Stochastic Deep Networks with Linear Competing Units for Transfer Learning

by

PhD Candidate: Konstantinos Kalais

Supervisor: Dr Sotirios Chatzis

Doctoral Dissertation

Cyprus University of Technology

Cyprus University of Technology

Faculty of Engineering and Technology

Department of Electrical Engineering, Computer Engineering and Informatics

Limassol, January 2024

Approval Form

Doctoral Dissertation

Stochastic Deep Networks with Linear Competing Units for Transfer Learning

Presented by

Konstantinos Kalais

Supervisor: Dr. Sotirios Chatzis, Associate Professor, Cyprus University of Technology

Signature _____

Member of the committee: Dr. Michael Sirivianos, Associate Professor, Cyprus University of Technology

Signature _____

Member of the committee: Dr. Sergios Theodoridis, Distinguished Professor, Aalborg University

Signature _____

Cyprus University of Technology

Limassol, January 2024

Copyrights

The approval of the dissertation by the Department of Electrical Engineering, Computer Engineering and Informatics does not imply necessarily the approval by the Department of the views of the writer.

Acknowledgments

I would like to thank my supervisor Dr. Sotirios Chatzis for his guidance, support and feedback he have provided throughout my studies, as well as the people that constitute the Statistical Machine Learning Laboratory in Cyprus University of Technology. I would also like to thank my friends and family for constantly encouraging me throughout my graduate years.

Abstract

Deep Learning (DL) has become the preferred approach to addressing various challenging machine learning (MLe) tasks, like computer vision, natural language processing, and speech recognition. Deep Neural Networks (DNN's) have achieved superior performance in those tasks compared to traditional MLe methods. However, they entail a huge number of weights causing them to make over-confident predictions, that may reduce their generalization capacity in hard problems, e.g. a Meta-Learning (ML) scenario. To mitigate this issue, researchers have applied Bayesian modeling to DNN's, where they employ Bayesian Neural Networks (BNN's) with more robust and tractable estimates of uncertainty in a model's predictions. That way, we can build safer MLe systems in safety-critical applications, such as healthcare, video recognition, and autonomous vehicle control.

DL models trained to solve a single task suffer from a common drawback: they cannot combine data from diverse tasks in order to learn new tasks in a future training round. Such models often require extensive training and data collection for each task individually, which can be time-consuming and data-intensive. This limitation has given rise to the importance of research in ML. This field aims to address these shortcomings by developing methods that allow existing models to efficiently learn from and adapt quickly to new tasks, by leveraging knowledge gained from previous tasks. Therefore, ML methods aim to make models more capable of generalizing well to unseen tasks with just a small amount of examples; this is the so-called problem of *few-shot learning*.

In this thesis, we aim to study how some existing DL methods for ML are used to tackle this phenomenon, and suggest a novel ML method regarding improving generalization capacity, predictive performance, and computational efficiency. Specifically, our proposed approach relies on the concepts of *stochastic* and *sparse learned representations*. In that way, we aim to define a sparse and stochastic network paradigm for ML, with novel network design principles compared to currently used ML models; we use stochastic deep networks with linear competing units in the context of model-agnostic ML. As we empirically show, our approach produces state-of-the-art predictive accuracy on few-shot image classification and regression experiments, as well as reduced predictive error on an active learning setting;

these improvements come with an immensely reduced computational cost.

These encouraging results, further motivate us to also examine the case where we do not have all tasks available beforehand, but they come in sequentially. In such a case, a DNN should learn to adapt to this continuous stream of data, effectively handling a major problem that affects DNN's in such settings, namely *catastrophic forgetting*. Continual Learning (CL) methods are designed to mitigate or reduce this issue. Specifically, such a method learns the DNN to accumulate new knowledge after a few training iterations on a new data distribution, and avoid drastically forgetting previously learned information from older tasks. Recently, researchers have developed various approaches in order to counteract this problem.

To address this challenge, this thesis proposes a radically different regard toward addressing catastrophic forgetting in CL tasks, and especially in a famous variant of CL called class-incremental learning (CIL). Our approach is founded upon the framework of stochastic local competition which is implemented in a task-wise manner. We have shown that it produces state-of-the-art predictive accuracy on few-shot image classification experiments, and imposes a considerably lower computational overhead compared to the current state-of-the-art.

Table of Contents

List of Tables and Figures

List of Publications

Konstantinos Kalais and Sotirios Chatzis. “Stochastic Deep Networks with Linear Competing Units for Model-Agnostic Meta-Learning”. In: International Conference on Machine Learning. 2022.

Konstantinos Kalais and Sotirios Chatzis. “Continual Learning via Winning Subnetworks That Arise Through Stochastic Local Competition”. In: International Conference on Learning Representations. 2024.

Chapter 1

Introduction to Deep Learning

Machine Learning (MLe) is a sub-field of Artificial Intelligence that deals with the capability of a machine to imitate intelligent human behavior. Specifically, it focuses on the use of statistical methods and algorithms trained to make classifications or predictions by obtaining key insights from data mining. First, we need to gather any type of data, like text, photos, numbers or time series data; this will consist the training data, which is the information the MLe model will be trained on. Then, depending on the data type, we choose a method in order to train the model and help it find patterns or make predictions. We might need to tweak the model by slightly changing its hyper-parameters in order to enhance or accelerate the training period, and to finally obtain more robust representations of the data and accurate predictive results. A proportion of the initial training data would be used as evaluation data, which tests how the model performs when it is shown previously unseen data.

Mathematically, a MLe task tries to learn a function $f : \mathbb{R}^N \to \mathbb{R}^M$ that maps inputs x from a distribution D_{in} to corresponding outputs y from a distribution D_{out}. Function f is usually parameterized by a set of trainable parameters $\boldsymbol{\theta}$ (weights $\boldsymbol{w}$), and the output is expressed as $y = f_{\boldsymbol{\theta}}(x)$. The training procedure aims to learn the optimal parameters $\boldsymbol{\theta}$ that would minimize the objective - *loss* - function, L, between predicted and target outputs. This metrical function compares the distance between target (real) and predicted output values and is different depending on the nature of the problem we are trying to solve. Generally, the (average) loss L_{av} is defined as:

$$L_{av}(\boldsymbol{w}|x) = \frac{1}{M}\sum_{i=1}^{M} L(y_{pred}^{(i)}, y_{real}^{(i)}) \tag{1.1}$$

where $y_{pred}^{(i)} = [f_{\boldsymbol{\theta}}(x)]^{(i)}$.

Deep Learning (DL) is a type of MLe based on Artificial Neural Networks (ANN's) with a large architecture, that retains multiple hidden layers helping to optimize and refine for accuracy. MLe algorithms can learn from small sets of data and may need human intervention when they get something wrong. However, DL algorithms requires big datasets that might contain diverse or/and unstructured data and can improve their outcomes through repetition, without the need of human correction. DL technology lies behind everyday products and services, such as voice-enabled TV remotes, digital assistants and self-driving cars. MLe and DL models are capable of different types of learning, which are: *supervised* (Nasteski (2017)), *unsupervised* (Dridi (2021)) and *reinforcement learning* (Li (2022b)). Supervised MLe processes labeled input and output training data, wheres unsupervised MLe relies on unlabelled or raw data and their internal structures to obtain patterns. Reinforcement learning trains a model through trial and error to take the best action by establishing a reward system.

The remainder of this thesis is organized as follows: In the rest of Chapter 1, we make a brief introduction to the basic structures and functionalities of ANN's, as well as describe deep architectures in a more thorough context. In Chapter 2, we present the mathematical background of the Bayesian Deep Learning fundamentals. In Chapter 3, we address the problem of Model-Agnostic Meta-Learning in a novel and very effective way. Specifically, our work brings to the forefront stochastic local winner-takes-all competition fundamentals. In Chapter 4, we move one step further and consider the case of iterative, Continual Learning. Relying on the solid foundation of stochastic competition principles, we introduce a radically novel and effective framework for class-incremental learning using deep networks. Finally in Chapter 5, we will provide the conclusions of the thesis accompanied with future research area topics.

1.1 Artificial Neural Networks

ANN's are biological inspired structures, that develop robust algorithms and data structures in order to investigate how simple models of biological brains can be used to solve difficult

computational tasks like the predictive modeling tasks we see in MLe.

The *perceptron* is the simplest computational module used to model a biological neuron in an ANN; it provides the foundation for later developing much larger networks. The idea about this simple structure was initially proposed by Jain et al. (1996). It is inspired by the processing of information employed by a single neural cell, called a *neuron*. A neuron receives input electrical signals via its dendrites, and they pass the signal information to the cell body. In a similar vein, the perceptron accepts N training instances $x = [x_i]_{i=1}^{N}$ as the input signal, calculates a weighted sum of them, using a weight matrix $\boldsymbol{w} \in \mathbb{R}^N$, and then pass it through an activation function to finally produce the neuron's output. As it is shown in Fig. 1.1, the output y is calculated by passing the linear computation of the sum $\sum_{i=1}^{N} w_i x_i$ plus a bias vector b through a non-linear activation function σ:

$$y(x) = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right) = \sigma\left(\boldsymbol{w}^T x + b\right) \tag{1.2}$$

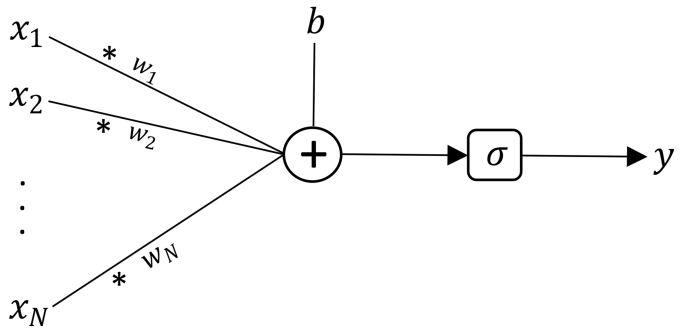


Figure 1.1: Illustration of a single-layer perceptron.

The perceptron is used for two-class classification problems, where a linear equation is used to separate the two classes. In this case, the output is defined as:

$$y(x) = \begin{cases} 1, & \text{if } \left(\boldsymbol{w}^T x + b\right) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1.3}$$

However, the single-layer perceptron has several limitations. First, its output can take on only of two values (0 or 1) due to the hard-limit activation function. Moreover,

perceptron can only be used in classification problems with linear separable learning sets (Minsky & Papert (1969)). Networks with more than one perceptron though can be used to solve more complex problems.

A multilayer perceptron (MLP) is a combination of multiple perceptrons, and consists of three types of layers: the input, output, and one or more hidden layer(s), as shown in Fig. 1.2. Neurons in a MLP can use any arbitrary activation function, whereas in perceptron must have an activation function that imposes a threshold, like ReLU or sigmoid (see Section 1.3 for more information on activation functions). More specifically, the dot products of the inputs with the corresponding weights are presented to the first hidden layer. Then, passing them through an activation function we obtain the output values of the first hidden layer. In a similar way, the output of the first hidden layer plays the role of input for the next hidden layer. We repeat this step until the output layer is reached. At the output layer, the calculations will be either used for a backpropagation algorithm (in the case of training) that corresponds to the activation function that was selected for the MLP, or a decision will be made (in the case of testing) based on the output values.

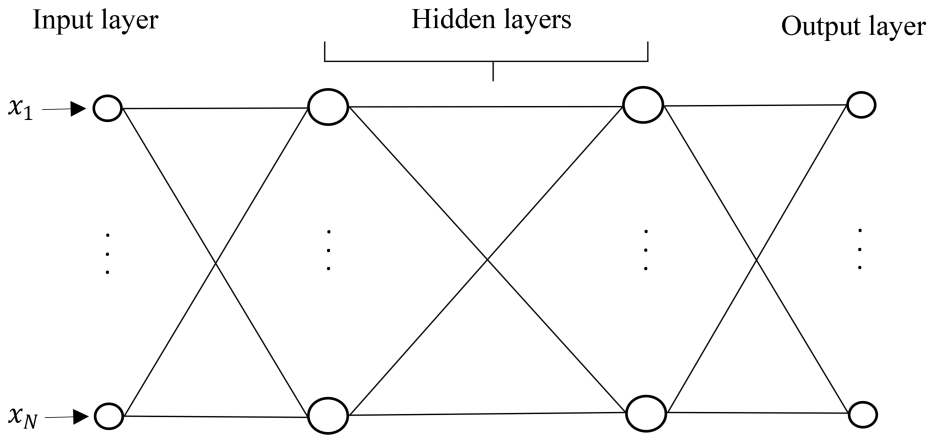


Figure 1.2: Illustration of a multilayer perceptron.

The ANN we have just described is a fully connected feed forward neural network. There are many types of neural networks, depending on their structure, data flow, hidden layers' depth, etc, that may include much more sophisticated functionalities. For example, convolutional neural networks (O'Shea & Nash (2015)) are usually applied to analyze visual

imagery, and can understand spatial relation between pixels of images better compared to a traditional MLP. Another type of neural network is the recurrent neural network (Schmidt (2019)), and can be used for problems related to sequence of words in a sentence for natural language processing (Khurana et al. (2022)) or sequence of sounds in speech recognition (Li (2022a)).

1.2 Backpropagation and Learning

At the training procedure of a DL task, we aim to find the optimal values of network weights that fit well the training instances. To update the values of the parameters in neural networks, we use an efficient algorithm, called *backpropagation* (Rumelhart et al. (1986)), for computing gradients with respect to the parameters using chain rule. Backpropagation is the learning mechanism that allows the MLP to iteratively adjust the weights in the network, with the goal of minimizing the loss function. In order for backpropagation though to be feasible, both the activation function and the weighted sum that combines inputs and weights in a neuron, must be differentiable. The derivative of these functions must be bounded, since *Gradient descent* (Ruder (2017)) is usually the optimization algorithm for finding the minimum of a function.

In each iteration of the gradient descent, we first compute the weighted sums of every neuron of the network. After feed-forwarding the sums through all layers, we compute the gradient of the loss function across all input and output pairs with respect to the network parameters. Then, we backpropagate the gradient values and update the weights of each layer using these values. We repeat this process until convergence, meaning that the current computed gradient for each input-output pair has not changed more than a specific threshold value, compared to the corresponding value of the previous iteration.

Let L be the (average) loss of Eq. (1.1). Thus, $L = L(f_{\boldsymbol{w}_t}(x), y_{real})$. The weight updates at t-th iteration of gradient descent follow the following equation:

$$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t - \gamma \frac{\partial L}{\partial \boldsymbol{w}_t} \tag{1.4}$$

where γ is the learning rate hyperparameter. A learning rate that is too small leads to slow convergence, whereas a large value of it can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge. So, choosing a proper learning rate is a challenging process and one should select it appropriately to the specific needs of the experimental settings. To solve this problem, the learning rate schedule is introduced (Robinds & Monro (1951)). Learning rate schedules try to adjust the learning rate between iterations as the training progresses. Two common techniques for learning rate schedule are: (i) linear annealing of the learning rate's value from a starting to an end point, and (ii) cosine annealing (Loshchilov & Hutter (2017)) that has the effect of starting with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again.

To formulate the mathematical expression for the backpropagation algorithm, let us consider - for simplicity - the loss function to be the mean squared error of each of the neurons in a layer. Let us also denote as K the number of layers, and N_K the number of neurons in layer K. For the weighted sum of j-th neuron $z_j^{(K)}$, there is now contributions from all neurons in the previous layer $(K-1)$. $a_j^{(K)}$ is the output from the activation function of j-th neuron in layer K. So, for layer K the loss function is defined as:

$$\begin{aligned} L &= \frac{1}{N_K}\sum_{j=1}^{N_K}(a_j^{(K)} - y_{real,j})^2 \\ z_j^{(K)} &= \sum_{i=1}^{N_{K-1}} w_{j,i}^{(K-1)} a_i^{(K-1)} + b_j^{(K-1)} \\ a_j^{(K)} &= \sigma(z_j^{(K)}) \end{aligned} \tag{1.5}$$

where $w_{j,i}^{(K-1)}$ is the weight between node i in layer $(K-1)$ and node j in layer K.

The equations for the gradient of loss function with respect to the activation and weights of the $(K-1)$ layer are:

$$\begin{aligned} \frac{\partial L}{\partial {w_{j,k}}^{(K-1)}} &= \frac{\partial z_j^{(K)}}{\partial w_{j,k}^{(K-1)}} \frac{\partial a_j^{(K)}}{\partial z_j^{(K)}} \frac{\partial L}{\partial a_j^{(K)}} \\ \frac{\partial L}{\partial {a_k}^{(K-1)}} &= \sum_{j=1}^{N_K} \left(\frac{\partial z_j^{(K)}}{\partial a_k^{(K-1)}} \frac{\partial a_j^{(K)}}{\partial z_j^{(K)}} \frac{\partial L}{\partial a_j^{(K)}} \right) \end{aligned} \tag{1.6}$$

In a similar vein, the equations for the gradient of loss function with respect to the weights of the $(K-2)$ layer are:

$$\begin{aligned}\frac{\partial L}{\partial w_{j,k}{}^{(K-2)}} &= \frac{\partial z_j^{(K-1)}}{\partial w_{j,k}^{(K-2)}} \frac{\partial a_j^{(K-1)}}{\partial z_j^{(K-1)}} \frac{\partial L}{\partial a_j^{(K-1)}} \\ \frac{\partial L}{\partial a_k{}^{(K-1)}} &= \sum_{j=1}^{N_K} \left(\frac{\partial z_j^{(K)}}{\partial a_k^{(K-1)}} \frac{\partial a_j^{(K)}}{\partial z_j^{(K)}} \frac{\partial L}{\partial a_j^{(K)}} \right)\end{aligned} \tag{1.7}$$

where $w_{j,k}^{(K-2)}$ is the weight between node k in layer $(K-2)$ and node j in layer $(K-1)$.

For the rest layers, we follow the exact same procedure to compute the gradient of loss L with respect to any weight in the network and propagate the gradient calculations back to the weight of interest.

The standard gradient descent algorithm have to run through all the samples in the training set to do a single update for a parameter in a particular iteration. Thus, if the training dataset is large, the gradient descent may take too long to run through the complete training set. However, by using the *Stochastic Gradient Descent (SGD)* (Robbins (1951)), we can use only one random training sample in order to compute the gradient of the loss function. In that way, we compute the gradient and weights updates in faster iterations, as well as the algorithm move towards an optimum more quickly. SGD follows the equation below:

$$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t - \gamma \frac{\partial L(\boldsymbol{w}_t | x_i)}{\partial \boldsymbol{w}_t}, \text{ where } x_i \sim D_{in}. \tag{1.8}$$

In case we want to perform a gradient descent update over a random subset of the training data, the optimization algorithm then is called *Mini-Batch Stochastic Gradient Descent*. It yields:

$$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t - \gamma \frac{\partial L(\boldsymbol{w}_t | x_{i:i+l})}{\partial \boldsymbol{w}_t}, \text{ where } x_{i:i+l} \subset D_{in}. \tag{1.9}$$

This way: (i) we reduce the variance of the parameter updates, which can lead to more stable convergence; and (ii) we can use the highly optimized matrix optimizations

common state-of-the-art DL libraries that make computing the gradient with respect to a mini-batch very efficient. We usually use a mini-batch size between 50 and 256, while it can be different depending on the experimental application.

1.2.1 SGD variants

Beyond the aforementioned gradient descent algorithms, there are few more advanced optimization algorithms used by the DL community. **Momentum** (Qian (1999)) is a method that helps accelerate SGD by orienting the gradient descent in the relevant direction and reducing oscillations. It does this by adding a fraction β of the update vector of the past time step to the current update vector:

$$\begin{aligned} \boldsymbol{w}_t &\leftarrow \boldsymbol{w}_t - \gamma \boldsymbol{v_t}, \\ \boldsymbol{v_t} &\leftarrow \beta \boldsymbol{v_t} + (1-\beta)\frac{\partial L}{\partial \boldsymbol{w}_t} \end{aligned} \tag{1.10}$$

Note that the momentum term β is usually set to 0.9 or a similar value. Generally, the momentum term increases the weight updates for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Adagrad (Ruder (2017)) is a variant of SGD that scales the learning rate for each parameter according to the history of gradients, so it is reduced for very large gradients and vice-versa. Also, other variants of SGD such as **RMSprop** (Mukkamala & Hein (2017)) and **Adam** (Kingma & Ba (2014)) are very frequently used in neural network training.

1.2.2 Loss functions

As we mentioned in Section 1, a loss function is a measure of the ability of a model to predict the expected outcome. In supervised learning, there are two main types of loss function: *regression* and *classification* loss functions. The regression loss functions are used for regression problems that involve predicting a real-valued quantity. Two common used

regression loss function are the *Mean Squared Error (MSE)* (Fürnkranz et al. (2010)) and the *Mean Absolute Error (MAE)* (Qi et al. (2020)). Classification loss functions are used for classification problems, where the output of the network is a vector of probabilities of the input belonging to different classes, and select the category with the highest probability. Such functions are the *Binary Cross-Entropy* and the *Categorical Cross-Entropy* (Zhang & Sabuncu (2018)).

MSE finds the average of the squared differences between predicted and target (real) outputs. So, the loss function of Eq. (1.1) becomes:

$$MSE(\boldsymbol{w}|x) = \frac{1}{M}\sum_{i=1}^{M}(y_{pred}^{(i)} - y_{real}^{(i)})^2 \tag{1.11}$$

MAE is used as an alternative to MSE in some cases, since MSE is sensitive to outliers which can dramatically effect the loss since the distance is squared. It finds the average of the absolute differences between the predicted and target outputs:

$$MAE(\boldsymbol{w}|x) = \frac{1}{M}\sum_{i=1}^{M}|y_{pred}^{(i)} - y_{real}^{(i)}| \tag{1.12}$$

Both MSE and MAE may be equal to 0; this happens when the average distance approaches 0, and the function's derivative at 0 is undefined. To tackle this challenge, **Huber** loss (Huber (1964)) was developed, and is defined as:

$$Huber(\boldsymbol{w}|x) = \begin{cases} \frac{1}{M}\sum_{i=1}^{M}(y_{pred}^{(i)} - y_{real}^{(i)})^2, & \text{if } |y_{pred}^{(i)} - y_{real}^{(i)}| \leq \delta \\ \frac{1}{M}\sum_{i=1}^{M}\delta\left(|y_{pred}^{(i)} - y_{real}^{(i)}| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases} \tag{1.13}$$

Binary Cross-Entropy loss function is used in binary classification models, where the model has to classify an input instance into one of two classes. Thus, to determine the loss between the predicted and the target values, it needs to compare the real values (0 or 1) with the probability the input lies into a specific category:

$$\text{Binary\_CE}(\boldsymbol{w}|x) = \frac{1}{M}\sum_{i=1}^{M} -\left(y_{real}^{(i)} \cdot \log p^{(i)} + (1 - y_{real}^{(i)}) \cdot \log\left(1 - p^{(i)}\right)\right) \tag{1.14}$$

where $p^{(i)}$ is the probability for the i-th input data point.

Categorical Cross-Entropy loss is used in cases where the number of classes is greater than two. Similar with Binary Cross-Entropy loss, the categorical variant is defined as:

$$\text{Categ\_CE}(\boldsymbol{w}|x) = -\frac{1}{M}\sum_{i=1}^{M}\sum_{c=1}^{C}\left(1_{y_{pred}^{(i)}\in C_c}\log p_{model}[y_{pred}^{(i)} \in C_c]\right) \tag{1.15}$$

where C is the total number of classes, the term $1_{y_{pred}^{(i)}\in C_c}$ is the indicator function of the i-th observation belonging to the c-th category, and $p_{model}[y_{pred}^{(i)} \in C_c]$ is the probability predicted by the model for the i-th observation to belong to the c-th category.

1.2.3 Overfitting

Overfitting is a common issue when training a MLe model and it occurs when the model gives accurate predictions for training data but not for new data. In that case, the model cannot generalize and fits too closely to the training dataset instead. Overfitting can happen due to various reasons, such as: (i) the size of the training dataset is not big enough to contain data samples in order to accurately represent all possible input data values, (ii) the training dataset contains too much noisy data that entail irrelevant information, (iii) model complexity is high, so it learns the noise within the training data.

Several researchers have proposed different techniques in order to address overfitting in DL; l_2 regularization, dropout, increased dataset size, artificial data augmentation and variational Bayesian approaches are characteristic such examples. Srivastava et al. (2014) introduced the **dropout** regularization that modifies the network structure. Specifically, it randomly drops out neurons and their connections from the neural network during training iteration, except for the output layer, or we can assign a probability p to all neurons in the network so that those with the highest probability are temporarily ignored from calculations; probability p is called *dropout rate* and is usually equal to 0.5. As it is shown in Fig. 1.3, this results in creating a smaller network with each pass on the training dataset.

Structured dropout is an extension of the traditional dropout technique by applying dropout not to individual neurons but to entire neural network layers or units. Instead of

randomly dropping out individual neurons, structured dropout involves dropping out entire units, channels, or other structured subsets of the network's parameters. This technique can be beneficial in scenarios where applying dropout at the individual neuron level may not be as effective, such as when dealing with convolutional layers (Ghiasi et al. (2018)), recurrent layers (Sarma et al. (2021)), or other specialized architectures.

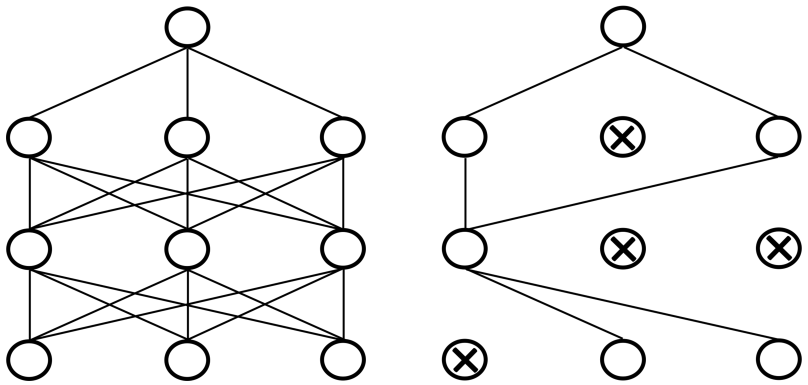


Figure 1.3: (a) Standard neural net, (b) After applying dropout.

When we drop different sets of neurons in the dropout technique, it's equivalent to training different neural networks, as in **Ensemble Methods** (Opitz & Maclin (1999)). Thus, dropout acts as an average method for the effects of large number of different networks. Ensembling combines predictions from separate MLe algorithms, that are called weak learners due to their poor predictive results, to get more accurate results. The two main ensemble methods are boosting and bagging. The former is an iterative technique that modifies the weight of an observation based on the last classification. If an observation was incorrectly classified, boosting increases the observation's weight. Bagging is a process for decreasing the variance in prediction by producing additional training data using combinations of the original data. In boosting, learns learn sequentially and adaptively to improve model predictions, whereas bagging trains the learns in parallel.

Another way of preventing overfitting is using an **Early stopping criteria** (Géron (2017)). This monitors the performance of the model for each training iteration on the validation set, and terminate the training as soon as the validation error reaches a minimum. As we can see in Fig. 1.4, as the training error is being reduced during training phase, so

does the model's error on the validation set. However, after a while, the validation error stops decreasing and starts to go back up. This indicates that the model has started to overfit the training data.

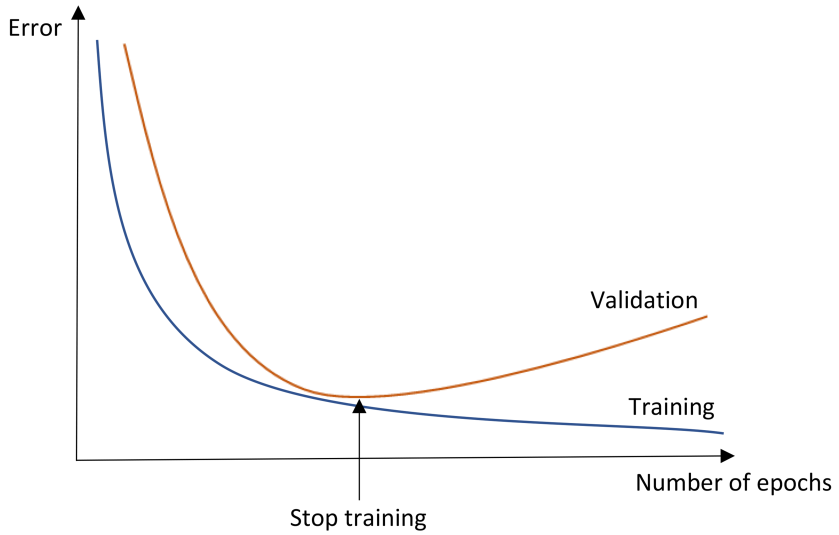


Figure 1.4: Early stopping procedure during a model's training

Another effective technique is **weight regularization** (Cortes et al. (2012)). Using as an example a linear regression problem, we denote that $y_{pred} = \boldsymbol{w}^T x = \sum_{i=1}^{N} w_i x_i$. l_1 and l_2 regularizations suggest adding an auxiliary extension on the loss function that penalises high values of network's weights. Thus, they yield:

$$\begin{aligned} l_1: \quad & \text{Loss}(\boldsymbol{w}|x) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}^T x - y_{real}^{(i)})^2 + \lambda \sum_{i=1}^{N} |w_i| \\ l_2: \quad & \text{Loss}(\boldsymbol{w}|x) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}^T x - y_{real}^{(i)})^2 + \lambda \sum_{i=1}^{N} w_i^2 \end{aligned} \tag{1.16}$$

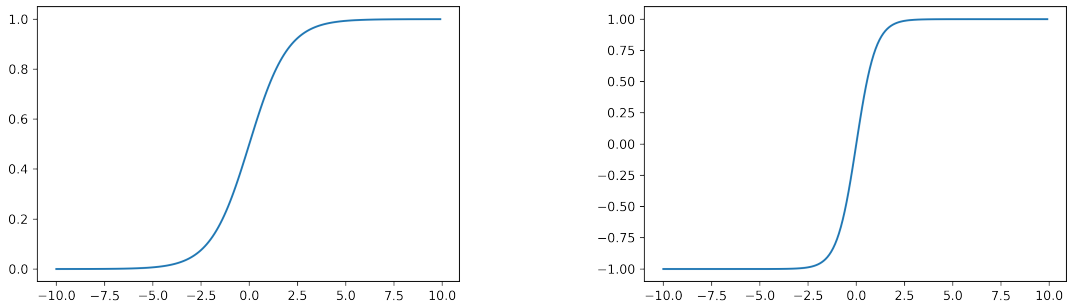
The key difference between these two techniques is that l_1 shrinks the less important feature's coefficient to zero and it is more suitable for feature selection when we have large number of features. Also, l_1 regularization is more robust to outliers than l_2.

1.3 Activation Functions

Utilizing depth in neural networks requires the usage of non-linear activation functions. If an ANN uses only linear multiplication between the layers, no matter how big or complex

its architecture is, the network would be equivalent to a single layer linear network. The goal of an activation function is to develop complex representations and functions based on the inputs, so that models can adapt to a variety of data and differentiate between the outputs. There is a large number of activation functions that have been proposed throughout the years. Choosing the type of activation function depends on the task the network is trained for and its architecture's parameters (number of layers, depth, etc). Two common used activation functions are the **sigmoid** and **tanh** (see Fig. 1.5):

$$\begin{aligned} \sigma(x) &= \frac{1}{1+e^{-x}} \quad \in \{0,1\} \\ \tanh(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \in \{-1,1\} \end{aligned} \tag{1.17}$$



(a) (b)

Figure 1.5: (a) Sigmoid, (b) Tanh.

Since $\sigma(x) \in \{0,1\}$, sigmoid function is commonly used for models where we have to predict the probability as an output. The derivatives of sigmoid and tanh are:

$$\begin{aligned} \frac{d\sigma(x)}{dx} &= \sigma(x)(1-\sigma(x)) = \frac{e^{x}}{(e^{x}+1)^{2}} \\ \frac{d\tanh(x)}{dx} &= 1-\left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right)^{2} = 1-\tanh^{2}(x) \end{aligned} \tag{1.18}$$

In both sigmoid and tanh, as $||x|| >> 0$, the gradient values approaches zero. So, the network ceases to learn and suffers from the *vanishing gradient* problem (Pascanu et al. (2013)). To fix this problem, the **Rectified Linear Unit (ReLU)** (Xu et al. (2015)) was proposed. If x is the input to an activation layer, then ReLU and its derivative are defined

as:

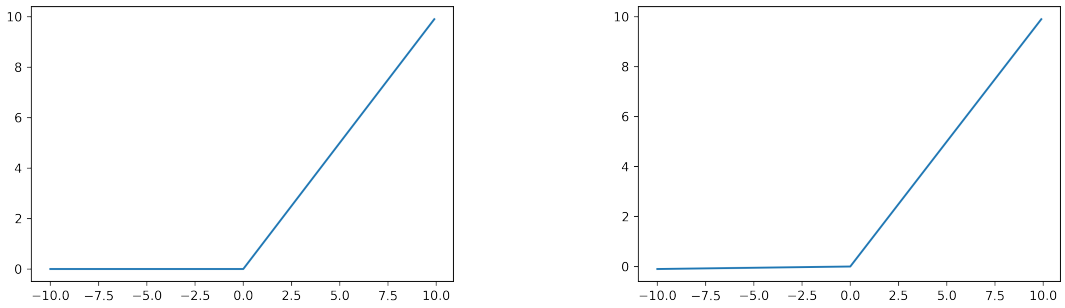
$$ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & \text{otherwise} \end{cases}, \qquad \frac{dReLU(x)}{dx} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1.19}$$

Compared to sigmoid and tanh functions, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient (Krizhevsky et al. (2012)). However, the derivative is zero for $x < 0$. During backpropagation, the weights of some neurons are not updated. This can create dead neurons that will never get activated.

Leaky ReLU (Xu et al. (2015)) is an improved version of ReLU function to solve this issue. Its type and derivative are defined as:

$$Leaky\_ReLU(x) = \begin{cases} 0.1x, & x < 0 \\ x, & \text{otherwise} \end{cases}, \qquad \frac{dReLU(x)}{dx} = \begin{cases} 1, & x \geq 0 \\ 0.01, & x < 0 \end{cases} \tag{1.20}$$

Both ReLU and Leaky ReLU can be seen in Fig. 1.6, for input instances of $x \in \{-10, 10\}$.



(a) (b)

Figure 1.6: (a) ReLU, (b) Leaky ReLU.

1.4 Convolutional Neural Networks

A *Convolutional Neural Network (CNN)* (O'Shea & Nash (2015), Krizhevsky et al. (2012)) is a special kind of ANN that significantly reduces the number of parameters in a deep

neural network with many neurons without compromising model quality. These networks preserve the spatial structure and can achieve state-of-the-art results on difficult computer vision (Patel & Patel (2020), Khan et al. (2018)) and natural language processing (Otter et al. (2019), Conneau et al. (2017)) tasks.

Let us assume we are dealing with a dataset of gray scale images with input size of 28 x 28 pixels each. If we use a traditional feedforward ANN, we will need 784 input weights. However, the flattening of the input image matrix to a long vector of values would lose all of the spatial structure in the image. A CNN would use fewer parameters and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data.

A CNN has typically three layers: a convolutional layer, a pooling layer, and a fully connected layer. The convolutional layer computes the dot product between a weight matrix, called *kernel*, and another matrix which is the restricted portion of the receptive field. The final output from this dot product is known as *feature map*. For the aforementioned example of an input matrix of size 28 x 28 and assuming a kernel size of 5 x 5, the convolutional layer will use a filter to map all the local 5 x 5 submatrices to their corresponding output neurons, producing an 24 x 24 output layer. For input images with 3 or more channels, a filter is applied; filter is one dimension higher than a kernel, and can be seen as stacked multiple kernels where each kernel corresponds to a particular channel.

The pooling layer (Gholamalinezhad & Khosravi (2020)) is placed between two convolutional layers. They are used to down-sample an input representation and reduce the layers size by keeping the important features while discarding others. This layer reduces complexity and improves the efficiency of the CNN. The final fully connected layer is used for the image classification process, based on the features extracted in the previous layers.

Using a very deep architecture in an ANN increases representation power; however, it increases training and computational complexities. *Residual Networks* (He et al. (2016b)) were introduced to make training of very large neural networks feasible. Specifically, they skip the training of few layers by utilizing a set of skip connections that act as shortcuts between layers. As we can see in Fig. 1.7, we take the case of an identical residual block

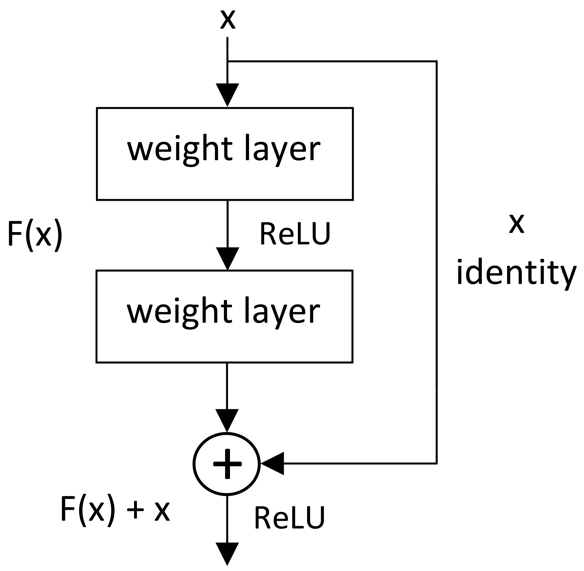


Figure 1.7: A residual block

where the input and output dimensions are the same. Instead of trying to learn a mapping function $H(x)$ from one layer to the next, we are trying to learn the difference (residual) between input and output: $F(x) = H(x) - x \Rightarrow H(x) = F(x) + x$. Using residual blocks inside a CNN, it allows efficient execution of multiple training iterations for deeper networks.

1.5 Local Competition in Neural Networks

There is an increasing body of evidence supporting that neurons in biological systems with similar functional properties are aggregated together in modules or columns where local competition takes place (Kandel et al. (1991), Andersen et al. (1969), Eccles et al. (1967), Stefanis (1969), Douglas & Martin (2004), Lansner (2009)). Few recent researches have found that networks that use non-sigmoidal activation functions such as ReLU, maxout (Goodfellow et al. (2013)) and local winner-takes-all (LWTA) (Srivastava et al. (2013)), have reduced training time and better predictive performance than sigmoidal networks. They have been widely used in complex DL tasks such as image classification (Krizhevsky et al. (2012)) and speech processing (Zeiler et al. (2013)). These activation functions are piecewise linear and not continuously differentiable. There have been found

numerous benefits from using such activation functions in a network architecture, such as catastrophic forgetting prevention (Carpenter & Grossberg (1988), McCloskey & Cohen (1989), Srivastava et al. (2013), Goodfellow et al. (2014)), improved flow of gradients (Goodfellow et al. (2013)), automatic gain control and noise suppression (Srivastava et al. (2013), Carpenter & Grossberg (1988), McCloskey & Cohen (1989)).

By using activation functions such as ReLU, maxout and LWTA, we replace non-linear units with local competition mechanisms among simpler linear units. Using maxout and LWTA, we utilize local competition between units that belong to the same group within a layer, and only part of the network is activated for any given input instance; in ReLU, the neuron's activation output equals to 0 whenever its weighted input sum is negative. As we can see in Fig. 1.8, maxout is a piecewise linear function that returns the maximum of the inputs. In a LWTA block, the units in a layer are considered to be divided into blocks of a fixed size. The output of only one unit is activated, and the rest are zeroed out (more information will be presented in Section 1.5.2). Each of the three aforementioned activation functions employs a local gating mechanism that allows non-zero activations, and errors during training, to propagate only through part of the network. The goal is to form such subnetworks with shared parameters that are trained in a way that each input example is gated to a specific subnetwork, and during inference, only the selected subnetwork is activated for a given example.

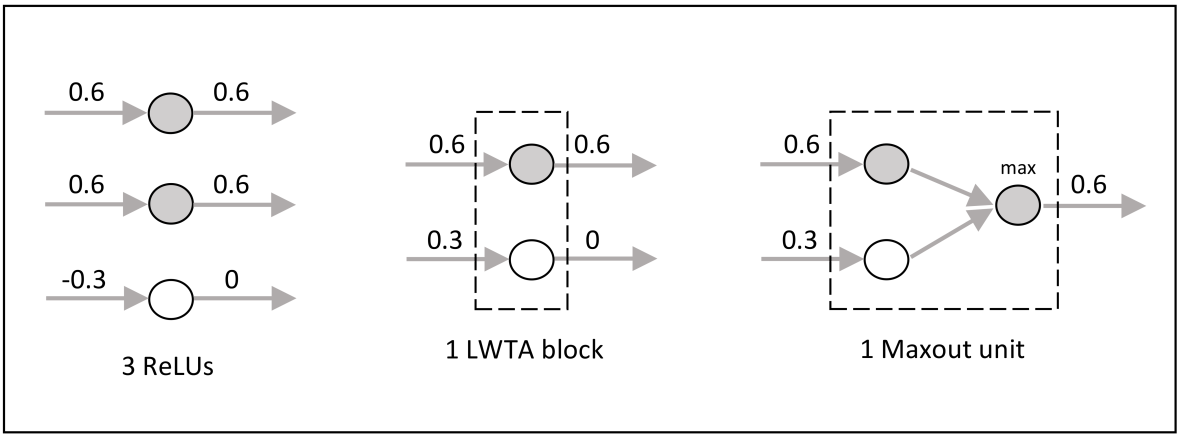


Figure 1.8: Comparison between activation functions: Non-zero activations and errors flow only through the active (grey) units.

1.5.1 LWTA layers in DL

LWTA layers are not new in the field of DL; see, e.g., Srivastava et al. (2013). Although not much work has been pursued along these lines, the recent works of Panousis et al. (2019), Panousis et al. (2021) and Voskou et al. (2021) have spurred some fresh interest in the field. These works have presented alternative implementations of the basic concepts of LWTA units in the context of diverse deep network architectures. Specifically, Panousis et al. (2019) propose a stochastic LWTA formulation which is founded upon the Indian Buffet process (IBP) prior, borrowed from nonparametric statistics; they use this architecture to effect data-driven network compression. Panousis et al. (2021) exploit the same technique to train adversarially-robust deep networks. On the other hand, Voskou et al. (2021) consider a different incarnation of stochastic LWTA architectures, which relies on sampling the winner from a Categorical posterior, driven from the layer input. This layer architecture is used to replace dense ReLU layers in Transformer networks (Vaswani et al. (2017)); it is then shown to yield important benefits in a Sign-Language Translation benchmark (Camgoz et al. (2018), Orbay & Akarun (2020)).

1.5.2 LWTA formulation

This Section describes the architecture of a network with local winner-takes-all (LWTA) activations. LWTA is a computational scheme used for replacing the classical form of neural networks with competing units for activation in each layer. Let denote that a LWTA-based network is consisted of layers, where each one is divided into R blocks. Each block r, $r = 1(1)R,$ includes J competing neurons (units) and it produces an output vector $\boldsymbol{y}_r \in \text{one\_hot}(J)$, given some input $\boldsymbol{x} \in \mathbb{R}^I$; the output is decided via local competition between its units. Each unit corresponds to an activation function $h_r^j, j = 1(1)J$, so the J components of the output $\boldsymbol{y}_r$ are defined as:

$$y_r^j = g(h_r^1, h_r^2, ..., h_r^J) \tag{1.21}$$

where $g(\cdot)$ is the competition function. The activation h_r^j of the j-th neuron in block r

is computed by:

$$h_r^j = \boldsymbol{w}_{r,j}^T \boldsymbol{x} \tag{1.22}$$

where $\boldsymbol{w}_{r,j}$ is the weight vector of unit j in block r, $\boldsymbol{x}$ is the input vector from the previous layer, and $\boldsymbol{W} \in \mathbb{R}^{I \times R \times J}$. In a conventional *hard* LWTA network, the final output reads:

$$y_r^j = \begin{cases} 1, & \text{if } h_r^j \geq h_r^o, \quad \forall o = 1(1)J, \ o \neq j \\ 0, & \text{otherwise} \end{cases} \tag{1.23}$$

To bypass the restrictive assumption of binary output, Srivastava et al. (2013) have proposed the following output expression:

$$y_r^j = \begin{cases} h_r^j, & \text{if } h_r^j \geq h_r^o, \quad \forall o = 1(1)J, \ o \neq j \\ 0, & \text{otherwise} \end{cases} \tag{1.24}$$

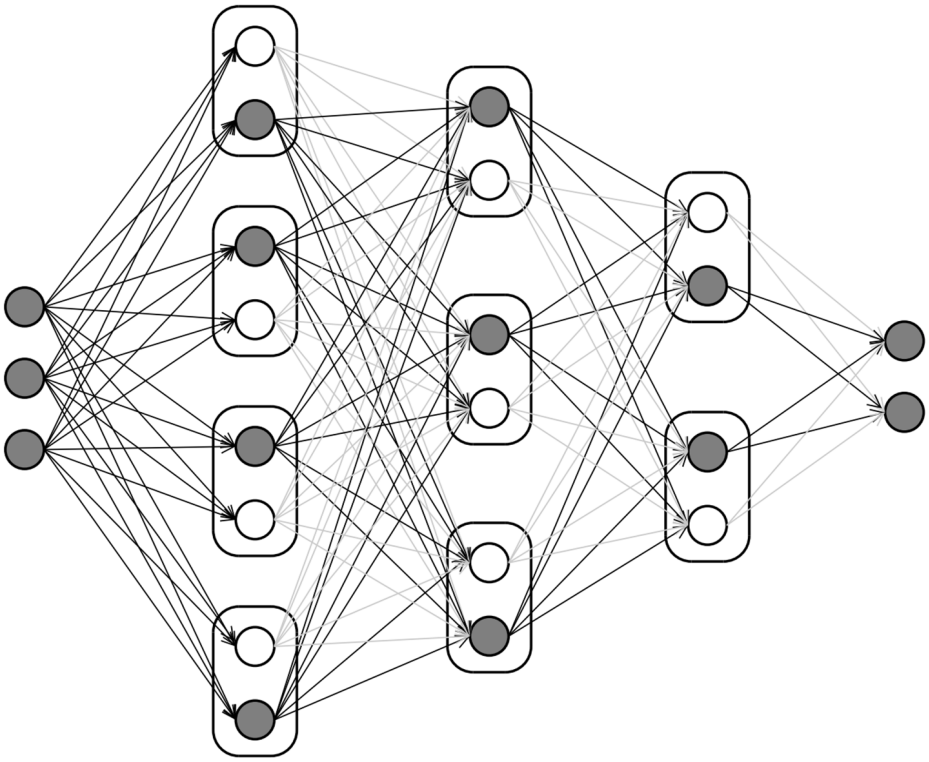


Figure 1.9: A LWTA-based network with blocks of size 2; rectangles denote the LWTA blocks and grey units denote winner units (Srivastava et al. (2013)).

For demonstration purposes, we show in Fig. 1.9 a graphical illustration of a LWTA-

based network with $J = 2$ competing units per block, and total number of 3 hidden layers with 4, 3, 2 blocks per layer.

During inference time, in each block the neuron with the *highest* activation will *turn off* the activation of the rest units and back-propagation will take place only through the winning part of the network. However, the above schemes do not respect a major aspect that is predominant in biological systems, namely *stochasticity*. It has been recently shown that stochastic competition mechanisms among locally competing units can offer important generalization capacity benefits for deep networks used in as diverse challenges as adversarial robustness (Panousis et al. (2021)), video-to-text translation (Voskou et al. (2021)), and model-agnostic Meta-Learning (Kalais & Chatzis (2022)). These aforementioned works employ a Bayesian framework to their neural networks and, thus, are able to obtain much more informative uncertainty estimates in their model predictions. In the next Chapter, we will present the fundamentals of this class of models, known as Bayesian neural networks.

Chapter 2

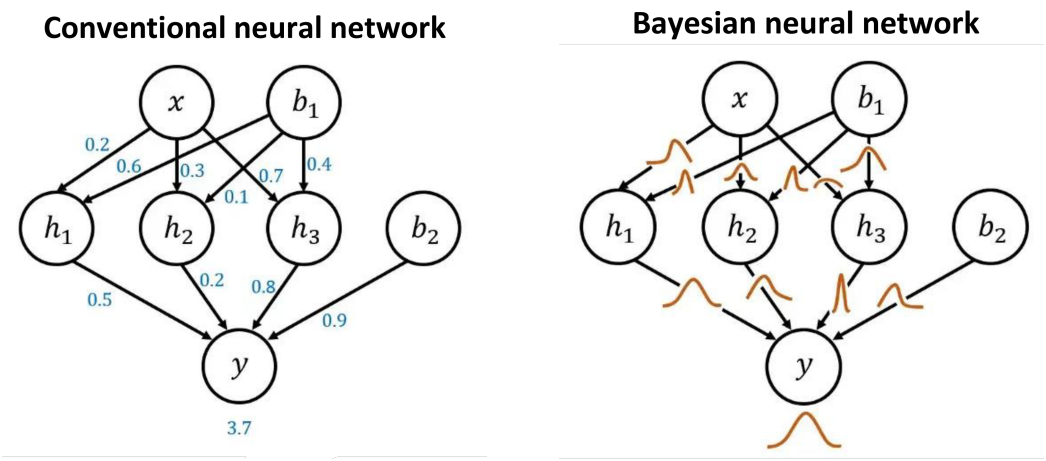
Bayesian Deep Learning

Traditional deep neural networks have shown to have unreliable estimated uncertainties. To mitigate this issue, researchers have developed methods and tools to apply Bayesian modeling to deep neural networks. This results in a class of models known as Bayesian neural networks (BNN's), whose uncertainty estimates are more reliable and informative. Bayesian deep learning that incorporates BNN's deals also with the vulnerability of traditional neural networks to overfit, which adversely affects their generalization capabilities (Szegedy et al. (2014)). Overfitting occurs when the model tunes optimally its parameters for the training set, that performance on unseen new examples suffers. These characteristics can provide important insights into developing models that are less prone to overfitting, memory efficient, perform better at predictive stage and tell us how uncertain our predictions are.

2.1 Bayesian Neural Networks

In a traditional neural network, weights $\boldsymbol{w}$ are assigned as a single value or point estimate, whereas in BNN weights are considered a probability distribution. If instead of learning the model's weights, we could learn a distribution over them, we would be able to estimate uncertainty over the weights so as to explain the trustworthiness of the prediction. Fig. 2.1 shows a schematic diagram of standard neural network, and a BNN where weights are normally distributed.

Let us consider a model with parameters (weights), $\boldsymbol{\theta} = \boldsymbol{w}$, and a parametric form $f_{\boldsymbol{\theta}}(x)$. Standard (frequentist) statistical modelling treats these parameters as fixed (non-random) quantities to be estimated by the observed data D. On the other hand, Bayesian Inference



(a) (b)

Figure 2.1: (a) Point estimate neural network, (b) BNN with a probability distribution over the weights

(BI) (Laurent Valentin Jospin & Bennamoun. (2020)) approaches $\boldsymbol{\theta}$ as random variables. More specifically, we assumed that we have some initial guess about the distribution of $\boldsymbol{\theta}$, which is called *prior distribution*, $P(\boldsymbol{\theta})$. After observing some data, we update the distribution of $\boldsymbol{\theta}$, and using *Bayes' Rule* (Webb (2017)) obtain the *posterior distribution*, $P(\boldsymbol{\theta}|D)$, as (Heckerman (2020)):

$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)} \tag{2.1}$$

where D is the known data, and $P(D|\boldsymbol{\theta})$ is the conditional probability that introduces the data into the equation. The term $P(D)$ expresses the marginal likelihood and operates as a normalizing constant. It is defined using the integral:

$$P(D) = \int_{\boldsymbol{\theta}'} p(D|\boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}' \tag{2.2}$$

where p is the probability density function of P.

However, it is typically expensive to compute the posterior distribution $P(\boldsymbol{\theta}|D)$ using Eq. (2.1) due to the pesky integral over all possible values of $\boldsymbol{\theta}$. For most neural networks, we cannot evaluate this integral analytically, so we have to employ a sort of sampling and approximating methods.

2.1.1 Monte Carlo integration

Sampling methods rely on *Monte Carlo (MC) integration* (Robert (2010)): the use of a finite set of random samples to approximate an expected value; it is expressed as:

$$P(D) = \int_{\boldsymbol{\theta}'} p(D|\boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}' = \mathbb{E}_{P(\boldsymbol{\theta})}[P(D|\boldsymbol{\theta})] \approx \frac{1}{N}\sum_{i=1}^{N} P(D|\boldsymbol{\theta}_i) \tag{2.3}$$

where the samples of network weights $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_N\}$ are drawn independently from the prior $P(\boldsymbol{\theta})$. For a more convenient usage of Eq. (2.3) and to avoid excessive computations, we can obtain a single MC sample.

2.1.2 Variational Inference

Sampling-based inference is good enough for many applications, especially where the sample space is reasonable. As network and dataset size grow, however, so grows the cost of estimating the posterior via sampling. *Variational Bayes* (VB) (Kingma & Welling (2014)) belongs to the bigger class of *Variational Inference* (VI) methods (Blei & McAuliffe (2017)); it consists an optimization-based technique for approximate BI, and provides a computationally efficient alternative to sampling methods. VB can deal with an intractable distribution like the posterior $P(\boldsymbol{\theta}|D)$. We first define a parametrized and tractable distribution, here called the *approximate posterior*, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$, and then tune the parameters $\boldsymbol{\phi}$ so that it better approximates the intractable distribution. More precisely, VB uses a distribution family Q, with a known analytical expression, and tries to find the closer to P member $q \in Q$. The task is now transformed into an optimization problem aiming for the optimal $\boldsymbol{\phi}$.

That is performed by minimizing a quantity, named *Kullback–Leibler (KL) divergence* (James (2011)), between $P(\boldsymbol{\theta}|D)$ and $Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$, to convey dissimilarity between two distributions. KL divergence is a non-symmetric measure of the difference between the two

probability distributions P and Q, and is defined as:

$$
\begin{aligned}
D_{KL}(P(\boldsymbol{\theta}|D)||Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})) &= \sum_i P(\boldsymbol{\theta}_i|D) \log \frac{P(\boldsymbol{\theta}_i|D)}{Q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i)} \\
D_{KL}(P(\boldsymbol{\theta}|D)||Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})) &= \int_{-\infty}^{\infty} p(\boldsymbol{\theta}|D) \log \frac{p(\boldsymbol{\theta}|D)}{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})} d\boldsymbol{\theta}
\end{aligned}
\tag{2.4}
$$

for discrete and continuous probability distributions respectively, where p and q denote the probability density functions of P and Q. KL divergence is always non-negative, $D_{KL} \in \mathbb{R}^+$, a result known as Gibbs' inequality (Bremaud (2012)), with $D_{KL}(P||Q) = 0$ if and only if $P = Q$.

In the case of continuous P and Q distributions, the calculation of their KL divergence integral can be computationally expensive, especially in the cases of unbounded multi-dimensional domains. Fortunately, the divergence between some specific distribution families is analytically solvable and can be easily reduced to a conventional (close) function of the corresponding distributions parameters, avoiding expensive integration. Such example is the KL between two Gaussian distributions $\{N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1}), N(x|\boldsymbol{\mu_2}, \boldsymbol{\sigma_2})\}$:

$$
\begin{aligned}
&D_{KL}(N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1})||N(x|\boldsymbol{\mu_2}, \boldsymbol{\sigma_2})) = \int_{-\infty}^{\infty} N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1}) \log \frac{N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1})}{N(x|\boldsymbol{\mu_2}, \boldsymbol{\sigma_2})} dx \\
&= \int_{-\infty}^{\infty} \left(N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1}) \log N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1}) - N(x|\boldsymbol{\mu_1}, \boldsymbol{\sigma_1}) \log N(x|\boldsymbol{\mu_2}, \boldsymbol{\sigma_2}) \right) dx \\
&= \frac{1}{2} \log(2\pi \boldsymbol{\sigma_2^2}) + \frac{\boldsymbol{\sigma_1^2} + (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^2}{2\boldsymbol{\sigma_2^2}} - \frac{1}{2}(1 + \log(2\pi \boldsymbol{\sigma_1^2})) \\
&= \log \frac{\boldsymbol{\sigma_2}}{\boldsymbol{\sigma_1}} + \frac{\boldsymbol{\sigma_1^2} + (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^2}{2\boldsymbol{\sigma_2^2}} - \frac{1}{2}
\end{aligned}
\tag{2.5}
$$

Using MC sampling of Eq. (2.3), the continuous KL divergence's integrals of Eq. (2.4) becomes:

$$
\begin{aligned}
&D_{KL}(P(\boldsymbol{\theta}|D)||Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})) \approx \frac{1}{N} \sum_{i=1}^{N} \log \frac{P(\boldsymbol{\theta}_i|D)}{Q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i)}, \ \boldsymbol{\theta}_i \sim P(\boldsymbol{\theta}|D). \\
&\approx \log \frac{P(\boldsymbol{\theta}_k|D)}{Q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_k)} = \log P(\boldsymbol{\theta}_k|D) - \log Q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_k), \ k \in \{1, \ldots, N\}, \text{ and } \boldsymbol{\theta}_k \sim P(\boldsymbol{\theta}|D).
\end{aligned}
\tag{2.6}
$$

2.1.3 Evidence Lower Bound

In the general case that P and Q do not belong to a distribution family that renders feasible the computation of the intractable KL divergence integral, we should still try to find a distribution Q that approximates the initial distribution P and minimizes the KL divergence between Q and P, which is formulated as:

$$
\begin{aligned}
&D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) = \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|D)} d\boldsymbol{\theta} = \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log \frac{p(D) q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}{p(D|\boldsymbol{\theta}) p(\boldsymbol{\theta})} d\boldsymbol{\theta} \\
&= \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})} d\boldsymbol{\theta} - \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log \frac{p(D|\boldsymbol{\theta})}{p(D)} d\boldsymbol{\theta} \\
&= D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta})) - \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log p(D|\boldsymbol{\theta}) d\boldsymbol{\theta} + \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log p(D) d\boldsymbol{\theta} \\
&\Rightarrow D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) = D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta})) - \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log p(D|\boldsymbol{\theta}) d\boldsymbol{\theta} + \log P(D) \\
&\Rightarrow \log P(D) - D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) = \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log p(D|\boldsymbol{\theta}) d\boldsymbol{\theta} - D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta})) \\
&\Rightarrow \log P(D) = D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) + L(\boldsymbol{\phi}) \\
&\Rightarrow L(\boldsymbol{\phi}) = \log P(D) - D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) \\
&\Rightarrow L(\boldsymbol{\phi}) \leq \log P(D) \text{ (lower bound), since } D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta}|D)) \geq 0
\end{aligned}
\tag{2.7}
$$

$L(\boldsymbol{\phi})$ is defined as the Evidence Lower Bound (ELBO) (Kingma & Welling (2014)) with respect to parameters $\boldsymbol{\phi}$. Since $\log P(D)$ is not dependent from parameters $\boldsymbol{\phi}$, instead of minimizing KL divergence between Q and P, we can maximize ELBO that is expressed as:

$$
L(\boldsymbol{\phi}) = \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \log p(D|\boldsymbol{\theta}) d\boldsymbol{\theta} - D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||P(\boldsymbol{\theta})) \tag{2.8}
$$

It seems that ELBO is a trade-off between the reconstruction accuracy against the complexity of the variational posterior. The KL divergence in Eq. (2.8) can be interpreted as a measure of the additional information required to express the posterior relative to the prior. The integral in Eq. (2.8) is the expected log-likelihood, and maximizing this term is equivalent to searching for a variational distribution that explains the data well.

2.1.4 Reparameterization and Gumbel-Softmax tricks

The main workflow of a model's training consists of the backpropagation algorithm, which uses dynamic programming to compute weight gradients of the network. These gradients are then used to minimize the optimization objective function via gradient descent. In a forward pass of a BNN, each layer's weights $\boldsymbol{w}$ are obtained through samples from the distribution $q_{\boldsymbol{\phi}}(\boldsymbol{w})$. However, we cannot backpropagate through samples. The problem of backpropagating through stochastic nodes can be circumvented if we can re-express the sample $\boldsymbol{w} \sim q_{\boldsymbol{\phi}}(\boldsymbol{w})$, such that gradients can flow to pass and update the parameters, without encountering stochastic nodes. *Reparameterization trick* (Maddison et al. (2017)) allows a gradient path through a non-stochastic node. We relegate the random sampling to a noise term which effectively separates it from the gradient flow. Technically, we employ an auxiliary random variable ϵ and a deterministic function $t(\boldsymbol{\phi}, \epsilon)$ where:

$$\boldsymbol{w} = t(\boldsymbol{\phi}, \epsilon),\ \epsilon \sim p(\epsilon),\ \text{is equivalent to}\ \boldsymbol{w} \sim q_{\boldsymbol{\phi}}(\boldsymbol{w}). \tag{2.9}$$

For example, assuming that the variational posterior of the weights is a Gaussian distribution, $N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma})$, we can draw a sample $\epsilon \sim N(0, 1)$ and express the weights $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \epsilon \tag{2.10}$$

In several stochastic deep learning setups though, we need to draw samples from trainable Categorical distributions. Performing backpropagation through reparameterized drawn samples becomes infeasible. Recently, the solution of introducing appropriate continuous relaxations has been proposed by different research teams, where they use the *Gumbel-Softmax relaxation trick* (Jang et al. (2017), Maddison et al. (2017)). Let $\boldsymbol{\eta}$ be the unnormalized probabilities of a considered Discrete distribution, and $\tau \in (0, \infty)$ be a hyperparameter referred to as the temperature of the relaxation that controls how closely the Categorical distribution is approximated by this continuous relaxation. Then, the

drawn samples $\boldsymbol{\xi}$ are expressed as differentiable functions of the form:

$$\boldsymbol{\xi} = \text{Softmax}((\log \boldsymbol{\eta} + g)/\tau), \text{ where } g = -\log(-\log U),\ U \sim \text{Uniform}(0, 1). \tag{2.11}$$

As $\tau \to 0$, the Softmax becomes an argmax and the Gumbel-Softmax distribution becomes the Categorical distribution. During training, we let $\tau > 0$ to allow gradients past the sample, and then gradually anneal the temperature τ (but not completely to 0, as the gradients would blow up).

2.1.5 Training

Blundell et al. (2015) introduced *Bayes by Backprop*, an efficient algorithm for learning a probability distribution on the weights of a neural network during backpropagation, and computing the gradients resulting from differentiating the ELBO with respect to the network weights. It is now a common method used for training a BNN (Goan & Fookes (2020), Neal (1996)). Let's assume that the variational posterior of the weights is a Gaussian, $Q_{\boldsymbol{\phi}}(\boldsymbol{w}) = N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma})$, and prior distribution is a Gaussian, $P(\boldsymbol{w}) = N(\mathbf{0}, \boldsymbol{I})$; $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ are the means and standard deviations of the Gaussian weight posteriors, respectively.

To train a BNN, we seek for minimizing an objective function. In our case it is the negative resulting ELBO, and this optimization process is equivalent with maximizing the ELBO of the model. The loss function is expressed as:

$$\begin{aligned} Loss(\boldsymbol{w}, \boldsymbol{\phi}) &= -\int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{w}) \log p(D|\boldsymbol{w}) d\boldsymbol{w} + D_{KL}(Q_{\boldsymbol{\phi}}(\boldsymbol{w})||P(\boldsymbol{w})) \\ &= -\int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\boldsymbol{w}) \log p(D|\boldsymbol{w}) d\boldsymbol{w} + D_{KL}(N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma})||N(\mathbf{0}, \boldsymbol{I})) \end{aligned} \tag{2.12}$$

Since the integral in Eq. (2.12) is intractable, we need to resort to MC sampling to approximate its value. Also, the KL divergence of the same equation will follow Eq. (2.5).

So, the loss function becomes:

$$Loss(\boldsymbol{w}, \boldsymbol{\phi}) \approx -\frac{1}{N}\sum_{i=1}^{N} \log p(D|\boldsymbol{w}_i) + \left(-\log \boldsymbol{\sigma} + \frac{\boldsymbol{\sigma}^2 + \boldsymbol{\mu}^2}{2} - \frac{1}{2}\right), \text{ where } \boldsymbol{w}_i \sim N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma}). \tag{2.13}$$

The final step of one cycle of the training process is to compute the gradients of loss function with respect to the parameters, and perform backpropagation to update their values. However, due to the stochasticity that arises from the obtained MC samples of Eq. (2.13), we need to employ again the reparameterization trick. Using Eq. (2.9), loss function becomes:

$$Loss(\boldsymbol{w}, \boldsymbol{\phi}) \approx -\frac{1}{N}\sum_{i=1}^{N} \log p(D|t(\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}, \epsilon_i)) + \left(-\log \boldsymbol{\sigma} + \frac{\boldsymbol{\sigma}^2 + \boldsymbol{\mu}^2}{2} - \frac{1}{2}\right), \tag{2.14}$$

where $\epsilon_i \sim N(\mathbf{0}, \boldsymbol{I})$.

Now, it is possible to effectively calculate the gradients with respect to the variational parameters $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ as follows:

$$\begin{aligned} \frac{\partial Loss(\boldsymbol{w}, \boldsymbol{\phi})}{\partial \boldsymbol{\mu}} &= -\frac{1}{N}\sum_{i=1}^{N} \frac{\partial \log p(D|t(\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}, \epsilon_i))}{\partial \boldsymbol{\mu}} + \boldsymbol{\mu} \\ \frac{\partial Loss(\boldsymbol{w}, \boldsymbol{\phi})}{\partial \boldsymbol{\sigma}} &= -\frac{1}{N}\sum_{i=1}^{N} \frac{\partial \log p(D|t(\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}, \epsilon_i))}{\partial \boldsymbol{\sigma}} + \left(-\frac{1}{\boldsymbol{\sigma}} + \boldsymbol{\sigma}\right). \end{aligned} \tag{2.15}$$

It is worth to notice that the posterior expectation of the data log-likelihood $\log p(D|t(\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}, \epsilon_i))$ is essentially the negative categorical cross-entropy between the data labels and the class probabilities generated by the last Softmax layer of a BNN's architecture.

2.1.6 Inference

During inference, we draw a set of B samples of the Gaussian connection weights from the trained posteriors $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$. This results in a set of B output logits of the network, which we average to obtain the final predictive outcome (*Bayesian Model Averaging* (Fragoso

(2015))):

$$f_{\boldsymbol{\phi}}(x) \approx \frac{1}{B}\sum_{b=1}^{B} f_{\boldsymbol{\phi}}(x; \boldsymbol{w}_b), \quad \text{where } \boldsymbol{w}_b \sim N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\sigma}). \tag{2.16}$$

A high value of B usually entails higher predictive accuracy but slower inference, whereas a smaller value B can cause increased output variance while requiring less time. The optimal B trade-off typically depends on the model training settings.

2.2 Types of Uncertainty Modelling in Neural Networks

2.2.1 Epistemic and Aleatoric Uncertainty

In this Section, we describe the two major types of uncertainty in deep learning: *epistemic uncertainty*, due to distributional mismatch between test and training data distributions, and *aleatoric (data) uncertainty*.

Epistemic uncertainty (Huang et al. (2021)) describes the errors associated to the lack of experience at specific regions of the feature space. So, it is inversely proportional to the density of training examples, and thus can be reduced by collecting data in those low density regions (Gal. (2016)). Although non-Bayesian models tend to perform poorly and give overconfident predictions in regions that lack data (Combalia et al. (2020)), Bayesian approaches to neural networks allow us to capture the epistemic uncertainty (Hüllermeier & Waegeman (2020)).

Aleatoric (data) uncertainty (Valdenegro-Toro & Saromo (2022)) is the type of uncertainty produced by the natural complexity of the data, such as class overlap, noisy or imprecise data, low-quality images or measurement errors (Kendall & Gal. (2017)). However, it cannot be reduced by collecting more data under the same experimental conditions. A trustworthy representation of this type of uncertainty is desirable especially

in safety-critical application domains such as medical systems and socio-technical systems (self-driving guide for Deaf and Hard of Hearing people). For example, in the binary classification problem where we have a large amount of data, the data that lie within the intersection region of the two class distributions will have higher aleatoric uncertainty than the data in either distribution but outside the intersection.

Fig. 2.2 represents a toy example of a linear process ($y = x$) that was sampled around $x = -2.5$ and $x = 2.5$. A sensor malfunction produces noisy measurements in the left blue cloud of observations, that lead to high aleatoric uncertainty. This uncertainty cannot be reduced by additional measurements, because the sensor keeps producing errors around $x = -2.5$ by design. On the other hand, high epistemic uncertainty is caused on the left, middle and right part of the blue clouds of points, where there are few or no observations for training. Given though more data in that space, this type of uncertainty would decrease.

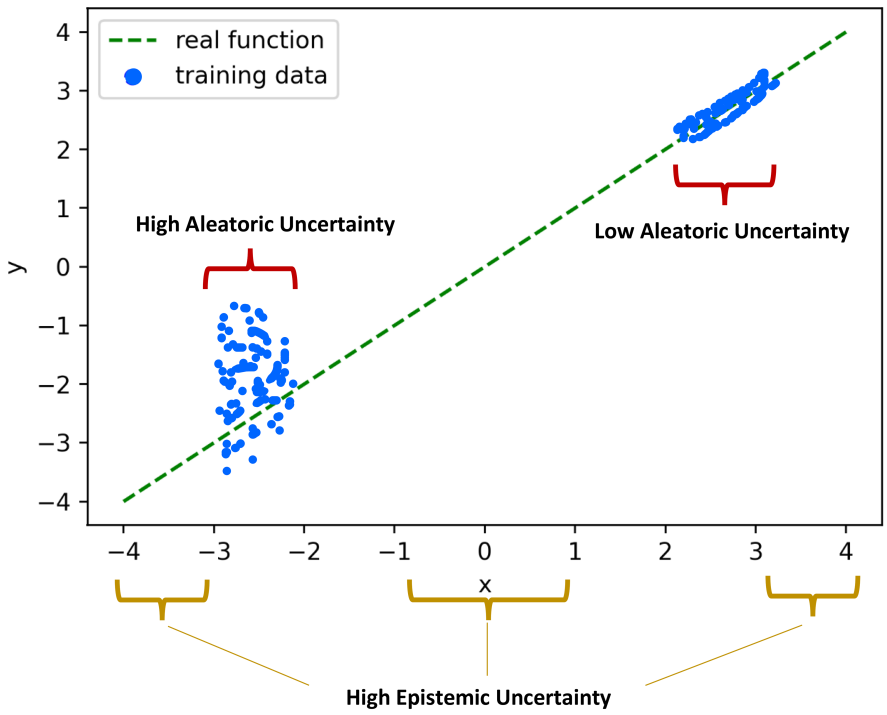


Figure 2.2: An exhibit of the different kinds of uncertainty in a linear regression context.

2.2.2 Methods for Uncertainty Estimation

Using **Bayesian Neural Networks**, we can successfully model both epistemic and aleatoric uncertainty. When aleatoric uncertainty is not dependent on the input data, it is

referred to as *homoscedastic uncertainty*; otherwise, it is called *heteroscedastic*. To model heteroscedastic uncertainty we have to change our loss functions. So, we can replace the loss function with the following:

$$\text{Loss} = \frac{||y - \hat{y}||_2}{2\sigma^2} + \frac{1}{2}\log\sigma^2 \tag{2.17}$$

where the model predicts a mean $\hat{y}$ and variance σ^2. As seen in the equation above, if the model produces a wrong prediction, it will be encouraged to attenuate the residual term, by increasing uncertainty σ^2. As for the term $\log\sigma^2$, it can be thought as learned loss attenuation that prevents the uncertainty term from growing infinitely large. In the homoscedastic uncertainty though, the uncertainty parameter will be a free parameter that we optimise, rather than a model output.

However, the modelling of epistemic uncertainty is a bit more challenging. It requires modelling distributions and their parameters using **Monte Carlo (MC) dropout** sampling (Gal. & Ghahramani (2016)). This technique places a Bernoulli distribution over the network's weights. Specifically, we can train a model with dropout, while at the testing phase we stochastically sample from the network with different random dropout masks. The statistics of this output distribution will reflect the model's epistemic uncertainty.

Concrete dropout (Gal et al. (2017)) extended MC dropout by regularizing the outputs of the units, instead of the weights. In fact, it applies a stochastic binary mask to randomly turn them off during training. This means that each unit has a probability of being dropped or kept, independently of the other units. These units learn to compensate for the missing ones, due to the stochastic nature of the dropout mask; that way, they learn more robust and diverse features.

Another way to model epistemic uncertainty in neural networks is **Deep Ensembles**. This technique, proposed by Lakshminarayanan et al. (2017), consists of training multiple deep neural networks with different architectures and initialization parameters, and then combining their predictions to estimate the uncertainty of the model. By training various models, each with different weights' values, we can cover a large scope of possible predictions and their corresponding uncertainties.

Chapter 3

Meta-Learning

When we train MLe models on problems with limited amounts of training data, we cannot usually get good predictive performance (Sculley et al. (2015), Lai (2019)). This comes in contrast to the human ability to quickly derive information from a range of different tasks, and then adapt to a new task with limited available new examples (Castro et al. (2008), Kühl et al. (2020)). In essence, this capability of the mind of learning how to learn (Black et al. (2006)) has inspired researchers to investigate the concept of Meta-Learning (ML) (Lake et al. (2017), Buysse et al. (2019)).

ML is the science of systematically observing how different MLe approaches perform on a wide range of learning tasks. Then, preserving knowledge from this experience, ML aims to learn new tasks much faster than otherwise possible. First, we need to collect *meta-data* that describe prior learning tasks and previously learned models. They comprise the algorithm configurations used to train the models, including hyperparameter settings, network architecture, evaluation accuracy, training time, and trained network weights. Then, we need to learn from this prior meta-data, in order to extract and transfer knowledge that guides the search for optimal models for new tasks.

In order to induce a ML algorithm for classification predictive modeling tasks, first we train the base classifiers (learners), and then the meta-classifier (meta-learner). The base learner deals with learning parameters for task-specific objective, while the meta-learner emphasizes learning general knowledge across different tasks. In the prediction phase, the base classifiers will output their classification outcomes, and then the meta-classifier will make the final prediction concerning the classification outcome, as a function of the base classifiers. The goal is to find the best score for a performance metric on the test harness. This optimization procedure can be usually performed by a human, too. Thus, we could

think of ourselves as meta-learners on a MLe project.

3.1 ML categories

Recent work in ML has produced state-of-the-art results in a variety of settings (Wang et al. (2016), Ba et al. (2016), Li & Malik (2017), Antoniou et al. (2018), Brock et al. (2018), Antoniou et al. (2019)). ML methods can be organized in three main categories:

(1) **Optimization-based** methods that include the optimization of the base-learner, and aim to learn meta-knowledge via model parameters in order to improve optimization performance using the meta-learner. A recent example of this type of method with state-of-the-art performance is *Model-Agnostic Meta-Learning* (MAML) algorithm, a work by Finn et al. (2017). MAML enables tuning the parameters of a trained network to quickly learn a new task with only a few gradient updates. To get away with the expensive second-order computations, several researchers proposed appropriate first-order approximations for MAML; such works are the *First-Order MAML* of Finn et al. (2017) and the *Reptile* algorithm of Nichol et al. (2018). The gradients used in the ML process, though, can entail large variances due to variance of both the data samples for each task and task samples from the task distribution. To face this challenge, Yang & Kwok (2022) have integrated a variance reduction method called *VR-Reptile* with a faster convergence rate and equal or/and improved predictive performance in few-shot classification benchmarks.

Furthermore, an important improved generalization performance in few-shot learning tasks is shown by *Meta-SGD* (Li et al. (2017)); apart from learning initialization parameters of the base-learner, they implement a scheme of learning a learning rate and an update direction for each parameter that are learned during training. However, this approach doubles the number of trainable parameters and computational perplexity. Antoniou et al. (2019) proposed *MAML++* that uses various modifications to MAML in order to create a model with improved generalization performance, faster convergence speed and improved inference and training computational efficiency. Another optimization-based algorithm is the *LSTM Meta-learner* (Ravi & Larochelle (2017)), achieving state-of-the art performance especially in few-shot learning settings. Specifically, being inspired by the

fact that gradient descent update rule has a very similar form to the cell update in an LSTM, it trains an LSTM based meta-learner, where the cell state represents the model parameters.

(2) **Metric-based** methods that focus on non-parametric metric-learning of the base-learner by comparing training and validation points and predicting the label of matching training points. The model learns an informative feature space that can be used to compute class predictions based on input similarity scores. The first method of metric-based ML is *Siamese networks* (Koch et al. (2015)). Hamilton et al. (2017) and Garcia & Bruna (2017) inspired from this idea, and using *Graph Neural Networks*, create a parametric information flow when comparing training and validation inputs. *Matching Networks* (Vinyals et al. (2016)) use cosine similarity for comparing inputs in the prediction phase and directly train tasks in the few-shot setting. *Prototypical networks* (Snell et al. (2017)) learn a metric space in which classification can be performed by computing distances to prototype representations of each class. Different from matching and prototypical networks that use a fixed similarity metric, *Relation Networks* (Sung et al. (2018)) use network functionalities to learn a domain-specific similarity function. Shyam et al. (2017) proposed a biologically inspired approach, called *Attentive Recurrent Comparators*, that avoid comparing entire inputs, and instead take multiple interleaved glimpses at various parts of the inputs that are being compared. When the learning tasks include a small number of samples, the metric-based methods are relatively fast at inference time, since they do not adjust task-specific elements. However, when there is a large amount of tasks, pair-wise input comparison can be computationally expensive. Finally, most metric-based methods can operate only under a supervised learning environment, where we have available the labels of the dataset instances.

(3) **Model-based** methods that design a model that updates its parameters rapidly with few training steps. This rapid parameter update can be achieved by its internal architecture or controlled by another meta-learner model. Such a method is the *Memory-Augmented Neural Networks* (Santoro et al. (2016)). They inspired the use of external memory from Neural Turing Machine (Graves et al. (2014)) and proposed a set of modifications

on the training setup and memory retrieval mechanisms to be used for classification. *Recurrent meta-learners* (Duan et al. (2017), Wang et al. (2016)) employ a process where the feed sequentially the entire training set into the model and then make predictions, in a reinforcement learning setting, for the evaluation set using the internal state of the model. Also, *Meta-Networks* (Munkhdalai & Yu (2017)) employs an architecture and training process used for rapid generalization across tasks. *SNAIL* (Mishra et al. (2018)) uses attention mechanisms with special temporal layers in order to improve memory capacity and pinpoint memories. Finally, Edwards & Storkey (2016) proposed *Neural Statistician* that learns the features of the datasets by using distances between the meta-features to make predictions. Compared to optimization-based approaches, model-based have been observed as a less efficient approach for generalization to out-of-distribution tasks (Finn & Levine (2018)). Also, they may not perform well when presented with large datasets (Hospedales et al. (2020)).

3.2 MAML

Finn et al. (2017) suggested a *model-agnostic* algorithm for ML, that can be applied to any model trained via gradient descent. The introduced MAML algorithm initializes model parameters in a way that can be quickly adapted to several types of new tasks.

Let us consider a model with parameters $\boldsymbol{\theta}$ and a parametric form $f_{\boldsymbol{\theta}}$. When the model is adapting to an unseen task T_i, MAML runs few steps of gradient descent that yields a task-specific parameter set, $\boldsymbol{\theta}'_i = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L_{T_i}(f_{\boldsymbol{\theta}})$; α is the step size hyperparameter and L_{T_i} denotes the loss on the task T_i. Subsequently, training proceeds to optimize the function $f_{\boldsymbol{\theta}'_i}$ with respect to the model parameters $\boldsymbol{\theta}$. Assuming that the batch of tasks has size M, we can define the targeted *meta-objective* as:

$$L_{meta}(\boldsymbol{\theta}) = \sum_{i=1}^{M} L_{T_i}(f_{\boldsymbol{\theta}'_i}) = \sum_{i=1}^{M} L_{T_i}(f_{\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L_{T_i}(f_{\boldsymbol{\theta}})}). \tag{3.1}$$

Optimization of this meta-objective over $\boldsymbol{\theta}$ yields the *outer-loop* update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{M} L_{T_i}(f_{\boldsymbol{\theta}_i'}) \tag{3.2}$$

where β stands for the *outer-loop* learning rate.

3.2.1 First-Order MAML

MAML involves expensive computations due to the second order updates of Eq.'s (3.1) and (3.2). So, Finn et al. (2017) have developed first-order approximations which is simplified as follows:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}'} \sum_{i=1}^{M} L_{T_i}(f_{\boldsymbol{\theta}_i'}) \tag{3.3}$$

In the final meta-step update, FOMAML computes the gradients with respect to the updated parameters' values $\boldsymbol{\theta}'$ and omits the gradients from $\boldsymbol{\theta}'$ to $\boldsymbol{\theta}$.

Nichol et al. (2018) have also attempted to propose a first-order gradient-based ML method named *Reptile*, where they apply some *inner-loop* gradient descent steps similar with MAML, but on the *outer-loop* update they subtract the initial parameters $\boldsymbol{\theta}$ from the updated ones $\boldsymbol{\theta}'$ as we mention below:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta(\boldsymbol{\theta}' - \boldsymbol{\theta}). \tag{3.4}$$

3.3 Bayesian ML

Learning a new task from a few examples can increase the amount of uncertainty in a neural network. Also, few-shot learning algorithms usually tend to overfit, since they neural networks they entail suffer from overparameterization. To tackle these issues, several researchers have considered Bayesian inference-driven methods for DL ML, extending upon older works built for conventional MLe approaches. Grant et al. (2018) examined MAML as a hierarchical model in a Bayesian sense, and presented a method for performing posterior inference. Several variants of this approach have been proposed based on different

Bayesian inference methods: (Finn et al. (2018), Yoon et al. (2018), Gordon et al. (2018), Nguyen et al. (2020)). In this thesis, we propose a different regard toward improving generalization capacity for deep networks in the context of ML (Kalais & Chatzis (2022)), where we use stochastic deep networks with linear competing units in the context of model-agnostic ML. We will analyze this proposed treatment of the ML problem in the following Section.

3.4 Stochastic LWTA Networks for Model-Agnostic Meta-Learning

3.4.1 Proposed Approach

In this Section, we attack the problem of model-agnostic ML. Specifically, we present a novel stochastic LWTA activations mechanism in a deep network architecture. The proposed type of network units results in sparse representations from each model layer, as the units are organized into blocks where only one unit generates a non-zero output. The main operating scheme of the introduced units relies on stochastic principles, as the network performs posterior sampling over competing units to select the winner. Therefore, the proposed networks are explicitly designed to extract input data representations of sparse stochastic nature, as opposed to the currently standard deterministic representation paradigm.

Our novel work proposes a different regard toward improving generalization capacity for deep networks in the context of ML. Specifically, this approach relies on the following main concepts: (i) **The concept of sparse learned representations**. For the first time in the literature of deep network-driven ML, we employ a mechanism that inherently learns to extract sparse data representations. This consists in replacing standard unit nonlinearities (e.g., ReLU) with a unit competition mechanism. Specifically, (linear) units are organized into blocks. Presented with some input, the units within a block engage in a competition process with only one winner. The outputs of all units except for the winner are zeroed

out; the output of the winner retains its computed value (LWTA architecture); and (ii) **The concept of stochastic representations**. We establish a stochastic formulation for the previously described competition process. Specifically, we postulate that, within a block of competing units, winner is selected via sampling from an appropriate Categorical posterior. The corresponding winning probability of each unit is proportional to its linear computation (thus depending on the layer input). Via this competition process, we yield stochastic representations from network layers, that is representations that may change each time we present to the network layer exactly the same input.

Our work is different from any previous approach, mentioned in Section 1.5.1, in various ways: (i) stochasticity does not stem from utilization of the IBP; we rather adopt an approach similar to Voskou et al. (2021); (ii) we do not use the proposed architecture as a replacement for a specific type of layer in a greater network architecture (Transformer) that remains largely unchanged; instead, we build completely new networks using these layers; (iii) we perform a full Bayesian treatment, by treating network weights as random variables; we do not employ this construction as a means of compressing the weights at prediction time, contrary to Panousis et al. (2019), Voskou et al. (2021); instead, we perform weight sampling at prediction time as a means of improving accuracy; and (iv) for the first time, we examine how these principles perform in the context of deep network driven ML. Note that, apart from LWTA architectures, other data-driven sparsity models have also been proposed recently, e.g. Lee et al. (2018), Kessler et al. (2021). However, none of these have been developed or evaluated in the context of an ML setting.

Based on the results from existing approaches, we posit that the proposed treatment of the ML problem, which combines learned representation sparsity and stochasticity, will be extremely beneficial to the deep learning community. We dub our approach Stochastic LWTA for ML (StochLWTA-ML).

We perform a variational Bayes treatment of the proposed model. We opt for a full Bayesian treatment, by also handling network weights as latent variables. That is, we elect to impose an appropriate prior over the network weights and fit approximate (variational) posteriors. As we evaluate our approach on image classification, sinusoidal regression and

active learning problems, we show that StochLWTA-ML offers a variety of advantages over the current state-of-the-art methods, namely: (i) incurring reduced predictive error rate compared to the currently state-of-the-art methods in the field; (ii) obtaining this performance with networks that comprise *an immensely reduced* number of trainable parameters, and therefore give rise to better computational efficiency and imposed memory footprint.

The remainder of this Section is organized as follows: Section 3.4.1.1 introduces our approach and provides the related training and prediction algorithms. In Section 3.4.2, we perform a thorough experimental evaluation of StochLWTA-ML, and compare our findings to the current state-of-the-art. In the final Section 3.4.3, we summarize our contribution and discuss our results.

3.4.1.1 Architecture

Let us denote as $\boldsymbol{x} \in \mathbb{R}^I$ an input vector presented to a dense ReLU layer of a conventional deep neural network, with corresponding weights matrix $\boldsymbol{W} \in \mathbb{R}^{I \times O}$. The output of the layer is the vector $\boldsymbol{y} \in \mathbb{R}^O$ and is fed to the subsequent layer. In a stochastic LWTA layer, a ReLU unit is replaced by J competing linear units, organized in one block. The input $\boldsymbol{x}$ is now presented to each block through weights that are organized into a three-dimensional matrix $\boldsymbol{W} \in \mathbb{R}^{I \times R \times J}$. Then, the j-th competing unit within r-th block computes the sum $\sum_{i=1}^{I}(w_{i,r,j}) \cdot x_i$. Competition means that, of the J units in the block, one unit (the "winner") will present its linear computation to the next layer; the rest will present zero values. Traditionally in the literature, the winner unit is selected to be the unit with greatest linear computation. Recently, stochastic competition principles have been considered, e.g. Panousis et al. (2019), Panousis et al. (2021), Voskou et al. (2021).

Let us denote as $\boldsymbol{y} \in \mathbb{R}^{R \cdot J}$ the output of a stochastic LWTA layer; this is composed of R subvectors $\boldsymbol{y}_r \in \mathbb{R}^J$ and is sparse, since all units except for one, in each block, yield zero values. Let us introduce the discrete latent indicator vector $\boldsymbol{\xi} \in \text{one\_hot}(J)^R$ to denote the winner units in the R blocks that constitute a considered stochastic LWTA layer. This vector comprises R component subvectors, where each component entails one non-zero

value at the index position that corresponds to the winner unit of the respective LWTA block. On this basis, the output $\boldsymbol{y}$ of the stochastic LWTA layer's $(r \cdot j)$-th component $\boldsymbol{y}_{r,j}$ is defined as:

$$\boldsymbol{y}_{r,j} = \boldsymbol{\xi}_{r,j} \sum_{i=1}^{I} (w_{i,r,j}) \cdot x_i \in \mathbb{R} \tag{3.5}$$

where we denote as $\boldsymbol{\xi}_{r,j}$ the j-th component of $\boldsymbol{\xi}_r$, and $\boldsymbol{\xi}_r \in \text{one\_hot}(J)$ holds the r-th subvector of $\boldsymbol{\xi}$.

In Eq. (3.5), we postulate that the latent winner indicator variables are drawn from a Categorical distribution which is proportional to the intermediate linear computation that each unit performs. Therefore, the stronger the linearity the higher the chance of the unit winning the stochastic competition within its block. In detail, we postulate that, a posteriori, the winner distributions yield:

$$q(\boldsymbol{\xi}_r) = \text{Categorical}\left(\boldsymbol{\xi}_r \middle| \text{softmax}(\sum_{i=1}^{I} [w_{i,r,j}]_{j=1}^{J} \cdot x_i)\right) \tag{3.6}$$

where $[w_{i,r,j}]_{j=1}^{J}$ denotes the vector concatenation of the set $\{w_{i,r,j}\}_{j=1}^{J}$. A graphical illustration of this stochastic architecture is provided in Fig. 3.1.

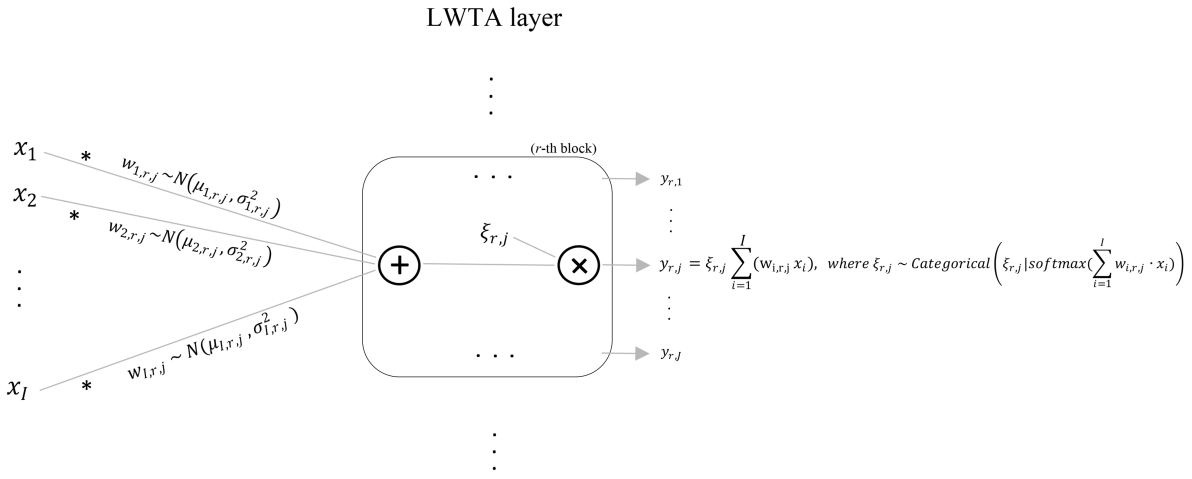


Figure 3.1: A zoomed-in graphical illustration of the r-th block of a stochastic LWTA layer. Input $\boldsymbol{x} = [x_1, x_2, \ldots, x_I]$ is presented to each unit in the block. The j-th component $\boldsymbol{y}_{r,j}$ of the block's output is obtained after computing latent variable $\boldsymbol{\xi}_{r,j}$.

As a network composed of such (StochLWTA) layers entails latent variables $\boldsymbol{\xi}$, we need to perform a Bayesian network treatment to perform effective parameter training.

We opt for a (approximate) stochastic gradient variational Bayes treatment (Kingma & Welling (2014)), for scalability purposes. This means that the used objective function takes the form of an evidence lower-bound (ELBO) objective, as we describe next. In our work, we take one step further: we also elect to infer a posterior density over the network weights $\boldsymbol{W}$, as opposed to obtaining point-estimates. This results in a second source of stochasticity for our approach, which may further increase its generalization capacity under uncertain conditions arising from limited training data availability. Note that the use of these posteriors is totally different from Panousis et al. (2019) and Voskou et al. (2021): therein, posterior variance is used for compressing posterior mean bit-precision; then, predictions are performed using only the compressed posterior mean. Instead, in our work we sample multiple times from the trained weight posterior and perform model averaging (in a Bayesian sense), as a means of increasing generalization capacity.

We postulate:

$$q(\text{vec}(\boldsymbol{W})) = N(\text{vec}(\boldsymbol{W})|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \tag{3.7}$$

where $\boldsymbol{\psi} \triangleq \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ are the means and variances of the Gaussian weight posteriors, respectively.

3.4.1.2 A Model-Agnostic ML Algorithm

Let us define an ML problem where we are given a training dataset D of tasks, T, that are governed by a distribution $P(T)$. We consider an N-way, K-shot learning setting, where each task contains K labelled examples for each of N available classes.

Our approach entails parameter sets $\boldsymbol{\theta}$ which coincide with the hyperparameter sets $\boldsymbol{\psi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ of the weights $\boldsymbol{W} \in \mathbb{R}^{I \times R \times J}$; these sets $\boldsymbol{\psi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ are the target of our MAML-type training algorithm.

Therefore, to perform model training, we have to first initialize the trainable parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ across layers. To this end, one can appropriately exploit popular initialization schemes, such as Glorot Uniform (Glorot & Bengio (2010)). Then, our approach entails an inner-outer loop scheme, in a vein similar to existing approaches, but with some crucial differences:

1. The stochastic nature of the postulated weights, $\boldsymbol{W}$, results in the updates taking place over the posterior means, $\boldsymbol{\mu}$ and variances, $\boldsymbol{\sigma}^2$.

2. The stochastic nature of both the inferred representations and the network weights themselves implies that proper training must rely on optimization of the ELBO function of the network. Let us consider an inner-loop dealing with task $T_i \sim P(T)$, with data $D_i = (X_i, Y_i) \subset D$. Let $\text{CE}(Y_i, f_{\boldsymbol{\psi}}(X_i; \hat{\boldsymbol{\xi}}, \hat{\boldsymbol{W}}))$ be the categorical cross-entropy between the data labels Y_i and the class probabilities $f_{\boldsymbol{\psi}}(X_i; \hat{\boldsymbol{\xi}}, \hat{\boldsymbol{W}})$ generated by the Softmax layer. Then, we yield:

$$L_{T_i}(\boldsymbol{\psi}) = -\text{CE}(Y_i, f_{\boldsymbol{\psi}}(X_i; \hat{\boldsymbol{\xi}}, \hat{\boldsymbol{W}})) - KL[\, q(\boldsymbol{\xi}) \,||\, p(\boldsymbol{\xi}) \,] - KL[\, q(\boldsymbol{W}) \,||\, p(\boldsymbol{W}) \,] \quad (3.8)$$

for task T_i with dataset D_i which is dealt with on the i-th training iteration.

Here, for simplicity and without harming generality, we consider that the weights prior $p(\text{vec}(\boldsymbol{W}))$ is a Gaussian distribution $N(\boldsymbol{0}, \boldsymbol{I})$ and the latent variables prior $p(\boldsymbol{\xi})$ is a symmetric Categorical distribution Categorical$(1/J)$. In addition, in our notation we stress that the output $f_{\boldsymbol{\psi}}(X_i; \hat{\boldsymbol{\xi}}, \hat{\boldsymbol{W}})$ depends on the winner selection process, which is stochastic, and the outcomes of sampling the network weights. Specifically, in our work we perform Monte-Carlo (MC) sampling using one reparameterized sample of the corresponding latent variables.

Let $\bar{\boldsymbol{\xi}}$ be the unnormalized probabilities of the Categorical distribution $q(\boldsymbol{\xi})$ (Eq. (3.6)). The sampled instances of $\boldsymbol{\xi}$, $\hat{\boldsymbol{\xi}}_{r,j}$, are expressed as (Maddison et al. (2017)):

$$\hat{\boldsymbol{\xi}}_{r,j} = \text{Softmax}((\log \bar{\boldsymbol{\xi}}_{r,j} + g_{r,j})/\tau), \ \forall r = 1, \ldots, R, \ j = 1, \ldots, J \quad (3.9)$$

where $g_{r,j} = -\log(-\log U_{r,j})$, $U_{r,j} \sim \text{Uniform}(0, 1)$, and $\tau \in (0, \infty)$ is a temperature factor that controls how closely the Categorical distribution is approximated by this continuous relaxation.

Similarly, the Gaussian weights yield: $\hat{w}_{t,r,j} = \mu_{t,r,j} + \sigma_{t,r,j}\hat{\epsilon}$, and $\hat{\epsilon} \sim N(0, 1)$.

On this basis, the KL divergences in Eq. (3.8) become:

$$KL[\, q(\boldsymbol{\xi}_{r,j} \,||\, p(\boldsymbol{\xi}_{r,j}) \,] = \mathbb{E}_{q(\boldsymbol{\xi}_{r,j})}[\log q(\boldsymbol{\xi}_{r,j}) - \log p(\boldsymbol{\xi}_{r,j})] \approx \log q(\hat{\boldsymbol{\xi}}_{r,j}) - \log p(\hat{\boldsymbol{\xi}}_{r,j}),\ \forall r,\ j \quad (3.10)$$

and

$$\begin{aligned} &KL[\, q(w_{t,r,j}) \,||\, p(w_{t,r,j}) \,] = \mathbb{E}_{q(w_{t,r,j})}[\log q(w_{t,r,j}) - \log p(w_{t,r,j})] \approx \\ &\log q(\hat{w}_{t,r,j}) - \log p(\hat{w}_{t,r,j}),\ \forall t = 1, \ldots, I,\ r,\ j \end{aligned} \quad (3.11)$$

Hence, the ELBO becomes:

$$L_{T_i}(\boldsymbol{\psi}) = -\text{CE}(Y_i, f_{\boldsymbol{\psi}}(X_i; \hat{\boldsymbol{\xi}}, \hat{\boldsymbol{W}})) - \sum_{r,j} (\log q(\hat{\boldsymbol{\xi}}_{r,j}) - \log p(\hat{\boldsymbol{\xi}}_{r,j})) - \sum_{t,r,j} (\log q(\hat{w}_{t,r,j}) - \log p(\hat{w}_{t,r,j})) \quad (3.12)$$

Therefore, we establish a MAML-type algorithm, where: (i) the used networks comprise blocks of stochastic LWTA units visually depicted in Fig. 3.1; (ii) the trainable parameters are the means $\boldsymbol{\mu}$ and variances $\boldsymbol{\sigma}^2$ of the synaptic weights; and (iii) the objective function of the inner-loop process is given in Eq. (3.12).

We summarize our training algorithm in Alg. 1.

Algorithm 1 Model training with StochLWTA-ML

Require: $P(T)$: distribution over tasks
Initialize $\boldsymbol{\psi} := \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$
Define outer-step size β and inner learning rate α
for i = 1,2, ... **do**
  Inner training loop:
  Sample task $T_i \sim P(T)$
  Compute $L_{T_i}(\boldsymbol{\psi})$ using Eq. (3.12)
  Compute adapted parameters with SGD: $\boldsymbol{\psi}_i' = \boldsymbol{\psi} - \alpha \nabla_{\boldsymbol{\psi}} L_{T_i}(f_{\boldsymbol{\psi}})$
  Outer training loop:
  Derive $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} + \beta(\boldsymbol{\psi}_i' - \boldsymbol{\psi})$
end for

3.4.1.3 Prediction Algorithm

We start our prediction algorithm (Alg. 2) by sampling a task $T' \sim P(T)$, with data $D' = (X', Y') \subset D$. Then, we draw a set of B samples of the Gaussian connection weights from the trained posteriors $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, and select the winning units in each block of the

network by similarly sampling from the posteriors $q(\boldsymbol{\xi})$. This results in a set of B output logits of the network, which we average to obtain the final predictive outcome:

$$f_{\boldsymbol{\psi}}(X';\tilde{\boldsymbol{\xi}},\tilde{\boldsymbol{W}}) \approx \frac{1}{B}\sum_{s=1}^{B} f_{\boldsymbol{\psi}}(X';\tilde{\boldsymbol{\xi}}_s,\tilde{\boldsymbol{W}}_s) \tag{3.13}$$

where $\tilde{\boldsymbol{\xi}}_s$ and $\tilde{\boldsymbol{W}}_s$ are sampled directly from the posteriors $q(\boldsymbol{\xi})$ and $q(\boldsymbol{W})$, respectively.

This concludes the formulation of the proposed model-agnostic ML approach.

Algorithm 2 Prediction with StochLWTA-ML

Require: Learned parameters $\boldsymbol{\psi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$
Sample task $T' \sim P(T)$
for $s = 1$ **to** B **do**
  Sample $\tilde{\boldsymbol{W}}_s \sim q(\boldsymbol{W})$ defined in Eq. (3.7)
  Sample $\tilde{\boldsymbol{\xi}}_s \sim q(\boldsymbol{\xi})$ defined in Eq. (3.6), for $\boldsymbol{W} = \tilde{\boldsymbol{W}}$ and $(x_i = x_i') \in X'$
  Compute output logits $f_{\boldsymbol{\psi}}(X';\tilde{\boldsymbol{\xi}}_s,\tilde{\boldsymbol{W}}_s)$, given the sampled values $\tilde{\boldsymbol{\xi}}_s$ and $\tilde{\boldsymbol{W}}_s$
end for
Use Eq. (3.13) to average over the resulting set of B logits and derive the final prediction.

3.4.2 Experiments

We evaluate our proposed model in various few-shot learning tasks: image classification, sinusoidal regression and active learning. After thorough exploration on the number of LWTA layers as well as the number of blocks for each layer and the competing units per block, we end up with using networks comprising 2 layers with 16 blocks and 2 competing units per block on the former layer, and 8 blocks with 2 units per block on the latter. The last network layer is a Softmax. Weight mean initialization, as well as point-estimate initialization for our competitors, is performed via Glorot Uniform. Weight log-variance initialization is performed via Glorot Normal, by sampling from $N(0.0005, 0.01)$. The Gumbel-Softmax relaxation temperature is set to $\tau = 0.67$.

In the inner-loop updates, we use the Stochastic Gradient Descent (SGD) (Robbins (2007)) optimizer with a learning rate of 0.003. For the outer-loop, we use SGD with a linear annealed outer step size to 0, and an initial value of 0.25. Additionally, all the experiments were ran with task batch size of 50 for both training and testing mode.

Prediction is carried out averaging over $B = 4$ output logits. The code was implemented in Tensorflow (Abadi et al. (2016)).

3.4.2.1 Classification

Table 3.1: N-way K-shot (%) classification accuracies on Omniglot, Mini-Imagenet and CIFAR-100

	Omniglot 20-way		**Mini-Imagenet 5-way**		**CIFAR-100 5-way**	
Algorithm	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Matching Nets (Santoro et al. (2016))	93.80	98.50	43.56	55.31	-	-
LSTM Meta-Learner (Ravi & Larochelle (2017))	-	-	43.44	60.60	-	-
MAML	95.80	98.90	48.70	63.11	-	-
FOMAML	-	-	48.07	63.15	-	-
Reptile	88.14	96.65	47.07	62.74	-	-
PredCP (Nalisnick et al. (2021))	-	-	49.30	61.90	-	-
Neural Statistician (Edwards & Storkey (2016))	93.20	98.10	-	-	-	-
mAP-SSVM (Triantafillou et al. (2017))	95.20	98.60	50.32	63.94	-	-
LLAMA (Grant et al. (2018))	-	-	49.40	-	-	-
PLATIPUS (Finn et al. (2018))	-	-	50.13	-	-	-
GEM-BML+ (Zou & Lu (2020))	96.24	98.94	50.03	-	-	-
DKT (Patacchiola et al. (2020))	-	-	49.73	64.00	-	-
ABML (Ravi & Beatson (2019))	-	-	45.00	-	49.50	-
BMAML (with 5 particles) (Yoon et al. (2018))	-	-	53.80	-		-
ABML (local)	90.21 $\pm$ 0.34	93.39 $\pm$ 0.09	44.23 $\pm$ 0.81	52.12 $\pm$ 1.01	49.23 $\pm$ 0.23	53.60 $\pm$ 0.39
BMAML (local)	96.92 $\pm$ 0.58	98.11 $\pm$ 2.03	53.10 $\pm$ 1.05	64.80 $\pm$ 0.93	52.60 $\pm$ 1.40	65.80 $\pm$ 0.04
PLATIPUS (local)	94.35 $\pm$ 0.87	98.30 $\pm$ 0.44	49.97 $\pm$ 0.97	63.13 $\pm$ 1.18	51.14 $\pm$ 0.48	63.61 $\pm$ 2.16
MAML (local)	95.48 $\pm$ 0.81	98.61 $\pm$ 0.49	48.60 $\pm$ 1.23	63.01 $\pm$ 1.28	50.67 $\pm$ 0.93	62.89 $\pm$ 0.77
FOMAML (local)	94.92 $\pm$ 0.71	98.12 $\pm$ 0.94	47.93 $\pm$ 0.67	63.10 $\pm$ 0.58	49.13 $\pm$ 0.61	63.80 $\pm$ 0.83
Reptile (local)	87.98 $\pm$ 1.18	96.36 $\pm$ 1.54	46.97 $\pm$ 0.95	62.53 $\pm$ 0.61	48.19 $\pm$ 0.74	63.45 $\pm$ 1.63
StochLWTA-ML	**97.79** $\pm$ 0.48	**98.97** $\pm$ 0.61	**54.11** $\pm$ 0.82	**66.70** $\pm$ 0.41	**54.60** $\pm$ 0.39	**66.73** $\pm$ 0.06

We first evaluate StochLWTA-ML on popular few-shot image classification datasets, and compare its performance to state-of-the-art prior results. In Table 3.1, we show how StochLWTA-ML performs on Omniglot 20-way (Lake et al. (2017)), Mini-Imagenet 5-way (Vinyals et al. (2016)) and CIFAR-100 5-way (Krizhevsky (2009)) few-shot settings. We compare our findings to state-of-the-art ML algorithms such as LLAMA and PLATIPUS as reported in Gordon et al. (2018), Amortized Bayesian Meta-Learning (ABML), MAML, FOMAML, Reptile and others. Using the original architectures with the same hyperparameters and data preprocessing as in Finn et al. (2017), we have also locally reproduced ABML, BMAML (with 5 particles), PLATIPUS, MAML, FOMAML and Reptile (dubbed "local" in Table 3.1).

For the local replicates, the results in Table 3.1 constitute average performance statistics and corresponding standard deviations (std's) over three runs, using different random seeds. For completeness sake, we also compare our findings to other state-of-the-art ML models

as reported in Finn et al. (2017), including Matching Nets and LSTM Meta-Learner. As we observe, our method outperforms the existing state-of-the-art in both the 1-shot and 5-shot settings. Besides, the reported std statistics on the locally reproduced experiments show consistency across all methods; thus, our stochastic approach is as resilient to seed initialization as the existing approaches.

In the following, we perform a diverse set of ablations that shed light to the characteristics and capabilities of our approach.

3.4.2.1.1 Does stochastic competition contribute to classification accuracy?

To check whether the accuracy improvements stem from the LWTA-induced sparsity or the proposed stochastic competition concept, we evaluate both our approach as well as MAML, FOMAML, ABML, BMAML and PLATIPUS, considering both "deterministic LWTA" and "stochastic LWTA" setups; deterministic LWTA networks have been adopted from Srivastava et al. (2013). As we see in Table 3.2, replacing ReLU with deterministic LWTA yields negligible differences. On the other hand, stochastic LWTA units yield a clear improvement in all cases. This improvement becomes even more important in the case of our approach, where we sample from stochastic weights.

Table 3.2: Ablation study (% classification accuracy)

		Omniglot 20-way		**Mini-Imagenet 5-way**	
Algorithm	**Network type**	1-shot	5-shot	1-shot	5-shot
MAML (local)	deterministic LWTA	95.52	98.15	48.88	63.15
	stochastic LWTA	95.91	98.78	49.61	64.03
FOMAML (local)	deterministic LWTA	95.01	98.18	48.11	63.54
	stochastic LWTA	95.80	98.41	49.24	64.54
ABML (local)	deterministic LWTA	90.30	93.64	44.31	52.27
	stochastic LWTA	91.21	93.91	45.11	53.31
BMAML (local)	deterministic LWTA	96.96	98.21	53.12	64.84
	stochastic LWTA	97.11	98.30	53.50	65.31
PLATIPUS (local)	deterministic LWTA	94.48	98.31	49.99	63.21
	stochastic LWTA	95.13	98.56	51.06	64.18
StochLWTA-ML	deterministic LWTA	96.95	98.63	53.12	64.93
	stochastic LWTA	**97.79**	**98.97**	**54.11**	**66.70**

3.4.2.1.2 Is there a computational time trade-off for the increased accuracy?

It is also important to investigate whether our approach represents a trade-off between accuracy and computational time compared to our competitors. To facilitate this investigation, in Table 3.3 we provide training iteration wall-clock times for our approach and the existing locally reproduced state-of-the-art, as well as the total number of iterations each model needs to achieve the reported performance of Table 3.1. It appears that our methodology takes 77% *less* training time than the less efficient algorithms ABML, BMAML, PLATIPUS, and is comparable to other approaches. This happens because our approach yields the reported state-of-the-art performance by employing a network architecture (that is, number of LWTA layers, as well as number of blocks and block size on each layer) that result in a total number of trainable parameters that is *one order of magnitude less* on average than the best performing baseline methods. This can be seen in the last three columns of Table 3.3 (dubbed D_A, D_B and D_C for Omniglot, Mini-Imagenet and CIFAR-100 respectively). In addition, training for our approach converges fast.

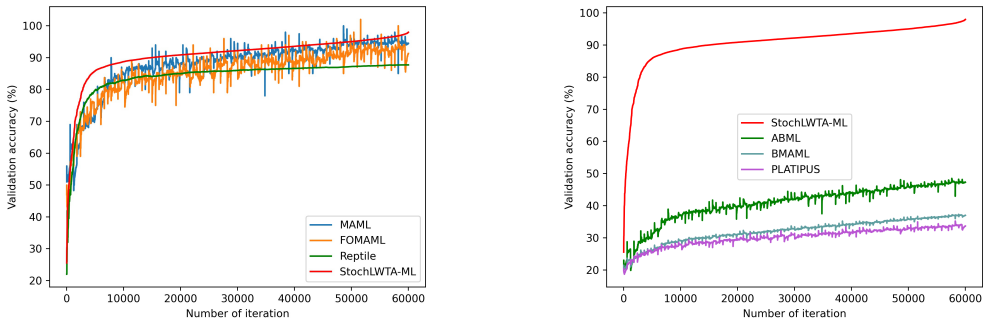
The situation changes when it comes to prediction: our approach imposes a slight computational time overhead compared to MAML, FOMAML and Reptile, but still *much less* than the time-consuming PLATIPUS. BMAML and ABML. This is a rather negligible increase when we are dealing with a low number of drawn samples, B. More information on the effect of sample size in our approach's performance are provided in Section 3.4.2.1.5.

Table 3.3: Performance comparison: average wall-clock time (in msecs), training iterations for each locally reproduced method and number of baselines' trainable parameters over the considered datasets of Table 3.1

Algorithm	Training	Prediction	Training iterations	D_A parameters	D_B parameters	D_C parameters
PLATIPUS (local)	1603.39	602.77	333600	560025	615395	580440
BMAML (local)	1450.31	514.43	301800	560025	615395	580440
ABML (local)	678.48	265.78	138000	224010	246158	232176
MAML (local)	288.25	103.28	60000	112005	123079	116088
FOMAML (local)	284.49	102.34	60000	112005	123079	116088
Reptile (local)	284.30	**102.27**	60000	113221	124613	117463
StochLWTA-ML	**282.90**	113.44	60000	**54549**	**60112**	**56745**

Finally, we provide an example of how training for our approach converges, and how this compares to the alternatives. We illustrate our outcomes on the Omniglot 20-way 1-shot benchmark; similar outcomes have been observed in the rest of the considered datasets. Fig.

3.2(a) compares StochLWTA-ML with prior traditional ML methods: MAML, FOMAML and Reptile. It becomes apparent that our approach converges equally fast to these competitors. Further, Fig. 3.2(b) compares StochLWTA-ML with the probabilistic ML models ABML, BMAML, PLATIPUS. Since these methods are quite time-consuming and less efficient regarding to memory consumption, StochLWTA-ML gives rise to an easier time training MAML based probabilistic model.



(a) (b)

Figure 3.2: ML algorithms' training convergence comparison

3.4.2.1.3 Effect of block size J

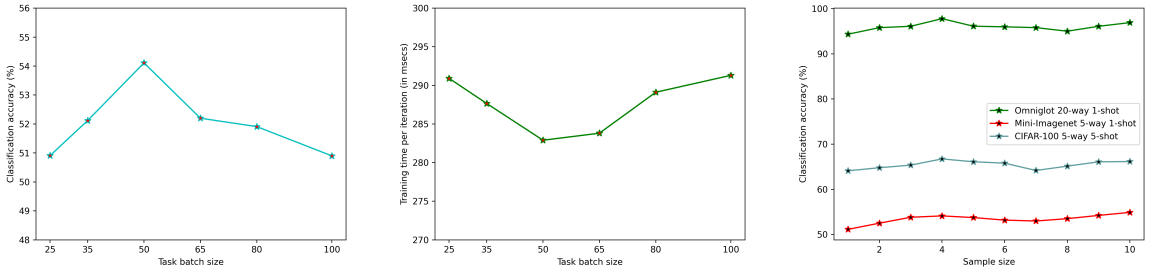
As it is presented in Table 3.4, increasing the number of competing units per block to $J = 4$ or $J = 8$ does not notably improve the results of our approach. On the contrary, it increases the number of trained parameters, thus leading to higher network computational complexity. This corroborates our initial choice of using $J = 2$ competing units per block in our approach.

Table 3.4: Effect of block size J in StochLWTA-ML's classification (%) accuracy

	Omniglot 20-way		**Mini-Imagenet 5-way**		**CIFAR-100 5-way**	
Number of units	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
$J = 2$	97.79	98.97	54.11	66.70	54.60	66.73
$J = 4$	96.33	98.55	53.99	66.65	54.51	66.13
$J = 8$	95.38	98.83	53.70	67.08	54.45	66.18

3.4.2.1.4 How does the task batch size affect StochLWTA-ML's performance?

For demonstration purposes, in Fig. 3.3(a) and 3.3(b) we illustrate the performance of our



(a) (b) (c)

Figure 3.3: (a) The effect of task batch size in StochLWTA-ML's predictive accuracy, (b) the effect of task batch size in StochLWTA-ML's training time per iteration (in msecs), and (c) the effect of sample size B in StochLWTA-ML's classification (%) accuracy

model on the Mini-Imagenet 5-way 1-shot setting with different task batch sizes. As we see, our model performs optimally with task batch size of 50 in terms of both training time and predictive accuracy. We have obtained similar results for all other considered datasets.

3.4.2.1.5 How does the sample size B at prediction time affect StochLWTA-ML's accuracy?

In Fig. 3.3(c), we illustrate how sample size B affects predictive accuracy on the Omniglot 20-way 1-shot, Mini-Imagenet 5-way 1-shot and CIFAR-100 5-way 5-shot settings. As we observe, an increase in sample size, B, does not always yield an accuracy increase. It seems that selecting $B = 4$ allows for the best predictive accuracy/computational complexity trade-off. We have obtained similar findings for the remainder of the considered experimental settings as well. Thus, we finally choose $B = 4$ throughout our experiments.

3.4.2.1.6 How important are the stochastic weights?

We perform an ablation study on Omniglot 20-way; the goal is to discern how much of an extra improvement Gaussian weights offer. Our results are shown in Table 3.5. Apparently, stochastic LWTA units offer the greatest fraction of the accuracy gains, but the Gaussian weights are still indispensable.

Table 3.5: Omniglot 20-way ablation study: Gaussian vs deterministic weights (point estimates).

Network type	**1-shot**	**5-shot**
ReLU (point estimates)	95.63	96.17
ReLU (Gaussians)	95.80	96.48
stochastic LWTA (point estimates)	97.68	98.85
stochastic LWTA (Gaussians)	**97.79**	**98.97**

3.4.2.1.7 How do the state-of-the-art methods perform with a parameter count reduced to be about the same as StochLWTA-ML?

We repeated Omniglot 20-way 1-shot experiments to evaluate all state-of-the-art, using a deep network of the same number of parameters and similar architecture as the proposed StochLWTA-ML model. To this end, we simply replaced the stochastic LWTA layers with dense ReLU layers of the same size, and dropped the Gaussians from the weights. This yielded between 2.2% and 3.5% reduction in classification accuracy, as we show in Table 3.6.

Table 3.6: Performance comparison: wall-clock time (in msecs), training iterations for each locally reproduced method, classification accuracy and number of baselines' trainable parameters over the Omniglot 20-way 1-shot benchmark

Algorithm	Training	Prediction	Training iterations	Accuracy (%)	Parameters
PLATIPUS (local)	490.67	221.91	107100	91.57	55817
BMAML (local)	479.03	200.68	103560	94.11	56321
ABML (local)	402.38	170.32	87000	87.92	55312
MAML (local)	272.89	91.24	60000	92.10	55917
FOMAML (local)	269.01	91.12	60000	92.83	55917
Reptile (local)	**268.72**	**90.83**	60000	85.60	56525
StochLWTA-ML	272.14	102.56	60000	**97.79**	**54549**

3.4.2.1.8 How does StochLWTA-ML perform with more parameters?

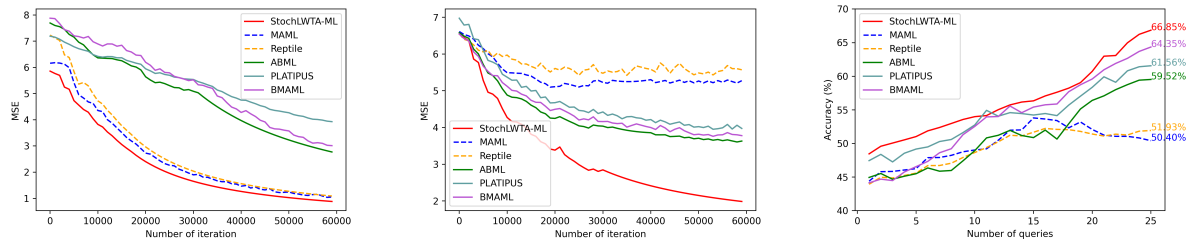
We repeated a classification experiment on Omniglot 20-way 1-shot by using a stochastic LWTA architecture of 4 convolutional layers. In this context, in each layer competition is performed among the feature maps of a convolutional kernel; this proceeds on a position-wise basis. The so-obtained convolutional deep network yields the same number

of parameters as in the state-of-the-art. Our experimental outcomes are conspicuous: our approach lost accuracy (the reported 97.79% accuracy reduced to 96.50%), while training time increased by four and inference time almost doubled; similar outcomes have been observed in the rest of the considered datasets.

3.4.2.2 Regression

In this Section, we compare our approach StochLWTA-ML with the locally reproduced baselines of Section 3.4.2.1, on two sinusoidal function regression problems. In the former case, we apply the default setting used in Finn et al. (2017); in the latter, we use a more challenging setting as proposed in Yoon et al. (2018), containing more uncertainty than the setting used in Finn et al. (2017). Specifically, the tasks distribution $P(T)$ is defined by a sinusoidal function $y = A\sin(\omega x + b) + \epsilon$, with amplitude A, frequency ω, phase b and observation noise ϵ. The parameters of each task are sampled from Uniform distributions $A \in [0.1, 5.0]$, $b \in [0.0, 2\pi]$, $\omega \in [0.5, 2.0]$, and observation noise ϵ from Normal distribution $N(0, (0.01A)^2)$. For each task, 10 input instances x are sampled from $[-5.0, 5.0]$. The network architecture employed for the baseline experiments consists of 2 hidden layers of size 64 with tanh activation. In our StochLWTA-ML case, we replace each hidden layer with a proposed stochastic LWTA one. Both architectures end up with a Softmax layer.

In Fig. 3.4, we illustrate the Mean Squared Error (MSE) performance on test tasks for both settings, after training all the models for 60000 iterations. Specifically, in Fig. 3.4(a) we observe that StochLWTA-ML, in the default setting, adapts equally fast as MAML and Reptile; thus, its convergence speed is much higher than the other time-consuming probabilistic methods ABML, BMAML and PLATIPUS. In the challenging setting of Fig. 3.4(b), the Bayesian methods StochLWTA-ML, ABML, BMAML and PLATIPUS can still perform well in a high uncertainty setting while non-Bayesian models MAML and Reptile fail to converge; this outcome is achieved due to the ability of Bayesian methods to reduce overfitting and generalize better. It is clear that our approach yields the most time-efficient method among the baselines.



(a) (b) (c)

Figure 3.4: Sinusoidal regression results: (a) MSE of default setting after 60000 training iterations, (b) MSE of challenging setting after 60000 training iterations, and (c) active learning setting

3.4.2.2.1 Is there a computational time trade-off for the reduced MSE?

In Table 3.7, we report the numbers of trainable parameters as an average over the default and challenging settings of the regression experiments. In a similar vein with the classification outcomes of Section 3.4.2.1.2, it appears that our methodology takes 77% *less* training time than the less efficient algorithms ABML, BMAML, PLATIPUS, and is comparable to other approaches. This demonstrates the effect of parameters' number reduction to the training process of a MAML based probabilistic model.

Table 3.7: Performance comparison: average wall-clock time (in msecs), and average number of baselines' trainable parameters over the two settings of regression experiments

Algorithm	Training	Prediction	Parameters
PLATIPUS (local)	359.23	135.74	21541
BMAML (local)	323.76	115.90	21541
ABML (local)	150.96	59.75	8622
MAML (local)	65.54	23.91	4315
FOMAML (local)	64.80	**23.02**	4315
Reptile (local)	64.63	23.30	4353
StochLWTA-ML	**63.11**	25.45	**2092**

3.4.2.3 Active learning with regression

To further evaluate the effectiveness of the proposed stochastic LWTA network, we now consider an active learning experiment, in the challenging setting described in Section 3.4.2.2. Specifically, we provide the models with 5 input observations x sampled from $[-5.0, 5.0]$. Then, each model queries to label 5 extra instances. The Bayesian methods

StochLWTA-ML, ABML, BMAML and PLATIPUS choose an item x^* that has the maximum variance across the sampled regressors. However, the non-Bayesian models MAML and Reptile, choose points randomly, since they do not entail a utility that handles uncertainty; as we demonstrate in Fig. 3.4(c), they fail to converge due to their inability to adapt to a more ambiguous training environment. Considering the Bayesian methods, our approach achieves the better predictive performance compared to the other baselines. This reaffirms the prevalence of our proposed network to another experimental scenario.

3.4.2.4 Additional experimental details

3.4.2.4.1 Further details on the used datasets

Omniglot is a dataset of 1623 characters from different alphabets, containing 20 examples per character scaled down to 28x28 grayscale pixels. The ratio between training and testings sets is 3:2, so after shuffling the character classes we randomly choose the first 974 classes for training and the remaining are left for testing. As for the Mini-Imagenet dataset, it has color images of size 84x84 and contains 100 classes with 600 examples from the ImageNet dataset. We randomly choose 45000 examples for the training phase and the rest constitute the testing population. The CIFAR-100 dataset consists of color images of size 32x32 and contains 100 classes with 600 images per class. We randomly choose 500 images per class for training and the rest 100 images per class constitute the testing population.

3.4.2.4.2 Few-Shot Classification Network Architectures

For the local replicates of prior ML algorithms in the experiments of our work, we follow the same architecture for the deep neural network as the one used by Vinyals et al. (2016). For Omniglot, the network is composed of 4 convolutional layers with 64 filters, 3 x 3 convolutions and 2 x 2 strides, followed by a Batch Normalization layer (Ioffe & Szegedy (2015)) and the final values of each layer are processed by an activation function. For both Mini-Imagenet and CIFAR-100, we use 4 convolutional layers with 32 filters to reduce overfitting like Ravi & Larochelle (2017), 3 x 3 convolutions followed by Batch Normalization layer and 2 $\times$ 2 max-pooling layer with the values of each layer finally

passed again through an activation block. The activation function used for the baseline experiments is: ReLU for Tables 3.1, 3.3, 3.6 and 3.7, and LWTA for Table 3.2.

3.4.2.4.3 What parameters do we count for the outcomes of Tables 3.3, 3.6 and 3.7?

The included parameters of each baseline for the outcomes of Tables 3.3, 3.6 and 3.7 are:

- PLATIPUS: $\Theta = \{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta^2, \boldsymbol{v}_\theta, \boldsymbol{\gamma}_p, \boldsymbol{\gamma}_q\}$
- BMAML: $\Theta = \{\theta^m\}_{m=1}^5$, for using 5 particles
- ABML: $\theta = \{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta^2\}$
- MAML: $\theta = \{\boldsymbol{\mu}_\theta\}$
- FOMAML: $\theta = \{\boldsymbol{\mu}_\theta\}$
- Reptile: $\theta = \{\boldsymbol{\mu}_\theta\}$
- StochLWTA-ML: $\theta = \{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta^2\}$

3.4.3 Conclusion

In this Section, we proposed a sparse and stochastic network paradigm for ML, with novel network design principles compared to currently used model-agnostic ML models. We introduced *stochastic LWTA activations* in the context of a *variational Bayesian treatment* that gave rise to a doubly-stochastic ML framework, bearing the promise of stronger generalization capacity. We evaluated our approach using standard few-shot image classification and regression benchmarks in the field, and showed that it outperformed the state-of-the-art in terms of both predictive accuracy, error rate and computational costs. The results have provide strong empirical evidence supporting our claims. Thus, the replacement of a standard ReLU layer with a stochastic LWTA one produces a quite promising network architecture that would be used in various DL tasks besides ML, e.g. Continual Learning.

Chapter 4

Continual Learning

Humans have the ability to acquire knowledge in a continuous manner, without losing previously learned knowledge (Shadmehr & Sandro (2012)). The skills and knowledge a human tries to learn in every day life are not a sequence of facts without context, but they are interrelated in the context of a task (Rajendra et al. (2021)). This enables the construction of new ideas or knowledge. MLe attempts to imitate this biologically inspired ability of sequential learning, in the context of Continual Learning (CL). CL, also referred to as Lifelong Learning (Thrun (1995)), aims to learn sequential tasks and acquire new information while preserving knowledge from previous learned tasks (Thrun & Mitchell (1995)). Recently, many applications have employed CL algorithms: sequential task processing (Thrun (1995)), streaming data processing (Aljundi et al. (2019)), healthcare (Lee & Lee (2021)), etc.

A famous variant of CL is class-incremental learning (CIL) (Belouadah & Popescu (2019), Gupta et al. (2020), Deng et al. (2021)). The main principle of CIL is a CL scenario where on each iteration we are dealing with data from a specific task, and each task contains new classes that must be learnt. The addressed problem poses the major challenge of preventing catastrophic forgetting (McCloskey & Cohen (1989)): as network parameters get modified to allow for learning the new classes, the training process tends to overwrite old knowledge about classes presented many iterations ago. This constitutes a key obstacle to achieving human-level intelligence.

Recently, there has been a large demand of using CIL from applications in industry and society. CIL offers a solution to various problems, such as: (i) *Memory restrictions.* Systems that have a small amount of space to store data cannot specialize in joint training strategies, since they can store only a limited set of examples per task. Thus, learning

must be done using CIL; (ii) *Data security and privacy restrictions.* Government systems containing sensible private data that cannot permanently save large amounts of data, can employ CIL. Also, healthcare systems that are prohibited by legislation to contain long-term storage of patients' data, CIL can be a solution (McClure et al. (2018)). And (iii) *Sustainable Information and communication technologies.* Training scenarios like learning GPT-2 (Brown et al. (2020)) entail excessive computational cost and carbon footprint. CIL provides more computationally efficient algorithms that require processing of new data only when updating the system.

4.1 CIL methods

Recently, researchers have proposed various approaches to solve catastrophic forgetting during CIL, which can be organized in three main categories: (1) *regularization-based* solutions that utilize knowledge distillation (Hinton et al. (2015), Li & Hoiem (2017)) or penalize changes in weights deemed important for previous tasks (Kirkpatrick et al. (2017), Zenke et al. (2017), Aljundi et al. (2018)); (2) *replay-based* methods that store in memory a collection of samples from previous tasks and replay them to retain the previous learned knowledge (Rebuffi et al. (2017), Saha et al. (2021)), and (3) *architecture-based* methods that split the dense model into task-related modules for knowledge transfer and acquisition (Rusu et al. (2016), Hung et al. (2019)).

Although the aforementioned CIL methods suppress catastrophic forgetting over sequentially arriving tasks, they often come at the price of increasing model size and computational footprint as new tasks arrive. To overcome this challenge, different research groups have recently drawn inspiration from the lottery ticket hypothesis (LTH) (Frankle & Carbin (2019)) to introduce the lifelong tickets (LLT) method (Chen et al. (2021)), the Winning SubNetworks (WSN) method (Kang et al. (2022)), and, more recently, the Soft-SubNetworks approach (Kang et al. (2023)). Specifically, LTH uses Iterative Magnitude Pruning that repeats multiple cycles of training, pruning, and weight rewinding to proves the existence of subnetworks of a neural network that can be trained to produce performance as good as the original network, with a much smaller size.

However, these recent advances are confronted with major limitations: (i) LLT entails an iterative pruning procedure, that requires multiple repetitions of the training algorithm for each task; (ii) the existing alternatives do not take into consideration the uncertainty in the used datasets, which would benefit from the subnetwork selection process being stochastic, as opposed to hard unit pruning. In fact, it has been recently shown that stochastic competition mechanisms among locally competing units can offer important generalization capacity benefits for deep networks used in as diverse challenges as adversarial robustness (Panousis et al. (2021)), video-to-text translation (Voskou et al. (2021)), and model-agnostic ML (Kalais & Chatzis (2022)).

Inspired from these facts, we propose a radically different regard toward addressing catastrophic forgetting in CIL. Our approach is founded upon the framework of stochastic local competition which is implemented in a task-wise manner. Specifically, our proposed approach relies upon the following novel contributions:

- **Task-specific sparsity in the learned representations.** We propose a novel mechanism that inherently learns to extract sparse task-specific data representations. Specifically, each layer of the network is split into blocks of competing units; local competition is stochastic and it replaces traditional nonlinearities, e.g. ReLU. Being presented with a new task, each block learns a distribution over its units that governs which unit specializes in the presented task. We dub this type of nonlinear units as *task winner-takes-all (TWTA)*. Under this scheme, the network learns a Categorical posterior over the competing block units; this is the winning unit posterior of the block. Only the winning unit of a block generates a non-zero output fed to the next network layer. This renders sparse the generated representations, with the sparsity pattern being task-specific.

- **Weight training strength regulation driven from the learned stochastic competition posteriors.** The network learns a global weight matrix that is not specific to a task, but evolves over time. During training, the network utilizes the learned Categorical posteriors over winning block units to dampen the update signal for weights that pertain to units with low winning posterior for the task at hand. In

a sense, the block units with high winning posterior tend to get a stronger weight training cycle.

- **Winner-based weight pruning at inference time.** During inference for a given task, we use the (Categorical) winner posteriors learned for the task to select the winner unit of each block; we zero-out the remainder block units. This forms a *task-winning ticket* used for inference. This way, the size of the network used at inference time is significantly reduced; pruning depends on the number of competing units per block, since we drop all block units except for the selected winner with maximum winning posterior.

We evaluate our approach, dubbed TWTA for CIL (*TWTA-CIL*), on image classification problems. We show that, compared to the current state-of-the-art methods in the field: (i) it offers a considerable improvement in generalization performance, and (ii) it produces this performance with a network architecture that imposes a significantly lower memory footprint and better computational efficiency.

The remainder of this Chapter is organized as follows: Section 4.1.1 briefly reviews related work. In Section 4.2, we introduce our approach and describe the related training and inference processes. In Section 4.3, we perform an extensive experimental evaluation and ablation study of the proposed approach. In the last Section 4.4, we summarize the contribution of this work.

4.1.1 State-of-the-art CIL methods comparison

Recent works in (Chen et al. (2021), Kang et al. (2022), Kang et al. (2023)) have pursued to build computationally efficient continual learners, that are robust to catastrophic forgetting, by drawing inspiration from LTH (Frankle & Carbin (2019)). These works compose sparse subnetworks that achieve comparable or/and even higher predictive performance than their initial counterparts. However, our work is substantially different from the existing state-of-the-art in various ways:

(i) Contrary to Chen et al. (2021), we do not employ iterative pruning, which repeats

multiple full cycles of network training and pruning, until convergence. Instead, we perform a single training cycle, at the end of which we select a (task-specific) subnetwork to perform inference for the task.

(ii) Kang et al. (2022) and Kang et al. (2023) select a subnetwork that will be used for the task at hand on the grounds of an optimization criterion for binary masks imposed over the network weights. Once this subnetwork has been selected, Kang et al. (2022) proceeds to train the values of the retained weights while Kang et al. (2023) trains a randomly selected subset of the weights of the whole network, to account for the case of a suboptimal subnetwork selection. On the contrary, our method attempts to encourage different units in a competing block to specialize to different tasks. Training is performed concurrently for the winner unit indicator hidden variables, the posteriors of which regulate weight updates, as well as the network weights themselves. Thus, network pruning comes at the end of weight updating and not beforehand. We posit that this regulated updating scheme, which does not entail a priori hard pruning decisions, facilitates generalization without harming catastrophic forgetting.

4.2 Proposed Approach

4.2.1 CIL definition

CIL objective is to learn a unified classifier from a sequential stream of data comprising different tasks that introduce new classes. CIL methods should scale to a large number of tasks without immense computational and memory growth. Let us consider a CIL problem T which consists of a sequence of n tasks, $T = \{(C^{(1)}, D^{(1)}), (C^{(2)}, D^{(2)}), \ldots, (C^{(n)}, D^{(n)})\}$. Each task t contains data $D^{(t)} = (\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)})$ and new classes $C^{(t)} = \{c_{m_{t-1}+1}, c_{m_{t-1}+2}, \ldots, c_{m_t}\}$, where m_t is the number of presented classes up to task t. We denote as $\boldsymbol{x}^{(t)}$ the input features, and as $\boldsymbol{y}^{(t)}$ the one-hot label vector corresponding to $\boldsymbol{x}^{(t)}$.

When training for the t-th task, we mainly use the data of the task, $D^{(t)}$; this may be augmented with a limited amount of additional data from the previous $t-1$ tasks, which

is retained in a memory buffer of limited size (Castro et al. (2018), Rebuffi et al. (2017)). Also, the learner has access to all the information and data $\{C^{(t-1)}, D^{(t-1)}\}$ of the model at the previous task $(t-1)$.

We consider learners-classifiers that are deep networks parameterized by weights $\boldsymbol{W}$, and we use $f(\boldsymbol{x}^{(t)}; \boldsymbol{W})$ to indicate the output Softmax logits for a given input $\boldsymbol{x}^{(t)}$. Facing a new dataset $D^{(t)}$, the model's goal is to learn new classes and maintain performance over old classes.

4.2.2 TWTA formulation

Let us denote as $\boldsymbol{x}^{(t)} \in \mathbb{R}^E$ an input representation vector presented to a dense ReLU layer of a traditional deep neural network, with corresponding weights matrix $\boldsymbol{W} \in \mathbb{R}^{E \times K}$. The layer produces an output vector $\boldsymbol{y}^{(t)} \in \mathbb{R}^K$, which is fed to the subsequent layers.

In our approach, a group of J ReLU units is replaced by a group of J competing linear units, organized in one block; each layer contains I blocks of J units. Within each block, different units are specialized in different tasks; only one block unit specializes in a given task t. The layer input is now presented to each block through weights that are organized into a three-dimensional matrix $\boldsymbol{W} \in \mathbb{R}^{E \times I \times J}$. Then, the j-th $(j = 1, \ldots, J)$ competing unit within the i-th $(i = 1, \ldots, I)$ block computes the sum $\sum_{e=1}^{E} (w_{e,i,j}) \cdot x_e^{(t)}$.

Fig. 4.1 illustrates the operation of a TWTA-based network architecture when dealing with task t. Due to the Gumbel-Softmax, for each task only one unit (the "winner") in a TWTA block will present its output to the next layer during forward passes through the network, while the rest are zeroed out. During backprop (training), the strength of the updating signal is regulated from the relaxed (continuous) output of the Gumbel-Softmax.

Let us consider the i-th block in a TWTA layer, comprising J units. We introduce the hidden winner indicator vector $\boldsymbol{\xi}_{t,i} = [\xi_{t,i,j}]_{j=1}^{J}$ of the i-th block pertaining to the t-th task. It holds $\xi_{t,i,j} = 1$ if the j-th unit in the i-th block has specialized in the t-th task (winning unit), $\xi_{t,i,j} = 0$ otherwise. We also denote $\boldsymbol{\xi}_t \in \{0,1\}^{I \cdot J}$ the vector that holds all the $\boldsymbol{\xi}_{t,i} \in \{0,1\}^J$ subvectors.

On this basis, the output of the layer, $\boldsymbol{y}^{(t)} \in \mathbb{R}^{I \cdot J}$, is composed of I subvectors $\boldsymbol{y}_i^{(t)} \in \mathbb{R}^J$

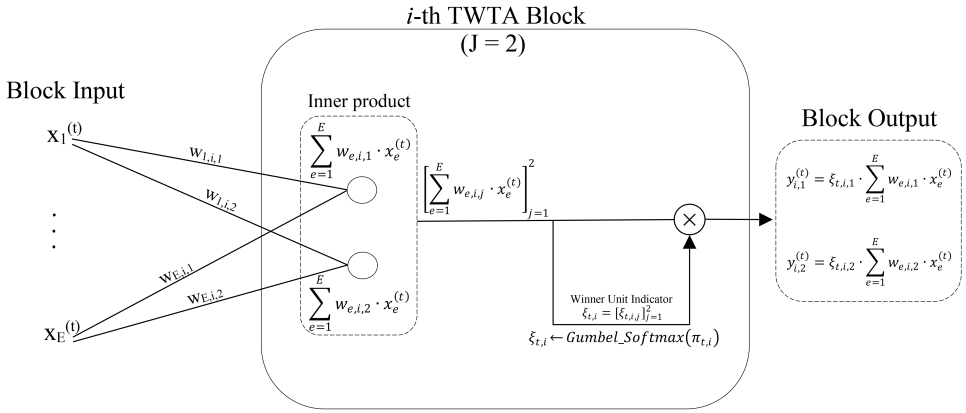


Figure 4.1: A detailed graphical illustration of the i-th block of a proposed TWTA layer; for demonstration purposes, we choose $J = 2$ competing units per block. Inputs $\boldsymbol{x}^{(t)} = \{x_1^{(t)}, \ldots, x_E^{(t)}\}$ are presented to each unit in the i-th block, when training on task t. Due to the TWTA mechanism, during forward passes through the network, only one competing unit propagates its output to the next layer; the rest are zeroed-out.

and is sparse, since all units except for one, in each block, yield zero values. Succinctly, we can write $\boldsymbol{y}_i^{(t)} = [y_{i,j}^{(t)}]_{j=1}^{J}$, where:

$$y_{i,j}^{(t)} = \xi_{t,i,j} \sum_{e=1}^{E} (w_{e,i,j}) \cdot x_e^{(t)} \in \mathbb{R} \tag{4.1}$$

We postulate that the hidden winner indicator variables are drawn from a Categorical posterior distribution that yields:

$$p(\boldsymbol{\xi}_{t,i}) = \text{Categorical}(\boldsymbol{\xi}_{t,i} | \boldsymbol{\pi}_{t,i}) \tag{4.2}$$

The hyperparameters $\boldsymbol{\pi}_{t,i}$ are optimized during model training, as we explain next. The network learns the global weight matrix $\boldsymbol{W}$, that is not specific to a task, but evolves over time. During training, by learning different winning unit distributions, $p(\boldsymbol{\xi}_{t,i})$, for each task, we appropriately mask large parts of the network, dampen the training signal strength for these parts, and mainly direct the training signal to update the fraction of $\boldsymbol{W}$ that pertains to the remainder of the network. We argue that this asymmetric weight updating scheme during backpropagation, which focuses on a small winning subnetwork, yields a model less prone to catastrophic forgetting.

In Fig. 4.1, we depict the operation principles of TWTA-based network. As we show

therein, the winning information is encoded into the trained posteriors $p(\boldsymbol{\xi}_{t,i})$, which are used to regulate weight training, as we explain in Section 2.1.5 of Chapter 2. This is radically different from Chen et al. (2021), as we do not search for optimal winning tickets during CIL via repetitive pruning and retraining for each arriving task. This is also radically different from Kang et al. (2023), where a random uniform mask is drawn for regulating which weights will be updated, and another mask is optimized to select the subnetwork specializing to the task at hand. Instead, we perform a single backward pass to update the winning unit distributions, $p(\boldsymbol{\xi}_{t,i})$, and the weight matrix, $\boldsymbol{W}$. Importantly, the updates of the former (winning unit posteriors) regulate the updates of the latter (weight matrix).

Inference. As we depict in Fig. 4.1, during inference for a given task t, we retain the unit with maximum hidden winner indicator variable posterior, $\pi_{t,i,j}$, in each TWTA block i, and prune-out the weights pertaining to the remainder of the network. Thus:

$$\text{winner}_{t,i} \triangleq \arg\max_{j} \pi_{t,i,j} \tag{4.3}$$

Apparently, this way the proportion of retained weights for task t is only equal to the $\frac{1}{J} * 100\%$ of the number of weights the network is initialized with.

4.2.3 A Convolutional Variant

Further, to accommodate architectures comprising convolutional operations, we consider a variant of the TWTA layer (Kalais & Chatzis (2023)), inspired from Panousis et al. (2019). In the remainder of this work, this will be referred to as the Conv-TWTA layer, while the original TWTA layer will be referred to as the dense variant. The graphical illustration of Conv-TWTA is provided in Fig. 4.2.

Specifically, let us assume an input tensor $\boldsymbol{X}^{(t)} \in \mathbb{R}^{H \times L \times C}$ of a layer, where H, L, C are the height, length and channels of the input. We define a set of kernels, each with weights $\boldsymbol{W}_i \in \mathbb{R}^{h \times l \times C \times J}$, where h, l, C, J are the kernel height, length, channels and competing feature maps, and $i = 1, \ldots, I$.

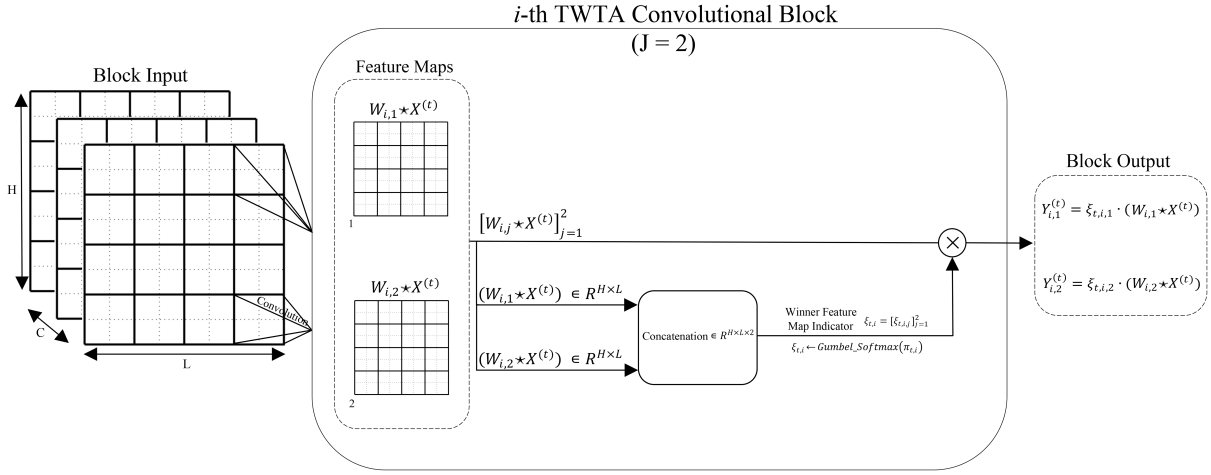


Figure 4.2: The convolutional TWTA variant; for demonstration purposes, we choose $J = 2$ competing feature maps per kernel. Due to the TWTA mechanism, during forward passes through the network, only one competing feature map propagates its output to the next layer; the rest are zeroed-out.

Here, analogously to the grouping of linear units in a dense TWTA layer, local competition is performed among feature maps in a kernel. Thus, each kernel is treated as a TWTA block, feature maps in a kernel compete among them, and multiple kernels of competing feature maps constitute a Conv-TWTA layer.

This way, the output $\boldsymbol{Y}^{(t)} \in \mathbb{R}^{H \times L \times (I \cdot J)}$ of a layer under the convolutional variant is obtained via concatenation along the last dimension of the subtensors $\boldsymbol{Y}_i^{(t)}$:

$$\boldsymbol{Y}_i^{(t)} = \boldsymbol{\xi}_{t,i} \cdot (\boldsymbol{W}_i \star \boldsymbol{X}^{(t)}) \in \mathbb{R}^{H \times L \times J} \tag{4.4}$$

where "$\star$" denotes the convolution operation.

Here, the winner indicator variables $\boldsymbol{\xi}_{t,i}$ are drawn again from the distribution of Eq. (4.2); they now govern competition among feature maps of a kernel.

Inference. At inference time, we employ a similar winner-based weight pruning strategy as for dense TWTA layers; for each task, the weights associated with one feature map (the winner) are retained while the rest are zeroed-out. Specifically, the winning feature map in a kernel i for task t is selected through arg max over the hidden winner indicator variable posteriors, $\pi_{t,i,j}\, \forall j$, similar to Eq. (4.3) (see also Fig. 4.2).

4.2.4 Training

For each task t, our approach consists in executing *a single* full training cycle. The performed training cycle targets both the network weights, $\boldsymbol{W}$, and the posterior hyperparameters $\boldsymbol{\pi}_{t,i}$ of the winner indicator hidden variables pertaining to the task, $p(\boldsymbol{\xi}_{t,i})\ \forall i$.

The vectors $\boldsymbol{\pi}_{t,i}$ are initialized at random, while the network weights, $\boldsymbol{W}$, "continue" from the estimator obtained after training on task $t-1$. We denote as $\boldsymbol{W}^{(t)}$ the updated weights estimator obtained through the training cycle on the t-th task.

To perform training, we resort to maximization of the resulting evidence lower-bound (ELBO) of the network. Let us consider the u-th training iteration on task t, with data batch $D_u^{(t)} = (X_u^{(t)}, Y_u^{(t)})$. Let $\text{CE}(Y_u^{(t)}, f(X_u^{(t)}; \boldsymbol{W}^{(t)}, \hat{\boldsymbol{\xi}}_t))$ be the categorical cross-entropy between the data labels $Y_u^{(t)}$ and the class probabilities $f(X_u^{(t)}; \boldsymbol{W}^{(t)}, \hat{\boldsymbol{\xi}}_t)$ generated from the penultimate Softmax layer of the network; here, $\hat{\boldsymbol{\xi}}_t = [\hat{\boldsymbol{\xi}}_{t,i}]_{i=1}^{I}$ is a vector concatenation of single Monte-Carlo (MC) samples drawn from the Categorical posteriors $p(\boldsymbol{\xi}_{t,i})$.

To ensure low-variance ELBO gradients with only one drawn MC sample from each posterior, we reparameterize these samples by resorting to the Gumbel-Softmax relaxation (Maddison et al. (2017)). The Gumbel-Softmax relaxation yields sampled instances $\hat{\boldsymbol{\xi}}_{t,i}$ under the following expression:

$$\begin{aligned} &\hat{\boldsymbol{\xi}}_{t,i} = \text{Softmax}(([\log \pi_{t,i,j} + g_{t,i,j}]_{j=1}^{J})/\tau) \in \mathbb{R}^J,\ \forall i \\ &\text{where}: g_{t,i,j} = -\log(-\log U_{t,i,j}),\ U_{t,i,j} \sim \text{Uniform}(0,1) \end{aligned} \tag{4.5}$$

and $\tau \in (0, \infty)$ is a temperature factor that controls how closely the Categorical distribution $p(\boldsymbol{\xi}_{t,i})$ is approximated by the continuous relaxation.

Note: We emphasize that this reparameterized relaxation is only applied to backprop training. As already discussed in Secs. 4.2.2 and 4.2.3, the feedforward pass is performed, by definition, by computing the task-wise discrete masks:

$$\tilde{\boldsymbol{\xi}}_{t,i} = \text{onehot}\left(\arg\max_j \pi_{t,i,j}, J\right) \in \mathbb{R}^J \tag{4.6}$$

We augment our loss function with the KL divergence between the pairs of Softmax

logits pertaining to $(t-1)$-th task data, obtained by using the winner masks $\tilde{\boldsymbol{\xi}}_{t-1,i}$ of the $(t-1)$-th task and either: (i) the weight updates $\boldsymbol{W}^{(t-1)}$ at the end of the training cycle on the $(t-1)$-th task; or (ii) the unknown weight updates $\boldsymbol{W}^{(t)}$ sought on the t-th task. This enforces consistency and regularization over the last model training cycle, without breaking our core model assumptions.

Finally, in our experiments we employ an additional regularization term, similar to previous work, e.g. (Chen et al. (2021), Kang et al. (2023)), only for comparability purposes. This regularization term is obtained as follows; consider task t:

1. We keep few exemplars X^{few} from the previous task, $t-1$, in a limited size buffer.

2. These may be possibly augmented with more similar unlabeled data, X', retrieved from some repository; selection is performed on the grounds of feature similarity with the data in the buffer, based on l_2 norm distance.

3. Then, we introduce an additional loss term that enforces similarity between the Softmax logits for $X^{KD} = (X^{few} \cup X')$ obtained by (i) using the old weight estimates, $\boldsymbol{W}^{(t-1)}$; and (ii) the sought updates, $\boldsymbol{W}^{(t)}$.

Note that, in our experiments, we also perform an ablation where the two aforementioned regularization terms are omitted during training (see Section 4.3.4.2).

In summary, the training criterion becomes:

$$\begin{aligned} L_u^{(t)} &= -\text{CE}(Y_u^{(t)}, f(X_u^{(t)}; \boldsymbol{W}^{(t)}, \hat{\boldsymbol{\xi}}_t)) - KL[\, p(\boldsymbol{\xi}_t) \,||\, q(\boldsymbol{\xi}_t) \,] \\ &- KL[\, f(X_u^{(t-1)}; \boldsymbol{W}^{(t)}, \tilde{\boldsymbol{\xi}}_{t-1}) \,||\, f(X_u^{(t-1)}; \boldsymbol{W}^{(t-1)}, \tilde{\boldsymbol{\xi}}_{t-1}) \,] - KD_{loss} \end{aligned} \tag{4.7}$$

where we consider that, a priori, $q(\boldsymbol{\xi}_{t,i})$ is a Categorical$(1/J)$; then, we obtain:

$$\begin{aligned} &KL[\, p(\boldsymbol{\xi}_{t,i}) \,||\, q(\boldsymbol{\xi}_{t,i}) \,] = \mathbb{E}_{p(\boldsymbol{\xi}_{t,i})}[\log p(\boldsymbol{\xi}_{t,i}) - \log q(\boldsymbol{\xi}_{t,i})] \approx \log p(\hat{\boldsymbol{\xi}}_{t,i}) - \log q(\hat{\boldsymbol{\xi}}_{t,i}) \Rightarrow \\ &KL[\, p(\boldsymbol{\xi}_t) \,||\, q(\boldsymbol{\xi}_t)] = \sum_{i=1}^{I} \Big(\log p(\hat{\boldsymbol{\xi}}_{t,i}) - \log q(\hat{\boldsymbol{\xi}}_{t,i}) \Big) \end{aligned} \tag{4.8}$$

where we use Eq. (4.5), that is the Gumbel-Softmax relaxation, $\hat{\boldsymbol{\xi}}_{t,i}$, of the Categorical hidden winner indicator variables $\boldsymbol{\xi}_{t,i}$.

On the other hand:

$$KL[\, f(X_u^{(t-1)}; \boldsymbol{W}^{(t)}, \tilde{\boldsymbol{\xi}}_{t-1}) \,||\, f(X_u^{(t-1)}; \boldsymbol{W}^{(t-1)}, \tilde{\boldsymbol{\xi}}_{t-1}) \,] = \log f(X_u^{(t-1)}; \boldsymbol{W}^{(t)}, \tilde{\boldsymbol{\xi}}_{t-1}) - \log f(X_u^{(t-1)}; \boldsymbol{W}^{(t-1)}, \tilde{\boldsymbol{\xi}}_{t-1}) \tag{4.9}$$

where we use Eq. (4.6), that is the pruned network pertaining to task $(t-1)$, obtained through application of the binary masks $\tilde{\boldsymbol{\xi}}_{t-1,i}$.

Finally, following Hinton et al. (2015), we define the knowledge distillation loss in (4.7) as:

$$KD_{loss} = -H\left(f(X_u^{KD}; \boldsymbol{W}^{(t)}, \tilde{\boldsymbol{\xi}}_{t-1})\right) \cdot \log H\left(f(X_u^{KD}; \boldsymbol{W}^{(t-1)}, \tilde{\boldsymbol{\xi}}_{t-1})\right) \tag{4.10}$$

where we again use the pruned network pertaining to task $(t-1)$, while $H(f(\cdot)) \triangleq \text{Softmax}(f(\cdot)/\gamma)$ and $\gamma = 2$.

4.3 Experiments

We evaluate on CIFAR-100 (Krizhevsky et al. (2012)), Tiny-ImageNet (Le & Yang (2015)), PMNIST (LeCun (1998)) and Omniglot Rotation (Lake et al., 2017). We randomly divide the 100 classes of CIFAR-100 into 10 tasks with 10 classes per task; the 200 classes of Tiny-ImageNet into 40 tasks with 5 classes per task; the 200 classes of PMNIST into 20 tasks with 10 classes per task; and in the case of Omniglot Rotation, we divide the available 1200 classes into 100 tasks with 12 classes per task. Also, we evaluate on the 5-Datasets (Saha et al. (2021)) benchmark, in order to examine how our method performs in case that cross-task generalization concerns different datasets.

We adopt the original ResNet18 network (He et al. (2016a)) for Tiny-ImageNet, PMNIST and 5-Datasets; we use a 5-layer AlexNet similar to Saha et al. (2021) for the experiments on CIFAR-100, and LeNet (LeCun (1998)) for Omniglot Rotation. In the case of our approach, we modify those baselines by replacing each ReLU layer with a layer of (dense) TWTA blocks, and each convolutional layer with a layer of Conv-TWTA blocks. More information is provided in Section 4.3.5.2.

For both the network weights, $\boldsymbol{W}$, and the log hyperparameters, $\log \boldsymbol{\pi}_{t,i}$, we employ

Glorot Normal initialization (Glorot & Bengio (2010)). At the first training iteration of a new task, we initialize the Gumbel-Softmax relaxation temperature τ to 0.67; as the training proceeds, we linearly anneal its value to 0.01. We use SGD optimizer (Robbins (2007)) with a learning rate linearly annealed to 0, and initial value of 0.1. We run 100 training epochs per task, with batch size of 40.

4.3.1 Experimental results

Table 4.1: Comparisons on CIFAR-100, Tiny-ImageNet, PMNIST, Omniglot Rotation and 5-Datasets. We report the mean and standard deviation of the classification accuracy (%), obtained over three experiment repetitions with different seeds; * are results obtained from local replicates. We set $J = 8$; thus, the proportion of retained weights for each task, after training, is equal to the $(\frac{1}{J} * 100 = 12.50)\%$ of the initial network.

Algorithm	**CIFAR-100**	**Tiny-ImageNet**	**PMNIST**	**Omniglot Rotation**	**5-Datasets**
GEM (Lopez-Paz & Ranzato (2017))	59.24 $\pm$ 0.61*	39.12 $\pm$ 0.72*	-	-	-
iCaRL (Rebuffi et al. (2017))	42.45 $\pm$ 0.27*	43.97 $\pm$ 0.40*	55.82 $\pm$ 0.71*	44.60 $\pm$ 0.61*	48.01 $\pm$ 1.03*
ER (Chaudhry et al. (2019))	59.12 $\pm$ 0.82*	37.65 $\pm$ 0.88*	-	-	-
IL2M (Belouadah & Popescu (2019))	53.24 $\pm$ 0.33*	47.13 $\pm$ 0.49*	60.12 $\pm$ 1.05*	51.31 $\pm$ 0.51*	55.93 $\pm$ 0.46*
La-MAML (Gupta et al. (2020))	60.02 $\pm$ 1.08*	55.45 $\pm$ 1.10*	80.82 $\pm$ 0.23*	61.73 $\pm$ 0.31*	75.92 $\pm$ 1.04*
FS-DGPM (Deng et al. (2021))	63.81 $\pm$ 0.22*	59.74 $\pm$ 0.64*	80.92 $\pm$ 0.63*	62.83 $\pm$ 0.93*	76.10 $\pm$ 0.13*
GPM (Saha et al. (2021))	62.40 $\pm$ 0.09*	56.28 $\pm$ 0.81*	83.51 $\pm$ 0.11*	74.63 $\pm$ 0.60*	80.75 $\pm$ 0.67*
SoftNet (80%)	48.52 $\pm$ 0.08*	54.02 $\pm$ 1.13*	64.02 $\pm$ 0.38*	55.82 $\pm$ 0.29*	57.60 $\pm$ 1.12*
SoftNet (10%)	43.61 $\pm$ 0.18*	47.30 $\pm$ 0.92*	57.93 $\pm$ 0.65*	46.83 $\pm$ 1.26*	52.11 $\pm$ 1.40*
LLT (100%)	61.46 $\pm$ 1.18*	58.45 $\pm$ 0.73*	80.38 $\pm$ 0.08*	70.19 $\pm$ 0.24*	74.61 $\pm$ 0.81*
LLT (6.87%)	62.69 $\pm$ 0.05*	59.03 $\pm$ 0.57*	80.91 $\pm$ 1.02*	68.46 $\pm$ 1.04*	75.13 $\pm$ 0.48*
WSN (50%)	64.41 $\pm$ 0.60*	57.83 $\pm$ 0.94*	84.69 $\pm$ 0.18*	73.84 $\pm$ 0.35*	82.13 $\pm$ 0.04*
WSN (8%)	63.24 $\pm$ 0.34*	57.11 $\pm$ 0.63*	83.03 $\pm$ 1.04*	72.91 $\pm$ 0.19*	79.61 $\pm$ 0.10*
TWTA-CIL (12.50%)	**66.53** $\pm$ 0.42	**61.93** $\pm$ 0.11	**85.92** $\pm$ 0.13	**76.48** $\pm$ 0.62	**83.77** $\pm$ 0.50

In Table 4.1, we show how TWTA-CIL performs in various benchmarks compared to popular alternative methods. We emphasize that the performance of SoftNet and WSN is provided for the configuration reported in the literature that yields the best accuracy, as well as for the reported configuration that corresponds to the proportion of retained weights closest to our method. Turning to LLT, we report how the method performs with no pruning and with pruning ratio closest to our method.

As we observe, our method outperforms the existing state-of-the-art in every considered benchmark. For instance, WSN performs worse than TWTA-CIL (12.50%), irrespectively of whether WSN retains a greater or a lower proportion of the initial network. Thus, our approach successfully discovers sparse subnetworks (*winning tickets*) that are powerful enough to retain previous knowledge, while generalizing well to new unseen tasks.

Finally, it is interesting to examine how the winning ticket vectors differentiate across tasks. To this end, we compute the overlap among the $\tilde{\boldsymbol{\xi}}_t = [\tilde{\boldsymbol{\xi}}_{t,i}]_i$ vectors, defined in Eq. (4.6), for all consecutive pairs of tasks, $(t-1, t)$, and compute average percentages. We observe that average overlap percentages range from 6.38% to 10.67% across the considered datasets; this implies clear differentiation.

4.3.2 Computational times for training CIL methods

In Table 4.2, we report training wall-clock times for TWTA-CIL and the locally reproduced state-of-the-art alternatives. Our experiments have been executed on a single Tesla-K40c GPU. It is apparent that our method yields much improved training algorithm computational costs over all the alternatives.

Table 4.2: Average training wall-clock time (in secs), c.f. Table 4.1.

Algorithm	**CIFAR-100**	**Tiny-ImageNet**	**PMNIST**	**Omniglot Rotation**	**5-Datasets**
GEM	3747.15	6856.32	-	-	-
iCaRL	1350.68	2449.05	1278.38	150.73	3404.49
ER	3869.78	7182.04	-	-	-
IL2M	3896.67	7487.16	2767.11	323.59	7924.12
La-MAML	4073.61	7560.34	3726.18	443.91	10430.52
FS-DGPM	4398.16	7839.63	4012.36	471.03	11323.63
GPM	4104.05	7481.03	3949.56	470.70	9803.71
SoftNet (80%)	4791.13	7984.05	3602.84	419.28	9760.44
SoftNet (10%)	3091.16	5529.12	2314.67	283.06	6412.02
LLT (100%)	3648.82	6746.32	2719.61	316.91	7746.85
LLT (6.87%)	1850.36	3449.52	1268.50	155.53	3613.81
WSN (50%)	4952.67	8074.94	3679.73	443.06	10413.92
WSN (8%)	3262.04	5915.86	2415.57	296.14	6837.21
TWTA-CIL (12.50%)	**1039.73**	**1914.63**	**859.27**	**127.41**	**2512.74**

4.3.3 Reduction of forgetting tendencies

To examine deeper the obtained improvement in forgetting tendencies, we report the *backward-transfer and interference* (BTI) values of the considered methods in Table 4.3. BTI measures the average accuracy change of each task as the considered network gets adapted to learn a new task; thus, it is immensely relevant to this empirical analysis. A smaller value of BTI implies lesser forgetting as the network gets trained on additional tasks. As Table 4.3 shows, our approach forgets less than the baselines on all benchmarks.

Table 4.3: BTI over the considered algorithms and datasets of Table 4.1; the lower the better.

Algorithm	CIFAR-100	Tiny-ImageNet	PMNIST	Omniglot Rotation	5-Datasets
iCaRL	13.41	6.45	8.51	18.41	23.56
IL2M	20.41	7.61	9.03	14.60	19.14
La-MAML	7.84	13.84	10.51	17.04	15.13
FS-DGPM	9.14	12.25	8.85	**13.64**	19.51
GPM	12.44	8.03	11.94	16.39	17.11
SoftNet (80%)	13.80	9.62	10.38	18.12	18.04
SoftNet (10%)	12.09	8.33	9.76	16.30	18.68
LLT (100%)	15.02	7.05	9.54	15.31	14.80
LLT (6.87%)	14.61	3.51	11.84	17.12	17.46
WSN (50%)	11.14	4.81	10.51	14.20	20.41
WSN (8%)	10.58	8.78	9.32	15.34	18.92
TWTA-CIL (12.50%)	**6.14**	**2.50**	**8.04**	**13.64**	**13.51**

4.3.4 Ablations

4.3.4.1 Effect of block size J

Further, we re-evaluate TWTA-CIL with various block size values J (and correspondingly varying number of layer blocks, I). In all cases, we ensure that the total number of feature maps, for a convolutional layer, or units, for a dense layer, which equals $I * J$, remains the same as in the original architecture of Section 4.3.1. This is important, as it does not change the total number of trainable parameters, but only the organization into blocks under the local winner-takes-all rationale. Different selections of J result in different percentages of remaining network weights at inference time, as we can see in Table 4.4 (datasets Tiny-ImageNet and CIFAR-100). As we observe, the "TWTA-CIL (12.50%)" alternative, with $J = 8$, is the most accurate configuration of TWTA-CIL.

Table 4.4: Effect of block size J; Tiny-ImageNet and CIFAR-100 datasets. The higher the block size J the lower the fraction of the trained network retained at inference time.

	Tiny-ImageNet			CIFAR-100		
Algorithm	Time	Accuracy	J	Time	Accuracy	J
TWTA-CIL (50%)	2634.02	61.32 ± 0.28	2	1493.79	65.73 ± 0.20	2
TWTA-CIL (25%)	2293.81	61.04 ± 0.13	4	1301.20	65.45 ± 0.10	4
TWTA-CIL (12.50%)	1914.63	**61.93 ± 0.11**	8	1039.73	**66.53 ± 0.42**	8
TWTA-CIL (6.25%)	1556.09	61.45 ± 0.12	16	801.46	65.89 ± 0.12	16
TWTA-CIL (3.125%)	1410.64	60.86 ± 0.55	32	785.93	65.40 ± 0.48	32

4.3.4.2 How does TWTA-CIL perform without the use of the additional regularization terms in the training criterion of Eq. (4.7)?

In Table 4.5 below, we provide full experimental CIL results for our method without the use of the two regularization terms; namely, these two terms are the knowledge distillation term on an external unlabelled dataset (Eq. (4.10)) and the KL divergence term concerning the previous state of the model (Eq. (4.9)). As we show, omitting these terms incurs an accuracy drop which does not exceed 1% in all cases. Thus, the two terms only offer an extra regularization option, which is of minor contribution to the overall method accuracy. In Table 4.6, we show how computational savings improve even more in that case.

Table 4.5: Comparisons on CIFAR-100, Tiny-ImageNet, PMNIST, Omniglot Rotation and 5-Datasets. We report the classification accuracies (%) for our approach; † denotes results for our method without the use of the two regularization terms, while * are the results reported in Table 4.1.

Algorithm	CIFAR-100	Tiny-ImageNet	PMNIST	Omniglot Rotation	5-Datasets
TWTA-CIL (12.50%)	65.83†	61.12†	85.38†	75.90†	83.04†
TWTA-CIL (12.50%)	**66.53***	**61.93***	**85.92***	**76.48***	**83.77***

Table 4.6: Average training wall-clock time (in secs), c.f. Table 4.5; † denotes results for our method without the use of the two regularization terms, while * are the results reported in Table 4.2.

Algorithm	CIFAR-100	Tiny-ImageNet	PMNIST	Omniglot Rotation	5-Datasets
TWTA-CIL (12.50%)	987.04†	1819.94†	814.05†	119.03†	2387.94†
TWTA-CIL (12.50%)	**1039.73***	**1914.63***	**859.27***	**127.41***	**2512.74***

4.3.4.3 Experimental results on task-incremental learning

Finally, we repeat the experiments on all datasets of Table 4.1 in a task-incremental learning (TIL) setting. Such a setting requires the correct task identifier (task-id) for each test sample to be given at inference. This means that: (i) at training time, we update only the latent indicator vector $\boldsymbol{\xi}$ that pertains to the known task at hand; (ii) at inference time, we use the inferred latent indicator vector $\boldsymbol{\xi}$ of the task we are dealing with, which was obtained at a previous training cycle. In all cases, we use the last update of the network weights $\boldsymbol{W}$.

In Table 4.7, we show how TWTA-CIL performs on CIFAR-100, Tiny-ImageNet, PMNIST, Omniglot Rotation, and 5-Datasets and compare our findings to current state-of-the-art algorithms; † denotes results adopted from Kang et al. (2022), while * are results obtained from local replicates. For the latter, the reported results are averages of the classification (%) accuracy and corresponding standard deviations obtained over three experiment repetitions with different random seeds. As we observe, our method outperforms the existing state-of-the-art in every considered benchmark.

Table 4.7: Performance in the benchmarks of Table 4.1 when task-id is known, i.e. addressing a TIL scenario.

Algorithm	**CIFAR-100**	**Tiny-ImageNet**	**PMNIST**	**Omniglot Rotation**	**5-Datasets**
EWC (Kirkpatrick et al. (2017))	72.77 $\pm$ 0.45†	-	92.01 $\pm$ 0.56†	68.66 $\pm$ 1.92†	88.64 $\pm$ 0.26†
GEM (Lopez-Paz & Ranzato (2017))	70.15 $\pm$ 0.34†	50.57 $\pm$ 0.61†	-	-	-
iCaRL (Rebuffi et al. (2017))	53.50 $\pm$ 0.81†	54.77 $\pm$ 0.32†	66.89 $\pm$ 0.82*	53.07 $\pm$ 1.13*	62.13 $\pm$ 0.71*
ER (Chaudhry et al. (2019))	70.07 $\pm$ 0.35†	48.32 $\pm$ 1.51†	-	-	-
La-MAML (Gupta et al. (2020))	71.37 $\pm$ 0.67†	66.90 $\pm$ 1.65†	90.02 $\pm$ 0.11*	70.62 $\pm$ 1.73*	86.92 $\pm$ 1.31*
FS-DGPM (Deng et al. (2021))	74.33 $\pm$ 0.31†	70.41 $\pm$ 0.30†	91.56 $\pm$ 0.19*	73.09 $\pm$ 1.14*	87.90 $\pm$ 0.33*
GPM (Saha et al. (2021))	73.18 $\pm$ 0.52†	67.39 $\pm$ 0.47†	94.96 $\pm$ 0.07†	85.24 $\pm$ 0.37†	91.22 $\pm$ 0.20†
SoftNet (80%)	60.13 $\pm$ 0.41*	63.81 $\pm$ 1.16*	75.30 $\pm$ 0.11*	62.04 $\pm$ 0.84*	70.38 $\pm$ 0.17*
SoftNet (10%)	55.01 $\pm$ 1.40*	57.15 $\pm$ 0.28*	68.54 $\pm$ 1.12*	55.92 $\pm$ 0.04*	66.01 $\pm$ 1.54*
LLT (100%)	72.65 $\pm$ 0.24*	69.13 $\pm$ 0.51*	92.80 $\pm$ 0.65*	82.30 $\pm$ 0.28*	86.42 $\pm$ 0.71*
LLT (6.87%)	73.08 $\pm$ 0.17*	70.15 $\pm$ 0.18*	92.12 $\pm$ 0.33*	81.63 $\pm$ 0.35*	86.03 $\pm$ 0.51*
WSN (50%)	76.38 $\pm$ 0.34†	69.06 $\pm$ 0.82†	96.24 $\pm$ 0.11†	79.80 $\pm$ 2.16†	93.41 $\pm$ 0.13†
WSN (8%)	74.12 $\pm$ 0.37*	71.40 $\pm$ 0.12*	95.80 $\pm$ 0.32*	83.72 $\pm$ 0.28*	92.13 $\pm$ 0.07*
TWTA-CIL (12.50%)	**77.61** $\pm$ 0.23	**72.28** $\pm$ 0.19	**96.43** $\pm$ 0.03	**86.53** $\pm$ 0.18	**93.81** $\pm$ 0.24

4.3.5 Additional experimental details

4.3.5.1 More details on the used datasets

5-Datasets is a mixture of 5 different vision datasets: CIFAR-10, MNIST (LeCun (1998)), SVHN (Netzer et al. (2011)), FashionMNIST (Xiao et al. (2017)) and notMNIST (Bui & Chang (2016)). Each dataset consists of 10 classes, and classification on each dataset is treated as a single task. PMNIST is a variant of MNIST, where each task is generated by shuffling the input image pixels by a fixed permutation. In the case of Omniglot Rotation, we preprocess the raw images of Omniglot dataset by generating rotated versions of $(90°, 180°, 270°)$ as in Kang et al. (2022). For 5-Datasets, similar to Kang et al. (2022), we pad 0 values to raw images of MNIST and FashionMNIST, convert them to RGB format to have a dimension of $3\times32\times32$, and finally normalize the raw image data. All datasets

used in Section 4.3 were randomly split into training and testings sets with ratio of 9:1. The number of stored images in the memory buffer - per class - is 5 for Tiny-ImageNet, and 10 for CIFAR-100, PMNIST, Omniglot Rotation and 5-Datasets.

The external unlabeled data are retrieved from 80 Million Tiny Image dataset (Torralba et al. (2008)) for CIFAR-100, PMNIST, Omniglot Rotation and 5-Datasets, and from ImageNet dataset (Krizhevsky et al. (2012)) for Tiny-ImageNet. We used a fixed buffer size of 128 for querying the same number of unlabeled images per class of learned tasks at each training iteration, based on the feature similarity that is defined by l_2 norm distance.

4.3.5.2 Modified Network Architecture details

ResNet18

The original ResNet18 comprises an initial convolutional layer with 64 3x3 kernels, 4 blocks of 4 convolutional layers each, with 64 3x3 kernels on the layers of the first block, 128 3x3 kernels for the second, 256 3x3 kernels for the third and 512 3x3 kernels for the fourth. These layers are followed by a dense layer of 512 units, a pooling and a final Softmax layer. In our modified ResNet18 architecture, we consider kernel size = 3 and padding = 1; in Table 4.8, we show the number of used kernels / blocks and competing feature maps / units, J, in each modified layer.

Table 4.8: Modified ResNet18 architecture parameters.

Layer Type	**Kernels / blocks** ($J = 2$)	**Kernels / blocks** ($J = 4$)	**Kernels / blocks** ($J = 8$)	**Kernels / blocks** ($J = 16$)	**Kernels / blocks** ($J = 32$)
TWTA-Conv	8	4	2	1	1
4x TWTA-Conv	8	4	2	1	1
4x TWTA-Conv	8	4	2	1	1
4x TWTA-Conv	16	8	4	2	1
4x TWTA-Conv	16	8	4	2	1
TWTA-Dense	16	8	4	2	1

AlexNet

The 5-layer AlexNet architecture comprises 3 convolutional layers of 64, 128, and 256 filters with 4x4, 3x3, and 2x2 kernel sizes, respectively. These layers are followed by two dense layers of 2048 units, with rectified linear units as activations, and 2x2 max-pooling after the convolutional layers. The final layer is a fully-connected layer with a Softmax

output. In our modified AlexNet architecture, we replace each dense ReLU layer with a layer of (dense) TWTA blocks, and each convolutional layer with a layer of Conv-TWTA blocks; in Table 4.9, we show the number of used kernels / blocks and competing feature maps / units, J, in each modified layer.

Table 4.9: Modified AlexNet architecture parameters.

Layer Type	**Kernels / blocks** ($J = 2$)	**Kernels / blocks** ($J = 4$)	**Kernels / blocks** ($J = 8$)	**Kernels / blocks** ($J = 16$)	**Kernels / blocks** ($J = 32$)
TWTA-Conv	8	4	2	1	1
TWTA-Conv	8	4	2	1	1
TWTA-Conv	16	8	4	2	1
TWTA-Dense	64	32	16	8	4
TWTA-Dense	64	32	16	8	4

LeNet

The LeNet architecture comprises 2 convolutional layers of 20, and 50 feature maps, followed by one feedforward fully connected layer of 500 units, and a final Softmax layer. In our modified LeNet architecture, we replace each of the 2 convolutional layers with one layer of Conv-TWTA blocks; the former retains 2 kernels / blocks of 8 competing feature maps, and the latter 6 kernels / blocks of 8 competing feature maps. The fully connected layer is replaced with a dense TWTA-layer, consisting of 50 blocks of 8 competing units.

4.4 Conclusion

In this Chapter, we presented a novel CIL method called TWTA-CIL. Different from recent works, we devised stochastic TWTA activations that produce task-specific representations that are sparse and stochastic; the goal was to attain greater generalization ability across incrementally learned tasks. From our empirical analysis, the devised task-wise winning ticket formulation appears to successfully identify sparse subnetworks, preserving accuracy much better than the alternatives. Notably, TWTA-CIL prunes the network well, training requires less computational budget, and forgetting is significantly improved. Finally, our method retains its competitive advantage in TIL settings as well.

Chapter 5

Conclusion

In this thesis, we have examined the efficacy of stochastic deep networks with linear competing units in the context of challenging MLe problems, namely model-agnostic Meta-Learning and Continual Learning. In particular:

(i) We addressed ML by considering deep networks with stochastic LWTA activations. Specifically, we proposed a novel paradigm for ML, called StochLWTA-ML, that relies on the concepts of sparse and stochastic learned representations. We aimed to improve the generalization capacity of deep neural networks, specifically in the context of ML. As we have shown, our approach achieved state-of-the-art performance in terms of both predictive accuracy and computational costs, under standard image classification benchmarks in the field. A graphical illustration of the proposed architecture composed of a stochastic LWTA layer is depicted in Fig. 5.1.

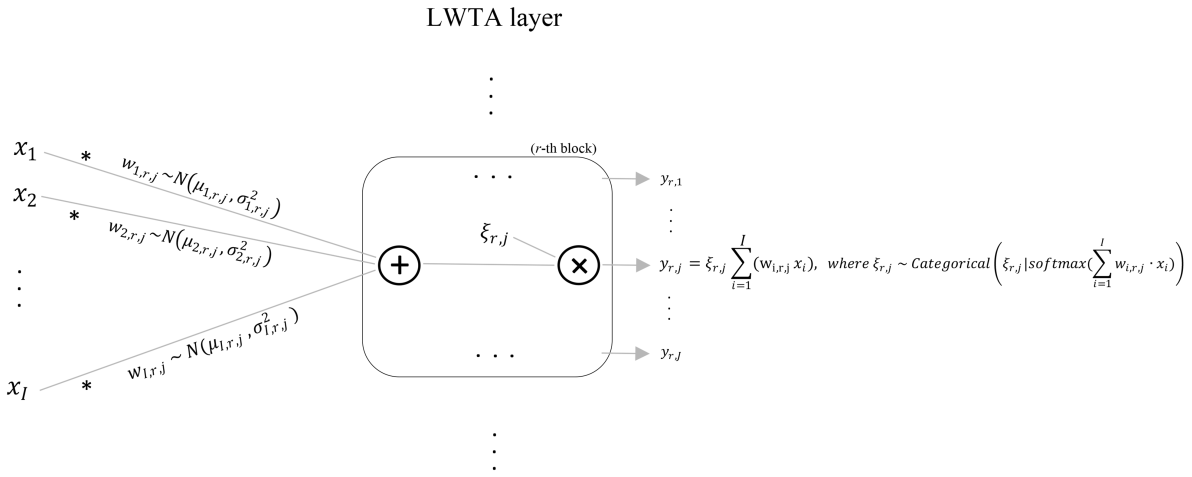


Figure 5.1: A zoomed-in graphical illustration of the r-th block of a stochastic LWTA layer; for a more detailed depiction, see Section 3.4.1.1 of Chapter 3.

In the future, we plan to study the effect of StochLWTA-ML in areas related to ML, such as Reinforcement Learning (Zhu et al. (2020)) and Automated Machine Learning

(AutoML) (He et al. (2021)).

(ii) We attempted to address catastrophic forgetting in class-incremental learning. To this end, we presented a novel CIL paradigm called TWTA-CIL. Different from recent works in this area, we employed stochastic TWTA activations that produce sparse task-specific and stochastic representations with greater generalization ability across tasks, that have been incrementally learned. As we have empirically shown, our method produced state-of-the-art predictive accuracy on few-shot image classification experiments, and imposed a considerably lower computational overhead compared to the current state-of-the-art. Fig. 5.2 illustrates the operation of a TWTA-based network architecture when dealing with task t.

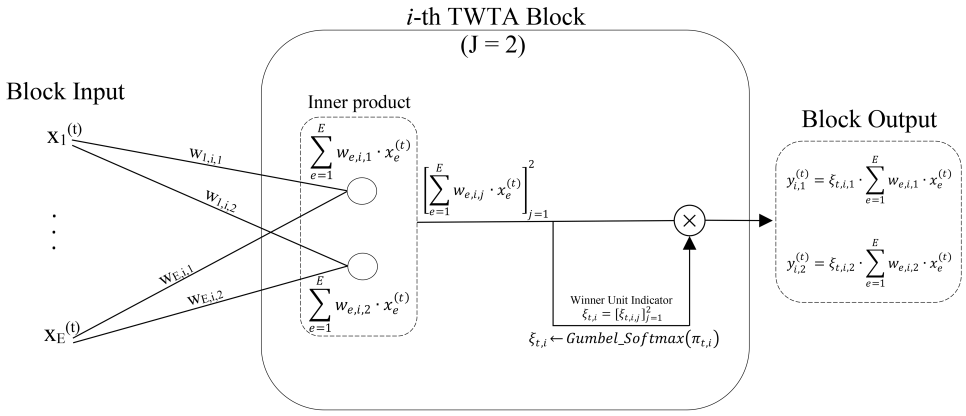


Figure 5.2: A graphical illustration of the i-th block of a proposed TWTA layer; for a more detailed demonstration, see Section 4.2.2 of Chapter 4.

One research direction that we have not considered in this work concerns studying the effect of the proposed TWTA-CIL method in the computer vision domain. Replacing a dense ReLU layer in Transformer networks with a proposed TWTA layer, could produce interesting findings under a Sign-Language Translation benchmark.

Another important research direction is to examine if we could employ our approaches in order to train adversarially-robust DNN's, and test them in either adversarial or data poisoning attack schemes.

Finally, while we focus mainly on image data in this thesis, it would also be interesting to investigate the effectiveness of our proposed approaches on other types of tasks, like speech tasks, including Automatic Speech Recognition, Speech Language Identifica-

tion, Translation and Retrieval. In summary, we believe that the proposed approaches, namely StochLWTA-ML and TWTA-CIL, will be extremely beneficial to the deep learning community.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., and Devin, M. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. pp. 265–283. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. pp. 139–154. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Aljundi, R., Kelchtermans, K., and Tuytelaars, T. Task-free continual learning. In *Conference on Computer Vision and Pattern Recognition*, 2019.

Andersen, P., Gross, G. N., Lomo, T., and Sveen, O. Participation of inhibitory and excitatory interneurones in the control of hippocampal cortical output. *UCLA Forum Med Sci*, 1969.

Antoniou, A., Storkey, A., and Edwards, H. Data augmentation generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition*, 2018.

Antoniou, A., Storkey, A., and Edwards, H. How to train your maml. In *International Conference on Learning Representations*, 2019.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. pp. 4331–4339. In *Neural Information Processing Systems*, 2016.

Belouadah, E. and Popescu, A. Il2m: Class incremental learning with dual memory. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

Black, P., McCormick, R., James, M., and Pedderd, D. Learning how to learn and assessment for learning: a theoretical inquiry. *Research Papers in Education*, 21:119–132, 2006.

Blei, David M., K. A. and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112, 2017.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. In *International Conference on Machine Learning*, 2015.

Bremaud, P. *An Introduction to Probabilistic Modeling.* Springer Science Business Media, 2012.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., and Agarwal, S. Language models are few-shot learners. In *Neural Information Processing Systems*, 2020.

Bui, V. and Chang, L. C. Deep learning architectures for hard character classification. 2016.

Buysse, H. M., Mets, K., and Latré, S. Fast task-adaptation for tasks labeled using natural language in reinforcement learning. 2019. URL https://arxiv.org/abs/1910.04040.

Camgoz, N. C., Hadfield, S., Koller, O., Ney, H., and Bowden, R. Neural sign language translation. In *Conference on Computer Vision and Pattern Recognition*, 2018.

Carpenter, G. A. and Grossberg, S. The art of adaptive pattern recognition by a self-organising neural network. *Computer*, 21(3):77–88, 1988.

Castro, F. M., Marin-Jimenez, M. J., Guil, N., Schmid, C., and Alahari, K. End-to-end incremental learning. pp. 233–248. In *Proceedings of the European Conference on Computer Vision*, 2018.

Castro, R., Kalish, C., Nowak, R., Qian, R., Rogers, T., and Zhu, J. Human active learning. In *Neural Information Processing Systems*, 2008.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. Continual learning with tiny episodic memories. In *ICML Workshop: Multi-Task and Lifelong Reinforcement Learning*, 2019.

Chen, T., Zhang, Z., Liu, S., Chang, S., and Wang, Z. Long live the lottery: The existence of winning tickets in lifelong learning. In *International Conference on Learning Representations*, 2021.

Combalia, M., Hueto, F., Puig, S., Malvehy, J., and Vilaplana, V. Uncertainty estimation in deep neural networks for dermoscopic image classification. pp. 3211–3220. In *Conference on Computer Vision and Pattern Recognition*, 2020.

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. Very deep convolutional networks for text classification. 2017. URL https://arxiv.org/abs/1606.01781.

Cortes, C., Mohri, M., and Rostamizadeh, A. L2 regularization for learning kernels. In *Uncertainty in Artificial Intelligence*, 2012.

Deng, D., Chen, G., Hao, J., Wang, Q., and Heng, P.-A. Flattening sharpness for dynamic gradient projection memory benefits continual learning. In *Neural Information Processing Systems*, 2021.

Douglas, R. J. and Martin, K. A. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci*, 27, 2004.

Dridi, S. Unsupervised learning - a systematic literature review. 2021.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Fast reinforcement learning via slow reinforcement learning. In *International Conference on Learning Representations*, 2017.

Eccles, J. C., Szentagothai, J., and Ito, M. The cerebellum as a neuronal machine. *Springer-Verlag*, 1967.

Edwards, H. and Storkey, A. Towards a neural statistician. 2016. URL https://arxiv.org/abs/1606.02185.

Finn, C. and Levine, S. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. In *Neural Information Processing Systems*, 2018.

Fragoso, Tiago M.and Neto, F. L. Bayesian model averaging: A systematic review and conceptual classification. *Statistical Science*, 2015.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Fürnkranz, J., Chan, P., Craw, S., Sammut, C., Uther, W., Ratnaparkhi, A., Jin, X., Han, J., Yang, Y., Morik, K., Dorigo, M., Birattari, M., Stützle, T., Brazdil, P., Vilalta, R., Giraud-Carrier, C., Soares, C., Rissanen, J., Baxter, R., and De Raedt, L. *Mean Squared Error*. 2010. doi: 10.1007/978-0-387-30164-8_528.

Gal., Y. *Uncertainty in deep learning*. 2016.

Gal., Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 2016.

Gal, Y., Hron, J., and Kendall, A. Concrete dropout. In *Advances in Neural Information Processing Systems*, 2017.

Garcia, V. and Bruna, J. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2017.

Ghiasi, G., Lin, T.-Y., and Le, Q. V. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, 2018.

Gholamalinezhad, H. and Khosravi, H. Pooling methods in deep neural networks, a review. 2020. URL https://arxiv.org/abs/2009.07485.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research 9*, pp. 249–256, 2010.

Goan, E. and Fookes, C. Bayesian neural networks: An introduction and survey. *Case Studies in Applied Bayesian Data Science*, 2020. URL https://arxiv.org/abs/2006.12024.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. pp. 1319–1327. In *International Conference on Machine Learning*, 2013.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations*, 2014.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. E. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2018.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. 2014. URL https://arxiv.org/abs/1410.5401.

Gupta, G., Yadav, K., and Paull, L. La-maml: Look-ahead meta learning for continual learning. In *Neural Information Processing Systems*, 2020.

Géron, A. *Hands-on Machine Learning with scikit-learn, keras, and tensorflow: concepts, tools, and techniques to build intelligent system.* O'Reilly Media, Inc., 2017.

Hamilton, W., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. pp. 1025–1035. In *Neural Information Processing Systems*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. pp. 770–778. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. pp. 770–778. In *Conference on Computer Vision and Pattern Recognition*, 2016b.

He, X., Zhao, K., and Chu, X. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 2021.

Heckerman, D. A tutorial on learning with bayesian networks. *Learning in Graphical Models*, 2020. URL https://arxiv.org/abs/2002.00269.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. 2015. URL https://arxiv.org/abs/1503.02531.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. 2020. URL https://arxiv.org/abs/2004.05439.

Huang, Z., Lam, H., and Zhang, H. Quantifying epistemic uncertainty in deep learning. 2021. URL https://arxiv.org/abs/2110.12122.

Huber, P. J. Robust estimation of a location parameter. *Annals of Statistics*, 53(1):73–101, 1964.

Hung, C.-Y., Tu, C.-H., Wu, C.-E., Chen, C.-H., Chan, Y.-M., and Chen, C.-S. Compacting, picking and growing for unforgetting continual learning. pp. 13669–13679. In *Advances in Neural Information Processing Systems*, 2019.

Hüllermeier, E. and Waegeman, W. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. 2020. URL https://arxiv.org/abs/1910.09457.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Jain, A. K., Mao, J., and Mohiuddin, K. M. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

James, J. *Kullback-Leibler Divergence*. 01 2011. URL https://doi.org/10.1007/978-3-642-04898-2_327.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Kalais, K. and Chatzis, S. Stochastic deep networks with linear competing units for model-agnostic meta-learning. In *International Conference on Machine Learning*, 2022.

Kalais, K. and Chatzis, S. Continual learning via winning subnetworks that arise through stochastic local competition. preprint, 2023.

Kandel, E. R., Schwartz, J. H., and Jessell, T. M. Principles of neural science. 1991.

Kang, H., Mina, R. J. L., Madjid, S. R. H., Yoon, J., Hasegawa-Johnson, M., Hwang, S. J., and Yoo, C. D. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, 2022.

Kang, H., Yoon, J., Madjid, S. R. H., Hwang, S. J., and Yoo, C. D. On the soft-subnetwork for few-shot class incremental learning. In *International Conference on Learning Representations*, 2023.

Kendall, A. and Gal., Y. What uncertainties do we need in bayesian deep learning for computer vision? pp. 5574–5584. In *Neural Information Processing Systems*, 2017.

Kessler, S., Nguyen, V., Zohren, S., and Roberts, S. Hierarchical indian buffet neural networks for bayesian continual learning. In *UAI*, 2021.

Khan, S., Rahmani, H., Shah, S., and Bennamoun, M. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8:1–207, 02 2018.

Khurana, D., Koli, A., Khatter, K., and Singh, S. Natural language processing: state of the art,current trends and challenges. *Multimedia Tools and Applications*, 2022. URL https://doi.org/10.1007/s11042-022-13428-4.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., and Grabska-Barwinska, A. Overcoming catastrophic forgetting in neural networks. volume 114(13), pp. 3521–3526. In *Proceedings of the national academy of sciences*, 2017.

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. volume 2. In *International Conference on Machine Learning deep learning workshop*, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, 2012.

Kühl, N., Goutier, M., Baier, L., Wolff, C., and Martin, D. Human vs. supervised machine learning: Who learns patterns faster? 2020. URL https://arxiv.org/abs/2012.03661.

Lai, Y. A comparison of traditional machine learning and deep learning in image recognition. *Journal of Physics: Conference Series*, 1314, 2019.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. pp. 6402–6413. In *Advances in Neural Information Processing Systems*, 2017.

Lansner, A. Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations. *Trends in neurosciences*, 32(3):178–186, 2009.

Laurent Valentin Jospin, Wray Buntine, F. B. H. L. and Bennamoun., M. Hands-on bayesian neural networks – a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17, 2020.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. 2015. URL http://vision.stanford.edu/teaching/cs231n/reports/2015/pdfs/yle_project.pdf.

LeCun, Y. The mnist database of handwritten digits. 1998.

Lee, C. and Lee, A. Clinical applications of continual learning machine learning. *Lancet Digit Health.*, 2(6), 2021.

Lee, J., Kim, S., Yoon, J., Lee, H. B., Yang, E., and Hwang, S. J. Adaptive network sparsification with dependent variational beta-bernoulli dropout. 2018. URL https://arxiv.org/abs/1805.10896.

Li, J. Recent advances in end-to-end automatic speech recognition. *Transactions on Signal and Information Processing*, 2022a.

Li, K. and Malik, J. Learning to optimize. In *International Conference on Learning Representations*, 2017.

Li, Y. Reinforcement learning in practice: Opportunities and challenges. 2022b. URL https://arxiv.org/abs/2202.11296.

Li, Z. and Hoiem, D. Learning without forgetting. volume 40(12), pp. 2935–2947. In *IEEE transactions on pattern analysis and machine intelligence*, 2017.

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-sgd: Learning to learn quickly for few shot learning. 2017. URL https://arxiv.org/abs/1707.09835.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, 2017.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *The Psychology of Learning and Motivation*, volume 24, pp. 109–164. 1989.

McClure, P., Zheng, C. Y., Kaczmarzyk, J. R., Lee, J. A., Ghosh, S. S., Nielson, D., and Bandettini, P.and Pereira, F. Distributed weight consolidation: a brain segmentation case study. In *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

Minsky, M. L. and Papert, S. A. Perceptrons. *Cambridge, MA: MIT Press*, 1969.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Mukkamala, M. C. and Hein, M. Variants of rmsprop and adagrad with logarithmic regret bounds. In *International Conference on Machine Learning*, 2017.

Munkhdalai, T. and Yu, H. Meta networks. In *International Conference on Machine Learning*, 2017.

Nalisnick, E., Gordon, J., and Hernández-Lobato, J. M. Predictive complexity priors. In *Neural Information Processing Systems*, 2021.

Nasteski, V. An overview of the supervised machine learning methods. *HORIZONS.B*, 4: 51–62, 2017.

Neal, R. M. *Bayesian Learning for Neural Networks*. 1996.

Netzer, Y., Wang, T.and Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Nguyen, C., Do, T.-T., and Carneiro, G. Uncertainty in model-agnostic meta-learning using variational inference. pp. 3090–3100. In *Winter Conference on Applications of Computer Vision*, 2020.

Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. 2018. URL https://arxiv.org/abs/1803.02999.

Opitz, D. and Maclin, R. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11*, pp. 169–198, 1999.

Orbay, A. and Akarun, L. Neural sign language translation by learning tokenization. pp. 222–228. In *15th IEEE International Conference on Automatic Face and Gesture Recognition*, 2020.

O'Shea, K. and Nash, R. An introduction to convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2015.

Otter, D. W., Medina, J. R., and Kalita, J. K. A survey of the usages of deep learning for natural language processing. 2019. URL https://arxiv.org/abs/1807.10854.

Panousis, K., Chatzis, S., Alexos, A., and Theodoridis, S. Local competition and stochasticity for adversarial robustness in deep learning. In *International Conference on Artificial Intelligence and Statistics*, 2021.

Panousis, K. P., Chatzis, S., and Theodoridis, S. Nonparametric bayesian deep networks with local competition. In *International Conference on Machine Learning*, 2019.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. 2013. URL https://arxiv.org/abs/1211.5063.

Patacchiola, M., Turner, J., Crowley, E. J., O'Boyle, M., and Storkey, A. Bayesian meta-learning for the few-shot setting via deep kernels. In *Neural Information Processing Systems*, 2020.

Patel, R. and Patel, S. A comprehensive study of applying convolutional neural network for computer vision. *International Journal of Advanced Science and Technology*, 6: 2161–2174, 01 2020.

Qi, J., Du, J., Siniscalchi, S. M., Ma, X., and Lee, C.-H. On mean absolute error for deep neural network based vector-to-vector regression. 2020. URL https://arxiv.org/abs/2008.07281.

Qian, N. On the momentum term in gradient descent learning algorithms. *Neural Networks : The Official Journal of the International Neural Network Society*, 12:145–151, 1999. URL http://doi.org/10.1016/S0893-6080(98)00116-6.

Rajendra, R., Sabin, M., Impagliazzo, J., Bowers, D., Daniels, M., Hermans, F., Kiesler, N., Kumar, A. N., MacKellar, B., McCauley, R., Nabi, S. W., and Oudshoorn, M. Professional competencies in computing education: Pedagogies and assessment. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education.*, 2021.

Ravi, S. and Beatson, A. Amortized bayesian meta-learning. In *International Conference on Learning Representations*, 2019.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. pp. 2001–2010. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017.

Robbins, H. A stochastic approximation method. In *Annals of Mathematical Statistics*, 2007.

Robbins, H. E. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Robert, C.P., C. G. *Monte Carlo Integration. In: Introducing Monte Carlo Methods with R.* Springer, New York, NY, 2010. URL https://doi.org/10.1007/978-1-4419-1576-4_3.

Robinds, H. and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Ruder, S. An overview of gradient descent optimization algorithms. 2017. URL https://arxiv.org/abs/1609.04747.

Rumelhart, D., Hinton, G., and Williams, R. Learning representations by back-propagating errors. *Nature*, 6088:533–536, 1986.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. 2016. URL https://arxiv.org/abs/1606.04671.

Saha, G., Garg, I., and Roy, K. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, 2016.

Sarma, A., Singh, S., Jiang, H., Zhang, R., Kandemir, M. T., and Das, C. R. Structured in space, randomized in time: Leveraging dropout in rnns for efficient training. In *Advances in Neural Information Processing Systems*, 2021.

Schmidt, R. M. Recurrent neural networks (rnns): A gentle introduction and overview. 2019. URL https://arxiv.org/abs/1912.05911.

Sculley, D., Holt, G., Golovin, D., Davydov, E., and Phillips, T. Hidden technical debt in machine learning systems. In *Neural Information Processing Systems*, 2015.

Shadmehr, R. and Sandro, M.-I. *Biological learning and control: how the brain builds representations, predicts events, and makes decisions.* MIT Press, 2012.

Shyam, P., Gupta, S., and Dukkipati, A. Attentive recurrent comparators. pp. 3173–3181. In *International Conference on Machine Learning*, 2017.

Snell, J., Swersky, K., and Zemel, R. S. Prototypical networks for few-shot learning. 2017. URL https://arxiv.org/abs/1703.05175.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, A., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhube, J. Compete to compute. In *Neural Information Processing Systems*, 2013.

Stefanis, C. Interneuronal mechanisms in the cortex. *UCLA Forum Med Sci*, 1969.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *Conference on Computer Vision and Pattern Recognition*, 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2014.

Thrun, S. A lifelong learning perspective for mobile robot control. *Elsevier*, 1995.

Thrun, S. and Mitchell, T. M. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

Torralba, A., Fergus, R., and Freeman, W. T. 80 million tiny images: A large data set for nonparametric object and scene recognition. volume 30(11), pp. 1958–1970. In *IEEE transactions on pattern analysis and machine intelligence*, 2008.

Triantafillou, E., Zemel, R., and Urtasun, R. Few-shot learning through an information retrieval lens. In *Neural Information Processing Systems*, 2017.

Valdenegro-Toro, M. and Saromo, D. A deeper look into aleatoric and epistemic uncertainty disentanglement. In *Conference on Computer Vision and Pattern Recognition*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning. In *Neural Information Processing Systems*, 2016.

Voskou, A., Panousis, K., Kosmopoulos, D., Metaxas, D., and Chatzis, S. Stochastic transformer networks with linear competing units: Application to end-to-end sl translation. In *International Conference on Computer Vision*, 2021.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. In *International Conference on Machine Learning*, 2016.

Webb, G. *Encyclopedia of Machine Learning and Data Mining*. Springer, Boston, MA., 2017. URL https://doi.org/10.1007/978-1-4899-7687-1_21.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017. URL https://arxiv.org/abs/1708.07747.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network. In *Conference on Computer Vision and Pattern Recognition*, 2015.

Yang, H. and Kwok, J. Efficient variance reduction for meta-learning. In *International Conference on Machine Learning*, 2022.

Yoon, J., Kim, T., Dia, O., Kim, O., Bengio, Y., and Ahn, S. Bayesian model-agnostic meta-learning. In *Neural Information Processing Systems*, 2018.

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V., Nguyen, P., Senior, A., Vanhoucke, V., and Dean, J. On rectified linear units for speech processing. pp. 3517–3521. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. volume 70:3987. In *Proceedings of machine learning research*, 2017.

Zhang, Z. and Sabuncu, M. R. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Conference on Computer Vision and Pattern Recognition*, 2018.

Zhu, H., Yu, J., Gupta, A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., and Levine, S. The ingredients of real-world robotic reinforcement learning. In *International Conference on Learning Representations*, 2020.

Zou, Y. and Lu, X. Gradient-em bayesian meta-learning. In *Neural Information Processing Systems*, 2020.