Entropy-based Stability-Plasticity for Lifelong Learning

Vladimir Araujo\textsuperscript{1,2,*}, Julio Hurtado\textsuperscript{2,3,*}, Alvaro Soto\textsuperscript{2}, Marie-Francine Moens\textsuperscript{1}
\textsuperscript{1}KU Leuven, \textsuperscript{2}Pontificia Universidad Católica de Chile, \textsuperscript{3}University of Pisa
vgaraujo@uc.cl, jahurtado@uc.cl, asoto@ing.puc.cl, sien.moens@kuleuven.be

Abstract

The ability to continuously learn remains elusive for deep learning models. Unlike humans, models cannot accumulate knowledge in their weights when learning new tasks, mainly due to an excess of plasticity and the low incentive to reuse weights when training a new task. To address the stability-plasticity dilemma in neural networks, we propose a novel method called Entropy-based Stability-Plasticity (ESP). Our approach can decide dynamically how much each model layer should be modified via a plasticity factor. We incorporate branch layers and an entropy-based criterion into the model to find such factor. Our experiments in the domains of natural language and vision show the effectiveness of our approach in leveraging prior knowledge by reducing interference. Also, in some cases, it is possible to freeze layers during training leading to speed up in training.

1. Introduction

Humans learn continuously throughout their lives, integrating new information to their knowledge to face new and changing environments. By contrast, artificial neural networks learn in a bounded environment, where the input distribution is assumed fixed. When the input distribution changes, the model must adapt its weights to perform correctly on the new task. Due to those modifications, the model overwrites previously learned patterns, creating interference between old and new tasks, causing a problem known as catastrophic forgetting\cite{29,34}. This excessive plasticity in the model is part of the stability-plasticity dilemma\cite{31,44}, which addresses the trade-off between modifying the parameters to learn a new task (plasticity) or keeping the parameters constant (stability) to avoid interference between tasks.

Several methods have been proposed to mitigate the stability-plasticity dilemma, focusing mainly on avoiding the catastrophic forgetting problem. Using different techniques to mitigate interference, these methods can be divided into two groups. The first group aims to restrict weight modifications by using regularization functions\cite{1,19,49} that minimize the modifications of key weight values. The second group uses gating functions\cite{15,27,39} to adaptively activate each weight depending on the context provided by the current task or input instance.

In this work, we follow the first group by proposing a model that aims to restrict weight modifications. We rely on evidence showing that the lower layers of a deep learning model capture general knowledge while the upper layers capture task-specific knowledge\cite{20,22,47,48}. Under this premise, in the case of a lifelong learning scenario, a model should update its layer weights based on how general or specific these layers should be. We propose the Entropy-based Stability-Plasticity (ESP) method, which relies on an entropy-based criterion to decide how much a model has to modify the weights in each of its layers. Specifically, ESP augments each layer of an encoder with a branch layer that computes an entropy-based plasticity factor during the forward pass, and dynamically updates the layer weights based on these plasticity factors during the backward pass. This way, when a new training example arrives, the model calculates how much we can update the weights of the model via a plasticity factor. We found in our experiments that in some cases, our method forces the model to freeze some layers, setting their gradients to zero, which encourages reusing past knowledge and reduces training time.

We demonstrate the effectiveness of our method experimentally by running a diverse set of experiments and comparing our results against well-known baselines. Unlike previous work in the field, we evaluated ESP on both, vision and natural language domains. The code is publicly available for further replicability and future research\footnote{https://github.com/vgaraujov/ESP-CL}.

2. Related Work

Previous methods have tackle the problem of Continual Learning (CL) using three main strategies. The first group of methods focus on limiting the plasticity of learning new
The typical approach penalizes weight modifications or freezes a subset of the model. This can be achieved by adding weight regularizations [19, 49], using masks to freeze parts of the model [15, 27, 28], or based those regularization on gradients behavior [4, 37].

The second strategy is to use dynamic architectures by increase network capacity and adding extra parameters [10, 36], or by finding new paths of relevant weights to solve each task [12], freezing used weights, and limiting learning of new tasks. Other approaches use different functions as components in the network, either Hypernetworks [45], Deep Artificial Neurons (DANs) [3], Compositional Structures [30, 32], or novel learning strategies [16], so that network components can be more flexible when learning new tasks.

The third strategy is based on memory-based methods. This strategy mitigate catastrophic forgetting by inserting data from past tasks into the training process of new tasks, continuously re-training previous tasks [13], either with raw samples [5, 6, 35], or minimizing gradient interference [4, 25]. Later works such as [23, 40] train generator functions (GANs) or autoencoders [18] to generate elements from past distributions. They seek memory-efficiency by generating examples instead of saving real data. Similarly, [2, 17] seek to be memory-efficient by saving feature vectors of instances from previous tasks, while learning a transformation from the feature space of past tasks to current ones. Other works use memory to create prototypes that can represent classes [7, 35], either for use as distillation or classification vectors.

3. Method

In this work, we consider a lifelong (continual) learning setup. Each task \( t \) consists of a new data distribution \( D^t = (X^t, Y^t) \), where \( X^t \) denotes the input instances and \( Y^t \) denotes the instance labels. The goal is to train a classification model \( f : X \rightarrow Y \) using data from a sequence of \( T \) tasks: \( D = \{ D^1, ..., D^T \} \). Following the Class Incremental Learning setup for CL [42], each task is presented sequentially to the model without a task descriptor. Also, in our setup, we only allow each item to be viewed once, such as in online learning scenarios [8, 26].

A model in this configuration consists of an encoder and a decoder. The encoder takes an input \( x \) and produces a vector representation. The encoder could be any kind of model, for instance, a Transformer [43] for text classification, or a ResNet [14] for image classification. The decoder is a linear transformation and a softmax layer to predict the class \( y \) of an input \( x \). Note that because there is no task descriptor, the decoder predicts across all classes.

Next, we explain the proposed method and how it is incorporated into the learning process.

![Figure 1. Overview of the method. During the forward step, the backbone processes an example and generates prediction and plasticity factor values for each block (left). During the backward pass, the plasticity factor is used to adjust the final amount of modification each layer will have (right).]

### 3.1. Entropy-based Stability-Plasticity (ESP)

Previous works attempt to solve the stability-plasticity dilemma by slowing down learning on certain weights based on how important they are to previously seen tasks [19, 49]. Although effective, these methods neglect that some layers learn general or task-specific patterns [20, 22, 47, 48] and constantly update the weights of the different layers, resulting in interference with the acquired knowledge. Based on this, ESP addresses the stability-plasticity dilemma using a mechanism that allows the model to decide how much each layer should be updated using an entropy-based criterion. This method favors the reuse of previous knowledge existing in the layers by means of little or no modification of their weights and the specialization of the layers in a specific task by means of a high modification of their weights.

Under the CL setup mentioned above, an encoder could be seen as a stack of processing blocks. As shown in Fig. 1, ESP extends each model block \( i \) with a side branch layer \( Br_i \) to generate a classification of the input \( x \):

\[
\hat{y}_i = W^2_i(\sigma(W^1_i f_i(x)))
\]

where \( f_i(x) \) is the output of the block \( i \), \( W^1_i \) and \( W^2_i \) are trainable linear layers, and \( \sigma \) is an activation function.

Later, the vector \( \hat{y}_i \) is used to compute the entropy of the prediction probability distribution for each block:

\[
E(\hat{y}_i) = \sum \hat{y}_{i,t} \log \hat{y}_{i,t}
\]

Finally, a plasticity factor (PF) is calculated as the complementary entropy value. Note that a softmax function is
applied first to find the proportion of entropy corresponding to each block $i$.

\[
PF = (1 - \text{softmax}(E(\hat{y})))
\]  

(3)

This whole process is slightly similar to previous work using branch classifiers and entropy for early exiting [11, 41, 46, 51]. However, in this case, the calculated factors provide the proportion by which the weights of each model layer should be modified. Intuitively, a high value of PF (low entropy) leads to a high modification of the weights, specializing them for the task. On the other hand, a low value of PF (high entropy) leads to smaller changes in weights, reducing interference and catastrophic forgetting.

3.2. Training

Analogous to EWC and SI, which have an additional step to find the importance of each weight after each new task, ESP needs to update each block’s branch layers $Br$ before training a new task. To do that, we first freeze the encoder and decoder and train only the attached layers with a subset of the training set (e.g., replay set). We train the layers as a classification problem, where the output of each layer is compared with ground truth using a Cross Entropy loss.

\[
Loss_i = \text{CrossEntropy}(\hat{y}_i, y_i)
\]  

(4)

Then, to train the backbone (encoder and decoder), the branch layers must be frozen. The reason for freezing the branch layers is to maintain the optimal quality of the decoder classifier. If the branch layers are not frozen, the model layers will no longer be optimized solely for the decoder classifier, which generally worsens its quality. Empirically, we found that joint optimization of branch layers and the backbone also leads to worst results.

This training step uses the information provided by the frozen branch layers to self-regulate the weight update. During the forward pass (Fig. 1 left), the model generates an output $y$ and each $Br$ generates the corresponding $PF$. During the backward pass (Fig. 1 right), the $PF$ scales the gradient of the corresponding block to control the modifications that the task wants to make to the model. Note the encoder and decoder are optimized with the same loss function of Eq. (4). The training of the model with ESP is summarized in Algorithm 1.

4. Experiments

In this paper, we test our approach in the domains of natural language and vision. For a fair and consistent comparison, we use the same CL setup (explained in Sec. 3) and the same baselines for both domains.

### Algorithm 1: ESP Training Process

**Components:**
- $D^t$: Dataset for task $t$.
- $F$: Model.
- $BC$: Branch Layers.

\[
\begin{align*}
BC \leftarrow & \text{TrainBC}(F, D^t) \\
\text{for } x, y \text{ to } D^t \text{ do} \\
& \textbf{# Forward pass} \\
& PF, \hat{y} \leftarrow F(x) \\
& \triangledown g \leftarrow \text{Loss}(y, \hat{y}) \\
& \textbf{# Backward pass} \\
& \triangledown g_{\text{new}} \leftarrow \text{UpdateGrad}(PF, \triangledown g) \\
& F \leftarrow \text{UpdateModel}(F, \triangledown g_{\text{new}})
\end{align*}
\]

4.1. Baselines

One of the most reliable approaches to overcoming catastrophic forgetting is **Replay** [13]. It involves storing a subset of previous inputs (e.g., sentences) and mixing them with more recent inputs to update the model. The replay subset is usually a percentage of data randomly taken from the training set of previous tasks. As our baseline, we use a standard replay strategy with commonly used percentages.

To compare our method, we consider several well-known methods that attempt to address the stability-plasticity dilemma to apply to our primary baseline. We provide a brief description of each below:

1. **Stability**: A method that keeps the encoder weights fixed and trains only the decoder (classifier).
2. **Plasticity**: A method with complete freedom to train the encoder and decoder (classifier).
3. **Linear Plasticity**: A method that, similarly to ESP, uses a factor to scale the gradient of each block. The factors are linearly spaced between 0 and 1 with respect to the model’s number blocks, where 0 is for the first block and 1 for the last one.
4. **O-EWC**: Online EWC [38] introduces a regularization term involving the Fisher information matrix that indicates the importance of each of the parameters to previous tasks.
5. **SI**: Synaptic Intelligence [49] adjusts the plasticity of the model by regularizing the modification of these important weights with a coefficient.

We also consider the class imbalance issue in our experiments. We train all methods in two scenarios: (1) using ONLY the replay set items, similar to [33] but without using a fixed amount of data, and (2) combining ALL data from the current task with the replay set. In the latter case, there
could be a significant class imbalance, but a more considerable amount of data would be available to train the model.

### 4.2. Natural Language

#### 4.2.1 Implementation Details

We use BERT [9], a Transformer-based [43] pre-trained language model, as the encoder. As decoder, following original BERT model, we use the first token (special token [CLS]) of the sequence and a classifier to predict the class. In addition, we use the default BERT vocabulary in our experiments. We use Adam optimizer with a learning rate of $3 \times 10^{-5}$ and a training batch of size 32.

#### 4.2.2 Datasets

We use publicly available text classification datasets from [50]: (1) AGNews classification, (2) Yelp sentiment analysis, (3) Amazon sentiment analysis, (4) DBPedia article classification and (5) Yahoo questions and answers categorization. We follow the same data processing described in [8]. In total, we have 575,000 training examples and 38,000 test examples with 33 classes from all datasets. In addition, we use the originally proposed dataset orders:

(i) Yelp → AGNews → DBPedia → Amazon → Yahoo
(ii) DBPedia → Yahoo → AGNews → Amazon → Yelp
(iii) Yelp → Yahoo → Amazon → DBpedia → AGNews
(iv) AGNews → Yelp → Amazon → Yahoo → DBpedia

#### 4.2.3 Results

Our results in the natural language experiments are shown in Tab. 1 and Tab. 2. The Plasticity model performs well compared to the Stability version. This was expected because the Stability model limits the flexibility of the model to acquire new knowledge. Interestingly, the Linear Plasticity model outperforms the Plasticity model in almost all experiments, supporting the hypothesis that lower blocks need minor updates, and modifying higher blocks leads to better results. On the other hand, we find that O-EWC and SI perform similarly or even worse than the Plasticity model in some cases. This is because these methods perform better under a setup in which a task id is provided.

In contrast, ESP outperforms all baselines when trained on all experiments. We found that the increase in performance is consistent in the ONLY and the ALL scenarios. Overall, ESP achieves an accuracy gain of 0.34 and 0.44 points on average (across all replay percentages) over SI and O-EWC, respectively. ESP also exceeds Linear Plasticity, which means that dynamic plasticity factors are useful to avoid forgetting.

### 4.3. Vision

#### 4.3.1 Implementation Details

For the visual experiments, as encoder we use a pre-trained ResNet-18 [14], and following previous works, a linear classifier for the decoder. Instead of using a branch clas-
sifier for each layer, here we use a branch classifier for each block of ResNet. The output of each block is reduced to one element per channel by averaging the values of the activation maps, this vector goes through the branch functions to find the scores.

We use SGD as our optimizer, using a learning rate of $10^{-3}$ and momentum factor equal to 0.9. We run all of our experiments using a batch size of 32. For EWC and SI, we try different values of regularization coefficient, at the end we use 2000 for EWC and 0.1 for SI.

### 4.3.2 Datasets

Following previous works [49], we use CIFAR10 [21] equally divided into 5 tasks. We use the implementation from Avalanche [24] to generate the different sequence. We run each experiments 3 times with different seeds and we average the results.

### 4.3.3 Results

Similar to the natural language experiments, we use a pre-trained model, specifically a ResNet pre-trained on ImageNet. However, a big difference between text and images is the number of elements saved in the replay set. As images weigh more than phrases, we do experiments saving between 1 and 5% of previous tasks.

Results shown in Tab. 3 and Tab. 4, shows that regularization methods have better results than Stability, Plasticity and Linear Plasticity. On the other hand, modifying the weights without a constraint does not achieve good results either. In general, there is no clear advantage between the regularization methods in either of the two scenarios. We hypothesize the results diverge from the natural language experiments for two reasons: The first is the difference in the number of blocks, indicating that ESP may take advantage of deeper networks. The second is because there is a difference in the input distribution between ImageNet and CIFAR10, we expand this hypothesis a bit more in Sec. 4.4.

The motivation behind regularization methods is to reduce the plasticity of the model to prevent forgetting. By minimizing the modification of relevant weights in the future training process, these methods force future tasks to reuse knowledge even if those patterns are irrelevant or hurtful to new tasks. For this reason, we believe it is essential for these methods to learn representations that may be useful across tasks. For example, if the model weights are too specific for a task and we freeze all layers, the model would not find a correct classification.

Given the above, we believe that one reason why regularization methods do not perform well in Class Incremental scenarios is the inability to find good representations and thus the need to start from a pre-trained model. To prove this hypothesis and compare our results, we change the pre-trained ResNet-18 to one initialize randomly. The results show that none of the three regularizing methods has good results, being outperformed by the Plasticity method in almost all replay percentages. The advantage of this method is that it has complete flexibility to adjust the weights. This advantage leads to the weights to learn new representations, not tied to representations particular to the previous tasks. It is of little use to reduce the modification of past relevant weights if they can not be reused for future tasks.

### 4.4. Further Analysis

This section discusses the plasticity factors resulting from our experiments in both domains. Fig. 2 shows the average of the plasticity factors for our experiments on natural language (Fig. 2a) and vision (Fig. 2b).

We use a pre-trained BERT, a 12 block model for natural language. Interestingly, the plasticity factor of the lower layers (1 to 4) is 0, and the factor constantly increases for the upper layers. Which means null modification in the
lower layers and high modification in the upper layers. Our method leverages existing general knowledge in lower layers, which is general knowledge, while updating upper layers with task-specific knowledge.

Regarding vision, we use a pre-trained ResNet-18, a four-block model. Here our method behaves slightly similar to the natural language model, where the block with higher modifications is the last one. However, unlike the natural language results, the first block has a higher plasticity factor than blocks 2 and 3. This result may be due to the data used for pre-train the model. The BERT model is pre-trained on a massive corpus of different domains and topics, promoting the lower layers to be general for any task. On the other hand, ResNet-18 was pre-trained on ImageNet, which has much higher resolution images than CIFAR10. The basic patterns are expected to be different between both datasets, explaining the high plasticity factor of the first block.

In general, the plasticity factors for natural language and vision are higher at the last layers. However, no one reaches complete plasticity, which means those layers retain some specialization acquired in previous tasks. Also, Fig. 2 it shows that different amounts of percentages of replay sets have similar results, indicating that the plasticity of the network mainly depends on the current input.

Finally, we argue that ESP could be computationally efficient for cases like natural language experiments. Fig. 3 shows the training runtime of ESP and baselines under the 20% replay setup. We use a GPU NVIDIA GeForce RTX 3090 and a CPU AMD EPYC 7502 for these experiments. The Stability model is the more efficient, with ~45 minutes, because it only updates the decoder layer. The Plasticity and Linear Plasticity model take ~127 minutes because the entire backbone is trained. On the other hand, O-EWC and SI methods take a total time of ~164 and ~167 minutes. For a fair comparison, we divide the total execution time into the backbone training time (blue) and the regularizer time (orange) because both models include an additional process to calculate the importance of the weights. O-EWC takes ~149 minutes of backbone training time and ~15 minutes to find the importance of each weight. SI takes ~150 minutes of backbone training time and ~17 minutes of regularization time. Note that the backbone training time of these methods is superior to the plasticity model because they include an additional calculation loss based on the importance of the previously calculated weights.

Concerning our method, ESP finishes its training in ~123 minutes. Analogous to O-EWC and SI, ESP has an additional process to tune the branch layer, which takes ~12 minutes. This time is similar compared to O-EWC and SI regularizer time. However, if we compare the training time of the spine, ESP is remarkably efficient. ESP takes ~111 minutes, which is less than other methods, including the Plasticity model. It happens because ESP sometimes forces the model not to update some layers, allowing those layers to be frozen on the fly, resulting in decreased training time.

5. Conclusion

In this paper, we introduced ESP, a method based on an entropy-based criterion to decide how much a model has to modify the weights of each of its layers. ESP augments each block of an encoder with branch layers that computes an entropy-based plasticity factor used to update layer weights dynamically. Our experiments in the natural language and vision domains show the effectiveness of our model in leveraging prior knowledge by not updating lower layers and specializing other layers by updating higher layers. In addition, we show that in the case of the natural language model, our method promotes computational efficiency since it forces not to update some layers.

Among the ideas for future work, we consider testing the hypothesis that ESP works better on networks with more blocks than a Resnet-18, such as vision Transformers. Also, we would like to extend ESP to an utterly online scenario.

Acknowledgement

We thanks the reviewers for their positive remarks and some valuable suggestions. This work was supported by the European Research Council Advanced Grant 788506 and the National Center for Artificial Intelligence CENIA FB210017, Basal ANID.

References

of the European Conference on Computer Vision (ECCV), pages 139–154, 2018. 1


3727


