.

3.1 Progressive Neural Networks

Progressive Neural Networks (PNN) proposed by DeepMind is an approach to prevent catastrophic forgetting by dynamically extending the network capacity with each task respectively (Rusu et al., 2016). In simple transfer learning a network is adjusted by first freezing the weights of all lower layers and typically only fine-tuning the parameters of the output layer, to perform well on new task. Not only does this disrupt the parameters for all previously trained tasks, but also assumes that the new task has representational overlap within the extracted features from the frozen layers. This might not be true, since new tasks can have orthogonal feature representations and might not be fully describable by the obtained feature set. To enable knowledge transfer between tasks and allow enough adaptiveness to learn new task specifics, they define the set of weights of a neural network as *columns* and with every added task they extend the previous columns by introducing additional weights. Starting with one column, consisting of L layers with n_i number of units, such that each layer is defined as $h_i^{(1)} = \mathbb{R}^{n_i}$, they train the network parameters $\Theta^{(1)}$ until convergence. Before switching to a new task, the weights of the previous task $\Theta^{(1)}$ are frozen and the columns are enlarged with a set of additional weights $\Theta^{(2)}$. These additional weights offer lateral connection to previous columns, such that $h_i^{(2)}$ receives input from $h_{i-1}^{(2)}$ and $h_{i-1}^{(1)}$, which ensures that at each layer the previously trained weights remain preserved and only additional features are learned. Note that the expansion factor is smaller than the initially defined weights, since they reuse previous columns features. Illustration 3.1 gives an overview of a three column neural network. Equation 3.1 offers a more formal description.

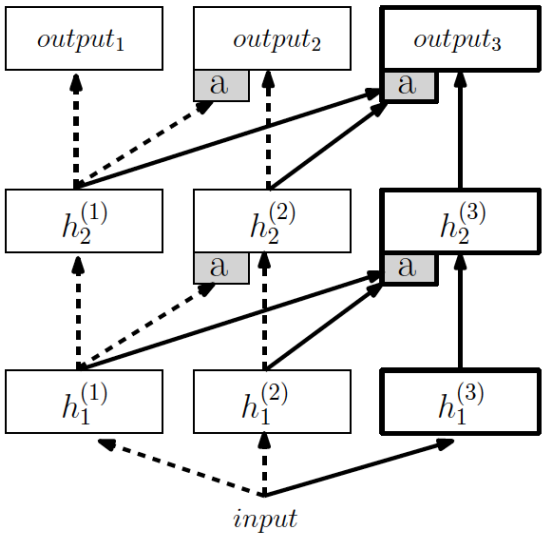


Figure 3.1: The first dashed column represents the initial network and the right handed columns include lateral connections to their predecessors, reusing their activations for further computation (Rusu et al., 2016).

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right), \tag{3.1}$$

whereas $W_i^{(k)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the weight matrix of layer i at column k, $U_i^{(k:j)} \in \mathbb{R}^{n_j \times n_j}$ represents the lateral connections of layer $i-1$ of column j, to layer i of column k. $h_0^{(.)}$ defines the input layer and in the implementation of PNN uses ReLUs for the non-linearity operation with $f = \max(0, x)$. In practice, the lateral connections will grow with the number of tasks and hence to reduce the increase of computation they apply a non-linearity mapping of previous features. This is achieved by projecting the anterior features as

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + U_i^{(k:j)} f(V_i^{(k:j)} \alpha_{i-1}^{(<k)} h_{i-1}^{(<k)})\right), \tag{3.2}$$

where they define $h_{i-1}^{(<k)} = [h_{i-1}^{(1)} \dots h_{i-1}^{(j)} \dots h_{i-1}^{(k-1)}]$ as a single vector of dimensionality $n_{i-1}^{(<k)}$, $V_i^{(k:j)} \in \mathbb{R}^{n_{i-1} \times n_{i-1}^{(<k)}}$ as the projection matrix and $\alpha_{i-1}^{(<k)}$ as a learnable scaling factor compensating for the different input scales. For dimensionality reduction of convolutional layers, 1×1 convolutions are applied and for fully connected layers they replace the linear lateral connections by a multilayer perceptron (MLP).

The bottom line 3.1.1 *Since the weights of previously trained task are never altered, this approach is innate to catastrophic forgetting by design. But the number of weights also grows with the number of tasks, which defines a limit to the maximum number of computation and memory one can provide. Furthermore, it is not guaranteed that the introduced activation noise from the lateral connection, does not*

disrupt learning of new tasks, if the number of tasks grows very large.

3.2 Elastic Weight Consolidation

Based on neuro-biological studies, which consider computational principles how memories are stored and retained through synaptic plasticity and the emergent neural interactions (Benna and Fusi, 2016), DeepMind (Kirkpatrick et al., 2017) proposed an algorithm that claims to imitate the excitatory task-specific synaptic consolidation of the mammalian brain. It is believed that the neocortex permanently memorizes previously learned tasks by adjusting the plasticity of its synapses enforcing the retention of important skills over long timescales. Training neural networks is achieved by optimizing a set of parameters θ over multiple iterations w.r.t. the data set $\mathcal{D}$. According to Hecht-Nielsen, 1989, many parameter configurations can be found, such that θ will result in similar performance. One can assume that over-parametrization enables us to find a solution that suits multiple tasks in a common sub-space. Given a two task problem, optimizing on task B with a parameter set θ_B^*, we can search for a close optimal solution compared to the previously found solution of task A, with parameters θ_A^*, such that by constraining the training procedure of task B to follow a path of low error on both tasks. This constrain can be implemented as quadratic regularization penalty. Figure 3.2 illustrates a schematic overview of Elastic Weight Consolidation (EWC). Viewing

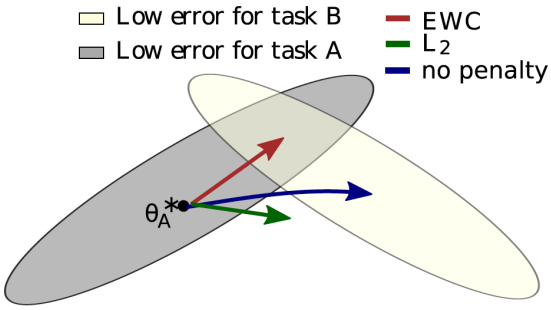


Figure 3.2: Illustration of the effect of EWC (red) on the weights updated, compared to L2 regularization (green) and no regularization (blue). It can be seen that EWC enforces the weights of θ_B^* to move in a region where θ_A^* and θ_B^* overlap. If simple L2 regularization is applied, this results in a sub-optimal update for both tasks. With no regularization term, only task B is favored and everything about task A will be forgotten (Kirkpatrick et al., 2017).

the training process of a neural network from a probabilistic perspective, gives the justification for the assumption of continuous learning so that we can train on both tasks sequentially. Due to Bayes' rule, the problem can be framed as a conditional probability $p(\theta|\mathcal{D})$ such that:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{3.3}$$

where $p(\theta)$ forms the prior probability w.r.t. the parameters trained on previous tasks and $p(\mathcal{D}|\theta)$ the likelihood of the data, and with the log-likelihood $\log p(\mathcal{D}|\theta)$ represent-

ing the negative of the loss function $-\mathcal{L}(\theta)$. Assuming that $\mathcal{D}$ can be split into two independent data sets $\mathcal{D}_A$ and $\mathcal{D}_B$, the above statement can be rephrased as:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B). \tag{3.4}$$

$\log p(\theta|\mathcal{D})$ is still describing the posterior probability of the entire dataset $\mathcal{D}$, yet the loss function to optimize on task B only depends on $\log p(\mathcal{D}_B|\theta)$, such that the posterior of task A is entirely encoded in $\log p(\theta|\mathcal{D}_A)$, therefore it is important to constrain the update on important weights of task A. Usually, the posterior is intractable and requires an approximation to be applicable. In the EWC paper a Laplace approximation was applied, originally proposed by MacKay, 1998, which approximates posteriors as a Gaussian distribution with mean θ^*_A and the diagonal precision of the Fisher information matrix F. The Fisher information matrix is easier to compute since it can be obtained through the first-order derivation during backpropagation. Furthermore, it represents the second-order derivative of a loss near a minimum with guaranteed positive semi-definite outcome. This approach is similar to the expectation propagation method. Each subtask can be defined as a factor of the posterior, which approximates an intractable probability distribution $p(\boldsymbol{x})$ with a tractable probability distribution $q(\boldsymbol{x})$. This is done by minimizing the Kullback-Leibler divergence $D_{KL}(p \mid\mid q)$ between them. By applying these approximations, we can state the new loss function as:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2, \tag{3.5}$$

where $\mathcal{L}_B(\theta)$ defines the loss function for task B and λ defines a hyperparameter, which establishing the rate of importance of the previous task A. This approach can be generalized by optimizing on a third task C, results in applying two separate penalties for A and B respectively or by summing up their penalties.

The bottom line 3.2.1 *This approach finds common weight settings of multiple tasks, which stabilizes the network updates on future tasks to reuse pre-computed features. The drawback is that it does not guarantee the avoidance of gradual forgetting over long training sequences on different tasks. Additionally, the Fisher information matrix is a point wise estimate of a trained task and does not provide additional information of the training trajectories.*

3.3 Less-Forgetting Learning

Jung et al., 2016, proposes a method related to transfer learning, which is more robust to catastrophic forgetting. The idea is to train a network and copy the pre-trained weights of the source network as initial weights for the target network. Before training on the new task they freeze the parameters of the softmax classifier and allow fine-tuning of the shared parameters. Afterwards they try to preserve the original activation distributions to match the classifier of the source network, while optimizing on the target network data as visualized in Figure 3.3

Given a multi-headed network output, where each classifier head represents a set of target classes for a given task, they first require an estimate of all previous layer

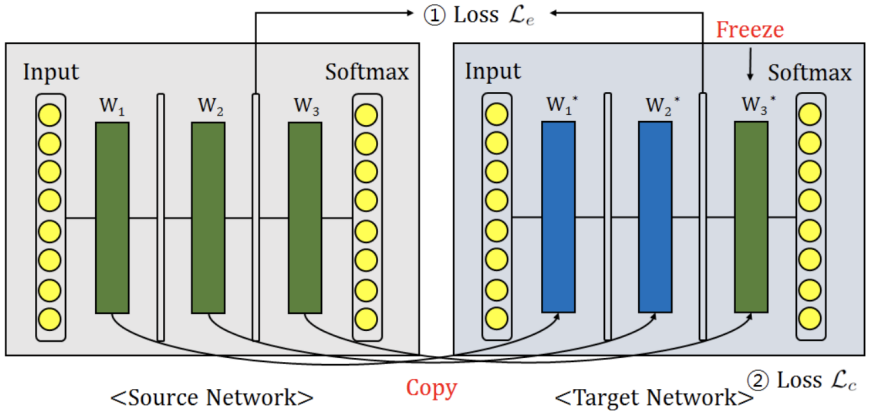


Figure 3.3: Schematic for describing less-forgetting learning (Jung et al., 2016).

probability distributions and after the optimization step, they try to reconstruct the same probabilities. The lower layers operate as mere feature extractors and the last layers operate as a linear classifier, usually implemented by a softmax function. This means that the extracted features obtained according to the weights before the softmax layer establish the decision boundaries to differentiate between the target classes. The features are an abstracted higher dimensional manifold of the original input data. This allows the softmax layer to easily perform a linear separation. To avoid disruptive updates of previous tasks the resulting decision boundaries must remain unchanged. After switching to a new task the source data is unavailable to verify the decision boundaries. By forwarding the target data activations to the source classifier Less-Forgetting Learning (LFL) fine-tunes parameters and regularizes the updates by measuring the shift in distributions. To correct the shift of feature representations, while optimizing on the new task, the following regularization term was applied to the loss function:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \hat{\boldsymbol{y}}; \theta^{(s)}, \theta^{(t)}) = \lambda_c \mathcal{L}_c(\boldsymbol{x}, \boldsymbol{y}, \hat{\boldsymbol{y}}; \theta^{(t)}) + \lambda_e \mathcal{L}_e(\boldsymbol{x}; \theta^{(s)}, \theta^{(t)}), \tag{3.6}$$

where $(x, y) \in \mathcal{D}^{(t)}$ are input and label samples from the target data set. $\hat{y}$ denotes the predictions from the target network, $\mathcal{L}_c$ and $\mathcal{L}_e$ represent the cross-entropy and Euclidean loss function respectively. $\theta^{(s)}$ and $\theta^{(t)}$ are the source and target weights. λ_c and λ_e are hyperparameters to regulate the task importance. The cost function for the cross-entropy loss is defined as:

$$\mathcal{L}_c(\boldsymbol{y}, \hat{\boldsymbol{y}}; \theta^{(t)}) = -\sum_{i=1}^{m} y_i \log(\hat{y}_i(x_i; \theta^{(t)})), \tag{3.7}$$

where m refers to the number of classes for the new target task. The cost function for the Euclidean loss is defined as:

$$\mathcal{L}_c(\boldsymbol{x}; \theta^{(t)}) = \frac{1}{2} \left\| \boldsymbol{v}_{L-1}(\boldsymbol{x}; \theta^{(s)}) - \boldsymbol{v}_{L-1}(\boldsymbol{x}; \theta^{(t)}) \right\|^2, \tag{3.8}$$

here $\boldsymbol{v}_{L-1}$ denotes the resulting feature vectors for the $L-1$ layer. If the number of tasks increases and requires to learn orthogonal features, this approach shows a quick decrease in performance, which will be further explore in the next chapter 4.

The bottom line 3.3.1 *According to the objective function, the authors of LFL demonstrate a novel approach to maintain similar features, which are required by previous task classifiers to maintain performance. This is achieved by correcting on target output probabilities while training on the new target data.*

3.4 Learning Without Forgetting

Learning Without Forgetting (LWF) (Li and Hoiem, 2017) is a novel sequential learning method and offers some similarities to the previously introduced LFL method. First, we divide the neural network weights into three main parts: $\theta_o, \theta_s, \theta_n$, which represents the old, shared and new weights respectively. Similar to LFL Li et al. trains the model by using only new data samples and regularizes the classifier distributions of old task classifiers. The task-specific parameters of the new and old tasks are usually fully connected layers at the top of the neural network. The set of shared parameters, which are feature extractors, are usually convolutional layers. When adding a new task, one also introduces new task-specific parameters θ_n to tune the distribution values with the output of the shared parameters and record the output of old task-specific parameters. The output layers are usually implementing a softmax layer, therefore, the output is usually a probability distribution over all task-specific classes. Afterwards, they jointly train all parameters with the addition of a regularization term to preserve old probabilities, similar to LFL. This approach mainly differs from jointly training a neural network on all tasks by only using the data from the new task, and estimating the distribution from the output layers of old tasks. Unlike in LFL, when training on a new task, the training procedure first freezes the weights of θ_o and θ_s, and updates only θ_n until convergence (warm-up step), and afterwards, all remaining parameters are jointly fine-tuned. Their experiments used a multinomial logistic loss:

$$\mathcal{L}_{\text{new}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\boldsymbol{y}_n \log \hat{\boldsymbol{y}}_n, \tag{3.9}$$

where $\boldsymbol{y}_n$ is the one-hot encoded ground truth label vector and $\hat{\boldsymbol{y}}_n$ the network output. To compute the loss between the recorded output of the original network and the current output, the proposed Knowledge Distillation loss from Hinton, Vinyals, and Dean, 2014, was used, which is a modified cross-entropy loss. This put more emphasize on smaller probabilities and showed better performance, compared to LFL:

$$\mathcal{L}_{\text{old}}(\boldsymbol{y}_o, \hat{\boldsymbol{y}}_o) = -H(\boldsymbol{y}'_o, \hat{\boldsymbol{y}}'_o) \tag{3.10}$$

$$= -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}, \tag{3.11}$$

where l is the number of labels and $y_o'^{(i)}, \hat{y}_o'^{(i)}$ are the recorded and current version of the probabilities, $y_o^{(i)}, \hat{y}_o^{(i)}$, defined as:

$$y_o'^{(i)} = \frac{\sqrt[T]{y_o^{(i)}}}{\sum_j \sqrt[T]{y_o^{(j)}}}, \qquad \hat{y}_o'^{(i)} = \frac{\sqrt[T]{\hat{y}_o^{(i)}}}{\sum_j \sqrt[T]{\hat{y}_o^{(j)}}}, \tag{3.12}$$

with $T > 1$, which increases the weight of small logit values, encoding better similarities. The experiments evaluated that $T = 2$ worked best. This defines the objective of LWF as:

$$\underset{\theta_o,\theta_s,\theta_n}{\operatorname{argmin}} = \lambda\mathcal{L}_{\text{old}}(Y_o, \hat{Y}_o) + \mathcal{L}_{\text{new}}(Y_o, \hat{Y}_o) + \mathcal{R}(\theta_o, \theta_s, \theta_n), \tag{3.13}$$

where λ denotes a measure of the importance of old tasks; if $\lambda > 1$ then the old tasks are favored and vice versa. In the case of multi-label classification or training multiple tasks at once, the losses are simply summed up. To minimize the loss for all tasks, they use stochastic gradient descent with weight decay of 0.0005.

The bottom line 3.4.1 *Although Less-Forgetting Learning and Learning Without Forgetting seem very similar, the authors of LWF show a more robust learning curve on multiple tasks. The issue of learning many tasks sequentially remains the same, since both LFL and LWF approaches try to track and correct multiple output distributions of long task sequences.*

3.5 Incremental Moment Matching

The authors of Incremental Moment Matching (IMM), proposed two methods (mean-IMM and mode-IMM) for incrementally matching the moments of a posterior distribution, when training sequentially on different tasks (Lee et al., 2017). Mean-IMM averages the parameter moments of multiple networks and mode-IMM uses the covariance information of the posterior of Gaussian distribution to match the moments. These methods have some parallels to EWC. Both methods alter the objective function by applying a regularization term to constrain forgetting when learning new tasks. Furthermore, these approaches can also be interpreted as an approximation of a sequential Bayesian learning method. Assuming that the posterior distribution of the random variables (network parameters) can be combined into a mixture of Gaussian (MoG) posterior according to each task and further on to a MoG representing all tasks. This results in combining their partial objective functions into one convex-like objective. Figure 3.4 illustrates the weight transition w.r.t. the regularized objectives. The assumption of representing all tasks as mixture of Gaussians posts a strong constrain on the trained parameters and has great implications on the behavior of the loss function. In fact, this assumption does not hold, since most datasets are not Gaussian distributed and since the model parameters can result in non-convex loss behavior. But in practice Le et al. showed that IMM offers great performance and validates the assumptions. Furthermore, they propose three different IMM implementations using transfer learning technique, such as weight-transfer, L2-transfer and drop-transfer to increase performance while training.

3.5.1 Matching Posterior Distributions

To approximate the posterior distribution $p_{1:K}$ of the network parameters θ via a Gaussian approximation $q_{1:K}$ for all K tasks combined, they evaluate the optimal $\mu^*_{1:K}$ and $\Sigma^*_{1:K}$, so that they approximate

$$p_{1:K} \equiv p(\theta|X_1, \ldots, X_K, y_1, \ldots, y_K) \approx q_{1:K} \equiv q(\theta|\mu_{1:K}, \Sigma_{1:K}), \tag{3.14}$$

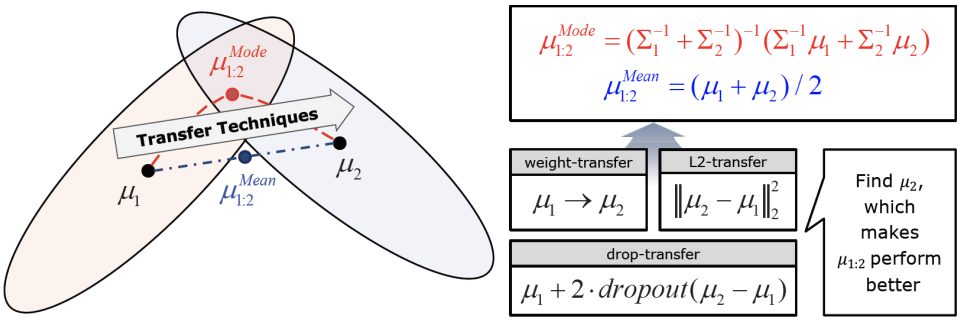


Figure 3.4: Geometric illustration of the incremental moment matching (IMM) method using two Gaussian mixtures to match the moments by regularizing the loss. mean-IMM averages the weights and mode-IMM finds the maximum of the mixture of Gaussian posteriors (Lee et al., 2017).

where $X_1, \ldots, X_K$ represents the input features of task $k = 1 \ldots K$ and $y_1, \ldots, y_K$ the targets of each task respectively. Since the distributions are matched in an incremental way, they define

$$p_k \equiv p(\theta|X_k, y_k) \approx q_k \equiv q(\theta|\mu_k, \Sigma_k), \tag{3.15}$$

optimizing for the training set (X_k, y_k). $\theta \in \mathbb{R}^d$, where d represents the number of trainable network parameters, which concludes in $\mu_k \in \mathbb{R}^d$ and $\Sigma_k \in \mathbb{R}^{d \times d}$.

3.5.2 Mean-based IMM

The objective function of the mean-IMM is to minimize the Kullback-Leibler divergence between q_k and $q_{1:K}$ by computing

$$\mu_{1:K}^*, \Sigma_{1:K}^* = \underset{\mu_{1:K}, \Sigma_{1:K}}{\operatorname{argmin}} \sum_{k=1}^{K} \alpha_k D_{KL}(q_k \,||\, q_{1:K}), \tag{3.16}$$

$$\mu_{1:K}^* = \sum_{k=1}^{K} \alpha_k \mu_k, \tag{3.17}$$

$$\Sigma_{1:K}^* = \sum_{k=1}^{K} \alpha_k \left(\Sigma_k + (\mu_k - \mu_{1:K}^*(\mu_k - \mu_{1:K}^*)^T) \right). \tag{3.18}$$

$\mu_{1:K}^*$ and $\Sigma_{1:K}^*$ represent the optimal solution of the KL-divergence and α_k a mixing ratio with $\sum_{k=1}^{K} \alpha_k = 1$, which weights the contribution of each task equally. After each task they average over the current and previous model parameters. When training on a new task the old model parameters are used as a L2 regularization, preventing large parameter updates.

3.5.3 Mode-based IMM

Unlike mean-IMM, mode-IMM uses the covariance information Σ_k of the posterior of Gaussian distributions. To obtain the mode while operating only on two parameter sets (current and previous task parameters) the Ray and Lindsay method was applied showing that any D-dimensional, K-component normal mixtures, can be defined on a $(K-1)$-dimensional hypersurface, which is guaranteed to include all critical points, including the modes of the D-dimensional mixture density (Ray and Lindsay, 2005). So all the modes for θ can therefore be given by

$$\left\{ \theta \;\middle|\; \theta = \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \mu_k \right)^{-1} \cdot \sum_{k=1}^{K} \alpha_k \Sigma_k^{-1},\ 0 < \alpha_k < 1 \text{ and } \sum_k \alpha_k = 1 \right\}. \tag{3.19}$$

Although there is no tight upper bound on the number of modes for the MoG, there is a guess that for all $D, K \geq 1$ it is ${}_{D+K+1}C_D$ (Améndola, Engström, and Haase, 2017). Given the MoG, one defines an approximation using the Laplacian method, where the logarithm of the function is expressed by the Taylor expansion:

$$\log q_{1:K} \approx \sum_{k=1}^{K} \alpha_k \log q_k + C \tag{3.20}$$

$$\approx -\frac{1}{2} \theta^T \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \right) \theta + \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \mu_k \right) \theta + C'. \tag{3.21}$$

This allows to compute the parameters $\mu^*_{1:K}$ and $\Sigma^*_{1:K}$:

$$\mu^*_{1:K} = \Sigma^*_{1:K} \cdot \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \mu_k \right) \tag{3.22}$$

$$\Sigma^*_{1:K} = \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \right)^{-1} \tag{3.23}$$

When implementing equation 3.23 the expression was inverted by multiplying with the identity matrix I and a constant scaling factor ϵ:

$$\left(\Sigma^*_{1:K} \right)^{-1} = \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \right)^{-1} \cdot \epsilon I. \tag{3.24}$$

Since computing the covariance matrix gives the runtime complexity of $\mathcal{O}(D^2)$, by adding the assumption that the parameters of the covariance matrix are not correlated, they show that it reduces the tractable dimensions to its diagonal. This results in a runtime complexity of $\mathcal{O}(D)$. To compute the covariance, the inverse of the Fisher information matrix was used, which represents the variance of the score, whereas the score is defined as the gradient of the log-likelihood w.r.t. θ. It adjusts the sensitivity of the likelihood function according to θ. The Fisher information matrix of the k-th task is given by:

$$F_k = E\left[\frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k) \cdot \frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k)^T \right], \tag{3.25}$$

where $\tilde{y} \sim p(y|x, \mu_k)$ and $x \sim \pi_k$, with π_k denoting a empirical distribution of X_k. The Fisher information matrix is used after each training process to update the trained parameters according to the previous and current parameter importance measure. This is done by preserving the previous task parameters and using the fine-tune parameters according to the new task. During training large updates are prevented through L2 regularization to previous tasks.

3.5.4 Transfer Techniques

In practice they proposed the usage of one of the following transfer learning techniques to improve learning when switching to a new task: weight-transfer, L2-transfer or drop-transfer.

Weight-transfer

The IMM paper uses by default weight-transfer for their experiments. This basically initializes the parameters for task μ_k with the values of the previous parameters μ_{k-1}. This method interpolates the solutions of multiple tasks on a surface merging all solutions into one, and allowing the loss function to behave nearly convex. These conclusions were drawn based on empirical analysis from Goodfellow, Vinyals, and Saxe, 2014, according to linear path analysis, where each loss and accuracy was evaluated w.r.t. a series of parameters such that $\theta = \theta_1 + \beta_1(\theta_2 - \theta_1) + \cdots + \beta_{k-1}(\theta_k - \theta_{k-1}) + \cdots + \beta_{K-1}(\theta_K - \theta_{K-1})$.

L2-transfer

L2-transfer is a descendant of L2-regularization and can be interpreted as a special case of EWC, where the prior distribution is a Gaussian with a covariance matrix λI. Unlike usual cases, where λ is chosen relatively high, they apply a very small value for λ, such that it defines a regularization term which smoothens the loss function, defining a convex-like space between μ_k and μ_{k-1}. This leads to the following objective function:

$$\ell(\theta) = \log p(y_k|X_k, \mu_k) - \lambda \left\|\mu_k - \mu_{k-1}\right\|^2. \tag{3.26}$$

Drop-transfer

Drop-transfer is a variant of dropout, where the zero values are replaced by the values of μ_{k-1}. During training, $\hat{y}_{k,i}$ replaces the weight vector of $y_{k,i}$, such that:

$$\hat{y}_{k,i} = \begin{cases} \mu_{k-1,i} & \text{if } i\text{-th node is turned off} \\ \frac{1}{1-p}\mu_{k,i} - \frac{p}{1-p}\mu_{k-1,i} & \text{otherwise} \end{cases}, \tag{3.27}$$

where p denotes the dropout ration. According to Srivastava et al., 2014, dropout can be interpreted as an ensemble of weak learners, hence they use this property to smoothen the transition between tasks, when optimizing w.r.t. the parameters of previous tasks. This gives them an average oversampled prediction, which they compensate during testing by multiplying the activations with the inverse of the dropout rate, since the output distribution of the ensemble is intractable.

The bottom line 3.5.1 *Incremental Moment Matching is a very effective method and offers deep theoretical insights on the geometrical properties when overlapping the moments while training on different tasks. But the moment mixture of Gaussians assumption is very strong for a large number of tasks, and training is also limited to an offline training procedure.*

3.6 PathNet

In 2017, DeepMind (Fernando, Banarse, et al., 2017) published a paper which was inspired by the Darwinian Neurodynamics designing how evolutionary algorithms might be implemented in the brain (Fernando, Szathmáry, and Husbands, 2012). Since this approach is partially determined by combinatorial heuristics, they focused their research on combining gradient-based optimization of neural networks with the usage of evolutionary algorithms to select the optimal pathways based on a new solution candidate of the sub-networks population. The layers are divided into smaller layer modules, where the selected modules form a task-specific network. By searching through multiple gradient optimized parameter populations when learning a new tasks this method showed not only efficient but also scalable results. Their core motivation is that multiple users do not have to start training from scratch, but are rather able to train on the same giant network. It uses the shared knowledge across tasks and finds different paths to solve a newly defined problem. To enable a distributed search of pathways, evolutionary based agents are used, which can work in parallel, learning disjoint weight updates and sharing these weights with other agents.

3.6.1 Modules

PathNet consists of a modular deep neural network having L layers, where each layer has M modules. Each module is composed of a neural network, containing convolutional or linear layers and a transfer function to match the input to outputs between modules. When forwarding activations through the network, only *active* modules are selected for processing, and each module output of a given layer is summed with its companion modules at the same layer, before passed on to the next active module of the next higher layer. A module is called active if it is present in the currently evaluated genotype. In the case of PathNet this is expressed in a matrix which enables feature extractors from layer modules. Each layer has a limited number of active modules, usually consisting of $N = 3$ or 4 entries. The final output layer, uses a multi-headed approach containing linear classifiers for each task respectively. The data structure which represents the active genotypes or pathways, is defined as a $N \times L$ dimensional matrix, storing integer values, for each layer respectively. Evolutionary algorithms are mainly defined by four phases, known as selection, recombination, mutation and fitness evaluation, they need to first train the networks and use the established error for the fitness evaluation. PathNet defines the selection phase by applying tournament selection, whereas two alleles compete against each other based on their evaluated fitness. In a broader context the recombination phase can be seen as randomly initialized and fine-tuned parameters of the neural network. The mutation phase is randomly permuting weights of trained networks, such as dropout or random inductive noise, to ensure a higher population diversity. To

evaluate the fitness, the negative classification error is used at each training step. The training procedure starts with randomly initializing P genotypes for each layer, then by selecting a genotype at random, and training its pathways for T epochs. This is followed by randomly selecting a second genotype which is also trained for T epochs. Afterwards, a copy of the winning pathways genotype overwrites the pathways of the losing one, after which it is mutated with a probability of $\frac{1}{N \times L}$, which adds an integer in the range of $[-2, 2]$ to it. Additionally, a local neighborhood search was applied, promoting spacial localization of network functionality. This training and evaluation procedure can be parallelized, since each worker only operates on its own parameter subset. The synchronization step is only required to compete in the tournament selection. Unless workers have finished, they write a large negative fitness value into the shared fitness array, to avoid being selected. When a worker has finished, it randomly selects B other genotypes and if only one is at least as fit or fitter than the current genotype, it overwrites its current genotype, otherwise, it continues to evaluate its own genotype. After a given number of epochs or after reaching a given threshold, the best pathways are selected for the current task and its weights are then fixed. All other parameters not relying in an optimal path are reinitialized and the training procedure is repeated for the next task.

The bottom line 3.6.1 *This paper has remarkable transfer learning capabilities, since it bases its search of optimal solutions on a population of different partially optimized modules, and where the fitness evaluation can even be computed in parallel. Because avoiding catastrophic forgetting also indirectly implies reuse of previous knowledge to solve future tasks, we can give great credit to their solution and frame their approach of being capable of passing on promising solution candidates over multiple tasks. But the drawback is that it requires a huge amount of computational resources and has a large memory footprint. Also the performance outcome is based on non-deterministic heuristics which not always finds an optimal solution. Reconstructing the same module pathways is very unlikely even with similar initial setups. In principle neural networks with backpropagation optimization are fully deterministic given the same initial conditions.*

3.7 PackNet

PackNet (Mallya and Lazebnik, 2017) counters catastrophic forgetting by learning task-specific sparse filters using network pruning techniques to create free parameters that can be used to learn future tasks. Given a neural network with N number of parameters θ_N, they train it on the first task t_0 with the full parameter set, afterwards prune unimportant parameters down to a pre-defined percentage, for example, 60%. The pruned parameters are then fine-tuned on the same task t_0, and define a sparse filter for task t_0. Afterwards, the filters are frozen and represent the solution filter for task t_0. For the next task t_1 they also pre-train the remaining free parameters and then fine-tune the subset. The forward pass computations include also the previously frozen parameters, but without updating the filter parameters. This procedure is repeated for K tasks, learning K task-specific filters. Figure 3.5 illustrates the training procedure of the task dependent filters. The two-step update procedure is necessary to compensate for the higher loss

caused by the applied pruning technique, since changes in the network connectivity has a high influence on the computed output.

3.7.1 Pruning Procedure

The pruning procedure is applied to the convolutional and fully connected layers according to a pre-defined percentage threshold. First, they sort the weights by their absolute magnitude, and the lower 50 to 75 percent are selected to be removed. For computational convenience, the a one-shot pruning approach was applied. To ensure stability of previous performance they only prune weights that are not assigned to previously trained tasks. The mentioned masks represent a sparse filter which is stored efficiently, since with an increasing number of tasks the previous task masks are a subset of the current mask, it requires at most $\log_2(N)$ number of bits to represent the masks.

3.7.2 Inference

During inference, it is important to notice that the model cannot perform simultaneous inference on all tasks since the activating units, which did not belong to previously trained filters, would disturb the resulting output.

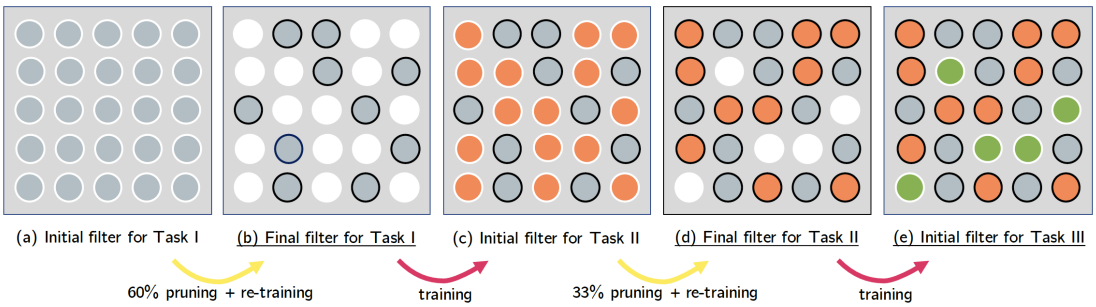


Figure 3.5: Illustration of the sequentially trained filters (Mallya and Lazebnik, 2017).

The bottom line 3.7.1 *This method is immune to catastrophic forgetting by design, since the task-specific sparse filters are kept fixed, but it requires an initial setting for its degree of sparsity and is limited to the fixed sized network capacity. Nonetheless, the parameter overhead is very small, since they need to add only one binary parameter selection masks for each task and their training process takes only* $1.5\times$ *longer compared to fine-tuning, since the second training step is running only for half the epochs. This is a very simple, yet effective training method to avoid forgetting on previous tasks and shows great performance on new tasks.*

3.8 Hard Attention to the Task

Serrà et al., 2018, proposed Hard Attention to the Task (HAT) that learns features by applying a hard attention mask to the output units such that

$$\boldsymbol{h}_l' = \boldsymbol{a}_l \odot \boldsymbol{h}_l, \tag{3.28}$$

where l denotes the current layer, $\boldsymbol{h}_l$ the activations at layer l and t the current task. In contrast to regular attention mechanism, where $\boldsymbol{a}_l^t$ forms a probability distribution over all tasks t, they apply a gated version of a single layer task embedding

$$\boldsymbol{a}_l^t = \sigma(s\boldsymbol{e}_l^t), \tag{3.29}$$

where σ defines an arbitrary gating mechanism and s a positive scaling factor on all layers $l = 1 \dots L-1$. The last layer $\boldsymbol{a}_L^t$ is a binary hard-coded mask according to a typical multi-head output.

The conducted experiments defined σ to be a sigmoid gating function, such that the gating mechanism arises similar behavior as *inhibitory synapsis*, activating or deactivating different units for each layer. To preserve information over multiple tasks they conditioned the gradient updates by freezing important weights according to a binary backward mask. This mask is the inverse of the accumulated attention vector, and obtained by recursively computing the element-wise maximum of the previous bit mask and the current mask:

$$\boldsymbol{a}_l^{\leq t} = \max(\boldsymbol{a}_l^t, \boldsymbol{a}_l^{\leq t-1}), \tag{3.30}$$

including the all zero vector $\boldsymbol{a}_l^{\leq 0}$. The inverse of the minimum of the cumulative update is applied to each gradient $g_{l,i,j}$ at layer l

$$g'_{l,i,j} = [1 - \min(a_{l,i}^{\leq t}, a_{l-1,j}^{\leq t})]g_{l,i,j}, \tag{3.31}$$

with the indices i and j corresponding to the units of the output layers l and input layers $(l-1)$ respectively.

A desired property of the attention vector $\boldsymbol{a}_l^t$ is to learn a binary hard attention. However, the training of the embeddings $\boldsymbol{e}_l^t$ must remain differentiable. To overcome this limitation, a scaling factor s is introduced which forces the creation of a pseudo-step function, which enables the hard attention encoding. During training s is incrementally linearly annealed, controlling the plasticity of the gradient flow and during testing $s = s_{\max}$, where $s_{\max} \gg 1$, defining $\boldsymbol{a}_l^t$ to behave almost like a step function. The following equation shows the annealing scheme of s during training:

$$s = \frac{1}{s_{\max}} + \left(s_{\max} - \frac{1}{s_{\max}}\right)\frac{b-1}{B-1}, \tag{3.32}$$

whereas $b = 1 \dots B$ is the batch index and B defines the total number of batches in an epoch. If $s_{\max} \to \infty$ then the attention $a_{l,i}^t \to \{0, 1\}$ behaves like a step function, otherwise if $s_{\max} \to 0$ then this results in $a_{l,i}^t \to \frac{1}{2}$ to enable similar gradient flow as in a regular sigmoid function.

3.8.1 Embedding Gradient Compensation

By empirically evaluating the results the authors from HAT came to the conclusion that because of the annealing scheme, the embeddings showed little changes. Without annealing $s = 1$, after one epoch the cumulative gradient showed a bell shaped curve, spanning the entire active sigmoid range $[-6, 6]$. When setting $s = s_{\max}$ it resulted in increasing the magnitude of the cumulative gradient, but decreases the spanning value

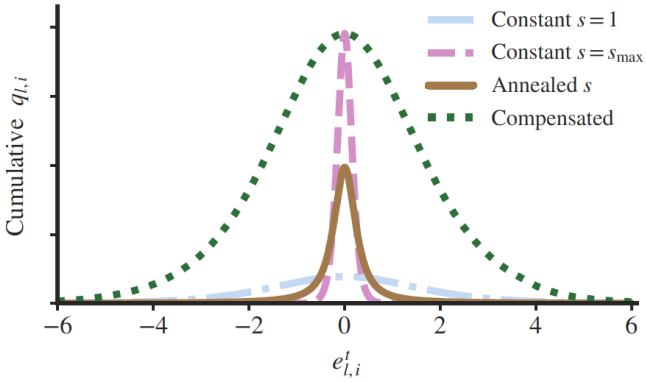


Figure 3.6: Illustration of the annealing effect of s on the gradients q of $\boldsymbol{e}^t$ (Serrà et al., 2018).

range. To ensure effective training both properties are required; a high magnitude and the full active value range of the sigmoid function, as illustrated in Figure 3.6.

By compensating the gradient the actual update is defined as

$$q'_{l,i} = \frac{s_{\max}[\cosh(se^t_{l,i}) + 1]}{s[\cosh(e^t_{l,i}) + 1]} q_{l,i}. \tag{3.33}$$

To further ensure numerical stability the gradients are clipped at $|se^t_{l,i}| \leq 50$, keeping $e^t_{l,i}$ in an active range of the standard sigmoid function $e^t_{l,i} \in [-6, 6]$.

3.8.2 Promoting Low Capacity

To promote efficient usage of the available network capacity they added a regularization term to the loss function. Since $a^t_{l,i} \to 1$ directly determines the units that will be dedicated to task t, they can use the set of previous attention vectors $A^{<t} = \{\boldsymbol{a}^{<t}_1, \ldots \boldsymbol{a}^{<t}_{L-1}\}$ and reduce the cost for $A^t = \{\boldsymbol{a}^t_1, \ldots \boldsymbol{a}^t_{L-1}\}$ if both values overlap. This masking promotes high reusage of existing features and requires a sparse set of additional units per task. The new loss function is denoted as

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}, A^t, A^{<t}) = \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) + cR(A^t, A^{<t}), \tag{3.34}$$

where $c \geq 0$ is the regularization constant, which controls the capacity spent on each task. The regularization term R is defined as a L1 regularization over A^t:

$$R(A^t, A^{<t}) = \frac{\sum_{l=1}^{L-1} \sum_{i=1}^{N_l} a^t_{l,i}(1 - a^{<t}_{l,i})}{\sum_{l=1}^{L-1} \sum_{i=1}^{N_l} 1 - a^{<t}_{l,i}}, \tag{3.35}$$

with N_l defining the number of units in each layer l and where the cumulative attentions over the past tasks $A^{<t}$ is promoting reuse of old features.

The bottom line 3.8.1 *This method is an effective and compact method to learn important features per task and prevents forgetting by freezing important weights. Compared to previously mentioned approaches it does not require major changes to the network topology as in Progressive Neural Networks. But similar to PackNet, they cannot perform simultaneous inference since the task-specific gating masks have to be manually pre-selected.*

3.9 Synaptic Intelligence

Zenke, Poole, and Ganguli, 2017, proposed Synaptic Intelligence (SI) to tackle catastrophic forgetting which is inspired by the adaptiveness of biological neural networks. The authors claim that in contrast to biological neural networks, where the plasticity of neurons is controlled by complex molecular reactions, the artificial counterparts are modeled to simplistic. ANN neurons require task-specific *importance* measures which are used to preserve features, proposing a three-dimensional state space per unit, instead of a zero-dimensional scalar output. By tracking the previous and current parameter values and maintaining a local *importance* estimate ω_k^{μ} for the global contribution of solving tasks μ, they penalize changes to important parameters θ_k according to a L2 regularization. The effect of catastrophic forgetting can be analyzed in a two-parameter case with θ_1 and θ_2 as shown in Figure 3.7. Following the optimization trajectories (black line) from the initial parameter state $\theta(t_0)$ towards $\theta(t_1)$ when training on task 1 and changing the objective towards task 2, the final state of $\theta(t_2)$ lies in an optimal region for task 2 and in a sub-optimal region for task 1. By regularizing the gradient trajectories while training on task 2 a sub-space can be found where the losses are minimal on both tasks, similar to Elastic Weight Consolidation. This also generalizes on more than

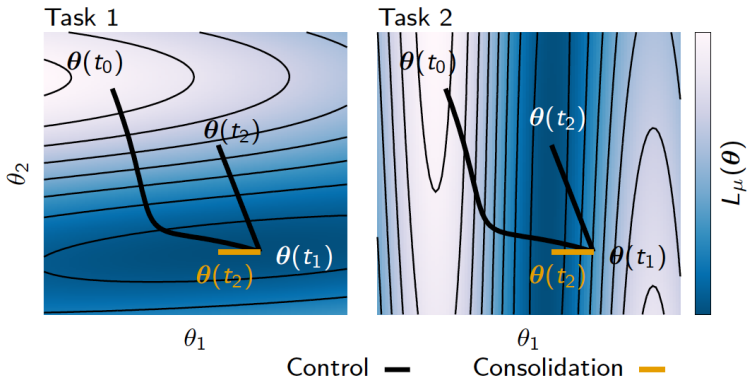


Figure 3.7: Two parameter example of the trajectory behavior on the fitness landscape. Blue color indicates little loss and white high loss (Zenke, Poole, and Ganguli, 2017).

two tasks and means that the main goal is to ensure updates that result in a minimal loss on all tasks. To enable this without knowledge about past tasks, they create a surrogate loss representing all previous losses. The surrogate loss is obtained through the parameter-based important measure which is tracked during training. Considering the

behavior of the objective function with an infinitesimal parameter update $\boldsymbol{\delta(t)}$ at each time step t, we can observe that the summed parameter changes of the gradient $\boldsymbol{g} = \frac{\partial L}{\partial \boldsymbol{\theta}}$ corresponds approximately to the differences in loss:

$$L(\boldsymbol{\theta}(t) + \boldsymbol{\delta}(t)) - L(\boldsymbol{\theta}(t)) \approx \sum_k g_k(t)\delta_k(t) \tag{3.36}$$

This basically means that each parameter change directly contributes to the total difference in loss. By integrating over the task-specific gradient trajectories according to each update step, we get an importance measure which corresponds to the parameter specific contribution and can be used to define a regularization term for the new surrogate loss. Equation 3.37 shows how the loss decomposes as a sum of individual parameters.

$$\int_{t^{\mu-1}}^{t^\mu} \boldsymbol{g}(\boldsymbol{\theta}(t))\boldsymbol{\theta}'(t)dt = \sum_k \int_{t^{\mu-1}}^{t^\mu} g_k(\theta(t))\theta_k'(t)dt \tag{3.37}$$

$$\equiv -\sum_k \omega_k^\mu \tag{3.38}$$

In practice, ω_k^μ is approximated as a running sum of parameter changes and negated since we are interested in minimizing a loss. When training on a new task μ large weight updates of important parameters can be prevented by penalizing movements away from the previous tasks $\nu < \mu$. To obtain the corresponding regularization we need to renormalize $\boldsymbol{\omega}^\nu$ according to the total parameter changes $\Delta_k^\nu \equiv \theta_k(t^\nu) - \theta_k(t^{\nu-1})$. The cumulative importance measure Ω_k^μ for the regularization of the new task μ is now defined as:

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{\omega_k^\nu}{(\Delta_k^\nu)^2 + \xi}, \tag{3.39}$$

where ξ represents a damping parameter avoiding zero division terms, and $(\Delta_k^\nu)^2$ ensures the same units as in the original loss L. The surrogate loss is now defined as the original loss L including a quadratic loss $\tilde{L}$, which approximates the summed loss over all previous tasks $L_\nu(\nu < \mu)$. Training on the new surrogate loss should be now equivalent to training on the original loss. Equation 3.40 shows the surrogate loss when learning a second task sequentially

$$\tilde{L}_\mu = L_\mu + c\sum_k \Omega_k^\mu(\tilde{\theta}_k - \theta_k)^2, \tag{3.40}$$

where c represents how much weight we put on keeping old tasks and the reference weights are defined as $\tilde{\theta}_k = \theta_k(t^{\mu-1})$. Although this approach is only defined for two tasks, in their paper they can empirically demonstrate that it also works on more than two tasks. ω_k^μ is continuously updated and Ω_k and $\tilde{\theta}$ are updated at the end of training of each task.

The bottom line 3.9.1 *Similar to EWC, the authors from HAT also prove that consolidation of weights can achieve great results and improve sequential training significantly. Since it is the only approach keeping an online estimate of the entire*

training trajectories (not only point estimates such as in EWC), one can argue that this is the best choice suited for a reinforcement learning approach.

3.10 Context-Dependent Gating

The authors of Context-Dependent Gating (XdG) focused their research on combining synaptic stabilization approaches with partial neuron updates (Masse, Grant, and Freedman, 2018). They analyzed the behavior of neural networks when trained with the previously mentioned EWC and SI approach in combination with a gating algorithm, which randomly masks neurons w.r.t. to the trained task. The research shows three measures how the gating strategy affects neurons when trained on multiple tasks in a sequential manner. XdG helps to significantly reduce catastrophic forgetting when training on a large amount of tasks $\gg 100$. The experiments were conducted with a fully connected neural network with two hidden layers and ReLU activation function. Figure 3.8 depicts different experimental network gatings while developing their method. A represents a two layer fully connected network without any gating, which was only enhanced by EWC or SI. B denotes a partially gated network (activations were multiplied by 0.5), with randomly chosen gates per neurons (according to a probability of 50%). The selected gates are always preserved and assigned to a corresponding task. C illustrates a partially gated network which consists of five sub-networks or modules and only one module per layer is activated for each task. The number of units for the su-networks are chosen such that the total number of units adds up to the original number of units of the full network. The last illustration shows their proposed XdG network where they apply full gating with either 0 or 1, and again each gating scheme is assigned to a particular task.

The bottom line 3.10.1 *XdG keeps only a partial sub-set of neurons active. In combination with synaptic stabilization they ensure consolidation of weights and only apply changes on an unimportant parameter subset.*

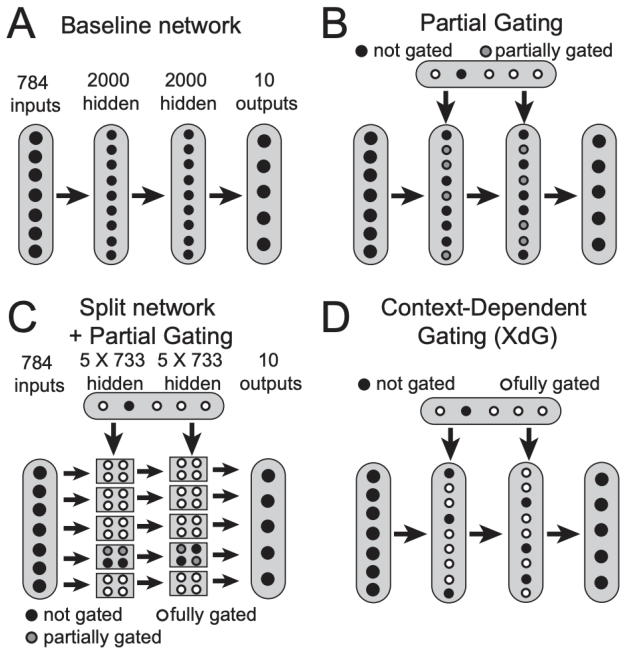


Figure 3.8: Illustration of the task dependent gating proposed by Masse, Grant, and Freedman, 2018. A shows a non-gated two layer fully connected network. B a partially gated network. It is gated by multiplying the activations with 0.5. C splits the layer parameters in modules and performs partial gating. D performs full gating with a binary bit mask.

Chapter 4

Comparison

> "Bring forward what is true, Write it so that it is clear, Defend it to your last breath!"
>
> *Ludwig Boltzmann*

In the previous chapter, we introduced several approaches to counter the effect of catastrophic forgetting. In this chapter, we will reflect on the published results and emphasize the benefits and drawbacks of the different approaches. The following criteria will be used as a guide to assess the quality of the different approaches:

- Benchmark: Learning versus forgetting
- Representational power of trainable parameters
- Deterministic learning behavior
- Neural plasticity for long term learning
- Online learning capability
- Single versus multi-phase training
- Ability for Context-dependent decision marking
- Simplicity of integration

As a baseline and frame of reference, we will use the framework and forgetting ratio benchmark introduced by Serrà et al., 2018. Their architecture is based on AlexNet with 3 convolutional layers of 64, 128, and 256 filters with 4×4, 3×3, and 2×2 kernel sizes, respectively, plus two fully-connected layers of 2048 units each. The last layer uses a fully-connected multi-head output per task, which is similar to a multi-task setup. While sequentially training on each task, the amount of forgetting is measured according to the forgetting ratio, which not only considers the accuracy of the current task, but also accuracy drops in past tasks. All weights except of the last layer are either shared or manipulated according to the proposed methods. The datasets are CIFAR-10 and CIFAR-100, since they are visualizable, human interpretable, and represent a certain degree of learning difficulty. We train on these datasets in an incremental way as proposed by Lopez-Paz and Ranzato, 2017. The group size of each dataset CIFAR-10 and CIFAR-100 is chosen as proposed by Serrà et al., 2018. We randomly select five times two classes from CIFAR-10 and five times twenty classes from CIFAR-100. The

training sequence is chosen according to a random seed. This setup ensures a certain level of difficulty, but remains trainable in a reasonable amount of time on a single GPU. Other proposed benchmarks used either too simplistic datasets, such as MNIST, or too idealistic training sequences, helping to solve future tasks after training the first task.

4.0.1 Forgetting Ratio

In contrast to the often used average mean accuracy of all tasks, the forgetting ratio focuses on both forgetting of previous tasks and adaptiveness towards the newest tasks. The average mean accuracy is biased according to its outlier-performance and does not reflect the sensitivity versus stability of learning. To receive a plottable scalar value for the forgetting quantity after each sequentially learned task, we use the *forgetting ratio* proposed by Serrà et al., 2018:

$$\rho^{\tau\leq t} = \frac{A^{\tau\leq t} - A^{\tau}_{R}}{A^{\tau\leq t}_{J} - A^{\tau}_{R}} - 1, \tag{4.1}$$

where $A^{\tau\leq t}$ is the accuracy measured on all tasks $\tau \leq t$ after sequentially learning task t, A^{τ}_{R} is the accuracy of the random stratified classifier and $A^{\tau\leq t}_{J}$ is the accuracy measured on the jointly learned multi-task setting. $\rho^{\tau\leq t}$ measures the accuracy of all tasks up until task t. To obtain the forgetting ratio (1) we need to train our models w.r.t. the approach we want to evaluate, (2) train the basic architecture on a joint data set version using Stochastic Gradient Descent (SGD) and (3) create randomly generated stratified classifier. To receive a single scalar value per task we then compute:

$$\rho^{\leq t} = \frac{1}{t}\sum_{\tau=1}^{t}\rho^{\tau\leq t}. \tag{4.2}$$

4.0.2 Analysis

At first, the introduced complexity of the listed approaches from chapter 3 might seem questionable and may suggest a more novel setting to counter catastrophic forgetting. One such example is based on differently scaled learning rates. Assuming a neural network with a small learning rate in the lower layers and a high learning rate in the last layer, where the last layer is shared across all tasks, this setup is object to forgetting, because no objective is embedded preventing destructive updates while learning new information. All weight settings will optimize towards the new objective and overwrite important features of previous tasks. The net effect is similar to learning plain SGD with a multi-head output. Over long training sequences old tasks will be forgotten regardless of the scale of the learning rate. LFL and LWF are more elaborate alternatives to altering learning rates, since they penalizes updates moving away from previous tasks. In Figure 4.1 we present an overview of the benchmark results from Serrà et al., 2018. We can see that HAT outperforms all approaches w.r.t. the forgetting ratio. We can also see that LFL and LWF are performing poorly, since freezing classifier weights and fine-tuning distributions that are not originating from the actual dataset does not prevent forgetting. Furthermore, the objective of LFL and LWF is not representative for a sequential learning setting. SGD alone is also forgetting older tasks continuously, since

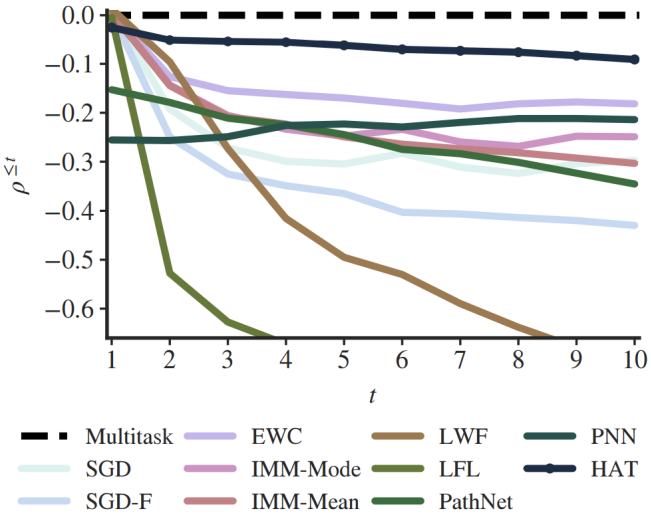


Figure 4.1: Forgetting ratio comparing the different approaches. PackNet, Synaptic Intelligence and Context-Dependent Gating is not shown, since the authors of HAT did not evaluate these algorithms. SGD-F is plain SGD with fine-tuning of the last layer (Serrà et al., 2018).

it does not include any objective to prevent destructive updates. What is not directly shown by the forgetting ratio, is that not only old tasks are disrupted, but also the representational power of some tasks may be reduced. This can only be detected when observing the relative accuracies compared to SGD. SGD is a good reference point for the current tasks, since it learns unrestricted on the current objective, but only performs well on multiple tasks if they are similar to each other. This is also the effect why the confidence intervals in Figure 4.1 sometimes become larger or narrower after training on a new task. However, regularizing the representational power of the models is only constructive if it occurs in a goal-oriented manner and ensures that task-specific parameters are preserved. This is also one of the key criteria why incremental moment matching shares the forgetting ratio similar to SGD, since averaging the moments of different tasks only preserves common features without prioritizing task-specific settings. EWC and SI trace the importance of individual weights by using either the Fisher information matrix or the path integral, and penalizes updates that would be destructive to previous tasks. These type of algorithms use structural regularization to update only specific parameters. If we observe the continuous learning capability of EWC to compute the Fisher information matrix, we notice that it requires a second optimization phase, as it is a pointwise estimate of the importance by forwarding the task-specific training samples through the network. This limits the capability of online learning applications, especially in a Reinforcement Learning setting, since we need to forward the entire state space to asses the importance measure to consolidate the weights. In Figure 4.1 we can see that EWC outperforms SGD, however, slightly under-performing PathNet, PNN and HAT. This is partially caused by the overestimation of the importance parameter and hence over-regularization.

PathNet's implementation is based on evolutionary algorithms, which selects and

combines layer-wise modules to learn new tasks. The setup of PathNet is similar to having multiple neural networks permuted and refined according to the selection criteria. This, of course, gives higher scores to match parameter constellations and improves the overall performance by updating only selective parts of the subset. However, using heuristics in a second stage update phase, not only introduces non-deterministic learning behavior but also requires higher computational expense. In analogy, this is similar to training n subsets of independent neural networks and exchanging weights between tasks, which seem to give promising results. From an online learning perspective, this computational overhead is not desirable. Furthermore, it requires external instructions to select the correct modules for inference. In fact, this issue is the same across all methods that share task-dependent weights for inference. HAT might outperform all methods, but it also requires the selection of task-dependent embeddings. This is a huge drawback from a continuous learning perspective since without knowledge of the correct context, one has to forward the input multiple times through the network and can only guess according to the maximum achieved scores to which class the current task belongs. The inference time also increases proportionally to the number of tasks.

Progressive Neural Networks (PNN) are by definition immune against catastrophic forgetting, since freezing weights of a trained network and extending it by lateral connections towards new weights avoids forgetting entirely, but also requires a pre-selection of the correct column for inference. An additional cost of PNNs relies upon the quickly growing number of parameters and destructive interference of activations. Assuming that two datasets are mutually exclusive, the network first needs to learn to suppress the destructive lateral signals and then learn the important task-specific features. This issue relates to the aspect of neural plasticity. Freezing weights is preventing destructive updates, but also prohibits constructive adaption of new tasks. For PNN this results in an increasing demand for additional weights. In comparison to HAT this sets an upper bound for the number of learnable tasks. HAT freezes important weights by using a backward hard attention mask to prevent destructive updates while training on the new objective. These masks are created during the forward pass and bitwise maximized with the total collection of masks from previous tasks. Their intend is to keep the creation of masks minimal per task, to ensure enough free capacity for future tasks. This reduction of available weights continues gradually if a good compression regularization is applied, but will eventually result in an exhaustion of neural plasticity. According to the authors of HAT, the plasticity of the neurons can be regained by unfreezing the bits of the backward mask, but according to experiments shown in chapter 6, such updates result in catastrophic forgetting of old tasks. The reason is that no other regularization technique is available to prevent destructive updates. If the weight distributions change, multiple tasks can be affected, which require proper matches of the hard attention masks. Furthermore, this method also requires a fixed knowledge about the desired embedding size, which is in conflict with continuous learning since we do not know in advance how many tasks we might want to learn. Changing the embedding size later requires full retraining of all tasks.

In theory, all methods except PNN are affected by the limitation of having to learn an infinite number of tasks with a finite number of parameters. The issue is rooted in the inability to introduce new weights without disrupting computations. In addition, switching between the task context avoids disrupting the computation of previous tasks,

but requires external interaction to chose the correct context regarding inference. These context-dependent inference problems also restricts the capability of an online learning settings. This favors synaptic stabilization based algorithms, such as SI and EWC and increases the demand to improve their performance. Since the paper of Serrà et al., 2018, did not offer a direct comparison between SI with EWC, we reconstructed the results in Figure 4.2 and additionally added the absolute performance in Figure 4.3 after training 10 tasks in a sequential manner relative to the baseline of SGD and HAT. Here we

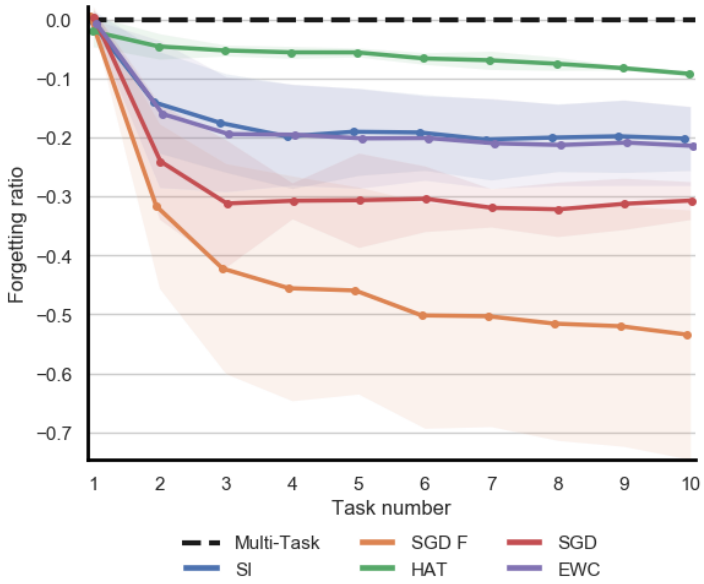


Figure 4.2: Forgetting ratio of Synaptic Intelligence (SI), Elastic Weight Consolidation (EWC), Hard Attention to the Task (HAT), Stochastic Gradient Descent with Dropout (SGD), and SGD with frozen weights (SGD-F) with a 95% confidence interval that the values rely within the shaded areas. The experiments are averaged over six random seeds.

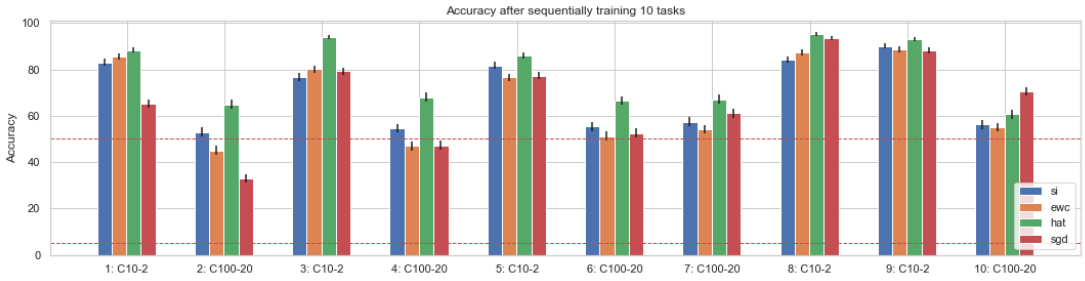


Figure 4.3: Accuracy plot after training SI, EWC, HAT and SGD in a sequential manner on 10 tasks. Red dashed lines at 50% and 5% mark the levels of randomly classifying classes by chance for CIFAR-10 with task size of two classes (C10-2) and CIFAR-100 with task size of 20 classes (C100-20) respectively. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

can see that SGD and HAT are much more adaptive to learning current task features. HAT performs very well on old tasks, but SI and EWC are stable across long training sequences. SGD-F performs worst, which does not forget any features but never adapts to new tasks properly. We also added the confidence intervals, which indicate volatility when training according to the expressiveness of the first tasks. If the first tasks includes features valid across multiple tasks, higher performance can be expected and vice versa. The main issue why EWC and SI performs worse compared to HAT relates not directly to forgetting, but due to the overestimated importance of parameters and, therefore, overregularization of new tasks. This means, they do not adapt to new tasks as easily as HAT and have a greater performance margin to the maximum possible task. Masse, Grant, and Freedman, 2018, showed that synaptic stabilization methods can improve with the issue of overestimated penalization by partially or fully gating neurons. This improves the absolute accuracy per task, but introduces the necessity to predetermine the context before inference. Their paper also emphasizes that the methods with the greatest potential to learn long term sequences are based on synaptic stabilization and not freezing parameters. The issue, how to improve assessing the parameter importance and receive the corresponding context during inference, remains. Apart from predetermining the context, Context-Dependent Gating (XdG) uses randomly created masks per task, and introduce another hyperparameter, specifying the percentage for the minimum amount of units to mask, similar to PackNet.

From an implementation point of view, PNN and PathNet introduce the highest complexity by requiring changes to the network architecture. This might be only a technical problem and is not relevant from a scientific point of view, but introduces limitations when trying to apply the methods on more elaborate architectures apart from AlexNet. The simplest and most effective adaption is based on synaptic stabilization since the changes mainly apply at the optimization criteria and the tracking of importance from gradients gradient.

The bottom line 4.0.1 *Synaptic stabilization algorithms perform best according to the trade-off between neural plasticity versus forgetting, but overestimate the importance of parameters. Context-dependent methods achieve high scores by better adapting the new task specific features, but require external context information for inference.*

Chapter 5

Context-Dependent Activations

> "People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on eachother, like a wall of mini stones."
>
> *Donald Knuth*

This chapter represents the main part of the thesis and introduces our method denoted as Context-Dependent Activations (XdA). As Context-Dependent Activations, we understand a non-linearity function that dynamically rectifies activations according to a trainable threshold.

5.1 Concept

The previous chapters demonstrated that EWC (Kirkpatrick et al., 2017) and SI (Zenke, Poole, and Ganguli, 2017) are fundamentally good at learning new tasks and remain stable towards previously learned tasks. Their main drawback is the overestimated parameter importance, which restricts learning of new objectives in comparison to plain SGD or HAT (Serrà et al., 2018). We could reduce this by adding XdG (Masse, Grant, and Freedman, 2018) and, thereby, compensate the amount of involved parameters causing destructive interference. When evaluating the model, this change requires external knowledge of the context, which we want to avoid. Furthermore, in a continuous learning environment one needs to learn a large number of tasks, such that it becomes possible to dynamically extend the representational power of the model by introducing new parameters. In brief, we can summarize our requirements as:

- Increasing the performance for learning new tasks
- Enable dynamic parameter extensibility
- Perform context-dependent updates
- Enable self-selective context prediction
- Remain online learning capable

5.2 Context-Dependent Activations

To increase the performance, we want to use a gating mechanism similar to XdG, which reduces the number of activations. But randomly generating binary bit masks to filter activations has the limitation that the masks become non-differentiable. The key aspect is to define a gating operation, which is first-order differentiable. HAT uses a trick to create a pseudo-step-function by scaling the embeddings with a hyperparameter s_{mas} and a sigmoid operation, which remains fully differentiable. The embeddings are independent of the input space and act as a fixpoint for the activation space. During the backward pass, the authors from HAT compensate for s_{max} by rescaling the gradients of the embeddings (Serrà et al., 2018).

If we analyze neural network operations excluding Dropout, the main operations are weighted summations at the neuron level, followed by a non-linearity function—often ReLU—and in cases with attention, a gating function. The ReLU activation function rectifies negative values to zero and maps positive values to its identity. This, by design, biases the activation space towards positive values as shown in Figure 5.1.

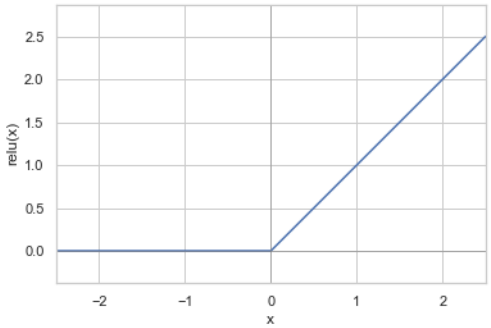


Figure 5.1: ReLU activation function.

If the activation function is followed by an attention mechanism, then the set of forwarded activations is reduced and operative updates are sparse. The threshold parameters of the non-linearity function must behave as rectifying fixpoint during the forward pass, but remain differentiable during the backward pass. This not only increases the representational power by allowing negative values, but also enables the network to determine important parameters. Such dynamically gated activations we define as Context-Dependent Activations (XdA) and Figure 5.2 illustrates the function behavior.

The illustrated thresholds in Figure 5.2 must be trainable, hence differentiable, which we will formally define in the next section.

5.3 Activation Threshold

To reduce the overestimation of parameters we need to train a threshold that filters activations and forwards only important values. The reduction of activations improves synaptic stabilization methods as shown by Masse, Grant, and Freedman, 2018, since

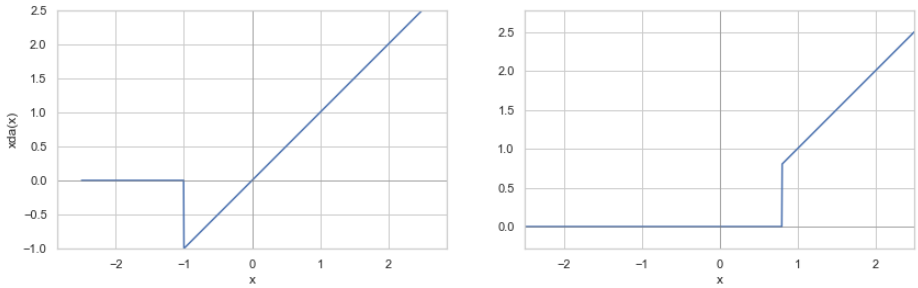


Figure 5.2: Two XdA activation functions, with the first example showing a negative threshold and the second example a positive threshold. All values below the threshold are set to zero, all above are mapped through the identity function. The function is not differentiable at the threshold point, similar to ReLU, but in practice, this is not an issue.

only a smaller subset of parameters is updated. During our initial evaluation, we experimented with a novel variant of the ReLU function defined as $f(x) = \max(x - \varphi, 0)$, where x denotes the pre-activations and φ the trainable threshold to enabled a context-based filtering. The resulting non-linearity mainly differs from ReLUs according to the subtraction of the threshold φ. We observed that backpropagating an error signal through φ offers only good performance on the current task, but does not prevent forgetting on older tasks. Moreover, the active value range of the activations was relatively small and left-skewed above zero according to the max operation as described in section 5.6. To enable a larger value range and ensure that φ improves the activations filtering, we wanted to allow negative values after the non-linearity and diverge φ relative to a secondary parameter fixpoint. Figure 5.2 illustrates the variable threshold φ.

According to our requirements we define the context-dependent operations to perform actions on the activation space $\boldsymbol{h} \in \mathbb{R}^{b \times n}$, where b refers to the batch size and n to the number of activations. The activations $\boldsymbol{h}$ are gated according to a trainable parameter set $\boldsymbol{\varphi} \in \mathcal{H}$, where $\mathcal{H} \in \mathbb{R}^n$. $\boldsymbol{\varphi}$ marks the trainable parameters. By shifting the values of $\boldsymbol{\varphi}$ or processing different values of the activation space $\boldsymbol{h}$, we can filter activations such that values are forwarded as an identity or set to zero. To realize this behavior during the forward pass we use an auxiliary binary mask $\boldsymbol{a} \in \{0, 1\}^{b \times n}$ which is computed for each layer by assigning

$$\boldsymbol{a}_k^i \leftarrow \boldsymbol{h}_k^i \geq \boldsymbol{\varphi}_k, \tag{5.1}$$

where k denotes the current layer, i represents the i-th element from the batch b and t the current task or threshold context. Now we can perform the Hadarmard product on the activation space and obtain a gated identity

$$\boldsymbol{h}'_k \leftarrow \boldsymbol{a}_k \odot \boldsymbol{h}_k \tag{5.2}$$

where we forward $\boldsymbol{h}'_k$ to the next layer. In addition to $\boldsymbol{\varphi}$ we track the moving average of the batches by introducing an additional variable $\boldsymbol{\mu} \in \mathcal{H}$. The initial values of the task-dependent parameters $\boldsymbol{\varphi}_{k,t}^{(0)}$ and $\boldsymbol{\mu}_{k,t}^{(0)}$ are set during the first forward pass according

to the task-dependent mean activations of the first batch $\bar{\boldsymbol{h}}_{k,t}^{(0)} \in \mathcal{H}$ such that

$$\boldsymbol{\varphi}_{k,t}^{(0)} = \boldsymbol{\mu}_{k,t}^{(0)} = \bar{\boldsymbol{h}}_{k,t}^{(0)}. \tag{5.3}$$

For each additional training step the exponential moving average of the mean activations is tracked via

$$\boldsymbol{\mu}_{k,t} \leftarrow \alpha \boldsymbol{\mu}_{k,t} + (1 - \alpha)\bar{\boldsymbol{h}}_{k,t}, \tag{5.4}$$

where α denotes the exponential decay.

5.4 Regularization

Since we used a non-differentiable operation during the forward pass to obtain the hard attention mask $\boldsymbol{a}_k$, we need to establish an alternative gradient pathway to make $\boldsymbol{\varphi}$ trainable. $\boldsymbol{\varphi}$ becomes differentiable by using the moving average of the activations $\boldsymbol{\mu}$ and extending our loss function with an additional regularization term $\mathcal{R}_B$:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \mathcal{L}_B(\boldsymbol{\theta}, \boldsymbol{\varphi}) + \lambda_1 \mathcal{R}_A(\boldsymbol{\theta}) + \lambda_2 \mathcal{R}_B(\boldsymbol{\varphi}), \tag{5.5}$$

where λ_1 and λ_2 are dimensionless scaling hyperparameter for each regularization respectively. The structure of the objective from equation 5.5 is almost identical to EWC or SI, since it uses an expression to optimize on the current objective denoted as $\mathcal{L}_B$ (task B) and a regularization term for synaptic stabilization $\mathcal{R}_A \in \mathbb{R}$ according to the previous task A as shown in section 3.2 and 3.9. The added regularization term $\mathcal{R}_B$ is used to maximize the absolute distance between the trainable parameters $\boldsymbol{\varphi}$ and the activation mean $\boldsymbol{\mu}$ such that

$$\mathcal{R}_B(\boldsymbol{\varphi}) = \sum_k \frac{\gamma}{s_k \sum_i |\varphi_{k,i} - \mu_{k,i}| + \gamma}, \tag{5.6}$$

where k denotes the layer, t the current task, γ a dimensionless factor to stabilize the cost and s_k a layer-wise hyperparameter to define the weighted importance of each layer. The net effect from this regularization is, that by maximizing the distance between $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$ we minimize the loss w.r.t. $\boldsymbol{\varphi}$. Maximizing according to the first norm also emphasizes sparsity in the activation space. Furthermore, by adding $\mathcal{R}_B$ to our new loss function, we introduced an interesting effect during training. We optimize two parameter sets $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}$ by alternating between them. When optimizing for $\mathcal{L}_B$, $\boldsymbol{\varphi}$ remains unchanged, behaving as a fixpoint and only the mean is updated. When optimizing for $\mathcal{R}_B$, $\boldsymbol{\varphi}$ maximizes its distance according to the unchanging mean. This training behavior is similar to *Generative Adversarial Networks* (GANs) proposed by Goodfellow, Pouget-Abadie, et al., 2014, where two networks play a game against each other until they reach a Nash equilibrium. In case of $\boldsymbol{\varphi}$ the training will stop after the distance reaches a local maximum, since increasing the distance between $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$ behaves equivalent to $\frac{1}{x}$.

To ensure parameter stability, we analyze the convergence behavior of the additional cost term by computing the Jacobian. The first order derivative of $\mathcal{R}_B$ w.r.t. $\boldsymbol{\varphi}$ is given

in equation 5.7.

$$\frac{\partial}{\partial \boldsymbol{\varphi}} \mathcal{R}_B(\boldsymbol{\varphi}) = \gamma \sum_k \frac{\partial}{\partial \boldsymbol{\varphi}_k} \left[\frac{1}{\gamma + s_k \sum_i |\varphi_{k,i} - \mu_{k,i}|} \right] \tag{5.7}$$

$$= -\gamma \sum_k \frac{s_k \sum_i \frac{\partial}{\partial \varphi_{k,i}} |\varphi_{k,i} - \mu_{k,i}|}{\left(\gamma + s_k \sum_i |\varphi_{k,i} - \mu_{k,i}|\right)^2} \tag{5.8}$$

$$= -\gamma \sum_k \frac{s_k \sum_i \frac{\varphi_{k,i} - \mu_{k,i}}{|\varphi_{k,i} - \mu_{k,i}|}}{\left(\gamma + s_k \sum_i |\varphi_{k,i} - \mu_{k,i}|\right)^2}. \tag{5.9}$$

The asymptotic behavior of equation 5.9 is that with the increasing distance between $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$ the gradient decays in a quadratic manner, stabilizing the fluctuating behavior of the regularization term, which is caused by the absolute value in the numerator. This derivation also proofs that the threshold parameters $\boldsymbol{\varphi}$ will always approach a fixpoint and settle to a stable state. We can empirically show how the regularization term behaves according to the maximization of distances between $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$. Figure 5.3 plots the saturation behavior measured by summing the absolute layer distances and shows that the layer distances decay over time and stabilize after a certain number of iterations. This further proofs that the XdA regularization term based on $\boldsymbol{\varphi}$ is guaranteed to converge. In addition, Figure 5.3 shows that by increasing the regularization constant λ_2 the training time is delayed and that the absolute distance between $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$ further increases. In terms of accuracy, the model that was longer trained also improved noticeably, which we will analyze in the subsequent experiments in chapter 6.

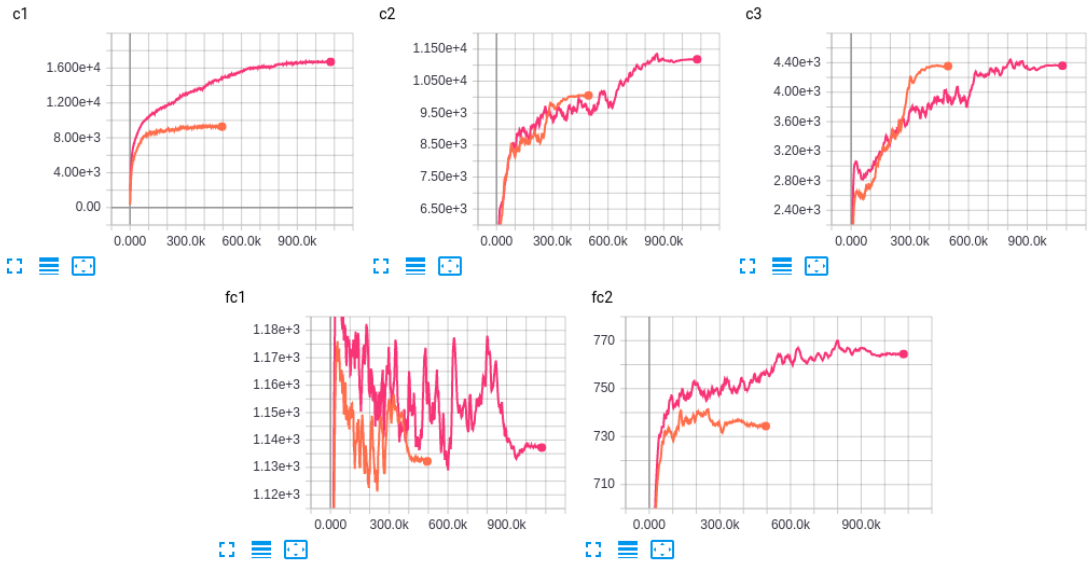


Figure 5.3: Saturation behavior of the AlexNet layers. c1, c2, c3 relates to the first, second and third convolutional layer and fc1, fc2 to the first and second fully connected layer respectively. Orange: without regularization $\lambda_2 = 0$. Pink: With high regularization $\lambda_2 = 100.0$ and $\gamma = 1e3$.

To observe the distribution behavior of the threshold parameters $\boldsymbol{\varphi}$ in contrast to

the means $\boldsymbol{\mu}$, Figure 5.4 compares the magnitudes of the first four network layers at different time steps. This demonstrates that the values are in the same active range

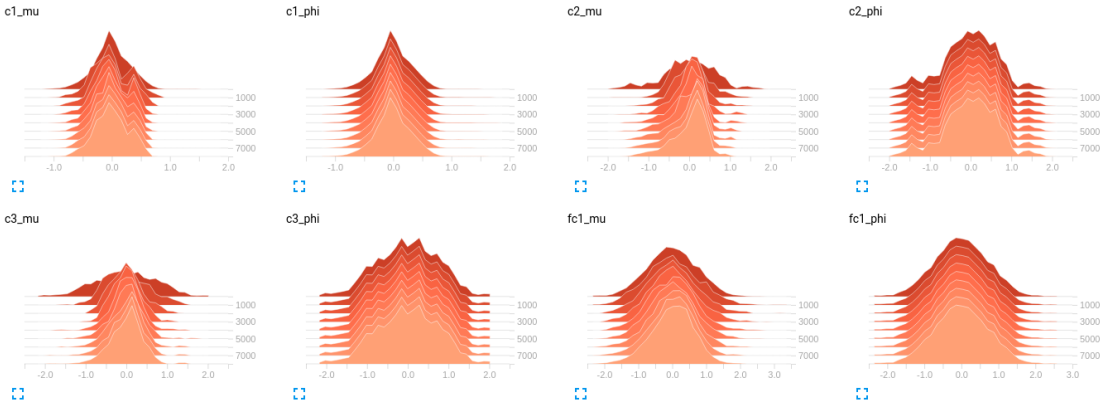


Figure 5.4: $\boldsymbol{\varphi}$ vs. $\boldsymbol{\mu}$ distribution of the layers (c1, c2, c3 and fc1) with $\lambda_2 = 1.0$ and $\gamma = 1e3$.

and normally distributed around zero, which is expected since all values are normalized before being processed as described in section 5.6.

In XdG we saw that for each task a different binary mask was selected, which altered the parameter updates according to the provided context. The effect of selecting only a subset of parameters for further computations, reduced disruptive interference of activations during the forward pass and destructive updates during the backward pass. It created the notion of task-dependent parameters. To relate equation 5.6 of XdA with XdG we can define a task-dependent regularization term $\mathcal{R}(\boldsymbol{\varphi}^t)$ according to task-dependent thresholds and moving average parameters

$$\mathcal{R}(\boldsymbol{\varphi}^t) = \sum_k \frac{\gamma}{s_k \sum_i |\varphi^t_{k,i} - \mu^t_{k,i}| + \gamma}. \tag{5.10}$$

This allows us to introduce new trainable parameters without disrupting previous tasks, but it introduces the issue of selecting the proper context for a task. If we need prior knowledge to manually pre-determine the correct context before inference, then the inference of a task becomes obsolete. In section 5.5 we present a novel method to counter this issue, by allowing XdA to self-predict its current task.

5.4.1 Memory Footprint

The introduced task-dependent parameters $\boldsymbol{\varphi}^t$ show a relative low memory footprint compared to already existing network parameters $\boldsymbol{\theta}$. Adding one parameter per activation value is proportionally cheap and resulted for our AlexNet architecture in a total parameter increase of approx. 2.5 percent per task.

5.5 Self-Task Prediction

If we introduce task-dependent trainable parameters $\boldsymbol{\varphi}^t$ we also introduce the issue of requiring knowledge of task t to pre-determine the correct threshold context. HAT, XdG, PNN, PathNet, and PackNet all require external information to select the correct task. To ensure the benefits of synaptic stabilization, we propose self-task inference by observing the activation space $\boldsymbol{h}$. Since the activation space is normalized according to its first and second moments using layer normalization, we can ensure similar magnitudes to compare the vectors. To compare the vectors we propose two variations.

5.5.1 Absolute Difference

We can apply the summed absolute difference as a metric to determine the context during the forward pass. When computing the distance between the activation vector and the task-dependent means $\boldsymbol{\mu}^t$, we predict the task t for each layer k by selecting the argument with the lowest error as shown in equation 5.11.

$$t = \underset{t}{\operatorname{argmin}} \sum_i |h_{k,i} - \mu_{k,i}^t|, \tag{5.11}$$

where h_k^i is the i-th activation of the k-th layer and $\boldsymbol{\mu}_k^t$ the mean activation of the t-th task. The experimental results regarding self-task prediction will be further discussed in chapter 6.

5.5.2 Cosine Similarity

As an alternative to the summed absolute distance metric we can use the cosine similarity and select the task which maximizes the score:

$$t = \underset{t}{\operatorname{argmax}} \frac{\sum_i h_{k,i}\mu_{k,i}^t}{\sqrt{\sum_i (h_{k,i})^2}\sqrt{\sum_i (\mu_{k,i}^t)^2}}. \tag{5.12}$$

This method is computationally more expensive, but also offered more stable results 6.

5.5.3 Runtime Complexity

The runtime complexity for computing the metrics during the forward pass is $\mathcal{O}(b*k*t)$, where b is the number of batch samples, k denotes the number of layers and t the number of tasks. Since k remains constant the computational demand grows linear according to the number of tasks. Although we stated that t is computed per batch sample, in practice, we implemented a batch-wise average for tests computing the forgetting ratio in chapter 6. For a very large number of tasks, we further propose to pre-compute hashes from the activation space and perform the similarity computation on a smaller subset.

5.6 Activation Normalization

To ensure numerical stability and to improve self-task prediction, we want to avoid the overestimation of outliers according to different magnitudes in the activation space.

Therefore, we normalize the activation space according to the LayerNorm operation proposed by Ba, Kiros, and Hinton, 2016, before computing the layer means $\boldsymbol{\mu}$, ensuring leveled value ranges. LayerNorm can be applied with and without additional parameters. We apply the parameterless approach. Using TensorBoard we show in Figure 5.5 that of distribution of the activation space after the normalization for different time steps during training. Although the distribution is slightly left-skewed, we can see that the values are positively and negatively distributed. This is the case, because we mask away values below a certain threshold and the range of values above this threshold is unbound. If we observe the higher layers, we see that in the beginning, the distributions are wider spread and continuously contract to a more spiked setting. This is partially caused by the gating function and by fine-tuning the parameters during training.

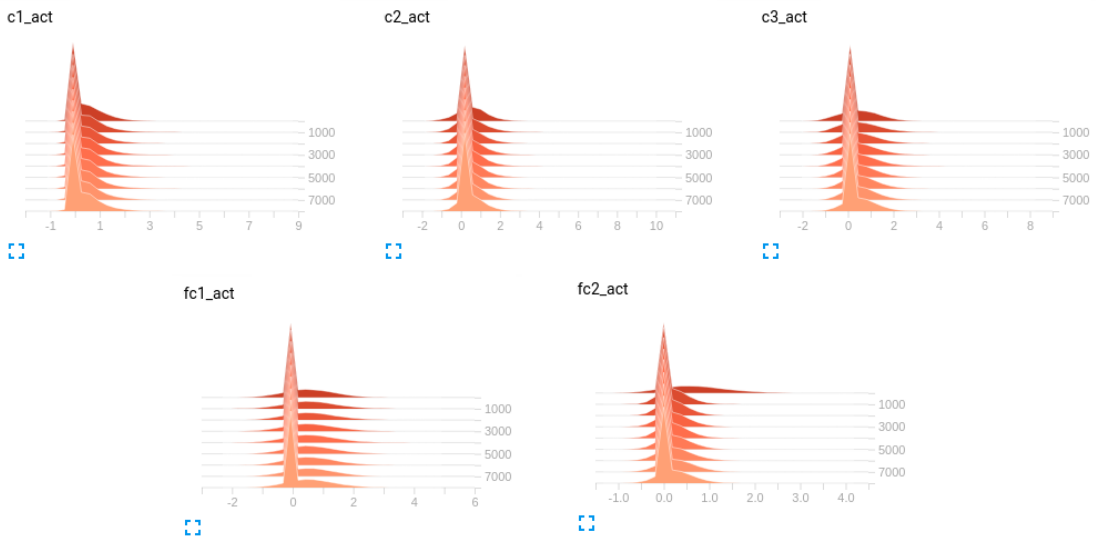


Figure 5.5: XdA activation distribution of CIFAR10 classes according to the convolutional (c1, c2, c3) and fully connected layers (fc1, fc2).

To compare the behavior of the ReLU and the XdA activation functions, we plot the distribution of activations in Figure 5.6. We can see that the ReLU activations are slightly left-skewed above zero and decay quickly afterwards, since all negative values are cut off. The more interesting aspect is the value range of ReLU since it is hardly above 0.5, which has indirect implications on the magnitude of the gradients. XdA offers a much larger value range since it implicitly uses normalization in addition to the identity mapping, which helps distribute values in both numerical directions. Furthermore, we have a direct connection to the loss function which also improves the gradient flow for the weight updates of $\boldsymbol{\varphi}$, ensuring longer training periods and allowing deeper network structures. Additionally, it is interesting to see that the distribution of XdA at layer fc1, offers a wider flattened range after the spiked activation values centered around zero. It is assumed to occur since this layer correlates all the activations from the last convolutional layer (c3).

Figure 5.7 shows how XdA benefits from the usage of LayerNorm, because without LayerNorm the distribution of the moving averages shifts drastically during the training

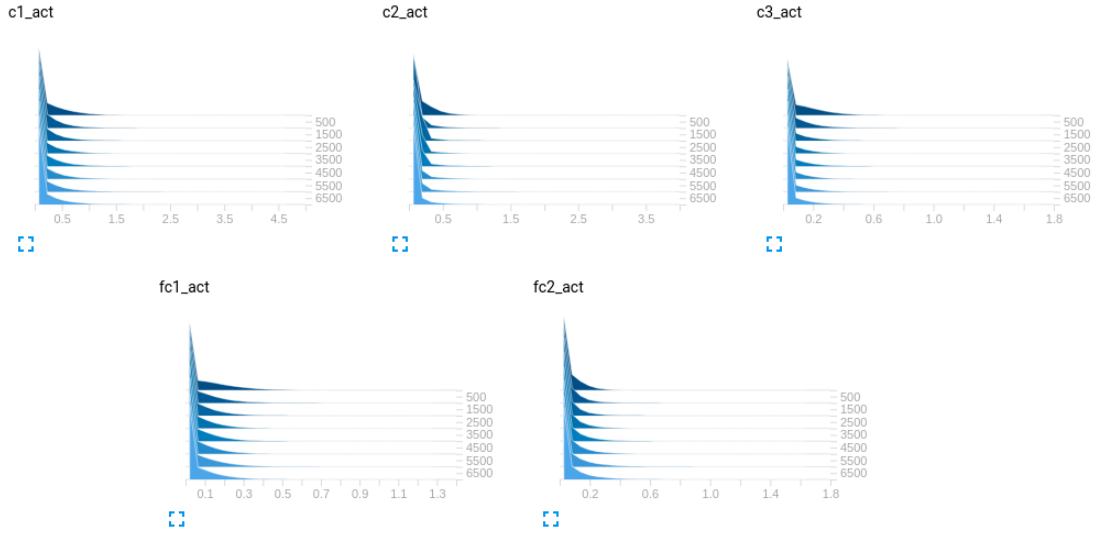


Figure 5.6: ReLUs activation distribution of CIFAR10 classes according to the convolutional (c1, c2, c3) and fully connected layers (fc1, fc2).

steps, before it settles to a stable fixpoint. This shift has an impact on the training of φ since it is initialized according to the first batch and may result in a local optimum while the actual mean moves away uncontrolled. This shift not only introduces a lot of noise during training, but while evaluating the models it reduced the overall performance, converging the threshold parameters into a pre-mature local minima. What we can see is that the trainable thresholds are similarly distributed with and without LayerNorm, and that only the magnitudes differ between both methods.

Finally, we analyzed the distribution of the activation means of XdA with LayerNorm and ReLUs with LayerNorm to show that the beneficial distribution of values of XdA does not solely rely on the LayerNorm transformation. Figure 5.8 shows XdA with LayerNorm and ReLU with LayerNorm. This illustrates that the value range of $\boldsymbol{\mu}$ is much larger for XdA compared to ReLU and emphasizes that LayerNorm is better performing with XdA, which not only improves the gradient flow of the neural network, but also offers better overall performance as we will show in the next chapter.

The bottom line 5.6.1 *In this chapter, we presented an alternative non-linearity function which in combination with synaptic stabilization methods not only improves the state-of-the-art benchmark results to overcome catastrophic forgetting, but also resembles a new tool for neural network architectures.*

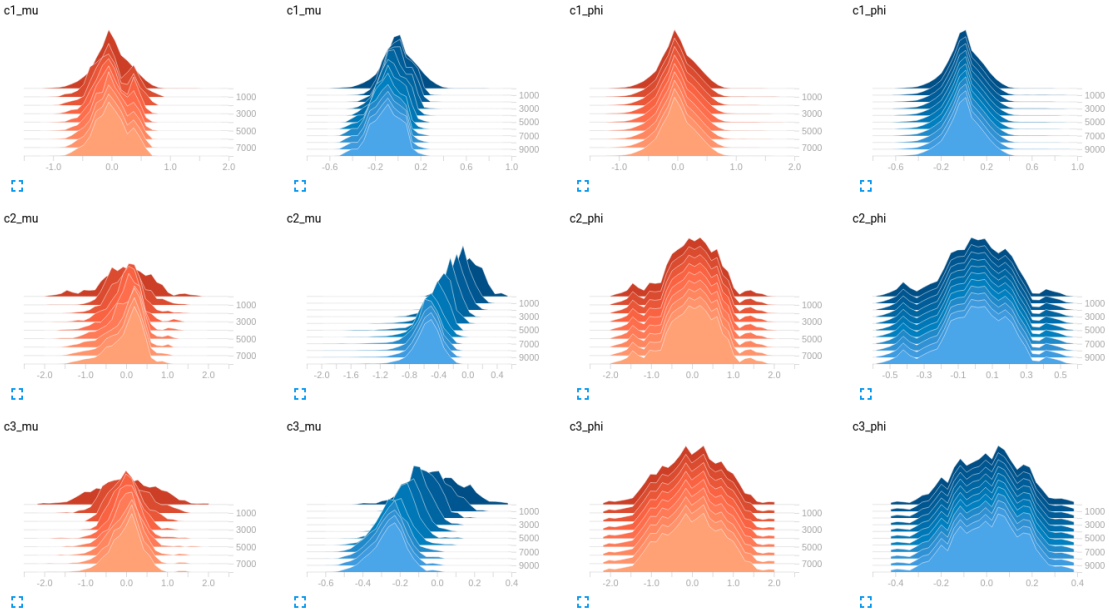


Figure 5.7: This diagram shows the shift of $\boldsymbol{\mu}$ from the convolutional layers and the trainable thresholds $\boldsymbol{\varphi}$. Orange: XdA with LayerNorm; Blue: XdA without LayerNorm.

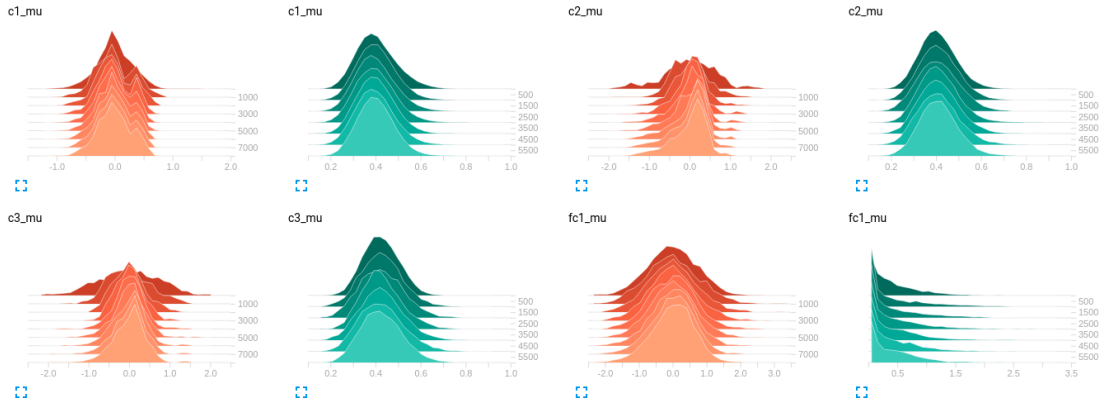


Figure 5.8: This diagram compares the shift of $\boldsymbol{\mu}$ from the convolutional layers and the first fully connected layer. Orange: XdA with LayerNorm; Green: ReLU with LayerNorm.

Chapter 6

Experiments

> "Intelligence is the ability to adapt to change."
>
> *Stephen Hawking*

The previous chapter 5 explained the application and the training procedure of context-dependent activations. Furthermore, if it analyzed the behavior of the means $\boldsymbol{\mu}$ and trainable threshold parameters $\boldsymbol{\varphi}$. In this chapter we will focus on the training benchmarks and improvements relative the HAT and variations of XdA.

6.1 Setup

To define a comparable baseline, we use the AlexNet architecture with Dropout, which was introduced in subsection 2.4.1. The layers of the neural network are initialized according to the PyTorch defaults, which are Xavier uniform distribution Glorot and Bengio, 2010. To remain comparable, we constrain the number of parameters for all methods to remain within the range of 6.7 to 7.6 million parameters. The datasets are CIFAR-10 and CIFAR-100, where we break out 10 tasks with five tasks per dataset as proposed by Lopez and Ranzato Lopez-Paz and Ranzato, 2017. For CIFAR-10 this means we randomly chose five times two classes and for CIFAR-100 we use twenty-five times twenty classes. We also alter between CIFAR-10 and CIFAR-100 tasks according to a random seed which we define during the initialization phase. The dataset is normalized before it is feed to the neural network according to the means $(125.3, 123.0, 113.9)$ and the standard deviations $(63.0, 62.1, 66.7)$ for each RGB channel respectively. The chosen optimizer is based on Stochastic Gradient Descent (SGD) with a learning rate of 0.05 and no momentum, because we are only interested in the native update and convergence behavior. We let the learning rate decay by a factor of three if no improvements were achieved after five epochs. The training of a task is stopped if a method exceeds 200 epochs or the learning rate decays below $1e{-}4$. As a reference, we trained models using plain SGD, SGD with frozen layers except of the last classifier (SGD-F), EWC, SI and HAT. We will chart the results relative to the results obtained by the XdA improvements and highlight different aspects in the subsequent sections. The following tests were computed with six random seeds, which are indicated by the confidence intervals at

the respective charts. The results are averaged to a mean point-wise score to ease the comparison between the presented methods.

6.2 Baseline

6.2.1 Method Comparison

The first Figure 6.1 covers the forgetting ratio of the XdA enhanced synaptic stabilization methods. We use a similar setup to HAT, where each task has its own task-specific threshold parameter φ^t and the parameters are manually selected when switching between tasks. The calculation of the forgetting ratio was introduced in subsection 4.0.1 and determines how much a method can adapt to new information without forgetting previous tasks. The closer the values are to zero the better the score. The black dashed

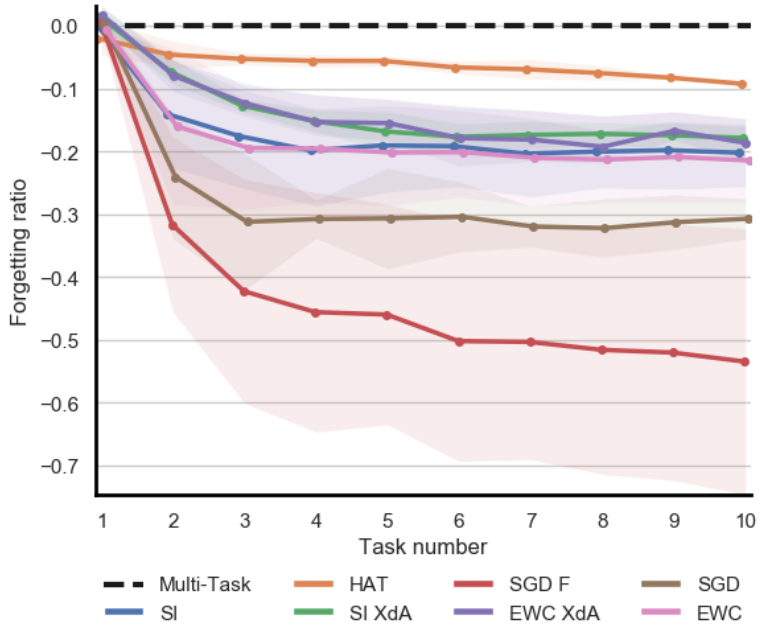


Figure 6.1: Forgetting ratio for SGD, SGD-F, SGD with multi-task, EWC, SI, HAT and XdA improved EWC and SI with a 95% confidence interval that the values rely within the shaded areas. The experiments are averaged over six random seeds.

line represents a model jointly trained on the all data samples using SGD, determining the maximum achievable score. Although the dashed line marks the maximum, both XdA approaches start slightly above this line at the first task. This is not an error, but shows the effect of context-dependent activation on plain SGD, since no regularization is applied and the optimizer can learn unrestricted. Additionally we can see that the confidence intervals become more narrow the better an approach handles forgetting and adapts to new tasks. This is congruent with our intuition, because a model that behaves sensitive to new information and forgets previous information provides different outcomes for multiple runs with different randomized seeds. This results in an increas-

ing score variance and the learning process becomes order dependent. SGD-F has the greatest variance and HAT shows the lowest variance during training. XdA improves both EWC and SI. SI is preferred, since it is applicable to online learning problems. HAT still outperforms both, but as the subsequent sections will show, this not always the case.

Although the forgetting ratio provides a good metric to compare the sensitivity versus stability of the individual models, it masks the absolute accuracies and removes the intuition of the actual task performances. To indicate the absolute accuracies and to get a more representative visualization of the individual task performances we plotted the 10 task accuracies in a bar chart as shown in the figures 6.3, 6.4 and 6.2. The tasks were trained in a sequential manner on a pre-defined seed and compare SI and EWC with and without XdA. After learning the 10th task we can, for example, see that the accuracy of SGD on the first task (1:C10-2) is almost random, since 50% marks the chance level for two classes on CIFAR-10. As we see in Figure 6.2, freezing the

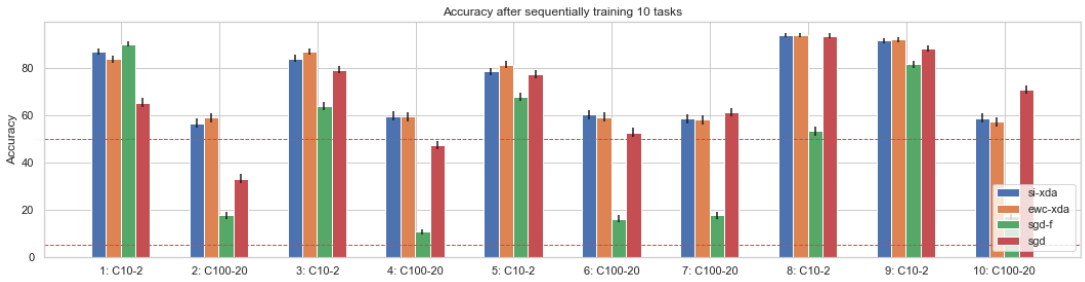


Figure 6.2: Absolute accuracies comparing SGD and SGD-F to SI-XdA and EWC-XdA after training 10 tasks sequentially. C10-2 denotes CIFAR-10 with 2 classes per task and C100-20 denotes CIFAR-100 with 20 classes per task. The 50% and 5% dashed red lines mark the random chance levels for CIFAR-10 and CIFAR-100 respectively. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

Figure 6.3: Absolute accuracies comparing SI and SI XdA with HAT and SGD. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

weights prevents forgetting on the first task, results into barely adapting subsequent tasks. HAT seems to be a better choice, but to understand the long term behavior of

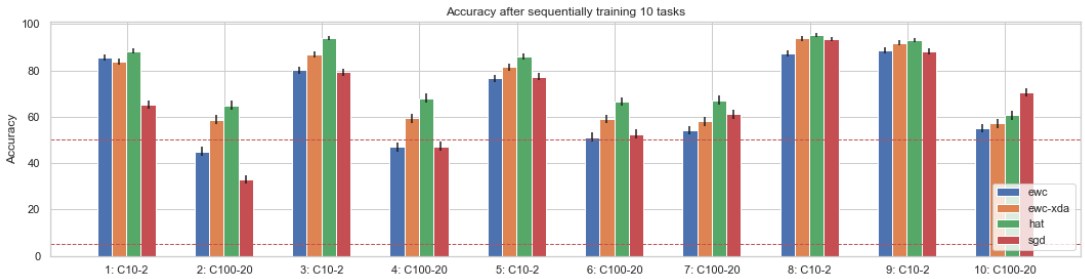


Figure 6.4: Absolute accuracies comparing EWC and EWC XdA. We see that EWC and SI behave almost similar. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

HAT, the freezing of weights requires further analysis.

6.2.2 HAT Capacity Exhaustion

Classifying images from CIFAR-10 and CIFAR-100 is very similar and relatively easy. To interpolate the long term test behavior of HAT, would require a different setup. Due to the limited computational resources, we can simulate this behavior by freezing weights from HAT. In Figure 6.5 we trained HAT not only freezing the weights of the first task but additionally 90% of its total weight capacity. This simulates the exhaustion of free parameters, similar to training multiple tasks in advance. HAT quickly drops almost to the same performance level of plain SGD and would continue dropping if we had a longer task sequence. We assume it performs slightly better on tasks that are similarly distributed according to already learned features and worse for tasks that require to learn orthogonal features. The embedding-based hard attention mask does not compensate the lack of adaptiveness in the system. It is also important to state, that the network has not reached full capacity after training on all 10 tasks. The last three layers conv3, fc1, and fc2 only reached about 96% of its total capacity. However, the HAT method could no longer adapt properly, meaning, that it could not continue learning with the remaining capacity. The setting for EWC and SI with XdA is unchanged, because it does not matter how many tasks were learned. They offer constant neural plasticity and synaptic stabilization.

In Figure 6.6 we show the absolute accuracies after training 10 tasks sequentially. As expected, we can observe that HAT performs well on CIFAR-10 similar tasks, which it was able to learn unrestricted, but it cannot further adapt to the more elaborate tasks from CIFAR-100.

6.2.3 HAT Weights Unfreezing

According to the authors of HAT, they can omit the issue of capacity exhaustion by unfreezing the backward masks of old tasks. This behavior was tested in Figure 6.7. It simulates long training sequences, where the capacity was exhausted and the model frees parameters of old tasks to allow continuous adaption. In the 10 task setup, we simulate this by unfreezing parameters before the predecessor to gain additional capacity while

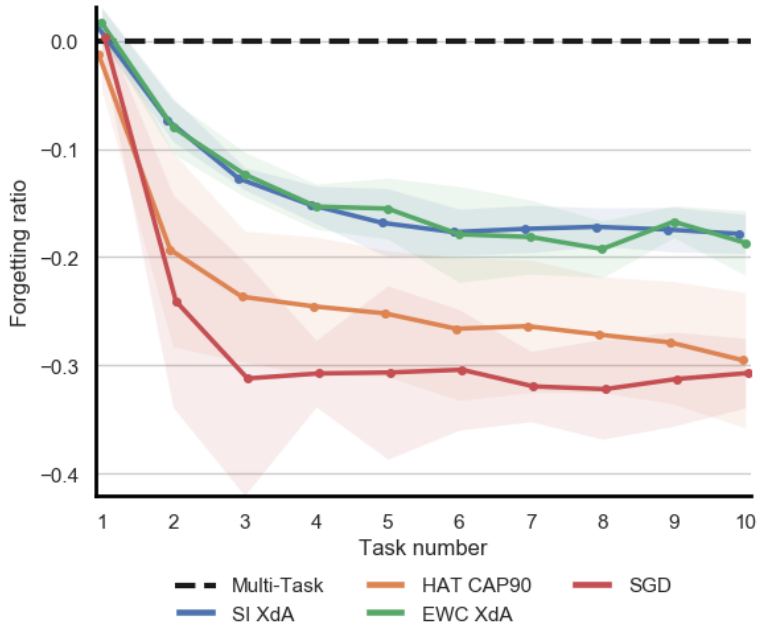


Figure 6.5: Forgetting ratio for HAT with 90% capacity frozen and XdA improved EWC and SI with a 95% confidence interval that the values rely within the shaded areas. The experiments are averaged over six random seeds.

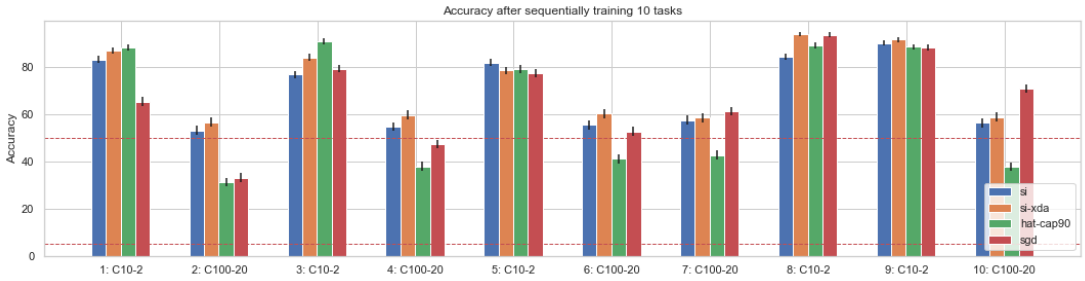


Figure 6.6: Absolute accuracies after training 10 tasks sequentially with HAT having 90% of weights exhausted after the first task. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

restricting us only on the previous task. We observe that the performance quickly drops since no other synaptic stabilization method prohibits destructive weight updates. This will continue until it levels slightly above the plain SGD line, because it offers some improvements over SGD by freezing the predecessor.

The plot clearly shows that HAT prevents catastrophic forgetting only through compression and by freezing weights. Even if the model learns meaningful features, it overwrites them when unfreezing old weights. It is also important to state, that we need to determine a scheme to unfreeze the weights, which is more difficult as it might seem. In the above section 6.2.2 we saw that before reaching the maximum capacity, the

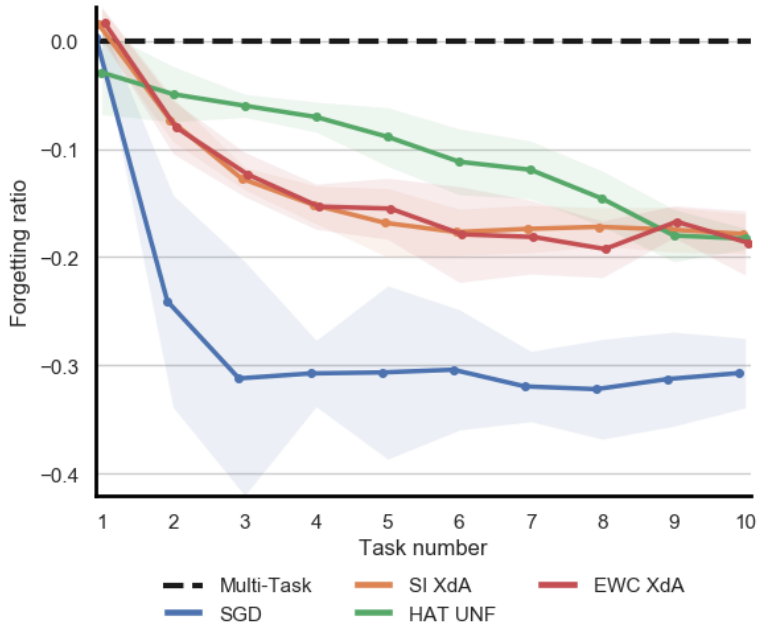


Figure 6.7: Forgetting ratio for HAT with unfrozen weights before the predecessor and XdA improved EWC and SI with a 95% confidence interval that the values rely within the shaded areas. The experiments are averaged over six random seeds. We can observe that after 10 tasks HAT with unfrozen parameters performs similar to EWC or SI with XdA.

network was no longer able to adapt. Although the first two convolutional layers c1 and c2 exhausted their full capacity, c3, fc1, and fc2 still offered free parameters. In the current setup, we also know the total number of tasks and the maximum possible accuracy per task. This allows us to estimate when our model starts to decrease its performance compared to the unrestricted SGD version. Given that we do not know the total number of tasks and their maximum performances, we need to define a learning scheme that dynamically freezes and unfreezes parameters without any reference point. In a reinforcement learning or continuous learning problem where we do not have the complete state space and retraining the model multiple times from scratch to create a performance estimate is unfeasible. We can no longer determine how well a HAT based model would perform. Additionally to the lack of baseline, the lack of context switching between the embeddings, makes HAT unpractical. For synaptic stabilization methods, the setup still does not change. The plasticity of the neurons is always given with the trade-off of forgetting some old or irrelevant features in favor of more promising ones.

For reference, we added the absolute accuracies after training 10 tasks sequentially, including the presented unfreezing scheme of HAT in Figure 6.8. For the first task, the accuracy of HAT dropped to the same level as SGD, although this is considered a novel task consisting only of two classes from CIFAR-10. This demonstrates that HAT offers no further stabilization except of freezing weights, which does not prevent overwriting important features when unfreezing parameters after the maximum capacity has been

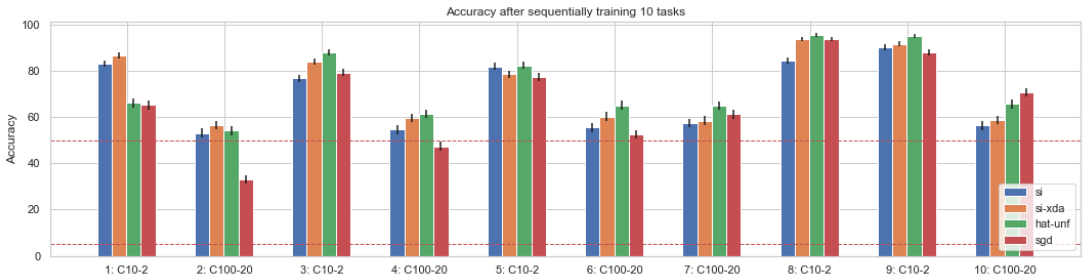


Figure 6.8: Absolute accuracies after training 10 tasks sequentially with HAT having all weights unfrozen before the direct predecessor task. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed.

reached.

6.3 Task Self-Prediction

An important aspect which we did not address until now, is that XdA with task-specific parameters also requires task pre-determination for inference. This section will focus on this issue, but only for Synaptic Intelligence, since the accuracies of SI and EWC are similar and we want to reference an online learning method, capable for continuous learning problems, such us in reinforcement learning environments. If a method requires the context information of the image in advance to associate it with the corresponding task, then the question arises why the prediction is still necessary? This is also what makes HAT unpractical and we will not further include it in the following charts. This requirement actually sorts out many approaches introduced in the related work chapter 3. SI with XdA and manual parameter selection is also not continuous learning capable but we will add it to the plots as a baseline for the following experiments.

We want to let the model predict the task context without human interference. Before XdA, SI was able to make predictions by consolidating the weights for all tasks. With the introduction of XdA and task-specific parameters, we restricted the application of EWC-XdA and SI-XdA to require external interaction. In chapter 5, we proposed a method which self-predicts its own parameter context, according to the summed absolute distance during the forward pass or according to the cosine similarity. By using the activation bias of each batch sample, we can self-predict the required parameters of each task from XdA. We denote the self-predictive version of XdA as Automated Context-Dependent Activation (AXdA). We showed that this gives even slightly better performance after the 10 task sequence according to the forgetting ratio compared to the non-optimized SI method. An additional configuration was added denoted as 1-XdA, which represents XdA with a single parameter setting shared across all 10 tasks. In Figure 6.9, we show all training results for six random seeds with three different XdA variations. The most interesting observation is that 1-XdA also slightly outperforms the manually selecting XdA version. We assume that this is either because of the small number of tasks or due to required improvements for selecting the task context. If we would increase the problem set, the additional parameters from XdA might outperform

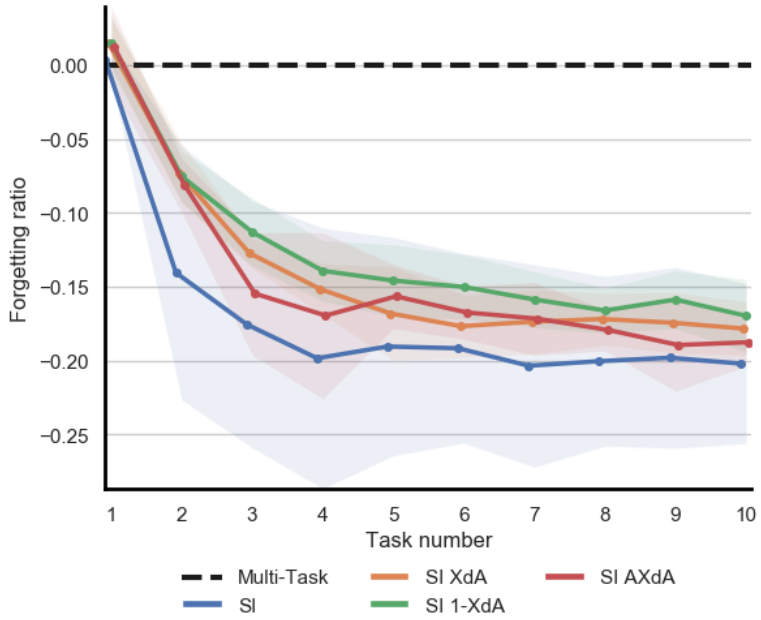


Figure 6.9: Forgetting ratio for Synaptic Intelligence with task self-prediction using AXdA, with manual task selection XdA (for comparison) and with a single parameter version for all tasks 1-XdA. We also added the 95% confidence intervals as in previous plots, averaged over six random seeds.

the 1-XdA version. By improving the parameter selection method we could gain performance on longer sequences. Additionally, by defining a regularization to reduce the set of active units we could also reduce the number of destructive updates. In any case, this would require a more in-depth analysis and exceeds the scope of this work. Either way, all approaches improve the non-optimized SI approach. We also observe that the AXdA version is slightly more unstable compared to the XdA version, but overall it behaves better than SI. By observing the confidence intervals, we see that SI has a much wider spread than AXdA. This indicates a more stable learning behavior and could be preferable to longer task sequences.

In Figure 6.11, we show the absolute accuracies after training 10 tasks sequentially to illustrate how AXdA performs relative to SI with XdA, 1-XdA and simple SI.

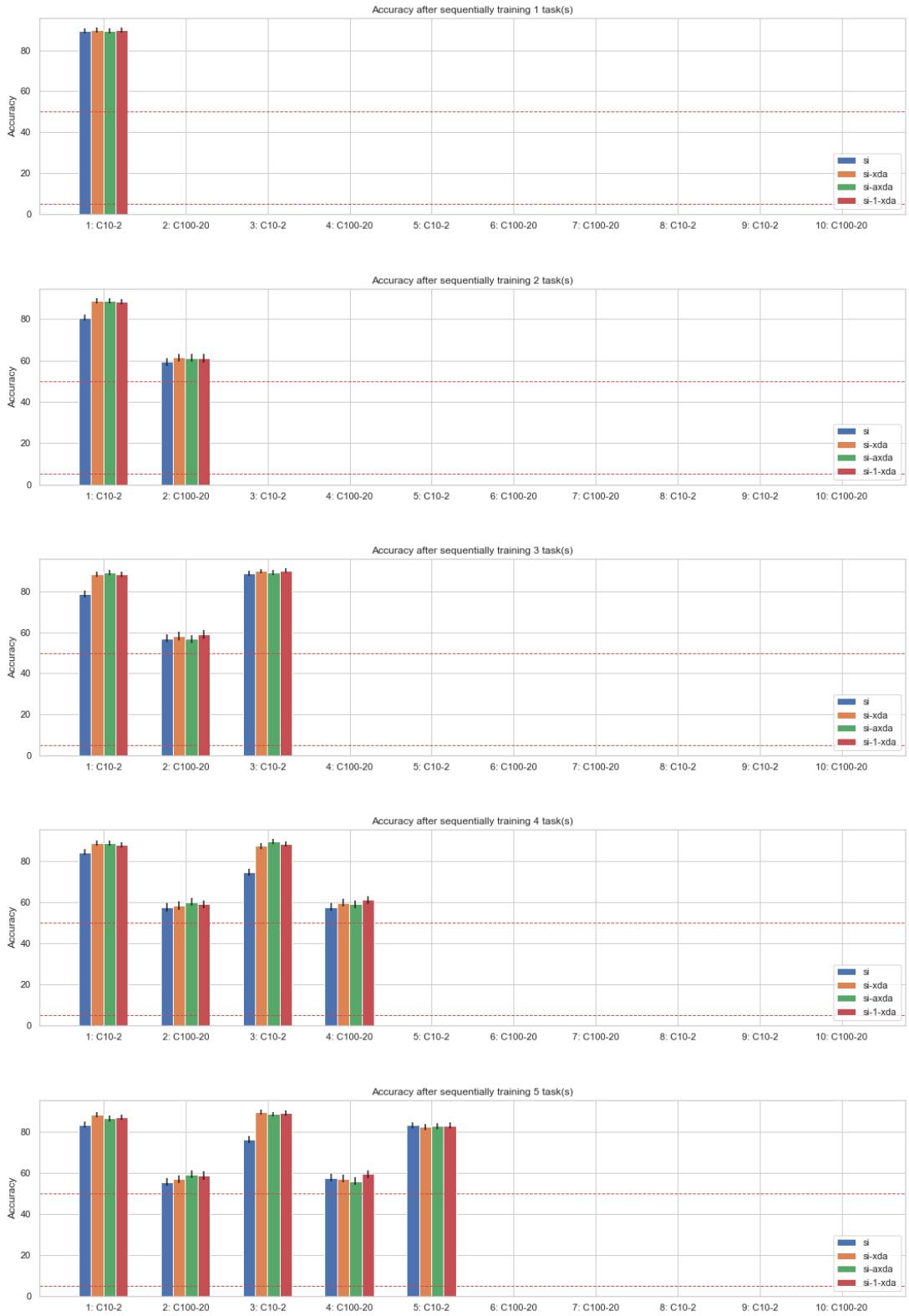


Figure 6.10: Absolute accuracies for all 10 tasks trained sequentially with self-predictive task inference of AXdA compared to XdA and 1-XdA. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed. (Part 1/2)

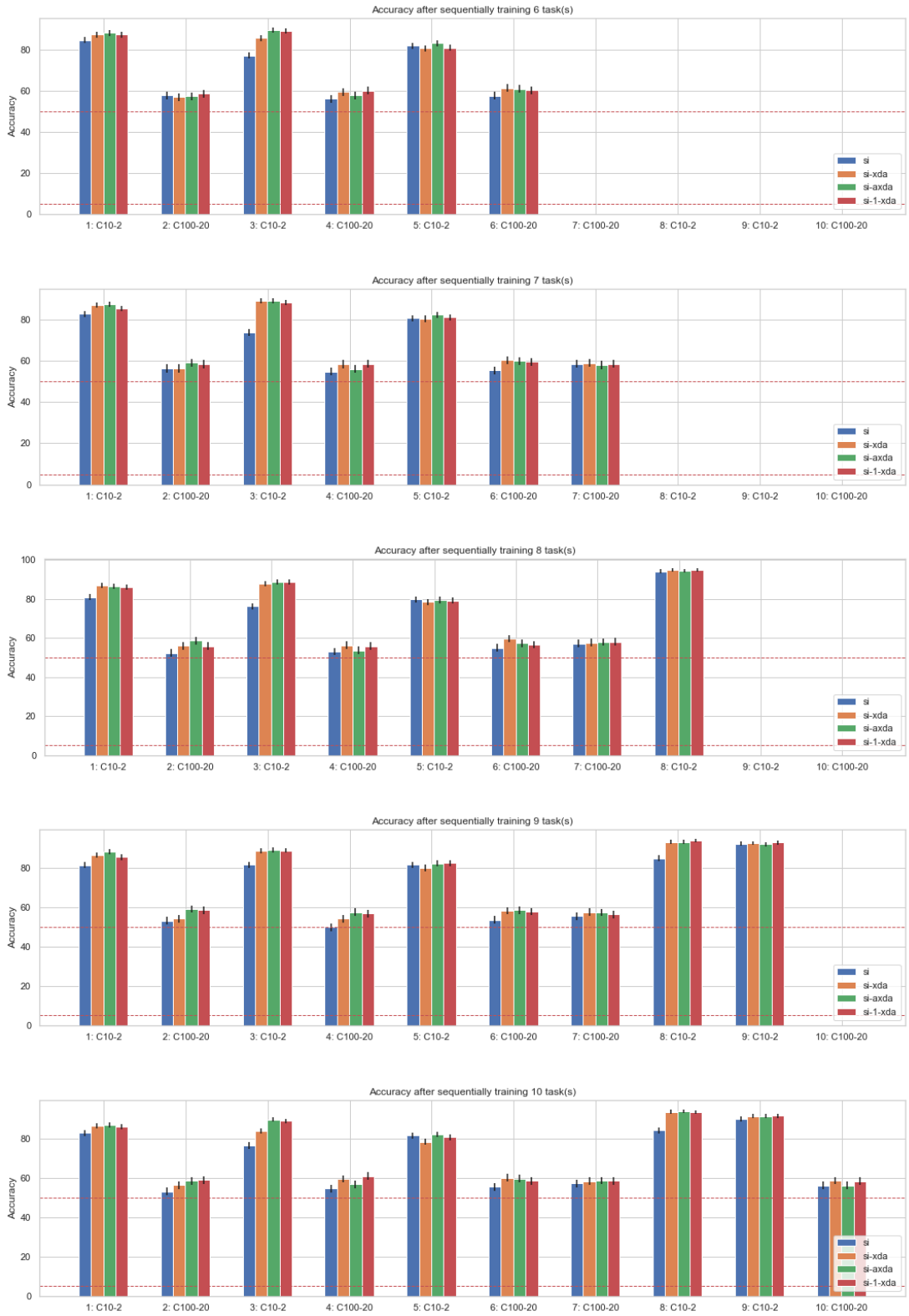


Figure 6.11: Absolute accuracies for all 10 tasks trained sequentially with self-predictive task inference of AXdA compared to XdA and 1-XdA. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed. (Part 2/2)

We can see that on average AXdA performs equally well compared to the other approaches that get their context manually set or have no context to select. To rule out random chance, we also trained a different model with an alternative distance metric, including a sub-optimal distance metric based on the mean squared error (MSE). This penalizes outliers more heavily and results in partially wrong task choices during training and inference, which heavily impacts the performance. We compare the MSE version of XdA with the cosine similarity and the summed absolute distances in Figure 6.12. We can observe that XdA with MSE has two major bumps after training on the 4th and 10th task. The forgetting ratio also considers intermediate performance drops, which explains why the overall performance of AXdA is partially worse than XdA. The confidence interval around the 4th task is much larger and emphasizes that during training the performance varied heavily. This indicates that the comparison criteria to select the correct task context is crucial and different metrics have an impact on the prediction stability. For example, by using the cosine similarity as a third metric we observe that the performance drops temporarily at task five, but outperforms the SAD-XdA after the 8th task. In Figure 6.14 we also added the absolute accuracies. Herby we demonstrated

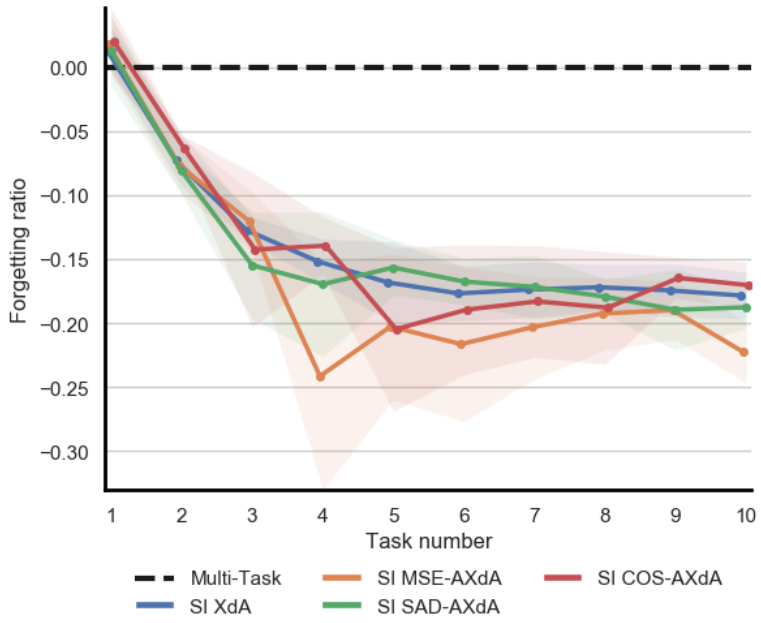


Figure 6.12: Forgetting ratio for SI with task self-prediction using cosine similarity as a metric SI-COS-AXdA with a 95% confidence interval that the values rely within the shaded areas. The experiments are averaged over six random seeds.

that task self-prediction based on the activation space can offer competitive results and context-dependent activations can be applied without any external interactions. This offers a new tooling for developing dynamically extensible networks.

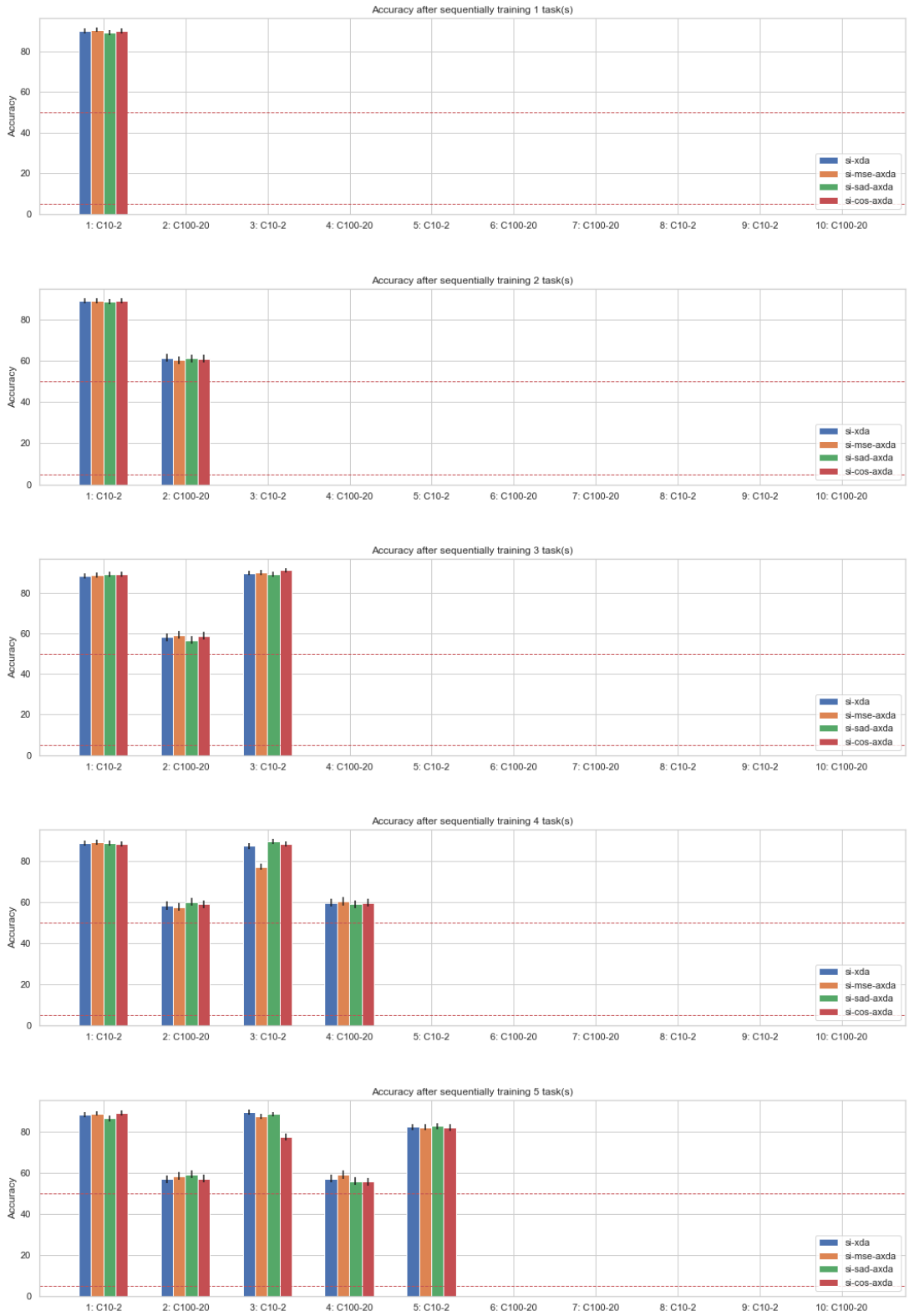


Figure 6.13: Absolute accuracies after training 10 tasks sequentially with self-predictive task inference of COS-AXdA. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed. (Part 1/2)

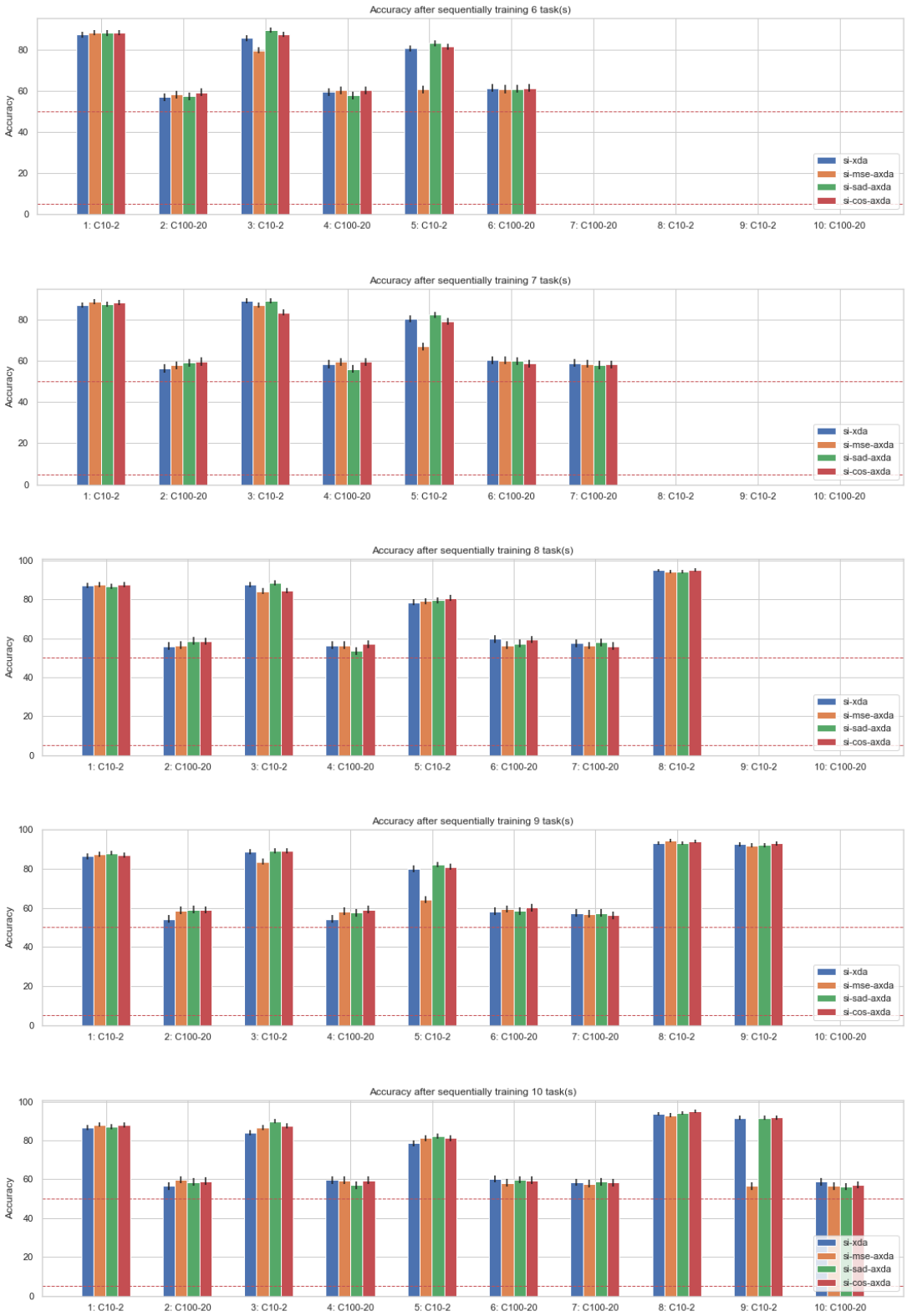


Figure 6.14: Absolute accuracies after training 10 tasks sequentially with self-predictive task inference of COS-AXdA. The error bars are computed according to the standard error of the mean based on four runs with a fixed seed. (Part 2/2)

6.4 Class Activation Maps

To visualize the activations of XdA, we added class activation maps after the first convolutional layer, as shown in Figure 6.15. These images illustrate the perception of

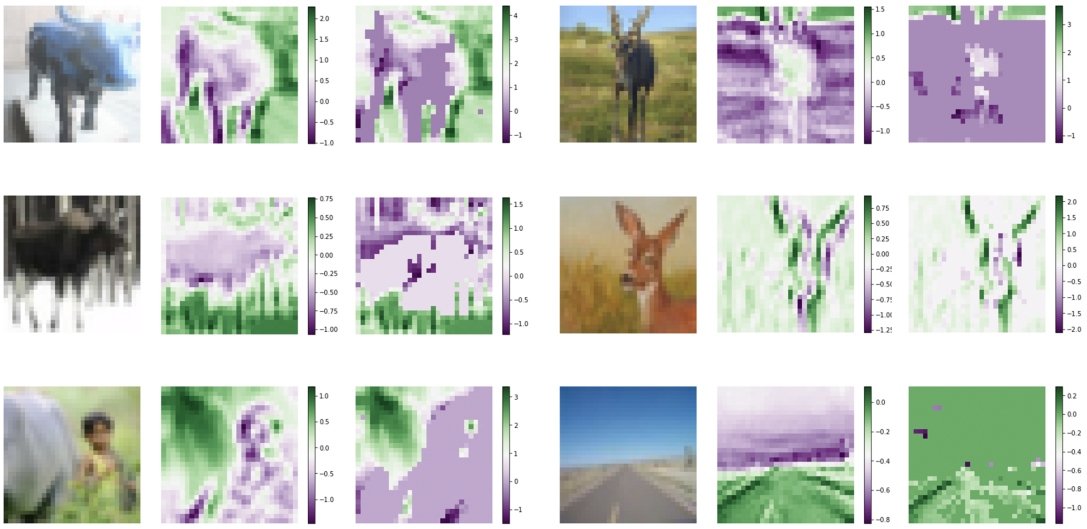


Figure 6.15: Class activation maps of the first convolutional layer (c1). Each row shows two image groups always starting with the original image, followed by the unchanged activation maps and the context-based activations.

the individual neurons. The uniformly colored patches indicate the rectified activation values from XdA. This illustrates the area of attention according to the class importance of XdA. We can see, that the image in the first row and first column, showing a bull, filters away the bull and classifies the image by focusing on its surrounding area. The last image, in the third row, fourth column, showing a picture of a road, filters away the sky and focuses on the actual image part. The class activation maps demonstrate that the network learns meaningful filters, selecting important features—similar to attention—to classify the image.

Chapter 7

Conclusion

> "I learned very early the difference between knowing the name of something and knowing something."
>
> *Richard P. Feynman*

7.1 Summary

This thesis analyzed and presented promising results for an alternative non-linearity function known as Context-Dependent Activations, which is capable of improving deep neural networks using synaptic stabilization for continuous learning tasks. It uses trainable threshold parameters to rectify the activation space and allows dynamic network extensibility without disrupting past task predictions. We also presented a novel method to avoid manual context selection and allowed the model to self-predict its own context by tracking the exponential moving average of the activation space. In addition, this non-linearity method outperformed ReLUs on single task predictions without requiring any architectural changes, which allows it to be easily integrated into existing network structures. Context self-prediction is a powerful tool which allows dynamic decisions making and reduces disruptive updates to non-contributing or irrelevant units. To further improve research in this field we propose several aspects to analysis in the next section.

7.2 Future Work

7.2.1 Long Sequence Experiments

In chapter 6 we observed that the single parameter setup of φ slightly outperformed the multi-parameter self-task prediction setup. This is an interesting result and opens up new questions regarding long sequence learning behavior and if the multi-parameter setup keeps improving over time. Based on the work from Masse, Grant, and Freedman, 2018, context-dependence improves long term synaptic stabilization methods and we

assume the same results for our work, but we leave the exploration of these setups open for future work.

7.2.2 Improving the Similarity Metric

In chapter 5 we proposed two metrics to compute the similarity metric between the tracked activation means and the provided sample activations. The absolute error and cosine similarity offered competitive results compared to the manual selected task predictions. We also observed that the total number of forwarded activations is constantly reduced to 50% with the absolute error metric. Further reduction may avoid further forgetting and may increase the task-specific accuracy.

7.2.3 Reduce Activation Overlap

The active units mark the parameter set which is altered by the backpropagation of the error term and the more updates overlap with previously important weights the more these alternations will contribute to forgetting of old tasks. By reducing the number of active units and the overlapping units per task, we can promote better adaptiveness to tasks without altering previously important tasks.

7.2.4 Hierarchical Context Selection

Self-task prediction requires $\mathcal{O}(k * t)$ number of computations during the forward pass, where k denotes the layer index and t the number of tasks. This increases training linearly, since the number of layers is fixed, but for very long task sequences it become unfeasible. Therefore, we suggest to use hashing methods as proposed by Weinberger et al., 2009, and switch between the threshold parameters. This, of course, has to be further tested and marks an interesting continuation of this work.

7.2.5 Reinforcement Learning Application

The implementation of Synaptic Intelligence with Context-Dependent Activations improved the performance of the 10 task setup using CIFAR-10 and CIFAR-100. It also allows an online learning setup to train in a reinforcement learning environment, but requires further analysis to determine the optimal number of task thresholds, regularization term and self-context prediction metric.

7.2.6 Dynamic Memory Allocation

In reinforcement learning, the number of tasks is often not inferable and would hinder the ability of Context-Dependent Activations to allocate additional parameters. To avoid manual task pre-determination we suggest to compute deviation between the tracked activation means and the sample activations and reallocate new memory if a certain percentage is exceeded. This would mark a novel method to increase the parameter size and in combination with self-task prediction would not require any external interference during the forward passes.

References

Literature

[1] Carlos Améndola, Alexander Engström, and Christian Haase. "Maximum Number of Modes of Gaussian Mixtures". abs/1702.05066 (2017). arXiv: 1702.05066. URL: http://arxiv.org/abs/1702.05066 (cit. on p. 35).

[2] Lei Jimmy Ba, Ryan Kiros Kiros, and Geoffrey E. Hinton. "Layer Normalization". *arXiv e-prints* (July 2016). arXiv: 1607.06450 [stat.ML] (cit. on pp. 22, 59).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: http://arxiv.org/abs/1409.0473 (cit. on p. 23).

[4] Marcus K. Benna and Stefano Fusi. "Computational principles of synaptic memory consolidation". *Nature neuroscience* (2016) (cit. on p. 29).

[5] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning*. 1st ed. Morgan & Claypool Publishers, 2016 (cit. on p. 7).

[6] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555 (cit. on p. 2).

[7] Rosangela Saher Cintra and Haroldo F. de Campos Velho. "Data Assimilation by Artificial Neural Networks for an Atmospheric General Circulation Model: Conventional Observation". *CoRR* abs/1407.4360 (2011) (cit. on p. 14).

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". *Under Review of ICLR2016 (1997)* (Nov. 2015) (cit. on p. 18).

[9] Chrisantha Fernando, Dylan Banarse, et al. "PathNet: Evolution Channels Gradient Descent in Super Neural Networks". *CoRR* abs/1701.08734 (2017). arXiv: 1701.08734. URL: http://arxiv.org/abs/1701.08734 (cit. on p. 37).

[10] Chrisantha Fernando, Eörs Szathmáry, and Phil Husbands. "Selectionist and Evolutionary Approaches to Brain Function: A Critical Appraisal". In: *Front. Comput. Neurosci.* 2012 (cit. on p. 37).

[11] Robert M. French. “Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks”. *In Proc. of the Annual Conf. of the Cognitive Science Society (CogSci)* (1991), pp. 173–178 (cit. on p. 8).

[12] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. *Biological Cybernetics* (1980), pp. 267–285 (cit. on p. 20).

[13] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. Vol. 9. Chia Laguna Resort, Sardinia, Italy, May 2010, pp. 249–256 (cit. on p. 62).

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html (cit. on p. 18).

[15] Ian J. Goodfellow, Mehdi Mirza, et al. “An empirical investigation of catastrophic forgetting in gradient-based neural networks”. *CoRR* abs/1312.6211 (2014). arXiv: 1312.6211. URL: http://arxiv.org/abs/1312.6211 (cit. on p. 24).

[17] Ian J. Goodfellow, Jean Pouget-Abadie, et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (cit. on p. 55).

[16] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. “Qualitatively characterizing neural network optimization problems”. *CoRR* abs/1412.6544 (2014). arXiv: 1412.6544. URL: http://arxiv.org/abs/1412.6544 (cit. on p. 36).

[18] Alex Graves, Greg Wayne, and Ivo Danihelka. “Neural Turing Machines”. *CoRR* abs/1410.5401 (2014). arXiv: 1410.5401. URL: http://arxiv.org/abs/1410.5401 (cit. on p. 1).

[19] Kaiming He et al. “Deep Residual Learning for Image Recognition”. *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 21, 24).

[20] Robert Hecht-Nielsen. “Theory of the Backpropagation Neural Network”. *In Proceedings of the International Joint Conference on Neural Networks, volume I, pages 593–605. Piscataway, NJ: IEEE* (1989) (cit. on p. 29).

[21] Geoffrey E. Hinton, Nitish Srivastava, et al. “Improving neural networks by preventing co-adaptation of feature detectors”. *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580 (cit. on p. 22).

[22] Geoffrey E. Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. *NIPS Workshop* (2014) (cit. on p. 32).

[23] Sepp Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. *Technical University Munich, Institute of Computer Science* (1991) (cit. on p. 17).

[24] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. *Neural Computation, 9(8):1735–1780* (1997) (cit. on p. 2).

[25] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993 (cit. on p. 24).

[26] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (cit. on p. 21).

[27] Heechul Jung et al. “Less-forgetting Learning in Deep Neural Networks”. *CoRR* abs/1607.00122 (2016). arXiv: 1607.00122. URL: http://arxiv.org/abs/1607.00122 (cit. on pp. 30, 31).

[28] Lukasz Kaiser et al. “Learning to Remember Rare Events”. *CoRR* abs/1703.03129 (2017). arXiv: 1703.03129. URL: http://arxiv.org/abs/1703.03129 (cit. on p. 2).

[29] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. *Proc. of the National Academy of Sciences of the USA, 114(13):3521–3526* (2017) (cit. on pp. 3, 29, 52).

[30] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 971–980. URL: http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf (cit. on pp. 18, 19).

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 23).

[32] Yann LeCun, Bernhard E. Boser, et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. *Neural Computation* (1989), pp. 541–551 (cit. on p. 20).

[33] Yann LeCun, Léon Bottou, et al. “Gradient-based learning applied to document recognition” (1998), pp. 2278–2324 (cit. on p. 23).

[34] Sang-Woo Lee et al. “Overcoming Catastrophic Forgetting by Incremental Moment Matching”. *In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), Advances in Neural Information Processing Systems (NIPS), volume 30, pp. 4655–4665. Curran Associates Inc.* (2017) (cit. on pp. 33, 34).

[35] H. Legg and M. Hutter. “Universal Intelligence: A Definition of Machine Intelligence”. *CoRR* abs/0712.3329 (2007). arXiv: 0712.3329. URL: http://arxiv.org/abs/0712.3329 (cit. on p. vii).

[36] Zhizhong Li and Derek Hoiem. “Learning Without Forgetting”. *IEEE Trans. on Pattern Analysis and Machine Intelligence pp (99):1–1* (2017) (cit. on p. 32).

[37] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.” Cambrindge, Massachusetts: University of Helsinki, 1970 (cit. on p. 14).

[38] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient Episodic Memory for Continuum Learning". *CoRR* abs/1706.08840 (2017). arXiv: 1706.08840. URL: http://arxiv.org/abs/1706.08840 (cit. on pp. 46, 62).

[39] Wenjie Luo et al. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks". *CoRR* abs/1701.04128 (2017). arXiv: 1701.04128. URL: http://arxiv.org/abs/1701.04128 (cit. on p. 21).

[40] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421. URL: https://www.aclweb.org/anthology/D15-1166 (cit. on p. 23).

[41] David J.C. MacKay. "Choice of Basis for Laplace Approximation". *Machine Learning* 33.1 (Oct. 1998), pp. 77–86. URL: https://doi.org/10.1023/A:1007558615313 (cit. on p. 30).

[42] Arun Mallya and Svetlana Lazebnik. "PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning". *CoRR* abs/1711.05769 (2017). arXiv: 1711.05769. URL: http://arxiv.org/abs/1711.05769 (cit. on pp. 38, 39).

[43] Nicolas Y. Masse, Gregory D. Grant, and David J. Freedman. "Alleviating Catastrophic Forgetting Using Context-Dependent Gating and Synaptic Stabilization". *CoRR* abs/1802.01569 (2018). arXiv: 1802.01569. URL: http://arxiv.org/abs/1802.01569 (cit. on pp. 4, 44, 45, 51–53, 76).

[44] Michael McCloskey and Neal J. Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". *In G. H. Bower (ed.)* 24 (1989), pp. 109–164 (cit. on p. 5).

[45] Pamela McCorduck. *Machines Who Think*. New York, NY, USA: W. H. Freeman & Co., 1979 (cit. on p. 1).

[46] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. "Regularizing and Optimizing LSTM Language Models". *CoRR* abs/1708.02182 (2017). arXiv: 1708.02182. URL: http://arxiv.org/abs/1708.02182 (cit. on p. 2).

[47] Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, eds. *Neural Networks: Tricks of the Trade - Second Edition*. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-35289-8 (cit. on p. 22).

[48] Surajit Ray and Bruce G. Lindsay. "The topography of multivariate normal mixtures". *Annals of Statistics, pages 2042–2065* (2005) (cit. on p. 35).

[49] Anthony Robins. "Catastrophic Forgetting, Rehearsal, and Pseudorehearsal". *Connection Science* 7.2 (1995) (cit. on p. 5).

[50] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* 1961 (cit. on p. 14).

[51] Sebastian Ruder. "An overview of gradient descent optimization algorithms". *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (cit. on p. 15).

[52] Sebastian Ruder. "An Overview of Multi-Task Learning in Deep Neural Networks". *CoRR* abs/1706.05098 (2017). arXiv: 1706.05098. URL: http://arxiv.org/abs/1706.05098 (cit. on p. 6).

[53] Andrei A. Rusu et al. "Progressive Neural Networks". *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671 (cit. on pp. 27, 28).

[54] Joan Serrà et al. "Overcoming catastrophic forgetting with hard attention to the task". *CoRR* abs/1801.01423 (2018). arXiv: 1801.01423. URL: http://arxiv.org/abs/1801.01423 (cit. on pp. 3, 7, 23, 39, 41, 46–48, 50, 52, 53).

[55] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". *Nature 529(7587):484* (2016) (cit. on p. vii).

[56] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (cit. on p. 24).

[57] Nitish Srivastava et al. "Dropout: a simple way to prevent Neural Networks from Overfitting". *Journal of Machine Learning Research 15(1):1929–1958* (2014) (cit. on p. 36).

[58] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". *MIT Press, Cambridge, MA* (2017) (cit. on pp. vii, 10, 13).

[59] Christian Szegedy et al. "Going Deeper with Convolutions". *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842 (cit. on p. 23).

[60] Kilian Q. Weinberger et al. "Feature Hashing for Large Scale Multitask Learning". *CoRR* abs/0902.2206 (2009). arXiv: 0902.2206. URL: http://arxiv.org/abs/0902.2206 (cit. on p. 77).

[61] Paul John Werbos. "Beyond Regression". New tools for prediction and analysis in the behavioral science. Cambrindge, Massachusetts: Harward University, Jan. 1975 (cit. on p. 1).

[62] Thomas Wiatowski and Helmut Bölcskei. "A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction". *CoRR* abs/1512.06293 (2015). arXiv: 1512.06293. URL: http://arxiv.org/abs/1512.06293 (cit. on p. 6).

[63] Bing Xu et al. "Empirical Evaluation of Rectified Activations in Convolutional Network". *CoRR* abs/1505.00853 (2015). arXiv: 1505.00853. URL: http://arxiv.org/abs/1505.00853 (cit. on p. 18).

[64] Baosong Yang et al. "Convolutional Self-Attention Networks". *CoRR* abs/1904.03107 (2019). arXiv: 1904.03107. URL: http://arxiv.org/abs/1904.03107 (cit. on p. 23).

[65] Greg Yang and Samuel S. Schoenholz. "Mean Field Residual Networks: On the Edge of Chaos". *CoRR* abs/1712.08969 (2017). arXiv: 1712.08969. URL: http://arxiv.org/abs/1712.08969 (cit. on p. 18).

[66] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". *CoRR* abs/1511.07122 (2015). arXiv: 1511.07122. URL: http://arxiv.org/abs/1511.07122 (cit. on p. 21).

[67] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual Learning Through Synaptic Intelligence". *Conference on Machine Learning pp. 3987–3995* (2017) (cit. on pp. 3, 8, 42, 52).

Online sources

[68] Soham Chatterjee. *Different Kinds of Convolutional Filters*. 2017. URL: https://stage.saama.com/blog/different-kinds-convolutional-filters/ (cit. on p. 21).

[69] Stefan Kojouharov. *Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data*. 2017. URL: https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463 (cit. on p. 10).

[70] C. Moyer. *How Google's AlphaGo Beat a Go World Champion*. 2016. URL: https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/ (cit. on p. vii).

[71] Michael Aaron Nielsen. *Neural Networks and Deep Learning*. 2016. URL: http://neuralnetworksanddeeplearning.com/ (cit. on p. 15).

[72] PyTorch Tutorials. *Word Embeddings - Encoding Lexical Semantics*. 2017. URL: https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html (cit. on p. 23).

[73] CS231n Stanford Blog. *CS231n Convolutional Neural Networks for Visual Recognition*. 2019. URL: http://cs231n.github.io/convolutional-networks/ (cit. on p. 20).

[74] University of Toronto Website. *The CIFAR-10 dataset*. 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html (cit. on pp. 25, 26).

[75] Wikipedia. *Convolutional Neural Network*. 2019. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (cit. on p. 20).

[76] Wikipedia. *Supervised learning*. 2019. URL: https://en.wikipedia.org/wiki/Supervised_learning (cit. on p. 9).