CS 839 Special Topics in AI: Deep Learning **Fall 2020**

Continual / Lifelong Learning III: SupSup - Supermasks in Superposition

Presenters: Akshata/Zifan *Scribes: Akshata/Zifan*

1 Background

Continual learning refers to the ability of learning different tasks sequentially with one model, which reuses the knowledge learnt before to guide the learning of new tasks. If a model is trained sequentially on a series of tasks in the traditional way, its performance on the earlier tasks degrades a lot as training goes on, a problem known as catastrophic forgetting.

The paper proposes a three-letter taxonomy to describe different scenarios of continual learning, which is summarized in Figure 1. The first and second letters specify if the task identity is provided during training and inference respectively (G if provided, N if not). The third letter specifies whether labels are shared (s) or not (u). GG is the simplest case where the task identity is given at both training and inference time, and the model performs the corresponding task directly. For example, for a model which predicts the breeds of dogs, cats or birds, if the model is told that the input picture is a dog, then it switches to the 'dog mode', looks at the distribution over the breeds of dogs and picks the one with the highest probability. In the GG case, whether the labels are shared or not does not make any difference since the model can take the corresponding outputs based on the task identity. When the task identity is unknown during inference, the model has to infer the task identity, switch to the corresponding mode, and then pick the answer from that task. If the labels of the tasks are not shared, the model needs to look at the distribution over all the classes across all tasks to predict the task identity, which is harder than the shared case. If we take the previous example but assume that the species of the input is unknown, the model has to look at the distribution of all breeds of dogs, cats and birds to guess which species it belongs to before making a prediction. In another example, if the tasks are predicting the breed of adult dogs, puppies and older dogs, but the labels are shared, the model only needs to look at the distribution of these shared labels, which is much easier. The NN case is the hardest because the model has to predict the task identity during both training and inference time. New tasks may be added during training when necessary, and how many tasks in total are unknown. For the NN case, only shared labels are considered, since the output domain is unknown if the labels are different for each task, which is unrealistic. The NNu scenario is invalid because unseen labels signal the presence of a new task, making the scenario actually GNu.

Previous works on continual learning lie in the following three categories:

- **Regularization based methods** These methods penalize the movement of parameters that are important for solving previous tasks in order to mitigate catastrophic forgetting. For example, Elastic Weight Consolidation (EWC) [9] uses the Fisher Information matrix to measure the importance of parameters.
- **Using exemplars, replay, or generative models** These methods explicitly or implicitly memorize data from previous tasks. iCaRL [11] updates the model for a new class with access to the exemplars from the previous classes.
- **Task-specific model components** These methods use different components for different tasks. For example, Batchensemble (BatchE) [12] learns a shared weight matrix on the first task and learn only a rank-one scaling matrix for each subsequent task. The final weight for each task is the elementwise

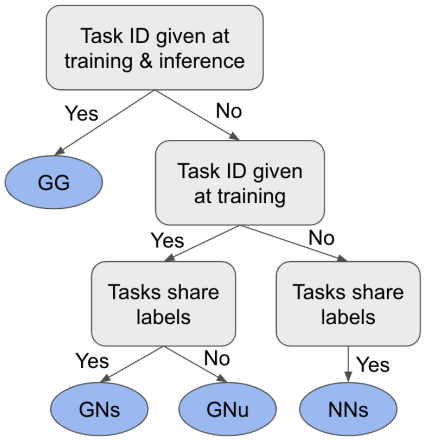

Figure 1: Different scenarios of continual learning.

product of the shared matrix and the scaling matrix. Parameter Superposition (PSP) [1] combines the parameter matrices of different tasks into a single one based on the observation that different weights are not in the same subspace.

The proposed method lies in the third category, and it is the only method in that category that handles all the scenarios. BatchE and PSP are selected as baselines because of their good performance, but they work only in the GG case.

2 Overview of SupSup

SupSup uses a single model to perform different tasks by taking different subnetworks for each task. The first Sup refers to Supermask, which is used to represent subnetworks. The second Sup refers to Superposition. Superposition is the ability of a quantum system to be in multiple states at the same time until it is measured. Here superposition refers to the mechanism that subnetworks for different tasks coexist in a single network, and the subnetwork to be used is decided by the input data.

SupSup utilizes the expressive power of subnetworks and views the inference of task identity as an optimization problem.

- **Expressive power of subnetworks** It is shown in the previous works [3, 10] that overparametrized networks, even without training, contain subnetworks that achieve impressive performance. SupSup uses the Supermask method [10] to find subnetworks for each task from a randomly initialized network. Each subnetwork is represented by a supermask. The supermasks are stored and then retrieved during inference time.
- **Inference of task identity as an optimization problem** If the task identity is not given at inference time, SupSup predicts it based on the entropy of the output distribution. The inference of task identity is framed as an optimization problem, which seeks a convex combination of the learnt supermasks to minimizes the entropy.

3 Supermask

Supermask [10] assumes that if a neural network with random weights is sufficiently overparameterized, it will contain a subnetwork that perform as well as a trained neural network with the same number of parameters. The intuition is that because the number of subnetworks grows exponentially as the size of the unpruned network increases, the probability that a good subnetwork does not exist become very small for an extremely overparameterized network.

There is an efficient algorithm called edge-popup that finds good subnetworks, which is proposed in [10]. Figure 2 shows an illustration of the edge-popup algorithm. Edge-popup assigns a score to each edge in the network, which is initialized randomly at first. In the forward pass, the edges corresponding to the top k percent scores are used, and the rest of them are disabled. In the backward pass, the scores of the edges are updated. The rule is that if the weighted output of in-node u is aligned with the negative gradient to the input of out-node v, we increase the score of that edge. The intuition is that the loss wants the input of v to be changed in the negative gradient direction. If the weighted output of u is aligned with that direction, we can reduce the loss by adding the edge between u and v to the subnetwork. It is shown that the solution converges to a good subnetwork. For example, from Resnet-50, they found a subnetwork that matches the performance of Resnet-34.

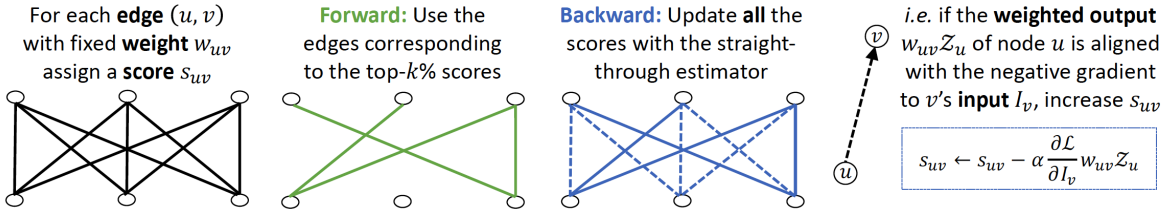

Figure 2: An illustration of the edge-popup algorithm.

4 Setup

In a standard l-way classification task, inputs x are mapped to a distribution p over output labels. A neural network f is parameterized by W and trained with a cross entropy loss, as given by (1).

$$p = f(x, W) \tag{1}$$

In continual learning setting, they consider k different l-way classification tasks. The input size remains constant across tasks. Output is given by the formula (2), where we perform elementwise dot product of weights W with mask M. Weights are frozen at initialization.

$$p = f(x, W \odot M) \tag{2}$$

5 Scenarios and Results

5.1 GG: Task ID given at training, given at inference

5.1.1 Introduction

In this setup, use different super mask for each task during training. For task 1 use a mask M^1, for task 2 use a mask M^2, and so on, as given in Figure 3 (left). Basically, learn a new binary mask (a new sub network for task), while keeping the weights fixed at the initialization, given by (3). For each new task, they either initialize the supermask randomly or use a running mean of all supermasks seen so far. At inference, since the task is known, the corresponding mask can be used.

$$p = f(x, W \odot M^i) \tag{3}$$

This is an extension of work by [8], where they have a similar idea, they using a pre-trained network as a backbone. They train on ImageNet and find different masks for different tasks. This is more expensive in terms of bytes, because the weights need to be saved. Whereas in supermasks, it is not required to store the base model weights W. Since W is random, it is sufficient to save the random seed used to generate these weights. These weights can be spun up on GPU as needed.

Also, SupSup requires minimal overhead to perform forward pass. Elementwise product by a binary mask can be implemented via memory access, i.e. selecting indices. Modern GPUs enable this speed up.

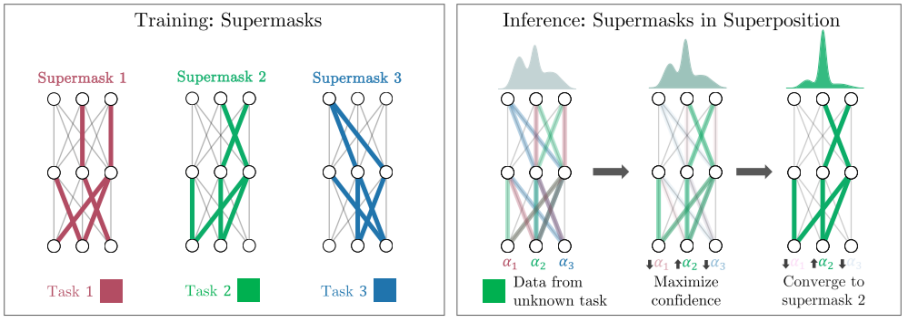

Figure 3: Training and Inference using SupSup.

5.1.2 Experiment Results

The authors provide the performance on SpliImageNet and SplitCIFAR100 dataset, as given in Figure 4.

In SplitImageNet [12] dataset, ImageNet [2] is partitioned into 100 different 10-way classification problems. In this experiment, they use a standard ResNet-50 [4] model. ResNet-50 takes traditionally a 100M byes to store. For Upper Bound, there is a model for each task, so there are 100 ResNet-50 models in total. In SupSup, they achieve good performance even when total number of bytes is less than single ResNet-50. Number of bytes are controlled by changing the sparsity of the supermasks.

SplitCIFAR100 randomly partitions CIFAR100 into 20 different 5-way classification problems. Here, transfer refers to the setting in which they initialize a new supermask with a running meaning of all the super maks seen so far. For Separate Heads, they train different heads for each task using a trunk. Trunk is all the layers except the final layer, and it is trained on task 1. For Separate Heads - Random W, they use a random trunk. SupSup Model outperformss similar size baselines and benefits from transfer.

Algorithm	Avg Top 1 Accuracy (%)	Bytes
Upper Bound	92.55	10222.81M
SupSup (GG)	89.58	195.18M
	88.68	100.98M
	86.37	65.50M
BatchE (GG)	81.50	124.99M
Single Model	-	102.23M

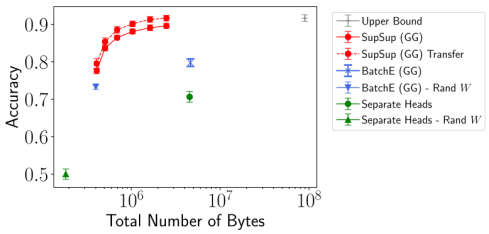

Figure 4: (left) SplitImagenet performance in Scenario GG. (right) SplitCIFAR100 performance in Scenario GG shown as mean and standard deviation over 5 seed and splits.

5.2 GNs & GNu : Task ID given during training only

5.2.1 Introduction

During training, they proceed exactly as scenario GG, where they obtain k learned supermasks. During inference, first infer the task and then use the corresponding supermask. So if the step 1 is done correctly, this is same as inference in GG scenario. For inferring task id, the authors consider a network with all supermasks in weighted superposition, as given in Figure 3 (right). First associate each of k learned supermasks M^i with coefficient α_i. The model output $p(\alpha)$ is then computed with a weighted superposition of all learned masks.

$$p = f(x, W \odot (\sum_{i=1}^{k} \alpha_i M^i)) \tag{4}$$

Intuition is that the correct mask should produce confident, low entropy, outputs. For e.g. if we train a model of dogs, and showed this model a picture of pug, then the model will be confident that it is a pug. But instead if we showed it a picture of cat, then it is likely that it will output a uniform distribution over all classes. Though this may not hold true always, if the model isn't well calibrated, it seems like in the tasks they consider, this intuition holds.

5.2.2 How to pick the supermasks?

One option is to try each mask individually and pick the one with the lowest entropy output. If there are k tasks, this would require k forward passes. If k is large say 1000, then we need to perform 1000 forward passes, which is computationally very expensive. Instead of this, given data from unknown task, supermasks can all be stacked together, each one weighted by α_i. The α_i's can be changed to maximize the confidence and minimize the entropy. To perform this they provide two optimization methods - One Shot and Binary.

One Shot Algorithm is given in Figure 5. Here the task is inferred using a single gradient step. The algorithm returns the coordinate of supermask for which entropy is maximally decreasing.

Binary aglorithm Algorithm. as given in Figure 6, is inspired from binary search. This algorithm takes a total of $\log k$ steps. At each step half of the tasks are ruled out.

5.2.3 Experiment Results

They provide the performance for GNu scenario only, since it is a more difficult task. In Figure 7, PermutedMNIST [6] dataset is used. Here each new task is a fixed random permutation of pixels of MNIST.

Algorithm 1 One-Shot$(f, \mathbf{x}, W, k, \{M^i\}_{i=1}^k, \mathcal{H})$

$\alpha \leftarrow \left[\frac{1}{k} \quad \frac{1}{k} \quad \ldots \quad \frac{1}{k}\right]$ ▷ Initialize α

$\mathbf{p} \leftarrow f\left(\mathbf{x}, W \odot \left(\sum_{i=1}^k \alpha_i M^i\right)\right)$ ▷ Superimposed output

return $\arg\max_i \left(-\frac{\partial \mathcal{H}(\mathbf{p})}{\partial \alpha_i}\right)$ ▷ Return coordinate for which objective maximally decreasing

Figure 5: One shot Algorithm

Algorithm 2 Binary$(f, \mathbf{x}, W, k, \{M^i\}_{i=1}^k, \mathcal{H})$

$\alpha \leftarrow \left[\frac{1}{k} \quad \frac{1}{k} \quad \ldots \quad \frac{1}{k}\right]$ ▷ Initialize α

while $\|\alpha\|_0 > 1$ **do** ▷ Iterate until α has a single nonzero entry

  $\mathbf{p} \leftarrow f\left(\mathbf{x}, W \odot \left(\sum_{i=1}^k \alpha_i M^i\right)\right)$ ▷ Superimposed output

  $g \leftarrow -\nabla_\alpha \mathcal{H}(\mathbf{p})$ ▷ Gradient of objective

  for $i \in \{1, \ldots, k\}$ **do** ▷ In code this **for** loop is vectorized

    if $g_i \leq$ **median** (g) **then**

      $\alpha_i \leftarrow 0$ ▷ Zero out α_i for which objective minimally decreasing

  $\alpha \leftarrow \alpha / \|\alpha\|_1$ ▷ Re-normalize α to sum to 1

return $\arg\max_i \alpha_i$

Figure 6: Binary Algorithm.

SupSup uses OneShot algorithm to infer task identity. SupSup outperforms baseline methods with access to task identity.

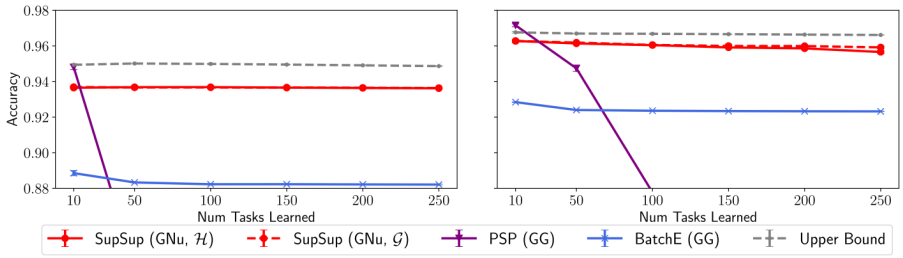

Figure 7: Results shown for PermutedMNIST with LeNet 300-100 (left) and FC 1024-1024 (right).

Results on RotatedMNIST dataset are given in Figure 8. Here the images are rotated by 10 degrees to form a new task with 36 tasks in total. "Full Batch" indicates that all 128 images are used to infer task identity. Otherwise, a single image is used to infer the task. In "Lower Bound" setup, a shared trunk of the network is trained continuously and a separate head is trained for each task. Figure 8 (left) shows that SupSup is able to infer task identity even when tasks are similar(rotation degree is 10). SupSup uses a full batch here since its a challenging problem, and it uses the binary algorithm to perform task identity inference. So we see that even with GNu scenarios, SupSup is able to outperform both the baseline models. Figure 8 (right) we see that SupSup outperforms BatchE. Since BatchE doesn't support GNu scenario, for fair comparison, they equip BatchE with task inference using OneShot Algorithm.

There is a question from the audience that why the performance of PSP improves as the angle of rotation increases. We did not give an explanation during the presentation. A possible reason is that PSP learns the angles sequentially in an ascending order, so that more forgetting happens to the small angles.

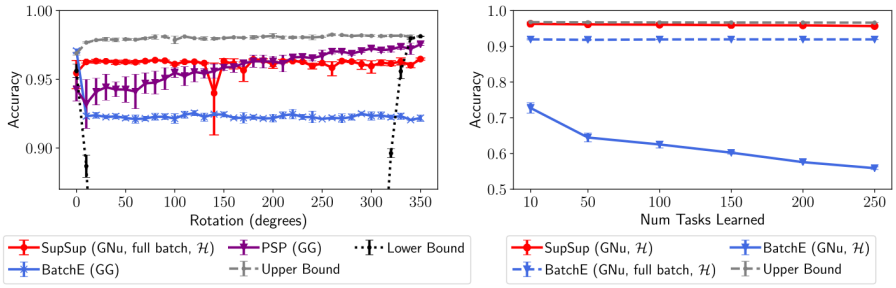

Figure 8: (left) Testing the FC 1024-1024 model on RotatedMNIST. (right) Experiment conducted with FC 1024-1024 on PermutedMNIST using an output size of 500, shown as mean and stddev over 3 runs.

5.3 Scenario NNs: Task ID not given at train or inference

5.3.1 Introduction

During training, SupSup tries to infer the task ID. If it is uncertain when inferring the task id, implying that the data doesn't belong to any tasks seen so far, it allocates a new supermask. So when the gradient of entropy (given by the following equation) is sort of uniform , the model uniformly prefers all supermasks seen so far, so a new task is allocated and k is incremented.

$$v = softmax(-\nabla_\alpha \mathcal{H}(p(\alpha))) \approx uniform \tag{5}$$

5.3.2 Experiment Results

In Figure 9, they try to learn 2500 permutations of MNIST in the NNs and GNu scenarios. This experiment is conducted using One Shot Algorithm and using a single image to infer the task identity. For NNs scenario, every 100 batches the model must choose to allocate a new mask or pick an existing mask using. Here we see that without access to task identity even during training, the model is able to learn 1000s of tasks. Accuracy dips in the last epoch, because they enforce a budget of a total of 2500 supermasks. So the model has allocated all super masks when it shouldn't have, and it runs out of supermasks in the end.

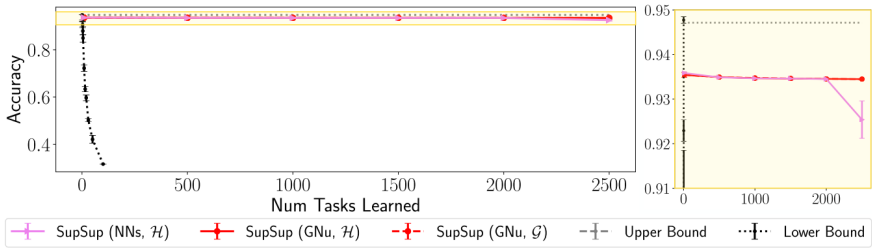

Figure 9: Results for both the GNu and NNs scenarios with the LeNet 300-100 model using output size 500. Computation.

6 Other Design Choices

6.1 Hopfield Network

The space required to store the masks grows linearly as the number of tasks increases. In order to further reduce the required space, the paper provides a design choice that encodes a series of masks into a Hopfield network [5].

A Hopfield network implicitly encodes a series of binary strings $z^i \in \{-1, 1\}^d$ with an associated energy function $E(z)$. Each z^i is a local minima of E, and can be recovered with gradient descent.

During training, when a new mask $m \in \{0, 1\}^d$ is learnt, we can encode $z = 2m - 1$ into the Hopfield network by updating the energy function.

During inference, when a new batch of data comes, we perform gradient descent on the following problem to recover the mask:

$$\min_z \frac{\gamma t}{T} E(z) + (1 - \frac{t}{T})\mathcal{H}(p) \tag{6}$$

where $E(z)$ is the energy function, $\mathcal{H}(p)$ is the entropy of the output distribution using the mask corresponding to z. T is the total number of steps, t is the step number, and γ is a hyperparameter. Minimizing E will push $m = (z + 1)/2$ to a mask encoded before, and $\mathcal{H}$ will push m to the correct mask. Note that as the gradient descent goes on, the strength of the Hopfield term increases while the strength the entropy term decreases.

Figure 10 shows the changes of hamming distance between the solution m and the correct mask m_i during 30 steps of gradient descent. The dataset is SplitMNIST which partitions MNIST into 5 different 2-way classification tasks, each containing consecutive classes from the original dataset. We can see that the solution converges to the correct mask in most of the cases.

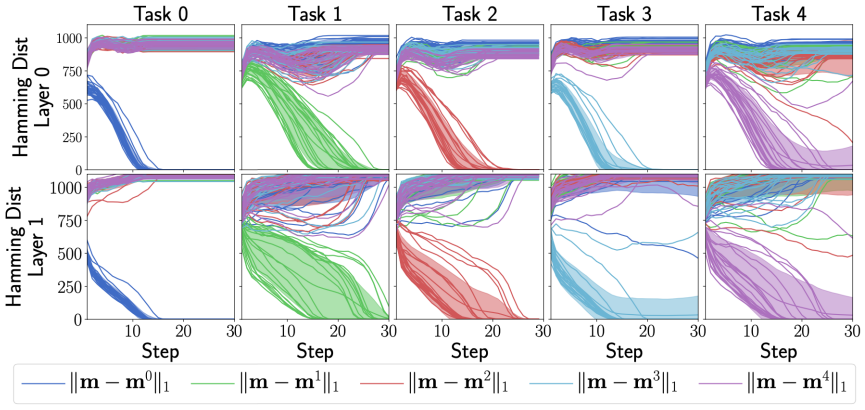

Figure 10: During Hopfield recovery, the solution converges to the correct mask.

Figure 11 shows the test accuracy on SplitMNIST with and without Hopfield network under different random seeds. We can see that the Supermask recovered from the Hopfield network results in similar accuracy compared to the correct mask most of the time. However, under certain random seeds, when the recovery fails, the accuracy drops a lot. Therefore, even the overall accuracy of the Hopfield SupSup is high, it may fail completely for certain batches.

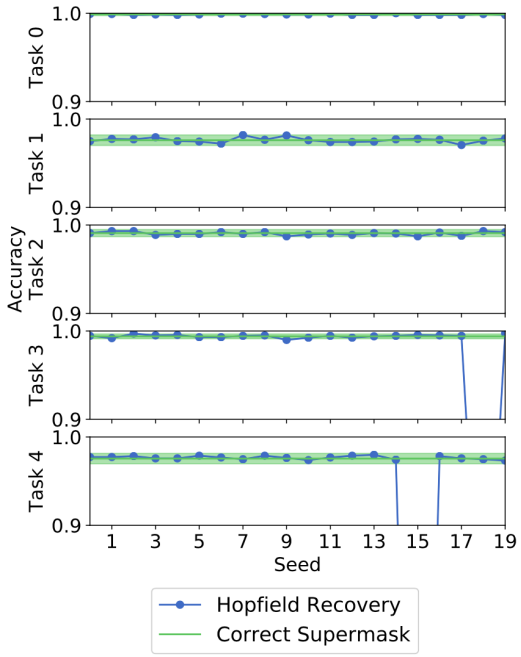

Figure 11: A comparison of the test accuracy of SupSup w/ and w/o Hopfield network.

6.2 Superfluous Neurons

In practice, the authors find it helps significantly to add extra neurons to the final layer, and use them as an alternative of the entropy to infer task identities. During training, the standard cross-entropy loss will push the values of the extra neurons down. The authors propose an objective $\mathcal{G}$ which encourages the extra neurons to have large negative values when the mask is correct. Specifically, $\mathcal{G} = \text{logsumexp(p)}$ where p is the output distribution. For one-shot inference, the inferred task identity is given by $\arg\max_i(-\frac{\partial \mathcal{G}(p(\alpha))}{\partial \alpha_i})$. Note that when we compute the gradients for $\mathcal{G}$, only the gradients w.r.t the extra neurons are enabled. The intuition is that given data from task i, network with mask i will minimize the values of the extra neurons as it was trained to, and networks with other masks will not because they are trained with data from different distributions.

Figure 12 compares the performance of the $\mathcal{H}$ and $\mathcal{G}$ objectives with the one-shot inference when the output size s varies. The dataset is PermutedMNIST which creates new tasks with a fixed random permutation of the pixels of MNIST. We can see that $\mathcal{G}$ performs comparable to or better than $\mathcal{H}$. The results also show the importance of the output size. The performance is much better with $s = 200$ than $s = 25$.

6.3 Transfer

If each Supermask is initialized randomly, the models for subsequent tasks cannot leverage the knowledge learnt from the previous tasks. In order to take advantage of the previous knowledge, SupSup with transfer initializes the score matrix used in the edge-popup algorithm for a new task with the running mean of the Supermasks for all the previous tasks.

Figure 13 shows the test accuracy of SupSup with and without transfer on splitCIFAR. We can see that

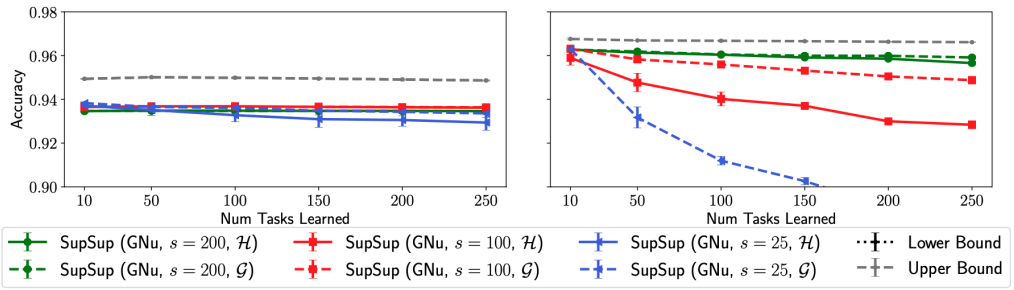

Figure 12: A comparison of $\mathcal{H}$ and $\mathcal{G}$ when the output size s varies. The result on the left is evaluated with LeNet 300-100 and that on the right is evaluated with FC 1024-1024.

SupSup with transfer performs better than the version without transfer, and requires less epoches.

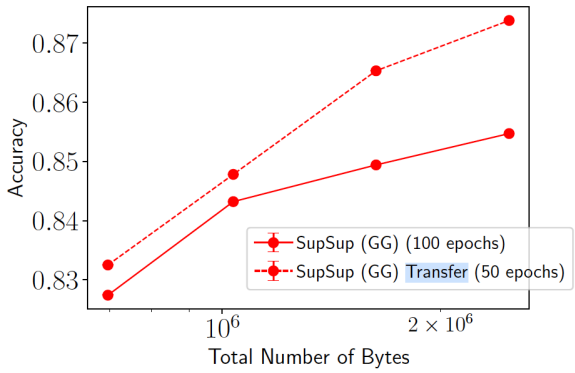

Figure 13: A comparison of SupSup w/ and w/o transfer under different model sizes.

7 Conclusion and Discussion

SupSup leverages the expressive power of subnetworks to perform a large amount of tasks with a single network. It infers task identities by solving an optimization problem when unknown. SupSup achieves the state-of-the-art performance when task identities are given, and performs well even when task identities are missing at both training and inference time. One limitation of SupSup is that task inference fails when models are not well calibrated and overly confident for the wrong task. In addition, it relies on the Supermask setting, and is hard to be generalized to the common settings where the weights of the models are trained.

There is a question from the audience that is softmax a good confidence measurement, given the fact that it becomes overly confident on out-of-distribution and adversarial examples. The answer is softmax actually suffers from the over-confidence issue, and an theoretical proof can be found in [7].

References

[1] Cheung, B., Terekhov, A., Chen, Y., Agrawal, P., and Olshausen, B. Superposition of

many models into one. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., pp. 10868–10877.

[2] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

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

[4] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[5] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences 79*, 8 (1982), 2554–2558.

[6] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks, 2017.

[7] Liu, W., Wang, X., Owens, J. D., and Li, Y. Energy-based out-of-distribution detection, 2020.

[8] Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights, 2018.

[9] Pascanu, R., and Bengio, Y. Revisiting natural gradient for deep networks, 2014.

[10] Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 11890–11899.

[11] Rebuffi, S., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5533–5542.

[12] Wen, Y., Tran, D., and Ba, J. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations* (2020).