ABSTRACT

In deep neural networks(DNNs), for a particular sample given only a subset of gates are active i.e., only a subset of gate are helping in producing the corresponding output. So while training as we learn the weights we can also think of learning the sub-networks which might help in learning the network better and might hold valuable information. Ongoing we'll analyze the importance of gates and the active sub-networks for that we use two components of neural network namely Neural Path Feature(NPFs) and Neural Path value(NPVs).In NPFs we encode the on/off state of the gates for a particular input and, for NPVs we store the weight of the Neural network. Despite being famous DNNs are always considered something hard to interpret.So to interpret DNNs using the above theory we disintegrate DNNs into simpler parts namely gating networks and weight networks which is referred as Deep Gate Network(DGN). To extend our understanding further we introduce Deep linearly Gated Network(DLGNs) extended version of DGNs to make DNN more feature interpretable.

In addition, we aim to investigate the capabilities of Deep Generative Networks (DGNs) and Deep Learning Generative Networks (DLGNs) by introducing the challenge of the neural network's inability to learn tasks in sequence, a significant hurdle for systems designed for sequential learning. This phenomenon, known as catastrophic forgetting, occurs when the neural network discards previously learned information upon being trained on new data. Although there are state-of-the-art methods for overcoming catastrophic forgetting, such as data replay, generative replay, multi-model, and multi-head approaches, they do not satisfy the constraints of Strict-Continual Learning (Strict-CL), which prohibits the use of data buffers or replays and negative feedback. Thus, we explore alternative methods to address catastrophic forgetting that adhere to the principles of Strict-CL.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Deep Neural Networks with Rectified Linear Unit(ReLU) activations, ReLU gating provides you with the property that you can write output as the product of pre-activation(q) and gating signal(1/0 for ReLU i.e., 1 for positive pre-activations otherwise 0). For a particular input there is a sub network that remembers the given memory i.e., Holds memory of it and sub-network gate will consists of gate with 1 as the gating value, this implies that only those input that is passed will be able to help in producing the output. The process of learning the neural network involves passing random initialized weights, as weights changes during learning the network so does the gate changes then the whole sub-network changes for different inputs. Thus we can argue that the gates, learned active sub-network will hold potential information about the learned network. So here the following set of goals to be fulfilled are:

1. To study the role of Gates in deep neural networks
2. Understanding the importance and learning of active sub-networks.

The above used methods are still considered as less interpretable because of the multiplication operation of weight matrix and non-linear activation these operations are considered to be entangled this entanglement might help in view of DNNs as it helps to learn the network layer by layer and a linear function will predict the output in the final layer but it doesn't help in interpreting the learnt feature because of the non-linear feature map. To achieve more understanding and increase our interpretability we should think our way out of the generalized layer-layer non linear feature learning and final layer weight learning concept and thus we have to move on to something that we can linearly interpret and learn our feature and perform unentangled way of learning weight in lifted space.

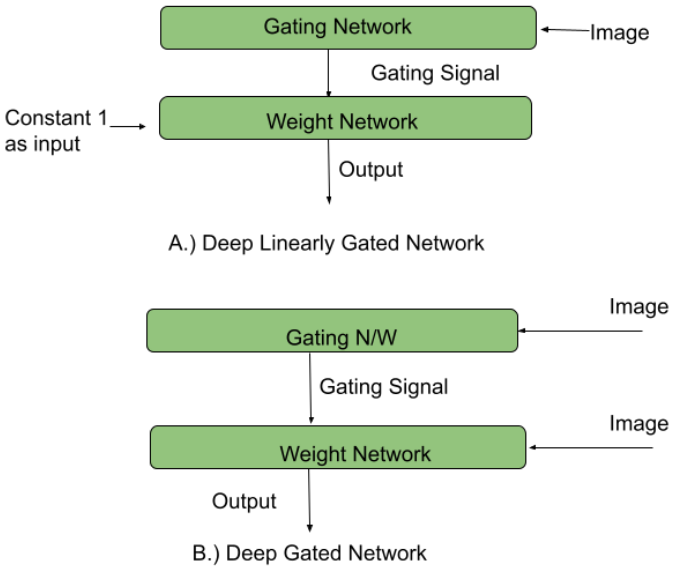

Figure 1.1: DLGN and DGN Working

1.1 DNN ABSTRACTION:

We interpret DNNs with the concept of gates, weights and paths which is :-

1. DNN seems to generate gates (1/0 states) in a layer by layer paradigm

2. The DNN then combine weights linearly with the higher dimension gated input information

thus from the above operation it is clear that DNN doesn't perform linear feature learning and also also performs weight learning in entangled manner.

1.2 DGNS ROLE:

DGNs to realize the importance of gates separate them from the weights. Thus in Deep Gated Network we use gating network that generates gate that is used as a mask for a separate network ,the weight network which combinedly with gate network produces the

output.

1.3 DLGN INTRODUCTION:

Deep Linear Gated Network are more interpretable version of DGNs/DNNs, DLGNs have gate network as deep linear network and weight network is always have constant 1 input. Thus DLGN performs both linear feature learning and linear weight training in lifted space. Hence DLGN will be using both the power of non-linearity and interpretability.

1.4 CONTINUAL LEARNING

When it comes to Machine Learning, Continual Learning refers to the ability to update the prediction model seamlessly, even when dealing with different tasks and data distributions. The goal is to retain the useful knowledge and skills learned over time while adapting to new information. This way, the system can continue to learn and improve without forgetting or interfering with what it has previously learned.

1.4.1 Catastrophic Forgetting

Continual learning of multiple tasks is often plagued by catastrophic forgetting, a phenomenon where acquiring new knowledge disrupts previously learned information. Forgetting is the primary challenge faced by Continual Learning algorithms.

The stability-plasticity dilemma captures the essence of catastrophic forgetting, as a model must be sufficiently plastic to learn new information while also maintaining stability to retain previously acquired knowledge. Traditional neural networks struggle to strike the right balance between stability and plasticity, with plasticity often dominating over stability.

1.4.2 Strict Continual Learning

This definition of continual learning describes a learning approach where neural networks are trained sequentially on various tasks and datasets. The process involves

training a neural network A with parameters α, over time intervals TI_i using task data TD_i, with i representing the current task and N being the total number of tasks. The training paradigm is characterized by several key conditions, including

- **Sequential training:** it refers to the process of training a model on a series of tasks one after the other, where the output of the model on the first task is used as input for the subsequent tasks. In this approach, a single set of model parameters is used for all tasks, and the model is fine-tuned for each new task.
- **No Negative Feedback:** in the sequential training process, the model is only trained on the current task without any negative feedback. This implies that any past or future data cannot be trained with the current task, which can limit the model's ability to generalize across tasks

Apart from above rule some methods change the neural network by introducing different layers for each task, which also violates the strict-CL condition.

1.4.3 Evaluating CL models

We are studying the challenge of continual learning where a single neural network model has to learn a series of tasks one by one, using only data from the current task during training. While many methods have been proposed to address catastrophic forgetting, comparing their effectiveness is difficult due to differences in experimental protocols. One crucial factor that affects difficulty is whether information about task identity is available during testing and if the model needs to identify the task explicitly when such information is not available.

There are 3 scenarios [11] based on the availability of task-id at test time or not. The easiest continual learning scenario is when models are always aware of the task they need to perform. This is known as **task-incremental learning (Task-IL)**. In the second case, known as **domain-incremental learning (Domain-IL)**, models do not have access to task identity information during testing. However, they only need to solve the current task without having to identify which task it is. The third case, referred to as **class-incremental learning (Class-IL)**, requires models to solve all the tasks they have learned so far and also identify the task they are presented with during testing.

CHAPTER 2

LITERATURE REVIEW

2.1 PHASE-I

2.1.1 Roles of gate in DNN

This work introduces us with the abstract and more interpretable view of Deep Neural Network. We consider a DNN with ReLU activation in which the ouptut can be denoted as the product of it's gating signal $G(p)$ and the pre-activation $p \in R$ as $p \cdot G(p)$ training further we see that the weight of DNN remain same across all the inputs, the states of the gates are one that changes across inputs. An active sub-network is formed for each input example, comprising only the gates that are active. These active sub-networks store information for their respective inputs, as only the weights that are contributing to the output are given priority. Training ahead we see that as the weights changes the gates and the active-subnetwork also changes for the corresponding inputs. Thus at the end we claim that the gates and the active subnetwork can potentially hold usefull information.

When using the ReLU activation function, the significance of gates encoded with 1/0 state has always been crucial. This is because such gates enable us to generate the output of a deep neural network by summing up the collective contributions of each individual path, while also allowing us to encode the 1/0 state for gates in a natural way that does not result in the loss of any valuable information. **Paths** contributes by giving product of the gate in the paths , weights in the paths and the signal fed at it's input. For input x and weight θ we can encode weight feature in NPVs ν_θ and encode gating information in NPFs $\phi_{\mathbf{x},\theta}$

The NPV parameter for a path is obtained by multiplying the weights in that path, while the NPF parameter is calculated by multiplying the input signal with the gates along that path.

We can represent the output like:

$$\hat{y}_\theta(x) = (\phi_{x,\theta}, \nu_\theta)$$

Input Layer	$k_{x,\theta}(0)$	x
pre-activation input	$l_{x,\theta}(a,c)$	$\sum_b \theta(a,b,c) \cdot k_{x,\theta}(b,c-1)$
Gating Values	$GV_{x,\theta}(b,c)$	$1_{(l_x\theta(a,c)>0)}$
Hidden Layer output	$k_{x,\theta}(a,c)$	$l_{x,\theta}(a,c) \cdot GV_{x,\theta}(b,c)$
Final Output	$\hat{y}_\theta(x)$	$\sum_{b\in\beta} \theta(a,b,c) \cdot k_{x,\theta}(b,c-1)$

Table 2.1: Input-Output-Expressions

Further to understand the importance of gates the next step is to separate gates(NPF) from weights(NPV) this configuration is achieved by DGN (Deep gated network) having 2 network identical in architecture namely a feature network that hold gating information and the a value/weight network that hold the weighting information. NPFs are parameterized by $\Theta^f \in R^{net}$ and NPVs are parameterised by $\Theta^v \in R^{net}$. Let $\theta^{DGN} = (\theta^f, \theta^v) \in R^{d^f_{net}} + R^{d^v_{net}}$ denotes the combined DGN parameters.

2.1.2 Regime of DGN

We have 2 DGN Configuration by configuring (i) the initialization of Θ^f_o and Θ^v_o (ii) The trainability of Θ^f_o and Θ^v_o. The 2 regime are as follows:-

DLNPF(Decoupled Learning of NPF): Here Θ^f is trainable, used Soft-ReLU with $\beta > 0$, Θ^f_o and Θ^v_o are independent and initialised at random.

FLNPF(Fixed Learnt NPF): Here Θ^f is non trainable, Here Θ^f_o is copied from pre-trained DNN ,Used Soft-ReLU with $\beta > 0$, Θ^v_o initialised at random.

2.1.3 Interpretable DNN:

The presented research work introduces us to Deep Linearly Gate Networks (DLGNs), which serve as an alternative to DGNs. DLGNs learn fundamental linear mappings of features, which are then employed to elevate the input to a more complex, higher-dimensional representation. This, in turn, facilitates the learning of a linear function through the utilization of weights. One of the problems with Deep Neural Networks (DNNs) is that they carry out linear computations at each layer. This results in the entanglement of the product of weight matrix and non-linear activation. Despite the success of this conventional manner of layer-by-layer feature learning and predicting output by linear function in the final layer the drawback of this way it does not shows the simple view/interpretation of learnt feature due to the entangled non-linear nature of feature map. The work majorly discuss how much of the non-linear entanglement can be disentangled without affecting the performance.We can completely disentangle by removing the non-linearity and making the entire network linear and making the network completely feature interpretable. This way of dealing with non-linearity arises with questions on performance issue. For achieving feature interpretability, one needs to make the features linear functions of the input. The role of the weights is to linearly combine the learnt features. However, doing this will lead to learning a overall linear function which is not so beneficial in general. Thus,to lifts the learnt linear features into a high dimensional space where the weights will act linearly a lifting operation is needed . In DNNs, this happens implicitly and the lifting is entangled and thus leading to difficulty in interpreting the learnt feature mapping . Our goal in this paper to develop a novel architecture which performs the following two functions in an explicit and disentangled manner: (i) F1-linear feature learning and (ii) F2-linear weight learning in the lifted space.This approach will helps both in feature interpretability and non-linearity of the network.To achieve this goal in which performs linear feature learning and linear weight learning in lifted space i.e., F1 and F2 this works proposes the Deep Linearly Gated Network where we completely remove the non-linearity by removing the ReLU

activation function from the gating network and thus performs F1 and we provide constant 1 input to the weight network thereby performing function F2

2.2 PHASE-II

Continual Learning(CL) refers to the way of learning where different data/tasks are presented sequentially to the model similar to the what human encounters in real world. But due to concurrent/distributed way of learning Neural Networks have been shown to to forget catastrophically, this term catastrophic forgetting is termed as the inability to predict correctly on the old trained data when new data is trained on the model.

Some of the conventional state of the art CL Methods are:-

- **Data Replay** involves using previously learned or future task data to train the neural network.
- **Multi-Heads** refers to using a separate output layer for each task in the neural network.
- **Generative Replay** involves using generative models to create additional training samples for the current task.
- **Multi-Models** involves using additional neural network models, in addition to the original model, to aid in the learning of new tasks

These State of the art methods surely solve the problem of catastrophic forgetting but doing so they violated the condition of Strict-CL stated above. Moving further we introduce you to methods which follows Strict-CL paradigms.

2.2.1 Elastic Weight Consolidation

The process of synaptic consolidation in the brain allows for ongoing learning by decreasing the flexibility of synapses that are crucial to previously acquired skills. A comparable technique is employed in artificial neural networks through an algorithm that restricts significant parameters from deviating too far from their previous values. This

section elucidates the reasons why a new task solution is anticipated to be found near an old one, the methods used to impose the constraint, and the approach taken to identify which parameters are significant.

Elastic Weight Consolidation (EWC)[12] technique, which is designed to protect the performance of a previously learned task A while learning a new task B. EWC achieves this by constraining the parameters to stay in a region of low error for task A, centered around θ^*_A. This constraint is implemented as a quadratic penalty and can be visualized as a spring anchoring the parameters to the previous solution. The stiffness of this "spring" should not be the same for all parameters but rather greater for those that are most important for performance during task A. The over-parameterization of the neural network makes it likely that there is a solution for task B θ^*_Bthat is close to the solution for task θ^*_A

Adopting a probabilistic perspective is helpful to rationalize the choice of constraint and identify the most crucial weights for a given task in neural network training. In this approach, the optimization of parameters corresponds to the search for their most likely values, given some data D. The conditional probability $p(\theta|D)$ can be calculated by utilizing Bayes' rule, which involves the prior probability of the parameters $p(\theta)$ and the probability of the data $p(D|\theta)$.

$$logP(\theta_0 \mid D_0) = logP(D_0 \mid \theta_0) + logP(\theta_0) - logP(D_0) \qquad (1)$$

In this method, we approximate the posterior distribution of the parameters as a Gaussian distribution, where the mean is determined by the parameters θ^*_A, and the diagonal precision is derived from the diagonal of the Fisher information matrix F. The Fisher information matrix has three essential properties, namely, (a) it is equivalent to the second derivative of the loss close to the minimum, (b) it can be calculated using only first-order derivatives, making it feasible for large models, and (c) it is positively semi-definite. It is worth noting that this technique is similar to expectation propagation, where each subtask is viewed as a factor of the posterior. Given this approximation, the function L that we

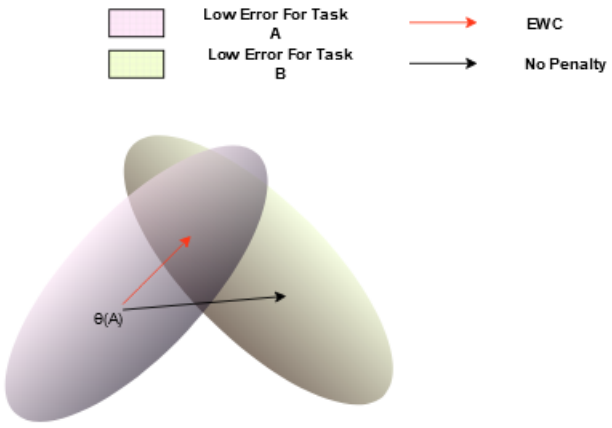

Figure 2.1: EWC allows retention of knowledge learned from task A while training on task B. It accomplishes this by identifying the parameters responsible for good performance on task A (pink parameter region) and on task B (green parameter region) in a parameter space. EWC then uses this information to update the parameters to a new solution for task B (red arrow) that does not cause significant deterioration of performance on task A.

minimize in EWC is:

$$L(\theta) = L_B(\theta) + \sum \frac{\lambda}{2} \cdot F_i(\theta_i - \theta^*_{A,i})^2 \qquad (2)$$

The variable $L_B(\theta)$ represents the loss exclusively for task B, while λ determines the relative significance of the previous task in comparison to the current one, and i serves as an identifier for each parameter.

2.2.2 Relevance Mapping Networks

The Relevance Mapping technique [8], which draws inspiration from the Optimal Overlap Hypothesis, seeks to attain an optimal representational overlap. This enables distinct tasks to utilize separate network parameters, while simultaneously enabling comparable tasks to possess a shared representational overlap. It is worth noting that this method accomplishes independence between network weights for different successive tasks, without the need for data replay.

Probabilistic view of Relevance Mapping

If we could study the importance of each node weight we can select non essential and apply zero mask to them without changing them and successfully learn the ground-reality/truth. let $M_P = (M_{P1}, M_{P2})$ be the set of mappings that explicitly represent the neuron-to -neuron connections of the network where $M_{P1} \in (0,1)^{d_2 X d_1}$ and $M_{P2} \in (0,1)^{d_3 X d_2}$. Relevance Mapping holds the idea of separability and distribution for the posteriors we learnt for the sequential tasks. The separation is given by.

$$P(\theta_1, M_{P_1} \mid D_1) \propto P(D_1 \mid \theta_{M_{P_1}}) P(\theta_{M_{P_1}}) \qquad (3)$$

The above equation is similar to the eq (1) above with introduced relevance mapping under the algorithm conditions. $\theta_{M_{P_1}}$ is only a subset of θ for which mask is non-zero. For the second task

$$P(\theta_{1:2}, M_{P_2} \mid D_{1:2}) \propto P(D_2 \mid \theta_{M_{P_1}}) P(\theta_1, M_{P_1} \mid D_1) \qquad (4)$$

The presence of independent relevance mappings causes the second term on the right to have no impact on optimizing the second task. This disconnects the parameter set θ_{MP_1} from further modification and results in the next task having a slightly restricted prior distribution, which we can denote as $\theta_2^{''}$. Despite this, the parameter set θ_{MP_1} remains accessible to the second task.

There are three possible scenarios that can occur with respect to the optimized parameters, denoted by k, where $M_{P_2}^k$ represents the optimized parameter for the second task and $M_{P_1}^k$ represents the optimized parameter for the first task.

- The first scenario is when $M_{P_2}^k$ is equal to $M_{P_1}^k$, which means that the value of $\theta_{M_{P_1}}^k$ is the same as the value of $\theta_{M_{P_2}}^k$.
- The second scenario is when $M_{P_1}^k$ is equal to 0 and $M_{P_1}^k$ is equal to 1.

- The third scenario is when $M^k_{P_2}$ is equal to 1 and $M^k_{P_1}$ is equal to 0, which results in the intersection of $\theta^k_{M_{P_1}}$ and $\theta^k_{M_{P_2}}$ being an empty set.

RMNs are capable of handling all three of these scenarios thanks to the Optimal Overlap (O2) hypothesis. When optimizing over n sequential tasks, eq (2) takes on a different form.

$$P(\theta_{1:n}, M_P \mid D_{1:n}) \propto \prod_{i=1}^{n} P(D_i \mid \theta_{M_{P_i}}) P(\theta_i^{''}) \tag{5}$$

In RMNs, the objective is not to eliminate representational overlap or generalize to all sequential tasks, but instead to leverage the over-parameterization property of ANNs and learn an optimal representational overlap for all tasks in the weight space. This approach supports the Optimal Overlap Hypothesis. There are no restrictions on the maps MP to minimize overlap with each other or a global loss function that takes into account the loss of individual tasks.

Each map MP for a given task plays a role in defining a subset of the final weight mapping of the ANNs

Below Algorithm 1, which incorporates Relevance Mapping, is used in traditional Supervised Continual Learning experiments to assess the effectiveness of Catastrophic Forgetting alleviation.

Algorithm 1: RMN Supervised Continual Learning

Input: The input for the algorithm consists of the following: data x and corresponding ground truth labels y for n tasks, a prune parameter μ, and task labels i paired with all x.

Given : The algorithm requires two inputs: parameters W and initialized relevance mappings M_P.

for *each task i* **do**
    $f(x_i; W, M_{P_i}) \rightarrow \hat{y}_i = \sigma((W \cdot MP_i) \cdot x_i)$
    Compute Loss: $L(\hat{y}_i, y_i)$
    Optional: Add Sparsity Loss: $L(\hat{y}_i, y_i) + (MP_i)_{l_0}$
    Backpropagate and optimize
    Prune $M_{P_i} \leq \mu$ only
    Stabilize parameters in f where $M_{P_i} = 1$
end
for *data x and ground-truth task label i* **do**
    Output: $f(x, i; W)$
end

2.2.3 Incremental Moment Matching (IMM)

The approach of incremental moment matching (IMM) [7]involves matching the moments of posterior distributions incrementally. In our study, we make use of a Gaussian distribution to estimate the posterior distribution of parameters. Our goal is to determine the optimal parameter $(\mu^*_{1:k})$ and $(\sum^*_{1:k})$ for the Gaussian approximation function $q_{1:k}$ based on the posterior parameter $(\mu_k, \sum_k)$ for each task k out of K sequential tasks.

In the following, we will describe two suggested moment matching techniques aimed at facilitating the continual learning of contemporary deep neural networks. These algorithms produce distinct moments of a Gaussian distribution, each utilizing a different objective function to accomplish this for a common dataset.

Mean-based Incremental Moment Matching (mean-IMM)

Mean-IMM is a technique that in each layer combines the parameters of two networks by taking a weighted average using mixing ratios α_k, where the sum of all α_k is equal to 1.

The goal of Mean-IMM is to minimize the local KL-divergence or the weighted sum of KL-divergences between each q_k and $q_{1:k}$. This can be understood as minimizing the difference between the probability distributions of the two networks, weighted by the mixing ratios, at each layer

$$\mu^*_{1:K}, \Sigma^*_{1:K} = \arg\min_{\mu_{1:K}, \Sigma_{1:K}} \sum_{k=1}^{K} \alpha_k \mathrm{KL}(q_k || q_{1:K}) \quad (6)$$

$$\mu^*_{1:K} = \sum_{k=1}^{K} \alpha_k \mu_k \quad (7)$$

$$\Sigma^*_{1:K} = \sum_{k=1}^{K} \alpha_k \left(\Sigma_k + (\mu_k - \mu^*_{1:K})(\mu_k - \mu^*_{1:K})^T\right) \quad (8)$$

Mode-based Incremental Moment Matching (mode-IMM)

Mode-IMM is a modified version of mean-IMM that incorporates the covariance information of the posterior Gaussian distribution. While a simple weighted average of the mean vectors of two Gaussian distributions may not necessarily yield the mode of the resulting mixture of Gaussians (MoG), the maximum of the distribution is typically of greater interest in discriminative learning. To address this, mode-IMM uses Laplacian approximation to approximate the MoG, whereby the logarithm of the function is expressed via a Taylor expansion.

$$\log q_{1:K} \approx \sum_{k=1}^{K} \alpha_k \log q_k + C = -\frac{1}{2}\theta^T \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1}\right) \theta + \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \mu_k\right)^T \theta + C' \quad (9)$$

$$\mu^*_{1:K} = \Sigma^*_{1:K} \cdot \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1} \mu_k\right) \quad (10)$$

$$\Sigma^*_{1:K} = \left(\sum_{k=1}^{K} \alpha_k \Sigma_k^{-1}\right)^{-1} \quad (11)$$

In this context, we make the assumption that the covariance matrices are diagonal, meaning that there are no correlations among the parameters. This assumption is beneficial since it reduces the number of parameters needed for each covariance matrix from $o(D^2)$ to $O(D)$, where D is parameter's dimension. To calculate the covariance,

we adopt the approach of using the inverse of a Fisher information matrix. The basic concept behind this approximation is that the square of the parameter gradients can serve as a reliable indicator of their precision, which is the inverse of their variance. The Fisher information matrix for the $k-th$ task, F_k, is defined as follows:

$$F_k = \mathbb{E}[\frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k) \cdot \frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k)^T] \quad (11)$$

The probability distribution used in the above equation follows from the assumptions that x is drawn from the empirical distribution of X_k, and the observed output $\tilde{y}$ is drawn from the distribution $\mathrm{p}p(y|x, \mu_k)^T]$ given input x and parameter μ_k.

2.2.4 Hard Attention

Hard attention to the task (HAT) [10] , which enables a neural network to learn new tasks while preserving the knowledge of previously learned tasks. HAT achieves this by learning almost-binary attention vectors using gated task embedding and back propagation. These attention vectors are used to create a mask that constrains updates to the network's weights during the learning of new tasks. The mask allows a portion of the weights to remain fixed, while the rest adapt to the new task. The authors demonstrate the effectiveness of HAT in image classification tasks. The authors use a layer-wise attention mechanism (as shown in Figure) to condition the neural network to the current task t. They do this by element-wise multiplication of the output of the units2 of layer L, H_L, with $H_l^{`} = a_l^t \cdot H_l$. However, unlike typical attention mechanisms that generate a probability distribution, the attention mechanism used in this approach is a gated version of a single-layer task embedding em_l^t,

$$a_l^t = \sigma(S^{`} \cdot em_l^t) \quad (12)$$

The gate function, $\sigma(x)$, ranges between 0 and 1, and the positive scaling parameter is denoted as $S^{`}$.

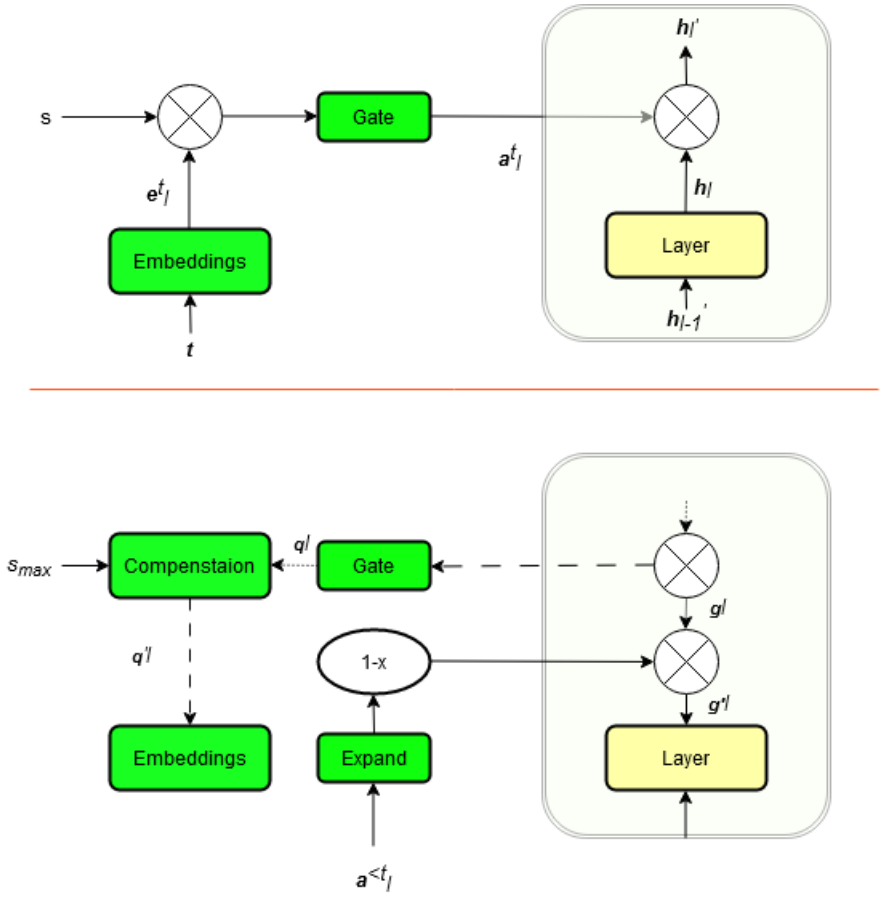

Figure 2.2: The proposed approach is illustrated in a schematic diagram that shows the forward pass (top) and backward pass (bottom).

The gating mechanism presented in Equation 1 aims to create hard attention masks, which can be binary and act as "inhibitory synapses." These masks can activate or deactivate the output of units in every layer, thus dynamically creating and destroying paths across layers. This capability allows for paths to be preserved when learning a new task. With HAT, individual-unit paths are learned and dimensioned automatically, affecting the individual layer weights. The authors note that HAT achieves this without the need for a separate path learning stage using genetic algorithms. Instead, the network and paths are learned together using backpropagation and stochastic gradient descent.

2.2.5 DLGN for Continual Learning

Deep Linear Gated Networks (DLGNs) are a variant of Deep Neural Networks (DNNs)/Deep Gated Networks (DGNs) that are more interpretable. DLGNs learn basic linear mappings of features, which are then used to enhance the input to a more complex, higher-dimensional representation. The gate network in DLGNs serves as a deep linear network, and the weight network always has a constant input of 1, making it easier to learn a linear function using weights. Therefore, DLGNs can perform both linear feature learning and linear weight training in a lifted space, providing both the power of non-linearity and interpretability. To alleviate Catastrophic Forgetting through DLGN we can use the properties of the model where it seperate the weights and the gate through NPVs ν_θ and NPFs $\phi_{\mathbf{x},\theta}$, for similar tasks the change in NPVs upon encountering multiple task can be minimized. we try to incorporate following ways to see the how the NPVs and NPFs behaves in sequential task training system.

- **First**, we try to have the hidden layers of a neural network the NPVs can be shared across all tasks, while still having multiple task-specific output layers. Also Known as hard parameter sharing ,The use of hard parameter sharing in multi-task learning reduces the risk of overfitting, with a study showing that overfitting of shared parameters is significantly smaller (by an order of N, where N is the number of tasks) than overfitting of task-specific parameters such as output layers.

- **Second**, used different learning rate for layers and neurons throughout the network. To preserve the general features learned by layers closer to the input, it is advisable to minimize the extent of changes made to them. For this reason, a very low learning rate (LR) is assigned to these early layers. The LR is then gradually increased per layer as we move deeper into the model. This approach ensures that the lower layers retain their general feature learning while the deeper layers adapt to more specific features of the task at hand.

- **Third** we try to have task specific layers by freezing or slow down learning significantly for the layer or neurons that we would like to reserve for that task. Despite its effectiveness, the approach of assigning a pre-defined structure for parameter sharing still has limitations. While it may be suitable for established computer vision problems, it can be prone to errors when applied to novel or less-studied tasks. This is because the pre-defined structure may not be optimal for the new task, potentially resulting in suboptimal performance or additional computational overhead in adapting the structure to the new task.

- **Fourth** We have tried to use Elastic Weight Consolidation over DLGNs. The Elastic Weight Consolidation (EWC) technique is specifically designed to preserve the performance of a previously learned task A while learning a new task B. Its main mechanism involves constraining the parameters to remain in a low-error region associated with task A, centered around the optimal parameter configuration denoted as (θ_A^*). This constraint is implemented as a quadratic penalty and can be conceptualized as a spring that anchors the parameters to the previous solution. Importantly, the stiffness of this "spring" is not uniform across all parameters but instead is higher for those that significantly influence performance in task A. Given the neural network's over-parameterization, it is highly probable that a solution for task B (θ_B^*) can be found close to the solution for task A(θ_A^*) .

CHAPTER 3

EXPERIMENTS/RESULTS PHASE-I

The work in [1] performs experiments in 2 phases first it incorporated the DGN based DNN architecture mainly using two different optimizers and two different datasets, second it incorporated the DLGN based DNN architecture mainly experimented variations of the infamous VGG-16 DNN model on two different datasets. Below we'll see the different models/variations that it followed for experiments for the above 2 phases and the main purpose of doing these experiments.

3.1 DEEP GATE NETWORK(DGN)

In this phase of experiments we investigate the role of gates and justify that learning of active sub-network improves generalization. We prove this by showing that gate encoded NPFs learning improves generalization.For this purpose it uses DGN framework.

3.1.1 Datasets:

We have used standard Datasets MINST and CIFAR-10.

3.1.2 Architecture:

For **CIFAR** we used **VCONV** a simple vanilla convolution neural network given as follows: Input Layer (32,32,3) followed by 4 convolution layers with stride (3,3) each and layer size 64,64,128,128 respectively. Then we have a fully connected(FC) layer with 256 size and a softmax activation for final prediction. For **MINST** we used **Fully Connected(FC)** architecture with $w = 128$ and $d = 5$.

3.1.3 Optimisers:

Here we have used Adam and SGD (Stochastic Gradient Descent) . For Adam we have used constant step size of $3e^{-4}$ and for SGD we used constant step sizes in the set

0.1,0.01,0.001 and use the best by experimenting. Batch size is same for both as 32.

3.1.4 DGN Result

DGN					
Architecture	Optimizer	Dataset	DLNPF	FLNPF	ReLU
FC	ADAM	MINST	97.05 ± 0.25	96.5 ± 0.37	97.85 ± 0.05
FC	SGD	MINST	96.25 ± 0.15	96.02 ± 0.10	96.29 ± 0.01
VCONV	ADAM	CIFAR-10	68.24 ± 0.02	71.06 ± 0.27	71.85 ± 0.86
VCONV	SGD	CIFAR-10	62.41 ± 0.32	63.12 ± 0.53	66.93 ± 0.23

Table 3.1: Results of DGN.

The table displays the average test accuracy of various NPFs learning settings, based on the best results obtained from each run. The models were trained until they reached a near-perfect accuracy of 100 , and the reported values are an average of five separate runs.

3.2 DEEP LINEARLY GATED NETWORK

For Deep Linearly Gated Network(DLGN) we have used VGG-16 architecture and further used its variations for the experiments.This Model DLGN and it's corresponding variants for each of the cases exploits lifitng and outperform the conventional model at every stage. First we'll see VGG-16's architecture and the we'll list the specification of each variation used.

3.2.1 VGG-16

Visual Geometry Group It is a CNN with many layers, VGG-16 have 16 layers, It has 13 convolution layer and 3 FC (Fully Connected Layer) Have 5 blocks of convolution layer Each of the 5 blocks have (2,2,3,3,3) No. of filters, Each of the layer have filter size (64,128,256,512,512) in their respective blocks , Three Fully Connected Layer, first 2 are of size 4096 and the 3rd has 1000 size. Now we'll study the different variation of VGG-16 architecture used with/without DLGN.

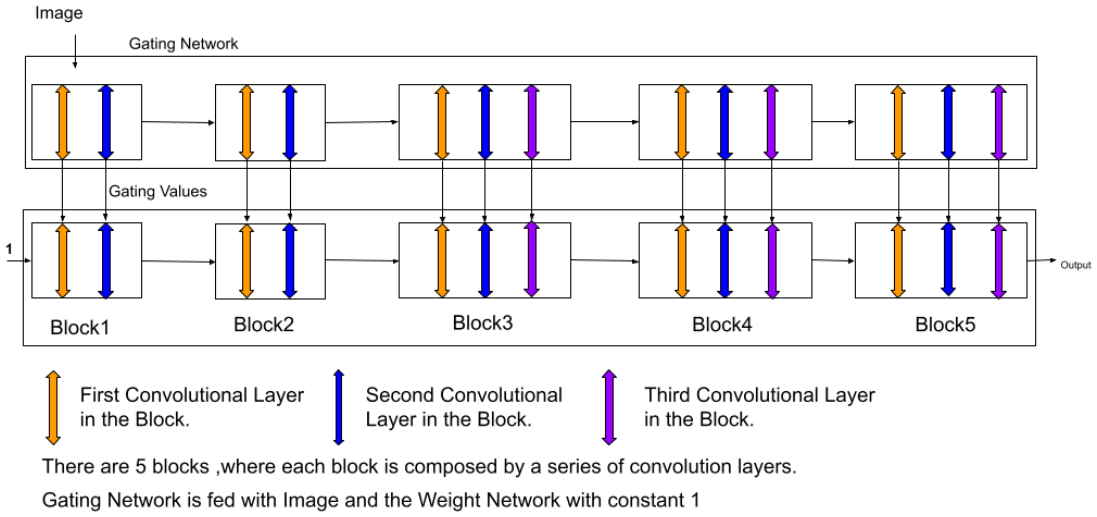

Figure 3.1: VGG-DLGN Block Dig

3.2.2 VGG-MAX

Same architecture as of VGG, Identity Function is used instead of ReLU. 13 hidden convolution layer. Max-Pooling is used at the end of each 5 blocks At last we have 3 FC Dense Layer Last we have Softmax as activation

3.2.3 VGG-DLGN

But we use 2 separate VGGs for Weight and Gates , Weight network is Called Neural Path Value (NPV), Gates network is Called Neural Path Feature (NPF), Identity Function is used instead of ReLU. NPVs are provided with constant 1 as input, NPFs are provided with actual image as input, 13 hidden convolution layer. Global Average Pooling is used at the end of 5 blocks, at last we have 3 FC Dense Layer at last we have Softmax activation.

3.2.4 VGG-DLGN-MAX

Same architecture as of VGG, identity Function is used instead of ReLU. 13 hidden convolution layer. Global Average Pooling is used at the end of each 5 blocks, at last we

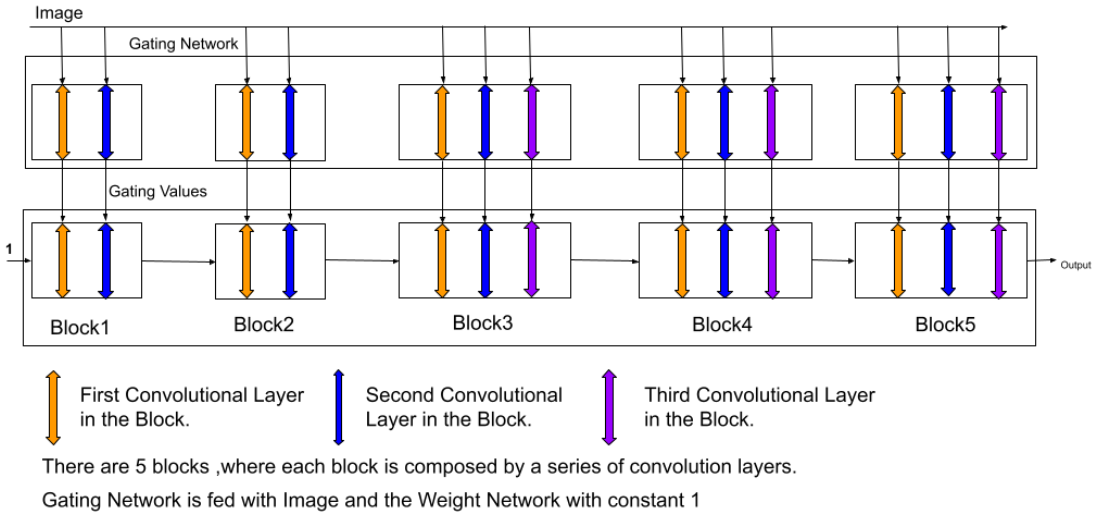

Figure 3.2: VGG-DLGN-SHALLOW Block Dig

have 3 FC Dense Layer, lastly we have Softmax activation. **VGG-DLGN-Shallow** Uses 2 separate VGGs for Weight and Gates Weight Network is Called Neural Path Value (NPV) Gates Network is Called Neural Path Feature (NPF). Identity Function is used instead of ReLU. NPVs are provided with constant 1 as input in each layer of in each block NPFs are provided with actual image as input in each layer of each block 13 hidden convolution layer. Global Average Pooling is used at the end of 5 blocks At last we have 3 FC Dense Layer Lastly we have Softmax as activation

3.2.5 DLGN Results

DLGN		
Model	Dataset	Accuracy
DLNG	CIFAR-10	85.15 ± 0.2
DLGN-MAX	CIFAR-10	91.07 ± 0.23
DLGN-SHALLOW	CIFAR-10	82.57 ± 0.02
DLGN	CIFAR-100	59.41 ± 0.37
DLGN-MAX	CIFAR-100	67.71 ± 0.16
DLGN-SHALLOW	CIFAR-100	56.81 ± 0.06

Table 3.2: Results of DLGN

3.3 EXPERIMENTS SPECIFICATION

We utilized various computing platforms, including Google Collab, Kaggle, and an HPCE Cluster to train our models. Specifically, for all VGG-16 based architectures, we employed the SGD optimizer with a momentum of 0.9. The learning rate was set to 0.01 for the first 0-400 iterations, 0.1 for iterations 400-32000, 0.01 for iterations 32000-48000, and 0.001 for iterations 48000-64000. A batch size of 128 was used, and each model was trained for 32 epochs.

CHAPTER 4

EXPERIMENTS/RESULTS PHASE-II

Now we perform experiments to alleviate our understanding of Continual Learning and catastrophic forgetting. First we see how a network which is capable of sequential learning shows the phenomena of catastrophic forgetting followed by some state of the art methods that goes against the idea of Strict-CL paradigms. Following previous ways to implement continual learning we perform experiments that minimize catastrophic forgetting while following the rule of continual learning.

4.1 OBSERVING CATASTROPHIC FORGETTING

An experiment on a simple CNN to observe the phenomena of Catastrophic Forgetting that shows us the loss of information while exposed to new tasks

4.1.1 Experiment details

- **Dataset:** used permuted-MNIST and created 5 task of 10 class each being different permutation of MNIST.
- **Architecture:** we have used a CNN with 2 convolution layers of size $(1, 10), (10, 20)$ respectively followed by a maxpool layer and 2 fully connected layer for 10 class classification using relu activation.

4.1.2 results

	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	98.25	-	-	-	-
Train 2	30.36	86.47	-	-	-
Train 3	27.30	29.02	88.22	-	-
Train 4	22.36	21.26	32.38	81.36	-
Train 5	16.60	17.91	17.89	35.23	86.32

Table 4.1: Observing Catastrophic forgetting

- **Table Reading: Train i** simply states that **Task i** is being trained after training **Task 1** to **Task [i-1]** in the increasing label sequence. And then Tested all the above task in the same order from **Task 1** to **Task i** .

- **Observation:** The above table clearly depicts the inability of neural network to learn multiple task sequentially. As soon as current task is trained completely the learned previous tasks are being overridden.

4.2 DATA-REPLAY FOR CONTINUAL LEARNING

Experimenting one of the STOA method for overcoming Catastrophic Forgetting. Data replay is a method for providing past data with the current data for training.

4.2.1 Experiment details

- **Dataset:** used Split-MNIST and created 5 task of 2 class each.

- **Architecture:** To perform a two-class classification task with the ReLU activation function, we employed a Convolutional Neural Network (CNN) architecture consisting of two convolution layers of dimensions $(1, 10)$ and $(10, 20)$, respectively. This was followed by a max-pooling layer and two fully connected layers. Used SGD as an optimizer and a learning rate of 0.001

4.2.2 Results

	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	94.52	-	-	-	-
Train 2	96.31	85.39	-	-	-
Train 3	97.52	90.36	88.22	-	-
Train 4	97.21	94.12	90.27	89.73	-
Train 5	97.63	95.21	94.88	94.12	92.01

Table 4.2: Continual Learning through Data Replay

- **Table Reading: Train i** simply states that **Task i** is being trained after training **Task 1** to **Task [i-1]** in the increasing label sequence. And then Tested all the above task in the same order from **Task 1** to **Task i** .

- **Observation:** As can be seen in the table Data Replay does works for the Continual learning but it violates the strict-CL policies i.e., trained data should never be feed again.

4.3 ELASTIC WEIGHT CONSOLIDATION (EWC)

Here [12] have demonstrated that it's feasible to develop neural networks that can retain their proficiency in tasks they haven't encountered for extended periods. Our strategy involves preserving knowledge of past tasks by deliberately reducing the rate of learning for the weights that are critical for those tasks.

4.3.1 Experiment details

- **Dataset:** used permuted-MNIST and created 5 task of 10 class each being different permutation of MNIST.
- **Architecture:** we have used a CNN with 2 convolution layers of size $(1, 10), (10, 20)$ respectively followed by a maxpool layer and 2 fully connected layer for 10 class classification using relu activation.

4.3.2 Results

EWC					
	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	98.91	-	-	-	-
Train 2	95.31	98.23	-	-	-
Train 3	95.22	96.36	97.29	-	-
Train 4	95.01	95.63	96.87	98.95	-
Train 5	94.91	95.08	95.48	95.23	97.12

Table 4.3: Elastic Weight Consolidation for Permuted Mnist

- **Table Reading: Train i** simply states that **Task i** is being trained after training **Task 1** to **Task [i-1]** in the increasing label sequence. And then Tested all the above task in the same order from **Task 1** to **Task i** .
- **Observation:** By selectively reducing the plasticity of weights, EWC enables the retention of knowledge acquired from previous tasks during new learning, thus preventing catastrophic forgetting of earlier skills. This approach bears similarities to neurobiological models of synaptic consolidation.

4.4 RELEVANCE MAPPING NETWORK

The work in [8] implements Relevance Mapping Network to overcome catastrophic forgetting using the concept of optimal overlap hypothesis. It doesn't rely on reusing

previously seen data, but rather focuses on ensuring that the network weights used for different tasks are independent of each other thus following the rule of strict-CL.

4.4.1 Experimental details

- **Dataset:** used permuted-MNIST dataset and partitioned it into 5 tasks, each containing 10 classes that corresponded to distinct permutations of the original MNIST dataset. You also employed the Split-MNIST dataset and created 5 tasks, with each task consisting of 2 classes. Additionally,used the CIFAR-100 dataset and divided it into 20 tasks, each containing 5 classes

- **Architecture:** Used multi layer preceptron(MLP) for both P-MNIST and S-MNIST that have 3 fully connected layer of size 100 used batch normalization and relu , and softmax for optimizer. For CIFAR-100 it uses 3 convolution block having 2 layers each of size $(32, 64, 128)$ each resp. followed by max-pool and relu and 2 fully connected layer.

4.4.2 Results

RMN					
	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	98.59	-	-	-	-
Train 2	98.59	94.07	-	-	-
Train 3	98.59	94.07	93.72	-	-
Train 4	98.59	94.07	93.72	94.03	-
Train 5	98.59	94.07	93.72	94.03	93.89

Table 4.4: Relevance Mapping Network for Permuted Mnist

RMN			
	P-MNIST	S-MNIST	CIFAR-100
Mean Accuracy	97.12	98.86	78.27

Table 4.5: Mean Accuracy for all 3 dataset

- **Observation:** Our results illustrate that our model can effectively mitigate catastrophic forgetting and retention issues, while still adhering to the strict requirements of a continual learning paradigm. Furthermore, we achieved state-of-the-art performance on various widely-used benchmark datasets, without compromising the conditions of continual learning.

4.5 INCREMENTAL MOMENT MATCHING

Incremental Moment Matching (IMM) [7] approach is based on the Bayesian neural network framework. This means that it incorporates uncertainty into the parameters of neural networks and calculates the posterior distribution. In IMM, the mixture of Gaussian posterior, where each component represents the parameters for a single task, is approximated as a single Gaussian distribution for a combined task.

4.5.1 Experimental details

- **Dataset:** Here we have used Disjoint MNIST with 2 task containing 5 classes each. i.e., Task 1 -$\{0, 1, 2, 3, 4\}$ and Task 2 - $\{5, 6, 7, 8, 9\}$

4.5.2 Result

IMM		
	Mean-IMM	Mode-IMM
Mean Accuracy	95.12	98.38

Table 4.6: Mean Accuracy Disjoint MNIST

- **observation:** The Mean-Incremental Moment Matching (Mean-IMM) approach achieves competitive results when compared to other models and balances the information between old and new networks. We can understand the success of IMM in the context of the Bayesian framework, specifically the Gaussian posterior. Additionally, we extended Mean-IMM to Mode-Incremental Moment Matching (Mode-IMM) by considering the interpretation of mode-finding in the mixture of Gaussian posterior.Furthermore, Mode-IMM has shown to outperform Mean-IMM and other comparative models in various datasets, making it a promising approach for continual learning in modern deep neural networks.

4.6 HARD ATTENTION MODEL

The proposed approach [10] utilizes a hard attention mechanism that enables the preservation of information from previous tasks without interfering with the learning of new tasks. Along with learning a new task, we also simultaneously learn almost-binary attention vectors using gated task embeddings, which are optimized through

backpropagation and minibatch stochastic gradient descent (SGD).

4.6.1 Experimental details

- **Dataset:** Used CIFAR-10 with 5 task 2 classes each.
- **Architecture:** a convolutional neural network with three layers of size 64, 128, 256 followed by max-pooling layer and relu as activation followed by 2 fully connected layers with softmax as output.
- **Evaluation:** The forgetting ratio is a metric used to compare task accuracies and determine the level of forgetting that has occurred. This ratio provides a general measurement of the extent to which previous knowledge has been lost

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

This equation calculates the forgetting ratio, denoted as $\rho^{\tau \leq t}$, which is used to quantify the amount of forgetting that occurs when learning a new task while sequentially learning previous tasks. The accuracy on task τ after sequentially learning task t is denoted as $A^{\tau \leq t}$. A^{τ}_{R} represents the accuracy of a random stratified classifier that uses the class information of task τ, and $A^{\tau \leq t}_{J}$ represents the accuracy on task τ after jointly learning t tasks in a multitask fashion

4.6.2 Results

HAT					
	$\rho^{\leq 1}$	$\rho^{\leq 2}$	$\rho^{\leq 3}$	$\rho^{\leq 4}$	$\rho^{\leq 5}$
Average forgetting ratio	-0.02	-0.04	-0.04	-0.05	-0.06

Table 4.7: Average forgetting ratio

- **Obesrvation:** hard attention mechanism can protect information from previous tasks while learning new tasks by focusing on a task embedding. This mechanism is easy to train together with the main model using backpropagation, and only adds a small number of weights to the base network, resulting in negligible overhead.

4.7 SELECTIVE LAYER ON DLGN

One approach to implementing task-specific layers is to selectively freeze or decelerate learning for certain layers or neurons that we want to specialize for a particular task. This

helps to ensure that these layers or neurons focus exclusively on learning features that are specific to that task, while minimizing interference from features learned for other tasks.

4.7.1 Experiment details

- **Dataset:** used permuted-MNIST and created 5 task of 10 class each being different permutation of MNIST. Further we used Office Home dataset with 4 task each being Art, Clipart, Product, Real-World with 65 categories.

- **Architecture:** we use 2 separate VGGs for Weight and Gates , Weight network is Called Neural Path Value (NPV), Gates network is Called Neural Path Feature (NPF), Identity Function is used instead of ReLU. NPVs are provided with constant 1 as input, NPFs are provided with actual image as input, 13 hidden convolution layer. Global Average Pooling is used at the end of 5 blocks, at last we have 3 FC Dense Layer at last we have Softmax activation

4.7.2 Results

DLGN					
	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	97.04	-	-	-	-
Train 2	20.32	94.74	-	-	-
Train 3	10.21	15.32	93.17	-	-
Train 4	10.56	13.71	20.56	90.19	-
Train 5	10.12	10.38	16.21	19.28	93.21

Table 4.8: Catastrophic Forgetting on DLGN

DLGN					
	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	97.04	-	-	-	-
Train 2	86.23	94.04	-	-	-
Train 3	78.35	82.23	89.71	-	-
Train 4	76.01	80.63	85.87	88.95	-
Train 5	72.08	78.48	83.23	86.12	89.30

Table 4.9: Accuracy for Selective Layer on DLGN for P-MNIST

- **Table Reading: Train i** simply states that **Task i** is being trained after training **Task 1** to **Task [i-1]** in the increasing label sequence. And then Tested all the above task in the same order from **Task 1** to **Task i** .

- **Observation:** By either freezing specific layers or significantly reducing their learning rate, we can reserve them for a particular task and achieve better accuracy

DLGN				
	Art	Clipart	Product	Real World
Art	70.71	-	-	-
Clipart	51.28	67.94	-	-
Product	52.01	67.12	72.34	-
Real World	53.21	58.21	62.21	71.12

Table 4.10: Accuracy for Selective Layer on DLGN for Office Home Dataset

compared to the baseline model. This indicates that task-specific layers are a promising approach for continual learning to mitigate catastrophic forgetting.

4.8 EWC ON DLGN

In this approach, the posterior distribution of the NPVs is approximated as a Gaussian distribution. The mean of this distribution is determined by the parameters θ_A^*, while the diagonal precision is derived from the diagonal elements of the Fisher information matrix F.

4.8.1 Experiment details

- **Dataset:** used permuted-MNIST and created 5 task of 10 class each being different permutation of MNIST.

- **Architecture:** we use 2 separate VGGs for Weight and Gates , Weight network is Called Neural Path Value (NPV), Gates network is Called Neural Path Feature (NPF), Identity Function is used instead of ReLU. NPVs are provided with constant 1 as input, NPFs are provided with actual image as input, 13 hidden convolution layer. Global Average Pooling is used at the end of 5 blocks, at last we have 3 FC Dense Layer at last we have Softmax activation

4.8.2 Results

- **Table Reading: Train i** simply states that **Task i** is being trained after training **Task 1** to **Task [i-1]** in the increasing label sequence. And then Tested all the above task in the same order from **Task 1** to **Task i** .

- **Observation:** Through the selective reduction of weight plasticity, EWC allows for the preservation of knowledge obtained from prior tasks while learning new ones, effectively preventing the phenomenon of catastrophic forgetting, where

DLGN					
	Task 1	Task 2	Task 3	Task 4	Task 5
Train 1	98.39	-	-	-	-
Train 2	96.74	95.39	-	-	-
Train 3	89.00	91.50	95.79	-	-
Train 4	84.20	83.42	92.54	95.57	-
Train 5	79.76	69.94	81.74	90.14	95.83

Table 4.11: Accuracy for EWC DLGN for P-MNIST

earlier skills are lost. This methodology shares similarities with neurobiological models of synaptic consolidation.

CHAPTER 5

RESULTS/CONCLUSIONS.

5.1 PHASE-I

- **Deep Gated Network**
  Using the DGN framework for experimentation, we noticed that during training, neural path features (NPFs) are learned and this type of learning leads to improved generalization. The information that is essential to us is stored in these NPFs. Our results demonstrate that FLNPF and DLNPF are comparable to simple ReLU based networks. This highlights the uniqueness of NPF learning in CNNs. By utilizing NPFs, we were able to achieve the original levels of accuracy, which led us to conclude that gates hold valuable information. We intend to further investigate the role of gates.

- **Deep Linearly Gated Network**
  Extensive experimenting using the DLGN framework we conclude that DLGNs architecture enjoys feature interpretability without sidelining accuracies. Some Key Observation from the experiments are:-As we move from shallow-linear model to DLGN-Shallow model we see an increase in performance. Adding Max-Pooling introduces non-linearity thus making the network less interpretable but gives a boost to the gain i.e., VGG-LIN-MAX performs better than VGG-LIN. we can arrange the interpretability of the models used in the above table as follows: Shallow Linear performs better than Deep Linear which in turn performs better than Deep Linear + MaxPool. By dual-lifting VGG-Lin-Max to VGG-DLGN-Max, we were able to improve performance on both datasets. Thus we can conclude that a network can be interpretable and perform at par with the conventional DNN with non-linearity.

5.2 PHASE-II

- By incorporating NPFs and NPVs, we can optimize the network to learn from multiple tasks in a more efficient and effective manner while minimizing catastrophic forgetting. Furthermore, the use of probabilistic methods, such as Bayesian inference, can also enhance the effectiveness of the DLGN approach by allowing for the estimation of task posteriors and adapting the network accordingly.
  In conclusion, DLGNs provide a promising solution to the problem of catastrophic forgetting in continual learning. By utilizing hard parameter sharing, varying learning rates, and task-specific layers, along with NPFs and NPVs, we can

optimize the network to learn multiple tasks while minimizing the risk of forgetting. Additionally, the use of probabilistic methods can further enhance the effectiveness of DLGNs in multitasking scenarios. Among the three approaches discussed earlier, the third and the fourth approach of task-specific layers with frozen or slow learning and EWC worked better than the other two, highlighting the importance of adaptability in continual learning and taking a probabilistic perspective proves beneficial in justifying the selection of constraints and identifying the most critical weights for a specific task in neural network training.

CHAPTER 6

FUTURE WORK

DLGNs offer a promising approach for implementing continual learning and addressing the challenge of catastrophic forgetting. Techniques such as Relevance Mapping, and probabilistic methods can be used to overcome these challenges by preserving existing knowledge while accommodating new information. By utilizing NPVs and NPFs, weight and gating information can be encoded separately, leading to better adaptability and interpretability. In addition, the use of posteriors provides a powerful tool for improving the efficiency and adaptability of DLGNs in multi-task settings, by enabling shared priors and better uncertainty quantification.By using Bayesian inference to compute these posteriors, the model can perform both parameter estimation and uncertainty quantification simultaneously, resulting in more robust and adaptive learning. Furthermore, this approach can be extended to multi-task learning by using shared priors for the model parameters across tasks, allowing for improved transfer learning and knowledge sharing between tasks. Overall, DLGNs have significant potential for enabling more efficient and adaptive machine learning systems

REFERENCES

[1] **Chandrashekar Lakshminarayanan and Amit Vikram Singh. Neural path features and neural path kernel**: Understanding the role of gates in deep learning. Advances in Neural Information Processing Systems, 33, 2020.

[2] **Szymon Mikler**. Resnets in tensorflow2. https://github.com/gahaalt/resnets-in-tensorflow2, 2019.

[3] **Chen, Zhiyuan, and Bing Liu. "Lifelong machine learning**." Synthesis Lectures on Artificial Intelligence and Machine Learning 12, no. 3 (2018): 1-207. Chapter 4.

[4] **Parisi, German I., Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter**. "Continual lifelong learning with neural networks: A review." Neural Networks (2019).

[5] **Kemker, Ronald, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan**. "Measuring catastrophic forgetting in neural networks." In Thirty-second AAAI conference on artificial intelligence. 2018.

[6] **Haiyan Yin, Peng Yang, Ping Li**. Mitigating Forgetting in Online Continual Learning with Neuron Calibration NeurIPS-2021

[7] **Sang-Woo Lee1 , Jin-Hwa Kim1 , Jaehyun Jun1 , Jung-Woo Ha2 , and Byoung-Tak Zhang1,3**. Overcoming Catastrophic Forgetting by Incremental Moment Matching NIPS 2017

[8] **Prakhar Kaushik Adam Kortylewski Alex Gain Alan Yuille**. Understanding Catastrophic Forgetting and Remembering in Continual Learning with Optimal Relevance Mapping

[9] **Chandrashekar Lakshminarayanan et al** .Rethinking Conventional Interpretations of Deep Neural Networks (Unpublished manuscript)

[10] **Joan Serrà, Dídac Surís, Marius Miron, Alexandros Karatzoglou**. Overcoming catastrophic forgetting with hard attention to the task

[11] **Gido M. van de Ven, Andreas S. Tolias**. Three scenarios for continual learning

[12] **James Kirkpatrick kirkpatrick, Razvan Pascanu, Neil Rabinowitz, and Raia Hadsell**. Overcoming catastrophic forgetting in neural networks