Continual Learning using Synaptic Intelligence

Aparna Bhutani
Center for Urban Science and Progress
New York University
New York, NY
`ab8473@nyu.edu`

Twishikana Bhattacharjee
Department of Computer Engineering
NYU - Tandon School of Engineering
New York, NY
`tb2517@nyu.edu`

Abstract

Catastrophic Forgetting is one of the biggest concerns in the current Applied Machine Learning realm. While Neural Networks have transformed the world of Machine Learning, the plasticity of Neural Networks are extremely low. The network needs to be retrained to align with any changes in the data distribution, leading to a lot of rework with additional computation overhead. This causes repercussions, making it forget whatever it had learnt initially. As the community tries to deal with this, a lot of research is being done to come up with a concise and feasible methodology to cope with *catastrophic forgetting*. In this paper, we have tried to implement the paper *Continual Learning through Synaptic Intelligence* by Zenke, Poole and Ganguli (2017). We have discussed their framework and the theory behind it and then implemented it to reproduce the findings mentioned by Zenke et al. (2017). We have also compared it against the performance of the EWC and GEM algorithm

1 Introduction

Artificial neural networks (ANNs) have become a strong weapon in the armory of applied machine learning, more often beating human performance in a variety of domain-specific tasks (LeCun et al., 2015). Inspired by the neurons in the human brain (Rosenblatt, 1958; Fukushima and Miyake, 1982), the design principles and learning methods of an ANN differ substantially from biological neural networks. In ANNs, parameters are learned on a dataset in the training phase, frozen and then used statically on new data in the test phase. To accommodate for new task which would typically be a change in Data Distribution, ANNs have to be retrained on the entire dataset to avoid overfitting and catastrophic forgetting (Choy et al., 2006; Goodfellow et al., 2013). As opposed to ANNs, biological neural networks exhibit continual learning in which they acquire new knowledge over a lifetime without forgetting anything that was learnt before. Thus, it is difficult to draw a clear line between a learning and a recall phase. The learning phase is comparable to the training phase in machine learning and the recall phase is comparable to test phase. Over the years, human brain has evolved to learn from all kinds of sensory data and update behaviour and response in accordance. While the inner working of this feat accomplished in the brain is a mystery, it seems possible that the biological performance in continual learning could rely on features implemented by the underlying biological connections that are not currently implemented in ANNs. Easily, one of the greatest gaps in the design of modern ANNs versus biological neural networks lies in the complexity of synapses. In ANNs, individual synapses (weights) are typically described by a single scalar quantity. On the other hand, a biological synapse make use of complex molecular machinery that can affect plasticity at different spatial and temporal scales (Redondo and Morris, 2011). In their paper, Zenke, Poole and Ganguli (2017) study the role of internal synaptic dynamics to enable ANNs to learn sequences of classification tasks. We have tried to reproduce their study and analyze the performance of the Synaptic Intelligence Method. We have also tried to evaluate and compare the performance of Synaptic Intelligence (SI) method to that of the Elastic Weight Consolidation (EWC) method and

Preprint. Under review.

Gradient Episodic Memory (GEM) method. The motivation behind the paper is that while simple, scalar one-dimensional synapses suffer from catastrophic forgetting, in which the network forgets previously learned tasks when trained on a novel task, this paper attempts to alleviate this problem by synapses with a more complex three-dimensional state space. In Synaptic Intelligence model, the synaptic state tracks the past and current parameter value, and maintains an online estimate of the synapse's "importance" toward solving problems encountered in the past. Simply put, it checks how important the particular synapse is towards solving a Task A. The importance measure is computed locally at each synapse during training, and represents the local contribution of each synapse to the change in the global loss. When the task changes, the important synapses are consolidated by preventing them from changing in future tasks. Thus the responsibility of learning in future tasks is imposed primarily on synapses that were unimportant for past tasks, thereby avoiding catastrophic forgetting of these past tasks.

2 The Synaptic Framework

The motivation behind the Synaptic Intelligence method is to build a structural regularizer that could be computed on the go and implemented at each synapse locally. The aim is to provide each synapse with a local measure of importance in solving tasks that the network has been trained on in the past. While training on a new Task, there is a penalty on changes to the important parameters to make sure that the memories of the old task is not overwritten. Thus, a class of algorithms was developed that would help keep track of an importance measure w_k^μ which reflects the past credit for improvements of task object L_μ for a task μ to individual synapses θ_k. For convenience, the term synapse is being used synonymously with the term parameter which is inclusive of weights between layers as well as biases. The process of training a neural network is characterized by a trajectory $\theta(t)$ in parameter space (Figure 1).

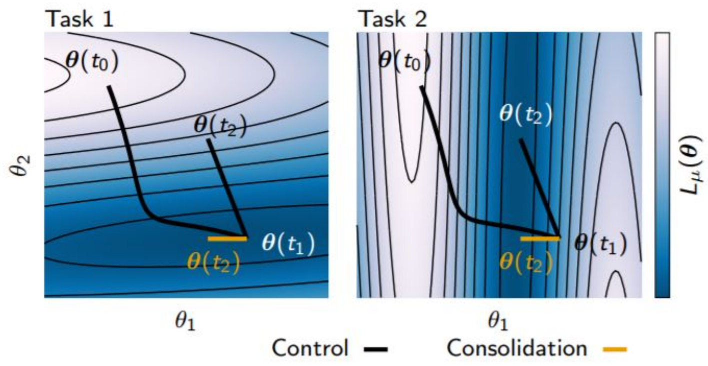

Figure 1: Taken from Continual Learning through Synaptic Intelligence Zenke et al. (2017). Schematic illustration of parameter space trajectories and catastrophic forgetting. Solid lines correspond to parameter trajectories during training. Left and right panels correspond to the different loss functions defined by different tasks (Task 1 and Task 2). The value of each loss function L_μ shown as a heat map. Gradient descent learning on Task 1 induces a motion in parameter space from $\theta(t_0)$ to $\theta(t_1)$. Subsequent gradient descent dynamics on Task 2 yields a motion in parameter space from $\theta(t_1)$ to $\theta(t_2)$. This final point minimizes the loss on Task 2 at the expense of significantly increasing the loss on Task 1, thereby leading to catastrophic forgetting of Task 1. However, there does exist an alternate point θ_2 labelled in orange, that achieves a small loss for *both* tasks.

The success of a training lies in finding learning trajectories for which the endpoint lies close to a minimum of the loss function L on all tasks. First, we consider the change in loss for an infinitesimal parameter update $\delta(t)$ at time t. In this case the change in loss is well approximated by the gradient $g = \frac{\delta L}{\delta \theta}$ and we can write

$$L(\theta(t) + \delta(t)) - L(\theta(t)) \approx \sum_k g_k(t)\delta_k(t) \tag{1}$$

which illustrates that each parameter change $\delta_k(t) = \theta_k'(t)$ contributes the amount $g_k(t)\delta_k(t)$ to the change in total loss. To compute the change in loss over an entire trajectory through parameter space we have to sum over all infinitesimal changes. This amounts to computing the path integral of the gradient vector field along the parameter trajectory from the initial point (at time t_0) to the final point (at time t_1):

$$\int_C g(\theta(t))d\theta = \int_{t_0}^{t_1} g(\theta(t)).\theta'(t)dt \tag{2}$$

As the gradient is a conservative field, the value of the integral is equal to the difference in loss between the end point and start point: $L(\theta(t_1)) - L(\theta(t_0))$. Crucial to our approach, we can decompose Equation 2 as a sum over the individual parameters

$$\int_{t^{\mu-1}}^{t^{\mu}} g(\theta(t)).\theta'(t)dt = \sum_k \int_{t^{\mu-1}}^{t^{\mu}} g_k(\theta(t)).\theta'_k(t)dt \equiv -\sum_k w_k^{\mu} \tag{3}$$

The w_k^{μ} now have an intuitive interpretation as the parameter specific contribution to changes in the total loss. The minus sign has been introduced in the second line because typically the interest is in decreasing the loss. In practice, we can approximate w_k^{μ} online as the running sum of the product of the gradient $g_k(t) = \frac{\delta L}{\delta \theta_k}$ with the parameter update $\theta'_k(t) = \frac{\delta \theta_k}{\delta t}$. For batch gradient descent with an infinitesimal learning rate, w_k^{μ} can be directly interpreted as the per-parameter contribution to changes in the total loss. In most cases the true gradient is approximated by stochastic gradient descent (SGD), resulting in an approximation that introduces noise into the estimate of g_k. As a direct consequence, the approximated per-parameter importance will typically overestimate the true value of w_k^{μ}. An attempt is made to use the w_k^{μ} information to improve the continual learning performance. We are trying to minimize the total loss$\mathcal{L} = \sum_{\mu} L_{\mu}$, the limitation being we do not have access to loss functions of tasks we were training on in the past. We only have access to loss function L_{μ} for a single task μ at any given time. Catastrophic forgetting occurs when minimizing L_{μ}in turn leads to substantial increase of cost on previous tasks L_v with $v < \mu$ (Figure 1). To avoid catastrophic forgetting of all previous tasks ($v < \mu$) while training task μ, we would attempt to avoid drastic changes to weights which were influential in the past. The importance of θ_k for a single task is based on two factors:

1. How much an individual parameter contributed to reduce the loss w_k^{μ} over the entire training trajectory.
2. How far it moved, i.e. $\Delta_k^v \equiv \theta_k(t^v) - \theta_k(t^{v-1})$

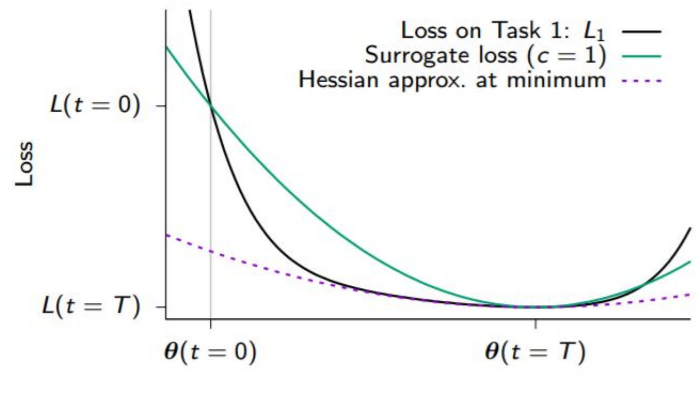

Figure 2: Taken from Continual Learning through Synaptic Intelligence Zenke et al. (2017). Schematic illustration of surrogate loss after learning one task. Consider some loss function defined by Task 1 (black). The quadratic surrogate loss (green) is chosen to precisely match 3 aspects of the descent dynamics on the original loss function: the total drop in the loss function $L(\theta(0)) - L(\theta(T))$, the total net motion in parameter space $\theta(0) - \theta(T)$, and achieving a minimum at the endpoint $\theta(T)$. These 3 conditions uniquely determine the surrogate quadratic loss that summarizes the descent trajectory on the original loss. Note that this surrogate loss is different from a quadratic approximation defined by the Hessian at the minimum (purple dashed line)

To avoid large changes to the important parameters, a modified cost function $\tilde{L}_{\mu}$ in which a surrogate loss is introduced. The surrogate loss helps approximate the summed loss functions of previous tasks $L_v(v < \mu)$. A quadratic surrogate loss is used that has the same minimum as the cost function of the previous tasks and yields the same w_k^{μ} over the parameter distance Δ_k . So, if we implement the learning on the surrogate loss in place of the actual loss function, it would result in the same final parameters and change in loss during training (Figure 2). For two tasks this is achieved by the following quadratic surrogate loss:

$$\tilde{L}_\mu = L_\mu + \underbrace{c \sum_k \Omega_k^\mu (\tilde{\theta}_k - \theta_k)^2}_{\text{surrogate loss}} \tag{4}$$

where,

c is a dimensionless strength parameter,

$\tilde{\theta}_k$ is the reference weight corresponding to parameters at the end of the previous task such that $\tilde{\theta}_k = \theta_k(t^{\mu-1})$,

Ω_k^μ is the per-parameter regularization strength that is represented as follows:

$$\Omega_k^\mu = \sum_{v<\mu} \frac{w_k^\mu}{(\Delta_k^v)^2 + \xi} \tag{5}$$

The term $(\Delta_k^v)^2$ in the denominator helps ensure that the regularization term carries the same unit as loss L.

A damping parameterξ is used to help bind the expression in cases where $\Delta_k^v) \to 0$.

The strength parameter c trades off old versus new memories. If path integral in Equation 3 is evaluated precisely, $c = 1$ would correspond to an equal weighting of new and old memories. But because of the noise in evaluation of path integral, c should typically be chosen smaller than the one we need to compensate.

Unless specified otherwise, w_k is continuously updated during training. On the other hand, Ω_k^μ which is the cumulative importance measure and $\tilde{\theta}$ which is the reference weight are only updated at the end of each task. After updating Ω_k^μ, w_k are set to 0.

To understand how the particular choices of Equation 4 and 5 affect learning, let us consider the example illustrated in Figure 1 in which we learn two tasks. We first train on Task 1. At time t_1 the parameters have approached a local minimum of the Task 1 loss L_1. But, the same parameter configuration is not close to a minimum for Task 2. Consequently, when training on Task 2 without any precautions, the L_1 loss may inadvertently increase (Figure 1, black trajectory). However, when θ_2 "remembers" that it was important to keeping L_1 low, it can utilize this knowledge during training on Task 2 by staying close to its current value (Fig. 1, orange trajectory). While this will almost inevitably result in a decreased performance on Task 2, this decrease could be negligible, whereas the gain in performance on both tasks combined can be substantial. The approach presented here is similar to EWC (Kirkpatrick et al., 2017) in that more influential parameters are pulled back more strongly towards a reference weight with which good performance was achieved on previous tasks. However, in contrast to EWC, the method here computes an importance measure online and along the entire learning trajectory, whereas EWC relies on a point estimate of the diagonal of the Fisher information metric at the final parameter values, which has to be computed during a separate phase at the end of each task and is also computationally heavy.

3 Theoretical Approach of the Synaptic Framework

The general approach taken here to implement Synaptic Intelligence recovers sensible Ω_k^μ in case of a simple and analytically tractable training scenario. Analysis is done to check how the w_k^μ and normalized Ω_k^μ (Equation 5) react in terms of the geometry of a basic quadratic error function

$$E(\theta) = \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*) \tag{6}$$

where

the minimum of the error function is at θ^*

H is a Hessian Matrix

Next, a batch gradient descent is considered on this error E. Thus, this can now be repesented as a continuous time differential equation

$$\tau \frac{d\theta}{dt} = -\frac{\delta E}{\delta \theta} = -H(\theta - \theta^*) \tag{7}$$

where τ is related to learning rate. On starting from an initial condition $\theta(0)$ at time $t = 0$, we find an equation that provides the exact solution to descent path.

$$\theta(t) = \theta^* + e^{-H\frac{t}{\tau}}(\theta(0) - \theta^*) \tag{8}$$

Equation 8 on being differentiated with respect to t yields a time dependent update direction

$$\theta'(t) = \frac{d\theta}{dt} = -\frac{1}{\tau} H e^{-H\frac{t}{\tau}}(\theta(0) - \theta^*) \tag{9}$$

The gradient, in accordance to classical gradient descent dynamics, obeys $g = \tau\frac{d\theta}{dt}$. Thus, the w_k^μ in Equation 3is computed as the diagonal elements of the matrix

$$Q = \tau \int_0^\infty dt \frac{d\theta}{dt}\frac{d\theta^T}{dt} \tag{10}$$

Thus, we can derive an explicit formula for Q that can be represented in terms of the eigenbasis of Hessian H. Let λ^α and u^α denote eigenvalues and eigenvectors respectively for Hessian H. Also, $d^\alpha = u^\alpha.(\theta(0) - \theta^*)$ be the projection of the discrepancy between initial and final parameters onto α'th eigenvector. Inserting Equation 9 to Equation 10, performing changes as per eigenmodes of $H$and then doing the integral yields, we get

$$Q_{ij} = \sum_{\alpha_\beta} u_i^\alpha d^\alpha \frac{\lambda^\alpha \lambda^\beta}{\lambda^\alpha + \lambda^\beta} d^\beta u_j^\beta \tag{11}$$

Now the time constant τ no longer governs the speed of descent as Q is now a time-integrated steady state quantity and is independent of τ. At a glance it would seem that Q depends on both eigenvectors and eigenvalues of the Hessian, and also on $\theta(0)$ in a complicated way. Let's first consider averaging Q over random initial conditions $\theta(0)$ such that collection of d^α constitutes a set of zero mean iid random variables with a variance σ^2. So the average is $\langle d^\alpha d^\beta \rangle = \sigma^2 d_{\alpha_\beta}$. Averaging over Q we get

$$\langle Q_{ij} \rangle = \frac{1}{2}\sigma^2 \sum_\alpha u_i^\alpha \lambda^\alpha u_j^\beta = \frac{1}{2}\sigma^2 H_{ij} \tag{12}$$

Thus after averaging over initial conditions, the Q matrix, which is available simply by correlating parameter updates across pairs of synapses and integrating over time, reduces to the Hessian, up to a scale factor dictating the discrepancy between initial and final conditions. Indeed, this scale factor theoretically motivates the normalization in Equation 5; the denominator in Equation 5, at zero damping, ξ averages to σ^2 and thus removes the scaling factor σ^2 in Equation 12 However, we need to figure out what Q_{ij} computes for a single initial condition. There are two scenarios in which the simple relationship between Q and the Hessian H is preserved without averaging over initial conditions.

First, consider the case when the Hessian is diagonal, so that $u_i^\alpha = \delta_{\alpha_i} e_i$ where e_i is the i'th coordinate vector. Then α and i indices are interchangeable and the eigenvalues of the Hessian are the diagonal elements of the Hessian: $\lambda^i = H_{ii}$. Then Equation11 reduces to

$$Q_{ij} = \delta_{i_j}(d^i)^2 H_{ii} \tag{13}$$

Normalizing Equation 5,at zero damping, removes the scale of movement in parameter space $(d^i)^2$. So the normalized Q matrix becomes *identical* to the diagonal Hessian

Secondly, consider an extreme case where Hessian is rank 1. Thus λ^1 is the only nonzero eigenvalue. Then Equation 11 reduces to

$$\frac{1}{2}(d^1)^2 u_i^1 \lambda_1 u_j^1 = \frac{1}{2}(d^1)^2 H_{ij} \tag{14}$$

Again, the Q matrix reduces to the Hessian upto a scale factor. The normalized importance now becomes the diagonal elements of the non-diagonal but low rank Hessian. The low rank Hessian is interesting case for continual learning as it leaves many directions in the synaptic weight space unconstrained by a given task thus leaving open excess capacity for synaptic modification to solve future tasks without interfering with performance on an old task.

It is important to stress that the path integral for importance is computed by integrating information along the *entire learning trajectory* (Figure 2). For a quadratic loss function, the Hessian is constant along this trajectory, and so a precise relationship is found between the importance and the Hessian.But for more general loss functions, where the Hessian varies along the trajectory, we cannot expect any simple mathematical correspondence between the importance Ω_k^μ and the Hessian at the endpoint of learning, or related measures of parameter sensitivity (Pascanu and Bengio, 2013; Martens, 2016; Kirkpatrick et al., 2017) at the endpoint.

4 Other Methods in Continual Learning

4.1 The Elastic Weight Consolidation Method

The EWC prevents catastrophic forgetting by constraining important parameters to stay close to their old values when performing new tasks. For example, when it learns a new task B, EWC protects the performance in task A by constraining the parameters to stay in a region of low error for task A centered around the set of weights and biases defined by θ_A. EWC also uses a posterior probability to justify this choice of constraint and decides which weights are most important for a task. In addition, it approximates the posterior as a Gaussian distribution with mean given by the parameters θ_A and a diagonal precision given by the diagonal of the Fisher information matrix F. The loss function used:

$$L(\theta) = L_B(\theta) + \sum_i^N \tfrac{2}{\lambda} F_i(\theta_i - \theta_{i,A}^*)^2$$

Here, the $L_B(\theta)$ is the loss function for new task only, the λ is the EWC lambda, which decides how important the previous task's parameters are to the new task, also there are i labels for each parameter. When EWC moving with the third task or more task, it tries to keep the network parameters close to the learned parameters previous tasks since the sum of two quadratic penalties is itself a quadratic penalty.

4.2 The Gradient Episodic Memory Method

The GEM method focuses on a feature called the *episodic memory* M_t which is used to store a subset of observed examples from the task t. So when we observe a the triplet (x, t, y), where x is the datapoint, y is the label and t is the task t, we try and solve the following:

$$\text{minimize}_\theta \quad l(f_\theta(x,t), y) \quad \text{subject to} \quad l(f_\theta, M_k)(f_\theta^{t-1}, M_k \; \forall \; k < t,)$$

where f_θ^{t-1} is predictor state at the end of task $t-1$ We observe it is unnecessary to store old predictors as long as we guarantee that the loss at previous tasks does not increase after each parameter update g. Also, assuming that the function is locally linear and that the memory is representative of the examples from past tasks, an increase in the loss of previous tasks can be identified by computing the angle between their loss gradient vector and the proposed update. Thus, the constraints can be rephrased as:

$$\langle g, g_k \rangle := \langle \tfrac{\delta l(f_\theta(x,t),y)}{\delta\theta}, \tfrac{\delta l(f_\theta, M_k)}{\delta\theta} \rangle \geq 0, \text{for all } k < t$$

To satisfying all the constraints, it is proposed to project the proposed gradient g to the closest gradient $\tilde{g}$ (in squared l_2 norm) which readjusts the equation as follows:

$$\text{minimize}_g \quad \tfrac{1}{2}\|g - \tilde{g}\|_2^2 \quad \text{subject to} \quad \langle \tilde{g}, g_k \rangle \leq 0 \; \forall \; k < t,$$

Further, treating this as a quadratic problem we observe the primal and the dual problem and on solving the dual problem for a v^* (optimal value of v) we can write projected gradient as:
$\tilde{g} = G^T v^* + g$

5 Implementation of the Synaptic Method

In their paper, Zenke, Poole and Ganguli (2017) analyzed their algorithm on three kinds of dataset to measure its performance. They tested the algorithm on **Split MNIST**, **Permutated MNIST** and split versions of **CIFAR-10** and **CIFAR-100** .

In the experiment with **Split MNIST** dataset, it was established that with synaptic consolidation turned off, i.e. $c = 0$, the performance of the network drops to chance levels but the model with consolidation on , i.e. $c = 1$, shows minor degradation in performance on the tasks.

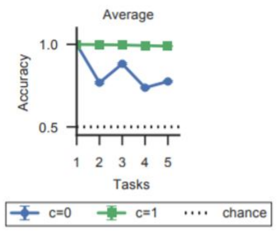

Figure 3: Taken from Continual Learning through Synaptic Intelligence Zenke et al. (2017).Average performance on Split MNIST. Observe a drop in the control setting graph where $c = 0$.

In the **Permutated MNIST** dataset, to establish a baseline for comparison the network is first trained without synaptic consolidation (c = 0) on all tasks sequentially. In this scenario the system exhibits catastrophic forgetting, i.e. it learns to solve the most recent task, but rapidly forgets about previous tasks. In contrast, when the synaptic consolidation is set to some value of $c > 0$, the same network retains high classification accuracy on Task 1 while being trained on the other 9 tasks. Moreover, the network learns to solve all other tasks with substantially high accuracy and performs only slightly worse than a network which had trained on all data simultaneously. To summarize the findings of their experiments, consolidation not only protected old memories from being slowly forgotten over time, but also allowed networks to generalize better on new tasks with limited data.

We tried implementing their idea with the hopes of trying to replicate their performance but the dataset we used was **Rotated MNIST**.

Experimenting on Rotated MNIST dataset
Our model consists of three fully connected hidden layers with 100 units in each hidden layer. We have used Rectified Linear Units(ReLUs) as activation function for each hidden layer and used standard cross-entropy loss. The optimizer used was Adam and the learning rate was set to 0.003. The batch size was set to 64. The model was trained. After training we evaluated the model on the 10 test sets. For the experiment, we set the value as $c = 0.152$.As a control, we ran the experiment on the model using $c = 0$. This way the surrogate loss regularization term would always be set to 0 and we could emulate the performance that Zenke et al. demonstrated using the Split MNIST dataset.

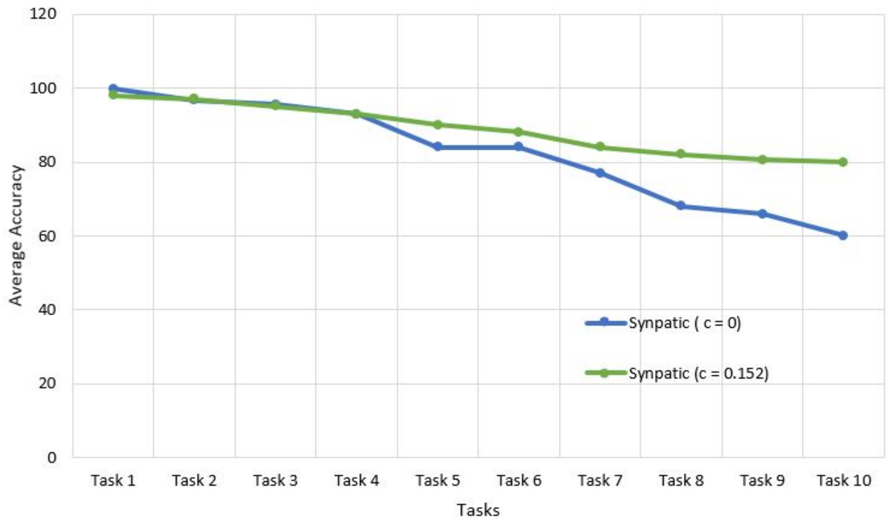

Figure 4: Performance of the Algorithm for training the Rotated MNIST dataset. (Blue) Control setting *c=0*; (Green) Experimental setting *c=0.152*. We were able to replicate Zenke et al.(2017) observation that without setting synaptic consolidation the performance gradually drops to almost pure chance whereas with synaptic consolidation the performance is substantially better

	Task 1	Task 2	Task 3	Task 4	Task 5	Taks 6	Task 7	Task 8	Task 9	Task 10
Synaptic (c=0)	99.8	96.7	95.5	93	84	84	77	68	66	60
Synaptic (c=0.152)	98	97	95	93	90	88	84	82	80.6	80

Table 1: Data of the training performance of the 10 Tasks on the Synaptic Intelligence Model

As per the data, we can clearly see that for the first task, the performance of the control setting $c = 0$ is almost 100% accurate. But the accuracy starts dipping with introduction of new tasks. Post task 5, we can observe a pretty sharp deterioration in the performance. On the other hand, the experiment setting $c = 0.152$ is not a 100% accurate with task 1. Also, with the increase in the number of tasks, the accuracy deteriorates. But this deterioration is much more tolerable that the deterioration exhibited by control setting $c = 0$. For $c = 0$, Task 1 to Task 10 performance goes from a 99.8% accuracy to a mere 60% accuracy, dropping by almost 40%. But for the experiment setting $c = 0.152$ accuracy goes from a 98% to 80% dropping by 18% only which is half of what the control setting.

Training vs Testing Performance

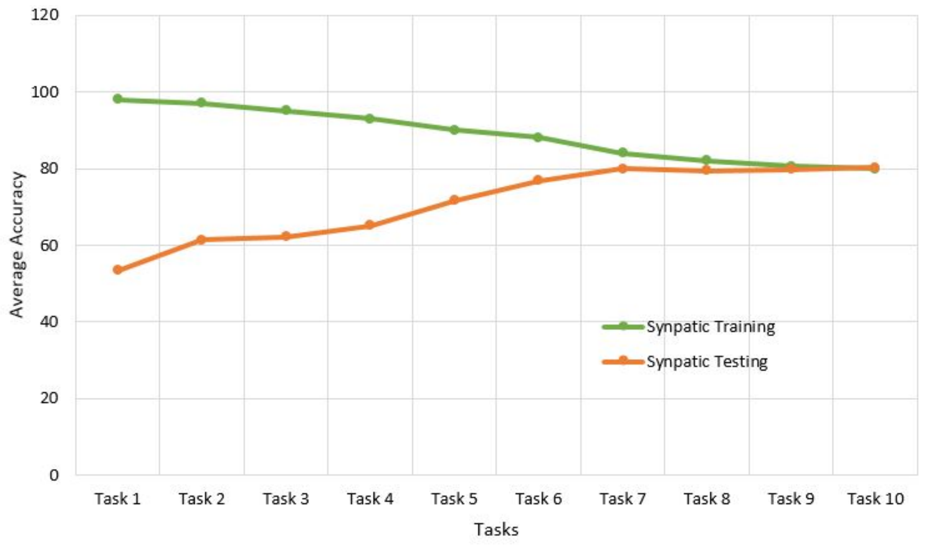

Figure 5: Performance of the Algorithm for training versus testing on the Rotated MNIST dataset.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10
Synaptic Testing	53.355	61.325	62.167	65.123	71.682	76.777	79.867	79.46	79.8	80.259
Synaptic Training	98	97	95	93	90	88	84	82	80.6	80

Table 2: Data comparing the Testing versus Training performance of Synaptic Intelligence method

The Testing Vs Training plot emulates the expected behaviour. As we train the model on more tasks, its accuracy decreases as we have already observed. But when we provide this model with a test sequence of data which is ordered Task 1, Task 2,..., Task 10, we observe a certain level of *forgetting*. With the test dataset, the model performs worst on the task that it had learnt in the very beginning and exhibits best performance in the task that it has learnt most recently. Thus, it successfully mimes the human tendency of forgetting the task it had learnt in the very beginning and effectively retaining the task it learnt at the very end. In fact, we see the same accuracy on both testing and training data for Task 10.

6 Comparative Study of the Synaptic Method

Once we completed our implementation of the Synaptic Intelligence Method with the two values of strength parameter c (0 and 0.152), we wanted to observe how this method fares against the other two popular continual learning method. We used the same **Rotated MNIST dataset** that we used in the previous implementation. The two tasks that the algorithm was trained in is the **Original MNIST dataset** and the **Rotated MNIST dataset**. We did a comparative study of the three algorithms.

Architecture used:

1. **EWC:** We used MLP as the basic network architecture for this experiment. The MLP model consists of four fully connected hidden layers with 512, 256,128,10 units in each hidden layer respectively and uses Rectified Linear Units(ReLUs) as activation function for each hidden layer. We also included a dropout layer (p=0.2) to prevent the overfitting of data. We tried different batch size ranging from 128 to 512, and finally decided to set batch size to 200 since this gave the highest accuracy.
2. **GEM:**We used MLP as the basic network architecture for this experiment. The MLP model consists of three fully connected hidden layers with 100 units in each hidden layer respectively and uses Rectified Linear Units(ReLUs) as activation function for each hidden layer. We used a learning rate of 0.003 and set the minibatches to 100. We used SGD optimizer and cross-entropy loss function for this experiment.
3. **Synaptic:** We used the same model setup as in section 5 to do the performance comparison on Rotated MNIST dataset.

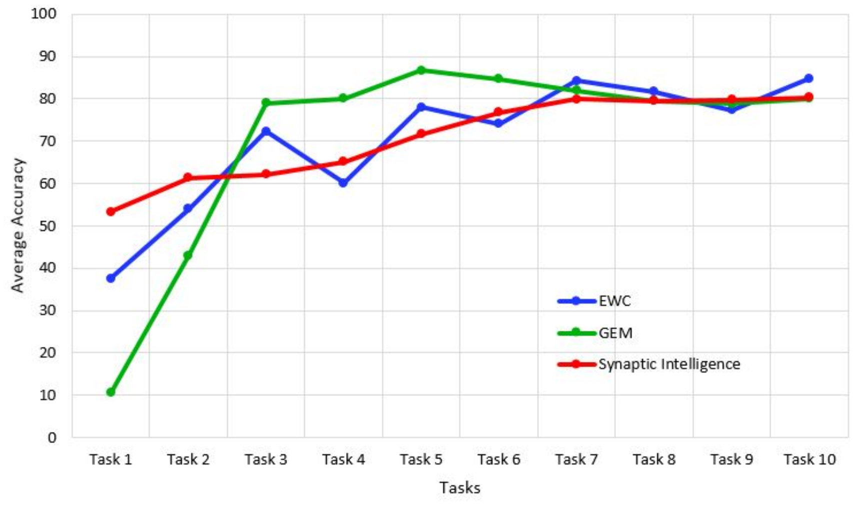

Figure 6: Performance Analysis of EWC, GEM and Synaptic Intelligence while testing on Rotated MNIST dataset.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10
EWC	37.55	53.942	72.26	60.028	77.984	74.09	84.28	81.65	77.26	84.764
GEM	10.6	42.98	78.89	79.98	86.66	84.7	81.99	80.01	79.5	78.88
Synaptic	53.355	61.325	62.167	65.123	71.682	76.777	79.867	79.46	79.8	80.259

Table 3: Data comparing the performance of the 10 Tasks on the three Continual Learning Models

	Average over Tasks	Range
EWC	70.38	47%
GEM	70.4	76%
Synaptic	70.97	26.9%

Table 4: Average over 10 Tasks for the three algorithms. Range calculates the difference between the worst and the best accuracy. (range is in % as accuracy is being measure in %)

As we compare the performance of each of these algorithms, we average their performance across all the 10 tasks. As per our observation (noted in Table 4), the average performance of all the three algorithms is pretty close but the *Synaptic Intelligence* algorithm performs slightly better than the rest. Also, if we observe the graph, both EWC and GEM perform very poorly on *Task 1* exhibit a case of *almost catastrophic forgetting* in the case of GEM. The performance of GEM does show an exponential increase but then shows a slight dip and then plateaus.While the GEM algorithm average well across the 10 tasks, the difference between the best and worst accuracy on individual tasks is a whopping 76%. In case of EWC, post *Task 2*, the algorithm almost alternates between a rise and a dip in its performance. Again, the average does not get majorly affected. We do observe a lower difference between best and worst accuracy (approximately 47%) but it is still substantial. With the Synaptic Intelligence model, we see the best average. Also, the difference between the worst and the best accuracy stands at 26.9%, which is the best among the three. Also, it emulates human-like retention the best by remembering the most recent task the best.

7 Summary

We studied the problem of catastrophic forgetting commonly encountered in continual learning scenarios and implemented the popular algorithms to alleviated the issue. Our major focus was on trying to overcome catastrophic forgetting by allowing individual synapses to estimate their importance for solving past tasks. Then by penalizing changes to the most important synapses, new tasks could be successfully learned with minimal interference to previously learned tasks.This approach computes the per-synapse consolidation strength in an online fashion and over the entire learning trajectory in parameter space, as opposed to the comparable EWC approach where synaptic importance is computed offline as the Fisher information at the minimum of the loss for a designated task. This approach requires individual synapses to not simply correspond to single scalar synaptic weights, but rather act as higher dimensional dynamical systems in their own right enabling it a lot more movement in the dimension. For the machine learning community to successfully be able to mimic human intelligence, we would need cognizant synapses to go hand in hand with deeper neural networks.

References

[1] Zenke, Friedemann, Ben Poole, and Surya Ganguli. *"Continual learning through synaptic intelligence."* In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 3987-3995. JMLR. org, 2017.

[2] Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan et al. *"Overcoming catastrophic forgetting in neural networks."* Proceedings of the national academy of sciences 114, no. 13 (2017): 3521-3526.

[3] Lopez-Paz, David, and Marc'Aurelio Ranzato. *"Gradient episodic memory for continual learning."* In Advances in Neural Information Processing Systems, pp. 6467-6476. 2017.

[4] Li, Zhizhong, and Derek Hoiem. *"Learning without forgetting."* IEEE transactions on pattern analysis and machine intelligence 40, no. 12 (2017): 2935-2947.

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

[6] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. *"Deep learning."* nature 521, no. 7553 (2015): 436-444.

[7] Ostapenko, Oleksiy, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. *"Learning to remember: A synaptic plasticity driven framework for continual learning."* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11321-11329. 2019.

[8] Fukushima, Kunihiko, and Sei Miyake. *"Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition."* In Competition and cooperation in neural nets, pp. 267-285. Springer, Berlin, Heidelberg, 1982.

[9] Choy, Min Chee, Dipti Srinivasan, and Ruey Long Cheu. *"Neural networks for continuous online learning and control."* IEEE Transactions on Neural Networks 17, no. 6 (2006): 1511-1531.

[10] Goodfellow, Ian J., Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. *"An empirical investigation of catastrophic forgetting in gradient-based neural networks."* arXiv preprint arXiv:1312.6211 (2013).

[11] Redondo, Roger L., and Richard GM Morris. *"Making memories last: the synaptic tagging and capture hypothesis."* Nature Reviews Neuroscience 12, no. 1 (2011): 17-30.

[12] Lahiri, Subhaneil, and Surya Ganguli. *"A memory frontier for complex synapses."* In Advances in neural information processing systems, pp. 1034-1042. 2013.

[13] Ziegler, Lorric, Friedemann Zenke, David B. Kastner, and Wulfram Gerstner. *"Synaptic consolidation: from synapses to behavioral modeling."* Journal of Neuroscience 35, no. 3 (2015): 1319-1334.

[14] Zenke, Friedemann, Everton J. Agnes, and Wulfram Gerstner. *"Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks."* Nature communications 6, no. 1 (2015): 1-13.

[15] Fusi, Stefano, Patrick J. Drew, and Larry F. Abbott. *"Cascade models of synaptically stored memories."* Neuron 45, no. 4 (2005): 599-611.

[16] Benna, Marcus K., and Stefano Fusi. *"Computational principles of synaptic memory consolidation."* Nature neuroscience 19, no. 12 (2016): 1697.