Exclusive Supermask Subnetwork Training for Continual Learning

Prateek Yadav & Mohit Bansal
Department of Computer Science
UNC Chapel Hill
{praty,mbansal}@cs.unc.edu

Abstract

Continual Learning (CL) methods focus on accumulating knowledge over time while avoiding catastrophic forgetting. Recently, Wortsman et al. (2020) proposed a CL method, SupSup, which uses a randomly initialized, fixed base network (model) and finds a supermask for each new task that selectively keeps or removes each weight to produce a subnetwork. They prevent forgetting as the network weights are not being updated. Although there is no forgetting, the performance of SupSup is sub-optimal because fixed weights restrict its representational power. Furthermore, there is no accumulation or transfer of knowledge inside the model when new tasks are learned. Hence, we propose EXSSNET (Exclusive Supermask SubNetwork Training), that performs exclusive and non-overlapping subnetwork weight training. This avoids conflicting updates to the shared weights by subsequent tasks to improve performance while still preventing forgetting. Furthermore, we propose a novel KNN-based Knowledge Transfer (KKT) module that utilizes previously acquired knowledge to learn new tasks better and faster. We demonstrate that EXSSNET outperforms strong previous methods on both NLP and Vision domains while preventing forgetting. Moreover, EXSSNET is particularly advantageous for sparse masks that activate 2-10% of the model parameters, resulting in an average improvement of 8.3% over SupSup. Furthermore, EXSSNET scales to a large number of tasks (100). Our code is available at https://github.com/prateeky2806/exessnet.

1 Introduction

Artificial intelligence aims to develop agents that can learn to accomplish a set of tasks. Continual Learning (CL) (Ring, 1998; Thrun, 1998) is crucial for this, but when a model is sequentially trained on different tasks with different data distributions, it can lose its ability to perform well on previous tasks, a phenomenon is known as catastrophic forgetting (CF) (McCloskey and Cohen, 1989; Zhao and Schmidhuber, 1996; Thrun, 1998). This is caused by the lack of access to data from previous tasks, as well as conflicting updates to shared model parameters when sequentially learning multiple tasks, which is called parameter interference (McCloskey and Cohen, 1989).

Recently, some CL methods avoid parameter interference by taking inspiration from the Lottery Ticket Hypothesis (Frankle and Carbin, 2018) and Supermasks (Zhou et al., 2019) to exploit the expressive power of sparse subnetworks. Given that we have a combinatorial number of sparse subnetworks inside a network, Zhou et al. (2019) noted that even within randomly weighted neural networks, there exist certain subnetworks known as supermasks that achieve good performance. A supermask is a sparse binary mask that selectively keeps or removes each connection in a fixed and randomly initialized network to produce a subnetwork with good performance on a given task. We call this the subnetwork as supermask subnetwork that is shown in Figure 1, highlighted in red weights. Building upon this idea, Wortsman et al. (2020) proposed a CL method, SupSup, which initializes a network with fixed and random weights and then learns a different supermask for each new task. This allows them to prevent catastrophic forgetting (CF) as there is no parameter interference (because the model weights are fixed).

Although SupSup (Wortsman et al., 2020) prevents CF, there are some problems with using supermasks for CL: (1) Fixed random model weights in SupSup limits the supermask subnetwork’s representational power resulting in sub-optimal performance. (2) When learning a task, there is no mechanism for transferring learned knowledge from previous tasks to better learn the current task. Moreover, the model is not accumulating knowledge over time as the weights are not being updated.
To overcome the aforementioned issues, we propose our method, ExSSNeT (Exclusive Supermask SubNetwork Training), pronounced as ‘excess-net’, which first learns a mask for a task and then selectively trains a subset of weights from the supermask subnetwork. We train the weights of this subnetwork via exclusion that avoids updating parameters from the current subnetwork that have already been updated by any of the previous tasks. In Figure 1, we demonstrate ExSSNeT that also helps us to prevent forgetting. Training the supermask subnetwork’s weights increases its representational power and allows ExSSNeT to encode task-specific knowledge inside the subnetwork (see Figure 2). This solves the first problem and allows ExSSNeT to perform comparably to a fully trained network on individual tasks; and when learning multiple tasks, the exclusive subnetwork training improves the performance of each task while still preventing forgetting (see Figure 3).

To address the second problem of knowledge transfer, we propose a $k$-nearest neighbors-based knowledge transfer (KKT) module that is able to utilize relevant information from the previously learned tasks to improve performance on new tasks while learning them faster. Our KKT module uses KNN classification to select a subnetwork from the previously learned tasks that has better than random predictive power for the current task and use it as a starting point to learn the new tasks.

Next, we show our method’s advantage by experimenting with both natural language and vision tasks. For natural language, we evaluate on WebNLP classification tasks (de Masson d’Autume et al., 2019) and GLUE benchmark tasks (Wang et al., 2018), whereas, for vision, we evaluate on SplitMNIST (Zenke et al., 2017), SplitCIFAR100 (De Lange and Tuytelaars, 2021), and SplitTinyImageNet (Buzzega et al., 2020) datasets. We show that for both language and vision domains, ExSSNeT outperforms multiple strong and recent continual learning methods based on replay, regularization, distillation, and parameter isolation. For the vision domain, ExSSNeT outperforms the strongest baseline by 4.8% and 1.4% on SplitCIFAR and SplitTinyImageNet datasets respectively, while surpassing multitask model and bridging the gap to training individual models for each task. In addition, for GLUE datasets, ExSSNeT is 2% better than the strongest baseline methods and surpasses the performance of multitask learning that uses all the data at once. Moreover, ExSSNeT obtains an average improvement of 8.3% over SupSup for sparse masks with $2 - 10\%$ of the model parameters and scales to a large number of tasks (100).

Furthermore, ExSSNeT with the KKT module learns new tasks in as few as 30 epochs compared to 100 epochs without it, while achieving 3.2% higher accuracy on the SplitCIFAR100 dataset. In summary, our contributions are listed below:

- We propose a simple and novel method to improve mask learning by combining it with exclusive subnetwork weight training to improve CL performance while preventing CF.
- We propose a KNN-based Knowledge Transfer (KKT) module for supermask initialization that
dynamically identifies previous tasks to transfer knowledge to learn new tasks better and faster.

- Extensive experiments on NLP and vision tasks show that ExSSN\text{e}T outperforms strong baselines and is comparable to multitask model for NLP tasks while surpassing it for vision tasks. Moreover, ExSSN\text{e}T works well for sparse masks and scales to a large number of tasks.

2 Motivation

Using sparsity for CL is an effective technique to learn multiple tasks, i.e., by encoding them in different subnetworks inside a single model. SupSup (Wortsman et al., 2020) is an instantiation of this that initializes the network weights randomly and then learns a separate supermask for each task (see Figure 7). They prevent CF because the weights of the network are fixed and never updated. However, this is a crucial problem as discussed below.

Problem 1 - Sub-Optimal Performance of Supermask: Although fixed network weights in SupSup prevent CF, this also restricts the representational capacity, leading to worse performance compared to a fully trained network. In Figure 2, we report the test accuracy with respect to the fraction of network parameters selected by the mask, i.e., the mask density for an underlying ResNet\text{18} model on a single 100-way classification on CIFAR100 dataset. The fully trained ResNet\text{18} model (dashed green line) achieves an accuracy of 63.9\%. Similar to Zhou et al. (2019), we observe that the performance of SupSup (yellow dashed line) is at least 8.3\% worse compared to a fully trained model. As a possible partial remedy, we propose a simple solution, SS\text{e}N\text{e}T (Supermask SubNet\text{work Training}), that first finds a subnetwork for a task and then trains the subnetwork’s weights. This increases the representational capacity of the subnetwork because there are more trainable parameters. For a single task, the test accuracy of SS\text{e}N\text{e}T is better than SupSup for all mask densities and matches the performance of the fully trained model beyond a density threshold. But as shown below, when learning multiple tasks sequentially, SS\text{e}N\text{e}T gives rise to parameter interference that results in CF.

Problem 2 - Parameter Interference Due to Subnetwork Weight Training for Multiple Tasks: Next, we demonstrate that when learning multiple tasks sequentially, SS\text{e}N\text{e}T can still lead to CF. In Figure 3, we report the average test accuracy versus the fraction of overlapping parameters between the masks of different tasks, i.e., the sparse overlap (see Equation 2) for five different 20-way classification tasks from SplitCIFAR\text{100} dataset with ResNet\text{18} model. We observe that SS\text{e}N\text{e}T outperforms SupSup for lower sparse overlap but as the sparse overlap increases, the performance declines because the supermask subnetworks for different tasks have more overlapping (common) weights (bold dashed lines in Figure 1). This leads to higher parameter interference resulting in increased forgetting which suppresses the gain from subnetwork weight training.

Our final proposal, ExSSN\text{e}T, resolves both of these problems by selectively training a subset of the weights in the supermask subnetwork to prevent parameter interference. When learning multiple tasks, this prevents CF, resulting in strictly better performance than SupSup (Figure 3) while having the representational power to match bridge the gap with fully trained models (Figure 2).

3 Method

As shown in Figure 1, when learning a new task $t_i$, ExSSN\text{e}T follows three steps: (1) We learn a supermask $M_i$ for the task; (2) We use all the previous tasks’ masks $M_1, \ldots, M_{i-1}$ to create a free parameter mask $M_i^{\text{free}}$, that finds the parameters selected by the mask $M_i$ that were not selected by any of the previous masks; (3) We update the weights corresponding to the mask $M_i^{\text{free}}$ as this avoids parameter interference. Now, we formally
describe all the step of our method EXSSNExT (Exclusive Supermask SubNetwork Training) for a Multi-layer perceptron (MLP).

**Notation:** During training, we can treat each layer \( l \) of an MLP network separately. An intermediate layer \( l \) has \( n_l \) nodes denoted by \( \mathcal{V}^{(l)} = \{v_1, \ldots, v_{n_l}\} \). For a node \( v \) in layer \( l \), let \( \mathcal{I}_v \) denote its input and \( \mathcal{Z}_v = \sigma(\mathcal{I}_v) \) denote its output, where \( \sigma(\cdot) \) is the activation function. Given this notation, \( \mathcal{I}_v \) can be written as \( \mathcal{I}_v = \sum_{u \in \mathcal{V}^{(l-1)}} w_{uv} \mathcal{Z}_u \), where \( w_{uv} \) is the network weight connecting node \( u \) to node \( v \). The complete network weights for the MLP are denoted by \( W \). When training the task \( t_i \), we have access to the supermasks from all previous tasks \( \{M_j\}_{j=1}^{i-1} \) and the model weights \( W^{(i-1)} \) obtained after learning task \( t_{i-1} \).

### 3.1 EXSSNExT: Exclusive Supermask SubNetwork Training

**Finding Supermasks:** Following Wortsman et al. (2020), we use the algorithm of Ramanujan et al. (2019) to learn a supermask \( M_i \) for the current task \( t_i \). The supermask \( M_i \) is learned with respect to the underlying model weights \( W^{(i-1)} \) and the mask selects a fraction of weights that lead to good performance on the task without training the weights. To achieve this, we learn a score \( s_{uv} \) for each weight \( w_{uv} \), and once trained, these scores are thresholded to obtain the mask. Here, the input to a node \( v \) is \( \mathcal{I}_v = \sum_{u \in \mathcal{V}^{(l-1)}} w_{uv} \mathcal{Z}_u m_{uv} \), where \( m_{uv} = h(s_{uv}) \) is the binary mask value and \( h(.) \) is a function which outputs 1 for top-k% of the scores in the layer with \( k \) being the mask density. Next, we use a straight-through gradient estimator (Bengio et al., 2013) and iterate over the current task’s data samples to update the scores for the corresponding supermask \( M_i \) as follows,

\[
s_{uv} = s_{uv} - \alpha \hat{g}_{s_{uv}}; \hat{g}_{s_{uv}} = \frac{\partial C}{\partial s_{uv}} = \frac{\partial C}{\partial \mathcal{Z}_v} w_{uv} \mathcal{Z}_u
\]

**Finding Exclusive Mask Parameters:** Given a learned mask \( M_i \), we use all the previous tasks’ masks \( M_1, \ldots, M_{i-1} \) to create a free parameter mask \( M_i^{\text{free}} \), that finds the parameters selected by the mask \( M_i \) that were not selected by any of the previous masks. We do this by (1) creating a new mask \( M_{i,j-1} \) containing all the parameters already updated by any of the previous tasks by taking a union of all the previous masks \( \{M_j\}_{j=1}^{i-1} \) by using the logical or operation, and (2) Then we obtain a mask \( M_i^{\text{free}} \) by taking the intersection of all the network parameters not used by any previous task which is given by the negation of the mask \( M_{1:i-1} \) with the current task mask \( M_i \) via a logical and operation. Next, we use this mask \( M_i^{\text{free}} \) for the exclusive supermask subnetwork weight training.

**Exclusive Supermask Subnetwork Weight Training:** For training the subnetwork parameters for task \( t_i \), given the free parameter mask \( M_i^{\text{free}} \), we perform the forward pass on the model as \( \text{model}(x, W \odot \hat{M}_i) \) where \( \hat{M}_i = M_i^{\text{free}} + ((1 - M_i^{\text{free}}) \odot M_i).detach() \), where \( \odot \) is the elementwise multiplication. Hence, \( \hat{M}_i \) allows us to use all the connections in \( M_i \) during the forward pass of the training but during the backward pass, only the parameters in \( M_i^{\text{free}} \) are updated because the gradient value is 0 for all the weights \( w_{uv} \) where \( m_{uv} = 0 \). While during the inference on task \( t_i \), we use the mask \( M_i \). In contrast, SSNet uses the task mask \( M_i \) both during the training and inference as \( \text{model}(x, W^{(i-1)} \odot M_i) \). This updates all the parameters in the mask including the parameters that are already updated by previous tasks that result in CF. Therefore, in cases where the sparse overlap is high, EXSSNExT is preferred over SSNet. To summarize, EXSSNExT circumvents the CF issue of SSNet while benefiting from the subnetwork training to improve overall performance as shown in Figure 3.

### 3.2 KKT: Knn-Based Knowledge Transfer

When learning multiple tasks, it is a desired property to transfer information learned by the previous tasks to achieve better performance on new tasks and to learn them faster (Biesielska et al., 2020). Hence, we propose a K-Nearest Neighbours (KNN) based knowledge transfer (KKT) module that uses KNN classification to dynamically find the most relevant previous task (Veniat et al., 2021) to initialize the supermask for the current task. To be more specific, before learning the mask \( M_i \) for the current task \( t_i \), we randomly sample a small fraction of data from task \( t_{i-1} \) and split it into a train and test set. Next, we use the trained subnetworks of each previous task \( t_1, \ldots, t_{i-1} \) to obtain features on this sampled data. Then we learn \( i-1 \) independent KNN-classification models using these features. Then we evaluate these \( i-1 \) models on the sampled test set to obtain accuracy scores which denote the predictive power of features from each previous task for the current task. Finally, we select the previous task with the highest accuracy.
on the current task. If this accuracy is better than random then we use its mask to initialize the current task’s supermask. This enables ExSSNeT to transfer information from the previous task to learn new tasks better and faster. We note that the KKT module is not limited to SupSup and can be applied to a broader category of CL methods that introduce additional parameters for new tasks.

4 Experiments
4.1 Experimental Setup and Training Details

Datasets: For natural language domain, we follow the shared text classification setup of IDBR (Huang et al., 2021), LAMOL (Sun et al., 2019), and MBPA++ (De Lange et al., 2019) to sequentially learn five text classification tasks; (1) Yelp Sentiment analysis (Zhang et al., 2015); (2) DB-Pedia for Wikipedia article classification (Mendes et al., 2012) (3) Yahoo! Answer for Q&A classification (Chang et al., 2008); (4) Amazon sentiment analysis (McAuley and Leskovec, 2013) (5) AG News for news classification (Zhang et al., 2015). We call them WebNLP classification tasks for easier reference. While comparing with the previous state-of-the-art text methods, we use the same training and test set as IDBR and LAMOL containing 115,000/500/7,600 Train/Val/Test examples. For our ablation studies, we follow IDBR and use a sampled dataset, please see Appendix Table 7 for statistics. Additionally, we create a CL benchmark using the popular GLUE classification tasks (Wang et al., 2018) consisting of more than 5k train samples. We use the official validation split as test data and use 0.1% of the train data to create a validation set. Our final benchmark includes five tasks; MNLI (353k/39k/9.8k), QQP (327k/36k/40k), QNLI (94k/10k/5.4k), SST-2 (60k/6.7k/872), CoLA (7.6k/856/1k). For vision experiments, we follow SupSup and use three CL benchmarks, SplitMNIST (Zenke et al., 2017) SplitCIFAR100 (Chaudhry et al., 2018) , and SplitTiny-ImageNet (Buzzega et al., 2020) datasets with 10, 100 and 200 total classes respectively.

Metrics: We follow Chaudhry et al. (2018) and evaluate our model after learning task t on all the tasks, denoted by T. This gives us an accuracy matrix A ∈ R^n×n, where a_{i,j} represents the classification accuracy on task j after learning task i. We want the model to perform well on all the tasks it has been learned. This is measured by the average accuracy, \( A(T) = \frac{1}{N} \sum_{k=1}^{N} a_{N,k} \), where N is the number of tasks. Next, we want the model to retain performance on the previous tasks when learning multiple tasks. This is measured by the forgetting metric (Lopez-Paz and Ranzato, 2017), \( F(T) = \frac{1}{N-1} \sum_{k=1}^{N-1} (\max_{t \in \{1,...,N-1\}} a_{k,t} - a_{N,t}) \). This is the average difference between the maximum accuracy obtained for task t and its final accuracy. Higher accuracy and lower forgetting are desired.

Sparse Overlap to Quantify Parameter Interference: Next, we propose sparse overlap, a measure to quantify parameter interference for a task i, i.e., the fraction of the parameters in mask \( M_i \) that are already updated by some previous task. For a formal definition refer to Appendix A.1

Previous Methods and Baselines: For both vision and language (VL) tasks, we compare with: (VL.1) Naive Training (Yogatama et al., 2019): where all model parameters are sequentially trained/finetuned for each task. (VL.2) Experience Replay (ER) (de Masson d’Autume et al., 2019): we replay previous tasks examples when we train new tasks; (VL.3) Multitask Learning (Crawshaw, 2020): where all the tasks are used jointly to train the model; (VL.4) Individual Models: where we train a separate model for each task. This is considered an upper bound for CL; (VL.5) Sup-sup (Wortsman et al., 2020). For natural language (L), we further compare with the following methods: (L.6) Regularization (Huang et al., 2021): Along with the Replay method, we regularize the hidden states of the BERT classifier with an L2 loss term; We show three Adapter BERT (Houlsby et al., 2019) variants, (L.7) AdaptBERT + FT where we have single adapter which is finetuned for all task; (L.8) AdaptBERT + ER where a single adapter is finetuned with replay; (L.9) MultiAdaptBERT where a separate adapter is finetuned for each task; (L.10) Prompt Tuning (Li and Liang, 2021) that learns 50 different continuous prompt tokens for each task. (L.11) MBPA++ (de Masson d’Autume et al., 2019) perform replay with random examples during training and does local adaptation during inference to select replay example; (L.12) LAMOL (Sun et al., 2019) uses a language model to generate pseudo-samples for previous tasks for replay; (L.13) IDBR (Huang et al., 2021) disentangles hidden representations into generic and task-specific representations and regularizes them while also performing replay. For vision task (V), we additionally compare with two popular regularization
Table 1: Comparing average test accuracy (and forgetting metric ↓) for multiple tasks and sequence orders with state-of-the-art (SotA) methods. Results with ↑ are taken from (Huang et al., 2021).

<table>
<thead>
<tr>
<th>Method (↑)</th>
<th>GLUE</th>
<th>WebNLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order (→)</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>Multitask</td>
<td>79.9 (0.0)</td>
<td>77.2 (0.0)</td>
</tr>
<tr>
<td>Individual</td>
<td>87.5 (0.0)</td>
<td>79.5 (0.0)</td>
</tr>
<tr>
<td>FT</td>
<td>24.7 (53.4)</td>
<td>20.8 (68.4)</td>
</tr>
<tr>
<td>AdaptBERT + FT</td>
<td>76.8 (3.8)</td>
<td>73.2 (3.0)</td>
</tr>
<tr>
<td>AdaptBERT + Replay</td>
<td>78.5 (0.0)</td>
<td>76.7 (0.0)</td>
</tr>
<tr>
<td>MultiAdaptBERT</td>
<td>76.3 (0.0)</td>
<td>66.3 (0.0)</td>
</tr>
<tr>
<td>Prompt Tuning</td>
<td>72.5 (8.8)</td>
<td>76.0 (2.8)</td>
</tr>
<tr>
<td>Regularization</td>
<td>77.7 (4.8)</td>
<td>75.1 (2.7)</td>
</tr>
<tr>
<td>MBPA++†</td>
<td>-</td>
<td>74.9 (-)</td>
</tr>
<tr>
<td>LAMOL†</td>
<td>-</td>
<td>76.1 (-)</td>
</tr>
<tr>
<td>IDBR</td>
<td>73.0 (6.8)</td>
<td>75.9 (2.7)</td>
</tr>
<tr>
<td>SupSup</td>
<td>78.3 (0.0)</td>
<td>75.9 (0.0)</td>
</tr>
<tr>
<td>SSN</td>
<td>78.4 (3.6)</td>
<td>76.3 (0.3)</td>
</tr>
<tr>
<td>EXSSN+T</td>
<td>80.5 (0.0)</td>
<td>77.0 (0.0)</td>
</tr>
</tbody>
</table>

Table 2: Average accuracy (Forgetting metric ↓) on all tasks for vision. For our method, we report the results are averaged over three random seeds.

<table>
<thead>
<tr>
<th>Method (S)</th>
<th>MNIST</th>
<th>CIFAR100</th>
<th>TinyImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiSup</td>
<td>96.2 (0.0)</td>
<td>55.0 (0.0)</td>
<td>45.7 (0.0)</td>
</tr>
<tr>
<td>Individual</td>
<td>99.7 (0.0)</td>
<td>77.5 (0.0)</td>
<td>53.7 (0.0)</td>
</tr>
<tr>
<td>Naive Sequential</td>
<td>50.3 (25.0)</td>
<td>19.3 (73.7)</td>
<td>11.5 (43.9)</td>
</tr>
<tr>
<td>EWC</td>
<td>96.1 (4.5)</td>
<td>32.4 (60.5)</td>
<td>20.5 (52.1)</td>
</tr>
<tr>
<td>SI</td>
<td>99.2 (0.6)</td>
<td>46.1 (57.8)</td>
<td>19.5 (46.2)</td>
</tr>
<tr>
<td>LwF</td>
<td>99.2 (0.8)</td>
<td>29.5 (70.2)</td>
<td>18.1 (56.5)</td>
</tr>
<tr>
<td>AGEM</td>
<td>98.3 (1.9)</td>
<td>52.1 (42.0)</td>
<td>21.6 (54.9)</td>
</tr>
<tr>
<td>ER</td>
<td>99.2 (0.6)</td>
<td>60.1 (27.5)</td>
<td>35.6 (36.0)</td>
</tr>
<tr>
<td>DER</td>
<td>98.9 (1.2)</td>
<td>62.5 (28.4)</td>
<td>35.9 (37.7)</td>
</tr>
<tr>
<td>DER++</td>
<td>98.3 (1.8)</td>
<td>62.5 (27.5)</td>
<td>36.2 (35.7)</td>
</tr>
<tr>
<td>CGATE</td>
<td>99.6 (0.0)</td>
<td>60.1 (0.0)</td>
<td>49.2 (0.0)</td>
</tr>
<tr>
<td>SupSup</td>
<td>99.6 (0.0)</td>
<td>62.1 (0.0)</td>
<td>50.6 (0.0)</td>
</tr>
<tr>
<td>SSN</td>
<td>99.7 (0.0)</td>
<td>23.9 (54.4)</td>
<td>49.6 (1.9)</td>
</tr>
<tr>
<td>EXSSN+T</td>
<td>99.7 (0.0)</td>
<td>67.3 (0.0)</td>
<td>52.0 (0.0)</td>
</tr>
</tbody>
</table>

Table 4: Average accuracy (Forgetting metric ↓) on all tasks for vision. For our method, we report the results are averaged over three random seeds.

4.2 Main Results

Q1. Does Supermask Subnetwork Training Help? In these experiments, we show that EXSSNeT outperforms multiple strong baseline methods including SupSup. For our main language experiments in Table 1, we sequentially learn multiple task orders, S1 - S5, corresponding to the GLUE and WebNLP benchmarks. These task orders are listed in Appendix Table 6. We report the average test accuracy (and forgetting in parentheses). For natural language, we perform better than previous SOTA CL methods in four out of five cases, across multiple task orders, and in aggregate. Specifically, on the GLUE benchmark, EXSSNeT is at least 2.0% better than other methods while avoiding CF. Furthermore, EXSSNeT either outperforms or is close to the performance of the multitasking baseline which is a strong baseline for CL methods.

For vision tasks, we split the MNIST, CIFAR100, and TinyImageNet datasets into five different tasks with an equal number of disjoint classes and report results. From Table 2, we observe that EXSSNeT leads to a 4.8% and 1.4% improvement over the strongest baseline for Split-CIFAR100 and Split-TinyImageNet datasets. Furthermore, both EXSSNeT and SupSup outperform the multitask baseline. Moreover, EXSSNeT bridges the gap to individually trained models significantly, for TinyImageNet we reach within 1.7% of individual
Table 3: Average test accuracies ↑ (and gains from KKT) when using the KKT knowledge sharing module.

<table>
<thead>
<tr>
<th>Method</th>
<th>S-MNIST</th>
<th>S-CIFAR100</th>
<th>S-TinyImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupSup</td>
<td>99.6</td>
<td>62.1</td>
<td>50.6</td>
</tr>
<tr>
<td>+ KKT</td>
<td>99.6 [+0.0]</td>
<td>67.1 [+5.0]</td>
<td>53.3 [+2.7]</td>
</tr>
<tr>
<td>SSNet</td>
<td>99.7</td>
<td>23.9</td>
<td>49.6</td>
</tr>
<tr>
<td>+ KKT</td>
<td>99.3 [+0.4]</td>
<td>33.5 [0.4]</td>
<td>51.8 [+2.2]</td>
</tr>
<tr>
<td>ExSSNeT</td>
<td>99.7</td>
<td>67.3</td>
<td>52.0</td>
</tr>
<tr>
<td>+ KKT</td>
<td>99.7 [+0.0]</td>
<td>70.5 [+3.2]</td>
<td>54.0 [+2.0]</td>
</tr>
</tbody>
</table>

Q2. Can KKT Knowledge Transfer Module Share Knowledge Effectively? In Table 3, we show that adding the KKT module to ExSSNeT, SSNeT, and SupSup improves performance on vision benchmarks. The experimental setting here is similar to Table 2. We observe across all methods and datasets that the KKT module improves average test accuracy. Specifically, for the Split-CIFAR100 dataset, the KKT module results in 5.0% and 3.2% improvement for SupSup and ExSSNeT respectively; while for Split-TinyImageNet, ExSSNeT + KKT outperforms the individual models. We observe a performance decline for SSNeT when using KKT because KKT promotes sharing of parameters across tasks which can lead to worse performance for SSNeT. Furthermore, ExSSNeT + KKT outperforms all other methods on both the Split-CIFAR100 and Split-TinyImageNet datasets. For ExSSNeT + KKT, the average sparse overlap is 49.6% across all three datasets (see appendix Table 11). These results suggest that combining weight training with the KKT module leads to further improvements.

Q3. Can KKT Knowledge Transfer Module Improve Learning Speed of Subsequent Tasks? Next, we show that the KKT module enables us to learn new tasks faster. To demonstrate this, in Figure 4 we plot the running mean of the validation accuracy vs epochs for different tasks from the Split-CIFAR100 experiment in Table 3. We show curves for ExSSNeT with and without the KKT module and omit the first task as both these methods are identical for Task 1 because there is no previous task to transfer knowledge. For all the subsequent tasks (Task 2,3,4,5), we observe that – (1) ExSSNeT + KKT starts off with a much better initial performance compared to ExSSNeT (2) given a fixed number of epochs for training, ExSSNeT + KKT always learns the task better because it has a better accuracy at all epochs; and (3) ExSSNeT + KKT can achieve similar performance as ExSSNeT in much fewer epochs as shown by the green horizontal arrows. This clearly illustrates that using the KKT knowledge-transfer module not only helps to learn the tasks better (see Table 3) but also learn them faster. For an efficiency and robustness analysis of the KKT module, please refer to Appendix A.4.2.

4.3 Additional Results and Analysis

Q4. Effect of Mask Density on Performance: Next, we show the advantage of using ExSSNeT when the mask density is low. In Figure 5, we show the average accuracy for the Split-CIFAR100 dataset as a function of mask density. We observe
that EXSSNeT obtains 7.9%, 18.4%, 8.4%, and 4.7% improvement over SupSup for mask density values 0.02, 0.04, 0.06, 0.08 respectively. This is an appealing property as tasks select fewer parameters which inherently reduces sparse overlap allowing EXSSNeT to learn a large number of tasks.

Q5. Can EXSSNeT Learn a Large Number of Tasks? SupSup showed that it can scale to a large number of tasks. Next, we perform experiments to learn 100 tasks created by splitting the Tiny-ImageNet dataset. In Table 4, we show that this property is preserved by EXSSNeT while resulting in a performance improvement over SupSup. We note that as the number of task increase, the sparse overlap between the masks also increases resulting in fewer trainable model weights. In the extreme case where there are no free weights, EXSSNeT by design reduces to SupSup because there will be no weight training. Moreover, if we use larger models there are more free parameters, leading to even more improvement over SupSup.

Q6. Effect of Token Embedding Initialization for NLP: For our language experiments, we use a pretrained BERT model (Devlin et al., 2019) to obtain the initial token representations. We perform ablations on the token embedding initialization to understand its impact on CL methods. In Table 5, we present results on the S2T¹ task-order sequence of the sampled version of WebNLP dataset (see Section 4.1, Datasets). We initialize the token representations using FastText (Bojanowski et al., 2016), Glove (Pennington et al., 2014), and BERT embeddings. From Table 5, we observe that – (1) the performance gap between EXSSNeT and SupSup increases from 0.8% → 7.3% and 0.8% → 8.5% when moving from BERT to Glove and FastText initializations respectively. These gains imply that it is even more beneficial to use EXSSNeT in absence of good initial representations, and (2) the performance trend, EXSSNeT > SSNeT > SupSup is consistent across initialization.

5 Related Work

Regularization-based methods estimate the importance of model components and add importance regularization terms to the loss function. Zenke et al. (2017) regularize based on the distance of weights from their initialization, whereas Kirkpatrick et al. (2017b); Schwarz et al. (2018) use an approximation of the Fisher information matrix (Pascual and Bengio, 2013) to regularize the parameters. In NLP, Han et al. (2020); Wang et al. (2019) use regularization to constrain the relevant information from the huge amount of knowledge inside large language models (LLM). Huang et al. (2021) first identifies hidden spaces that need to be updated versus retained via information disentanglement (Fu et al., 2017; Li et al., 2020) and then regularize these hidden spaces separately.

Replay based methods maintain a small memory buffer of data samples (De Lange et al., 2019; Yan et al., 2022) or their relevant proxies (Rebuffi et al., 2017) from the previous tasks and retrain on them later to prevent CF. Chaudhry et al. (2018) use the buffer during optimization to constraint parameter gradients. Shin et al. (2017); Kemker and Kanan (2018) uses a generative model to sample and replay pseudo-data during training, whereas Rebuffi et al. (2017) replay distilled knowledge from the past tasks. de Masson d’Autume et al. (2019) employ episodic memory along with local adaptation, whereas Sun et al. (2019) trains a language model to generate a pseudo-sample for replay.

Architecture based methods can be divided into two categories: (1) methods that add new modules over time (Li et al., 2019; Veniat et al., 2021; Douillard et al., 2022); and (2) methods that isolate the network’s parameters for different tasks (Kirkpatrick et al., 2017a; Fernando et al., 2017; Mallya and Lazebnik, 2018; Fernando et al., 2017). Rusu et al. (2016) introduces a new network for each task while Schwarz et al. (2018) distilled the new network after each task into the original one. Recent prompt learning-based CL models for vision (Wang et al., 2022a,b) assume access to a pre-trained model to learn a set of prompts that can potentially be shared across tasks to perform CL this is orthogonal to our method that trains from scratch. Mallya and Lazebnik (2018) allocates parameters to specific tasks and then trains them in isolation which limits the number of tasks that can be learned. In contrast, Mallya et al. (2018) use a frozen pretrained model and learns a new

<table>
<thead>
<tr>
<th>Method</th>
<th>FastText</th>
<th>Glove</th>
<th>BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupSup</td>
<td>54.01</td>
<td>55.52</td>
<td>74.0</td>
</tr>
<tr>
<td>SSNeT</td>
<td>60.41 [+6.4]</td>
<td>59.78 [+4.3]</td>
<td>74.5 [+0.5]</td>
</tr>
<tr>
<td>EXSSNeT</td>
<td>62.52 [+8.5]</td>
<td>62.81 [+7.3]</td>
<td>74.8 [+0.8]</td>
</tr>
</tbody>
</table>

Table 5: Ablation result for token embeddings. We report average accuracy ↑ [and gains over SupSup]
mask for each task but a pretrained model is crucial for their method’s good performance. Wortsman et al. (2020) removes the pretrained model dependence and learns a mask for each task over a fixed randomly initialized network. ExSSNeT avoids the shortcomings of Mallya and Lazebnik (2018); Mallya et al. (2018) and performs supermask sub-network training to increase the representational capacity compared to (Wortsman et al., 2020) while performing knowledge transfer and avoiding CF.

6 Conclusion

We introduced a novel Continual Learning method, ExSSNeT (Exclusive Supermask SubNetwork Training), that delivers enhanced performance by utilizing exclusive, non-overlapping subnetwork weight training, overcoming the representational limitations of the prior SupSup method. Through the avoidance of conflicting weight updates, ExSSNeT not only improves performance but also eliminates forgetting, striking a delicate balance. Moreover, the inclusion of the Knowledge Transfer (KKT) module propels the learning process, utilizing previously acquired knowledge to expedite and enhance the learning of new tasks. The efficacy of ExSSNeT is substantiated by its superior performance in both NLP and Vision domains, its particular proficiency for sparse masks, and its scalability up to a hundred tasks.

Limitations

Firstly, we note that as the density of the mask increases, the performance improvement over the SupSup method begins to decrease. This is due to the fact that denser subnetworks result in higher levels of sparse overlap, leaving fewer free parameters for new tasks to update. However, it is worth noting that even in situations where mask densities are higher, all model weights are still trained by some task, improving performance on those tasks and making our proposed method an upper bound to the performance of SupSup. Moreover, in general, a sparse mask is preferred for most applications due to its efficiency.

Secondly, we have focused on the task incremental setting of continual learning for two main reasons: (1) in the domain of natural language processing, task identities are typically easy to obtain, and popular methods such as prompting and adaptors assume access to task identities. (2) the primary focus of our work is to improve the performance of supermasks for continual learning and to develop a more effective mechanism for reusing learned knowledge, which is orthogonal to the question of whether task identities are provided during test time.

Moreover, it is worth noting that, similar to the SupSup method, our proposed method can also be extended to situations where task identities are not provided during inference. The SupSup paper presents a method for doing this by minimizing entropy to select the best mask during inference, and this can also be directly applied to our proposed method, ExSSNeT, in situations where task identities are not provided during inference. This is orthogonal to the main questions of our study, however, we perform some experiments on Class Incremental Learning in the appendix A.4.3.

Acknowledgements

We thank Marc’Aurelio Ranzato for the helpful discussions to formulate the initial idea. We thank the reviewers and Xiang Zhou, Swarnadeep Saha, and Archiki Prasad for their valuable feedback on this paper. This work was supported by NSF-CAREER Award 1846185, DARPA Machine-Commonsense (MCS) Grant N66001-19-2-4031, Microsoft Investigator Fellowship, and Google and AWS cloud compute awards. The views contained in this article are those of the authors and not of the funding agency.

References


Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR 2009*.


A1 Sparse Overlap to Quantify Parameter Interference

Next, we propose a measure to quantify parameter interference for a task $i$, i.e., the fraction of the parameters in mask $M_i$ that are already updated by some previous task. We define sparse overlap as the difference between the number of parameters selected by mask $M_i$ and $M_i^{free}$ divided by the total parameters selected by $M_i$. Formally, we define sparse overlap (SO) between current supermask $M_i$ and supermasks for previous tasks $\{M_j\}_{j=1}^{i-1}$ as,

$$SO(M_i, \{M_j\}_{j=1}^{i-1}) = \frac{\text{sum}(M_i) - \text{sum}(M_i^{free})}{\text{sum}(M_i)}$$

and $M_i^{free} = M_i \land \neg(\lor_{j=1}^{i-1}(M_j))$

where $\land, \lor, \neg$ are logical and, or, and not operations.
Table 6: Task sequences used in text experiments. For the GLUE dataset, we use order corresponding to decreasing train data size. Sequence S2-S8 are from (Huang et al., 2021; de Masson d’Autume et al., 2019; Sun et al., 2019).

Table 7: Statistics for sampled data used from Huang et al. (2021) for hyperparameter tuning. The validation set is the same size as the train set. Class means the number of output classes for the text classification task. Type is the domain of text classification.

A.2 Space, Time, and Memory Complexity of ExSSNeT

For training, we store an additional set of scores on GPU with size as the model weight. The additional GPU memory required is a small fraction because the model activations account for a huge fraction of the total GPU memory. Our runtime is similar to training the weight of a model with < 5% overhead due to the logical operations on masks and masking weight during the forward passes. For training time comparisons refer to Appendix Table 13. On the disk, we need to store \( k \times |W| \) updated weights of 32-bits and boolean mask which takes 1-bit for each parameter. Hence, we take \( \max(|W|, k \times t, |W|) \times 32 + |W| \times 1 \) bits in total as in the worst case we need to store all \(|W|\) model weights.

A.3 Experimental setup and hyperparameters

Unless otherwise specified, we obtain supermasks with a mask density of 0.1. In our CNN models, we use non-affine batch normalization to avoid storing their means and variance parameters for all tasks (Wortsman et al., 2020). Similar to (Wortsman et al., 2020), bias terms in our model are 0 and we randomly initialize the model parameters using \textit{signed kaiming constant} (Ramanujan et al., 2019). We use Adam optimizer (Kingma and Ba, 2014) along with cosine decay (Loshchilov and Hutter, 2016) and conduct our experiments on GPUs with 12GB of memory. We used approximately 6 days of GPU runtime. For our main experiment, we run three independent runs for each experiment and report the averages for all the metrics and experiments. For natural language tasks, unless specified otherwise we initialize the token embedding for our methods using a frozen BERT-base-uncased (Devlin et al., 2018) model’s representations using Huggingface (Wolf et al., 2020). We use a static CNN model from Kim (2014) as our text classifier over BERT representations. The model employs 1D convolutions along with \textit{Tanh} activation. The total model parameters are \( \sim 110M \) Following Sun et al. (2019); Huang et al. (2021), we evaluate our model on various task sequences as provided in Appendix Table 6, while limiting the maximum number of tokens to 256. Following (Wortsman et al., 2020), we use LeNet (Lecun et al., 1998) for SplitMNIST dataset, a ResNet-18 model with fewer channels (Wortsman et al., 2020) for SplitCIFAR100 dataset, a ResNet50 model (He et al., 2016) for TinyImageNet dataset. Unless specified, we randomly split all the vision datasets to obtain five tasks with disjoint classes. We use the codebase of DER (Buzzega et al., 2020) to obtain the vision baselines. In all our experiments, all methods perform an equal number of epochs over the datasets. We use the hyperparameters from Wortsman et al. (2020) for our vision experiments.

For the ablation experiment on natural language data, following Huang et al. (2021), we use a sampled version of the WebNLP datasets due to limited resources. The reduced dataset contains 2000 training and validation examples from each output class. The test set is the same as the main experiments. The dataset statistics are summarized in Table 7. For WebNLP datasets, we tune the learning rate on the validation set across the values \{0.01, 0.001, 0.0001\}, for GLUE datasets we use the default learning rate of the BERT model. For our vision experiments, we use the default learning rate for the dataset provided in their original implementation. For TinyImageNet, SplitCIFAR100, SplitMNIST dataset, we run for 30, 100, and 30 epochs respectively. We store 0.1% of our vision datasets for replay while for our language experiments we use 0.01% of the data because of the large number of datasets available for them.
### Additional Results

#### A.4.1 Results on ImageNet Dataset

In this experiment, we take the ImageNet dataset (Deng et al., 2009) with 1000 classes and divide it into 10 tasks where each task is a 100-way classification problem. In Table 8, we report the results for ExSSNeT and the strongest vision baseline method, SupSup. We omit other methods due to resource constraints. We observe a strong improvement of 6.7% of ExSSNeT over SupSup, indicating that the improvements of our methods exist for large scale datasets as well.

#### A.4.2 Analysis of Efficiency, Runtime, and hyperparameters of the KKT module

Firstly, we would like to note that the KKT module is lightweight and efficient because it only runs once for each task before we start training on it and only uses a few batches to estimate the relevant mask. Given that we perform multiple epochs over the task’s data, the cost of the KKT module becomes negligible in comparison to it and runs in almost similar clock time as without it. The runtime on splitcifar100 datasets with 100 epochs for ExSSNeT is 168 minutes and for ExSSNeT + KKT is 173 minutes which is a very small difference.

Second, there are two main hyperparameters in the KKT module – (1) \( k \) for taking the majority vote of top-\( k \) neighbors fixed for KKT module fixed.

Next, in Table 10, we use a fixed \( k = 10 \) and vary the number of batches used for KKT with a batch size of 64 and report the performance of ExSSNeT.

From this table, we observe that the performance increases with \( k \) and then starts to decrease but in general most values of \( k \) work well.

From both of these experiments, we can observe that the KKT module is fairly robust to different values of these hyperparameters but carefully selecting them hyperparameters can lead to slight improvement.

#### A.4.3 Class Incremental Learning

We performed Class Incremental Learning experiments on the TinyImageNet dataset (10-tasks, 20-classes in each) and used the One-Shot algorithm from SupSup (Wortsman et al., 2020) to select the mask for inference. Please refer to Section-3.3 and Equation-4 of the SupSup paper (Wortsman et al., 2020) for details. From Table 12, we observe that ExSSNeT outperforms all baseline methods that
do not use Experience Replay by at least 2.75%. Moreover, even with the need for a replay buffer, ExSSNeT outperforms most ER-based methods and is comparable to that of DER.

A.4.4 Sparse Overlap Numbers

In Table 11, we report the sparse overlap numbers for SupSup, SSNeT, and ExSSNeT with and without the KKT knowledge transfer module. This table corresponds to the results in main paper Table 3.

A.4.5 Average Accuracy Evolution

In Figure 6, we plot $\sum_{t \leq T} A_t$ vs $t$, that is the average accuracy as a function of observed classes. This plot corresponds to the SplitCIFAR100 results provided in the main paper Table 2. We can observe from these results that Supsup and ExSSNeT performance does not degrade when we learn new tasks leading to a very stable curve whereas for other methods the performance degrades as we learn new tasks indicating some degree of forgetting.

A.4.6 Runtime Comparison across methods

In this Section, we provide the result to compare the runtime of various methods used in the paper. We ran each method on the sampled version of the WebNLP dataset for the S2 task order as defined in Table 6. We report the runtime of methods for four epochs over each dataset in Table 13. Note that the masking-based method, Supsup, SSNeT, ExSSNeT takes much lower time because they are not updating the BERT parameters and are just finding a mask over a much smaller CNN-based classification model using pretrained representation from BERT. This gives our method an inherent advantage that we are able to improve performance but with significantly lower runtime while learning a mask over much fewer parameters for the natural language setting.

A.4.7 Validation results

In Table 14, we provide the average validation accuracies for the main natural language results presented in Table 1. We do not provide the validation results of LAMOL (Sun et al., 2019) and MBPA++ (de Masson d’Autume et al., 2019) as we used the results provided in their original papers. For the vision domain, we did not use a validation set because no hyperparameter tuning was performed as we used the experimental setting and default param-

<table>
<thead>
<tr>
<th>Method</th>
<th>BufferSize</th>
<th>TinyImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>0</td>
<td>7.92</td>
</tr>
<tr>
<td>oEWC</td>
<td>0</td>
<td>7.58</td>
</tr>
<tr>
<td>LwF</td>
<td>0</td>
<td>8.46</td>
</tr>
<tr>
<td>ER</td>
<td>200</td>
<td>8.49</td>
</tr>
<tr>
<td>A-GEM</td>
<td>200</td>
<td>8.07</td>
</tr>
<tr>
<td>iCARL</td>
<td>200</td>
<td>7.53</td>
</tr>
<tr>
<td>DER</td>
<td>200</td>
<td>11.87</td>
</tr>
<tr>
<td>DER++</td>
<td>200</td>
<td>10.96</td>
</tr>
<tr>
<td>SupSup</td>
<td>0</td>
<td>10.27</td>
</tr>
<tr>
<td>ExSSNeT</td>
<td>0</td>
<td>11.21</td>
</tr>
</tbody>
</table>

Table 12: Results for CIL setting.
<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime (in minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multitask</td>
<td>200</td>
</tr>
<tr>
<td>Finetune</td>
<td>175</td>
</tr>
<tr>
<td>Replay</td>
<td>204</td>
</tr>
<tr>
<td>AdapterBERT + FT</td>
<td>170</td>
</tr>
<tr>
<td>AdapterBERT + Replay</td>
<td>173</td>
</tr>
<tr>
<td>MultiAdaptBERT</td>
<td>170</td>
</tr>
<tr>
<td>Regularization</td>
<td>257</td>
</tr>
<tr>
<td>IDBR</td>
<td>258</td>
</tr>
<tr>
<td>SupSup</td>
<td>117</td>
</tr>
<tr>
<td>SSNeT</td>
<td>117</td>
</tr>
<tr>
<td>EXSSNeT</td>
<td>117</td>
</tr>
</tbody>
</table>

Table 13: Runtime comparison of different methods used in the text experiments.

<table>
<thead>
<tr>
<th>Method</th>
<th>GLUE</th>
<th>WebNLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order (→)</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>Random</td>
<td>33.3</td>
<td>7.14</td>
</tr>
<tr>
<td>Multitask</td>
<td>80.6</td>
<td>77.5</td>
</tr>
<tr>
<td>FT</td>
<td>14.0</td>
<td>22.9</td>
</tr>
<tr>
<td>Replay</td>
<td>79.7</td>
<td>75.2</td>
</tr>
<tr>
<td>AdapterBERT + FT</td>
<td>25.1</td>
<td>19.1</td>
</tr>
<tr>
<td>AdapterBERT + Replay</td>
<td>78.6</td>
<td>74.7</td>
</tr>
<tr>
<td>MultiAdaptBERT</td>
<td>83.6</td>
<td>76.7</td>
</tr>
<tr>
<td>Regularization</td>
<td>75.5</td>
<td>75.0</td>
</tr>
<tr>
<td>IDBR</td>
<td>77.5</td>
<td>75.8</td>
</tr>
<tr>
<td>SupSup</td>
<td>78.1</td>
<td>76.0</td>
</tr>
<tr>
<td>SSNeT</td>
<td>77.2</td>
<td>76.3</td>
</tr>
<tr>
<td>EXSSNeT</td>
<td>80.1</td>
<td>77.1</td>
</tr>
</tbody>
</table>

Table 14: Average validation accuracy (↑) for multiple tasks and sequence orders with previous state-of-the-art (SotA) methods.

A.4.8 Effect of Task Order and Number of Tasks

Following Huang et al. (2021), we conduct experiments to study the effect of task length and order in the language domain. We use task sequences of lengths three and five, with multiple different task orders on the sampled data (Section 4.1, Table 6, and Appendix) to characterize the impact of these variables on the performance. In Table 15, we present the average test accuracy averaged over three different random seeds. We observe that across all six different settings our method performs better compared to all the baseline methods. Our methods bridge the gap toward multitask methods’ performance, leaving a gap of 0.36% and 1.19% for lengths three and five sequences, respectively.

A.5 Additional Model Details

A.5.1 Algorithm for EXSSNeT

In Algorithm 1, we provide a pseudo-code for our method EXSSNeT for easier reference and understanding. We also attach our working code as supplementary material to encourage reproducibility.
<table>
<thead>
<tr>
<th>Model (↓)</th>
<th>Length-5 WebNLP</th>
<th>Length-3 WebNLP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Order (→)</td>
<td>S2</td>
</tr>
<tr>
<td>Random</td>
<td>7.14</td>
<td>7.14</td>
</tr>
<tr>
<td>MTL</td>
<td>75.09</td>
<td>75.09</td>
</tr>
<tr>
<td>Finetune</td>
<td>32.37</td>
<td>32.22</td>
</tr>
<tr>
<td>Replay†</td>
<td>68.25</td>
<td>70.52</td>
</tr>
<tr>
<td>Regularization†</td>
<td>72.28</td>
<td>73.03</td>
</tr>
<tr>
<td>AdaptBERT</td>
<td>30.49</td>
<td>20.16</td>
</tr>
<tr>
<td>AdaptBERT + Replay</td>
<td>69.30</td>
<td>67.91</td>
</tr>
<tr>
<td>IDBR†</td>
<td>72.63</td>
<td>73.72</td>
</tr>
<tr>
<td>SupSup</td>
<td>74.01</td>
<td>74.04</td>
</tr>
<tr>
<td>SSNet</td>
<td>74.5</td>
<td>74.5</td>
</tr>
<tr>
<td>EXSSNET</td>
<td>74.78</td>
<td>74.72</td>
</tr>
</tbody>
</table>

**Table 15:** Average test accuracy reported over task sequences for three independent runs on sub-sampled data. Results with † are taken from Huang et al. (2021).

**Figure 7:** This is a canonical model diagram for SupSup. In SupSup, the model weights are always fixed at the random initialization $W^{(0)}$. For each task SupSup learns a new mask (in this case $M_1, M_2, M_3$) over the weights $W^{(0)}$. A mask selectively activates a subset of weights for a particular task. This subset of selected weights forms a subnetwork inside the full model which we refer to as the supermask subnetwork. For example, when learning Task 2, SupSup learns the mask $M_2$ (the weights activated by the mask are highlighted in green) over the fixed weight $W^{(0)}$. These highlighted weights along with the participating nodes are the subnetwork formed by mask $M_2$. Whenever a prediction is made for Task 2 samples, this mask is selected and used to obtain the predictions. Please note that the model weights $W^{(0)}$ are never updated after their random initialization. Hence, for SupSup there is no learned knowledge sharing across tasks. This is in contrast to our setup in Figure 1, where for the first task the mask is learned over the weights $W^{(0)}$ but once the mask is selected the weights of the corresponding subnetwork are also updates to obtain new weight $W^{(1)}$. Then the next task’s mask is learned over these new set of weights $W^{(1)}$ and so on. Also note that in Figure 1, we do not show the KKT knowledge transfer module here to avoid confusion.
ACL 2023 Responsible NLP Checklist

A  For every submission:

☑️ A1. Did you describe the limitations of your work?
  
  
  Section 7

☒ A2. Did you discuss any potential risks of your work?
  
  Our work has no additional risk apart from the risk of the components involved.

☑️ A3. Do the abstract and introduction summarize the paper’s main claims?
  
  Abstract and Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
  
  Left blank.

B  ☒ Did you use or create scientific artifacts?

  Left blank.

☐ B1. Did you cite the creators of artifacts you used?
  
  No response.

☐ B2. Did you discuss the license or terms for use and/or distribution of any artifacts?
  
  No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
  
  No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected/used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect/anonymize it?
  
  No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
  
  No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train/test/dev splits, etc. for the data that you used/created? Even for commonly-used benchmark datasets, include the number of examples in train/validation/test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
  
  No response.

C  ☑ Did you run computational experiments?

  Section 4

☑️ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
  
  Appendix A.3

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.
C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
   *Section 4.1 and Appendix A3*

C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
   *Section 4*

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
   *Not applicable. Left blank.*

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
   *Left blank.*

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
   *No response.*

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants’ demographic (e.g., country of residence)?
   *No response.*

☐ D3. Did you discuss whether and how consent was obtained from people whose data you’re using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
   *No response.*

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
   *No response.*

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
   *No response.*