Improving and Understanding Variational Continual Learning

Siddharth Swaroop[*], **Cuong V. Nguyen**[*], **Thang D. Bui**[†], **Richard E. Turner**[*]
[*] University of Cambridge, UK, {vcn22,ss2163,ret26}@cam.ac.uk
[†] University of Sydney, Australia, thang.buivn@gmail.com

Abstract

In the continual learning setting, tasks are encountered sequentially. The goal is to learn whilst i) avoiding catastrophic forgetting, ii) efficiently using model capacity, and iii) employing forward and backward transfer. In this paper, we explore how the Variational Continual Learning (VCL) framework achieves these desiderata on two benchmarks in continual learning: split MNIST and permuted MNIST. We first report significantly improved results on what was already a competitive approach, achieved by establishing a new best practice approach to mean-field variational Bayesian neural networks. We then look at the solutions in detail. This allows us to obtain an understanding of why VCL performs as it does, and we compare the solution to what an 'ideal' continual learning solution might be.

1 Introduction

In the real world, inputs can change over time, suffering from covariate and dataset shift, and often the size of datasets makes revisiting old data computationally prohibitive. Continual Learning, where we receive data in an online fashion, allowing tasks to change and be added over time, focusses on these types of problems. An ideal continual learning solution would, when it first sees data, efficiently use its capacity to solve the task, without any forgetting of contributions from old data. Once it reaches its capacity, it would forget the 'least important' information from old tasks, such that it incorporates backward and forward transfer: new data can be useful for increasing performance on old tasks (backward transfer), and information from old tasks can be used to perform better on new tasks (forward transfer).

We shall specifically focus on the continual image classification setting, a standard continual learning problem tackled in research. Due to their success on batch image classification, neural networks are often employed. However, naive techniques such as vanilla neural networks suffer from catastrophic forgetting [3, 7]. In order to combat this, many approaches regularise parameter updates by penalising against changes in 'important' parameters for previous tasks. For example, Elastic Weight Consolidation (EWC) [6, 9] regularises using Fisher information matrices from previous tasks. Synaptic Intelligence (SI) [12] compares the (rate of change of) the objective's gradient and the parameters to decide each parameter's importance to previous tasks, and EWC and SI can be combined [2]. In Progress & Compress [10], an active column is trained on each new task (with some layerwise connections to the 'knowledge base'), which is then distilled into a knowledge base, using a modified version of EWC to prevent catastrophic forgetting.

However, these approaches often do not maintain accurate parameter uncertainty estimates after training on each task. Uncertainty is extremely useful in deciding the important information from previous tasks, allowing for less important information to be forgotten. A Bayesian approach to learning parameters should automatically keep uncertainty estimates, and so should perform well in the continual learning setting. We therefore consider learning a Bayesian neural network, where

Continual Learning Workshop, 32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

we set the posterior from the previous task as the prior for a new task. Exact Bayesian inference is not tractable, and so we approximate the posterior after every task. Although some of the previous approaches are closely related to approximate inference in a Bayesian neural network, we use the variational objective function, following the Variational Continual Learning (VCL) framework [8], and introduced in Section 2. Variational Inference (VI) is known to provide better uncertainty estimates that other approaches in this setting [1].

In Nguyen et al. [8], results were reported for multi-head split MNIST and permuted MNIST (both with and without episodic memory). In this paper, we report improved results on both these tasks, and look in detail at the solutions reached by VCL. This leads to some valuable insights on the behaviour we might expect from a 'good' continual learning model, both in terms of model capacity usage and forward/backward transfer, while also shedding some light on properties of mean-field variational Bayesian neural networks.

2 Improvements to Variational Continual Learning

In VCL, we adopt a Bayesian approach to learning parameters of a model, using the previous task's posterior as the new task's prior when we see new data. We approximate each task's posterior by a variational distribution, $q_t(\theta) \approx p_t(\theta|\mathcal{D}_\mathbf{t})$, obtained by using Monte Carlo variational inference, optimising the objective (for every task t)

$$\mathcal{L}^t_{\mathrm{VCL}}(q_t(\theta)) = \sum_{n=1}^{N_t} \mathbb{E}_{\theta\sim q_t(\theta)} \left[\log p(y_t^{(n)}|\theta, \mathbf{x}_t^{(n)})\right] - \mathcal{KL}\left(q_t(\theta)||q_{t-1}(\theta)\right). \tag{1}$$

There are two terms in Equation 1, which we shall refer to as the likelihood term (the left hand term with Monte Carlo estimates), and the $\mathcal{KL}$ term. For further details on the optimisation process, please see the original VCL paper [8]. We consider fully-connected multi-layer perceptrons in this paper, with ReLU activation functions, and use a Gaussian mean-field approximation over each of the weights, as in the original paper. We place a standard normal Gaussian over each parameter as the initial prior ($p_0(\theta)$), and can calculate the $\mathcal{KL}$ term in Equation 1 analytically. We use ADAM [4] to optimise for the means and variances of each parameter. We do not use episodic memory enhancement (coresets) in this work.

2.1 Improved results

We established a simple scheme for substantially improving results over Nguyen et al. [8]. Crucially, we optimise for a much longer time. Although progress can sometimes appear to stall during optimisation, in reality progress is just extremely slow. We also firstly introduce the local reparameterisation trick during the Monte Carlo sampling of the network [5], and secondly initialise all weights with small, random means (of the order 10^{-1}) and small variances (of the order 10^{-3}) before every task. Without these additional two tricks, we would have to wait for much longer in order to reach convergence. Although these improvements may seem technically trivial, we find that they are practically very important, greatly impacting results.

In the multi-head split MNIST task, we have to sequentially solve five binary classification tasks from the MNIST dataset: {0v1}, {2v3}, {4v5}, {6v7}, {8v9}. The challenge in split MNIST is to obtain good performance on new tasks while retaining performance on old ones. We ran a one hidden layer model with 200 units for 600 epochs (with 256 batch size), sharing the lower level weights between tasks. See Table 1 for results. We achieve a final test accuracy of 98.5$\pm$0.4% (mean over 10 runs, with standard deviation), an increase from 97.0% reported earlier [8].

We also consider permuted MNIST. In permuted MNIST, we receive tasks sequentially, where each task is the standard (10-way) MNIST classification task, with the pixels having undergone a fixed random permutation, different for each task. Ideally, a network with two or more hidden layers would use lower layer(s) to 'de-permute' the images, and higher layer(s) to solve MNIST, which is then constant between tasks. We ran a two hidden layer model with 100 units in each hidden layer for 800 epochs (with 1024 batch size), achieving a final average test accuracy of 93$\pm$1% (mean over 5 runs, with standard deviation), as compared to 90% reported earlier [8] (Table 1). For reference, training

Table 1: Final average test accuracy on split MNIST and permuted MNIST

Task	VCL (this paper)	Previous VCL [8]	EWC [6]	SI [12]
Permuted MNIST	93±1%	90%	84%	86%
Split MNIST	98.5±0.4%	97.0%	63.1%	98.9%

the same network but seeing all the data together (batch mode) has an accuracy of 97% (this is an upper bound on the performance possible with this model and inference scheme).

3 Discussion

We now look into how VCL continually learns tasks, exploring how it uses its model capacity, primarily by looking at the learnt weights in the network.

3.1 Split MNIST

Appendix A shows plots of weights into and out of each unit after training on each task. Even though we have 200 units in the single hidden layer, it appears as if only one unit is being used per each of the five tasks; the remaining units are pruned out as part of the optimisation process. These active (un-pruned) units are plotted in Figure 1. This effect is similar to that observed in Trippe and Turner [11], with entire units pruned out, as opposed to just individual weights. The pruned units appear to have input weights at or near the prior (standard normal Gaussian), with output weights near a delta function (zero mean, small variance), therefore minimising their effect on the output prediction. Removing all pruned units from the network does not change the network's predictions (and therefore accuracy).

This pruning effect seems to be due to the choice of inference scheme. We leave a detailed review of the pruning process for future work, including characterising the mechanism and explaining it mathematically. Intuitively, the pruning effect can be explained by looking at the optimisation function (Equation 1). By reducing the effect of a unit on the output prediction (setting output weights to have zero mean and small variance), the input weights to the unit can be set to their prior. The increase in the $\mathcal{KL}$ term due to the small variance of the output weights are offset by the reduction in the $\mathcal{KL}$ term from the numerically more input weights. Provided the likelihood term does not change too much, a pruned solution is therefore more optimal. In this paper, we now focus on what our pruned solutions reveal about how our model approaches continual learning tasks, and debate whether the pruning effect is a feature or a bug for continual learning.

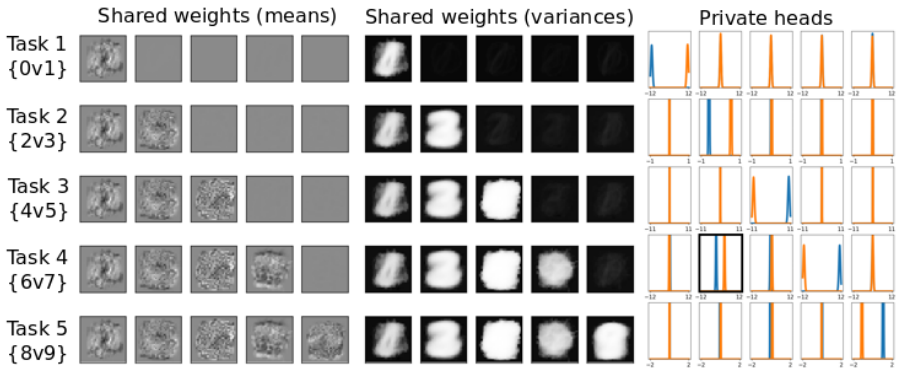

Figure 1: The active units learnt during split MNIST. Rows correspond to stages of continual learning. Left: means, Centre: variances, Right: output weights for each task's two classes.

As the model only uses 1 hidden unit per task, it has learnt to use a fraction of its total capacity to successfully learn binary classifications in the split MNIST experiment. The high test accuracies indicate this is an efficient use of model capacity. Remaining, unused units can be used for other

tasks that we may see in the future. This implies that pruning is a beneficial feature for continual learning, forcing the model to efficiently use its capacity.

Additionally, the pruning effect allows us to see some forward and backward transfer (see Figure 1), both important qualities in a good continual learning solution. Forward transfer is visible when previous tasks' active units have non-zero weights for subsequent tasks. For example, unit 2, which was learnt after task 2 (classifying digits {2v3}), has non-zero output weights after task 4 (classifying {6v7}). The model therefore uses some information about the task {2v3} in solving the task {6v7}. Although less visible in the plots, there is also backward transfer in the same units: unit 2's input weights change slightly after training on task 4 ({6v7}), potentially changing test accuracy on task 2. In this case however, any backward transfer does not result in different test accuracies; this could be because there is no potential for improvement given the high test accuracies involved.

3.2 Permuted MNIST

We now look into how the two hidden layer model approaches permuted MNIST. Appendix B shows figures of the weights, showing there is still pruning. The numbers of active (un-pruned) units after training on each task are summarised in Figure 2.

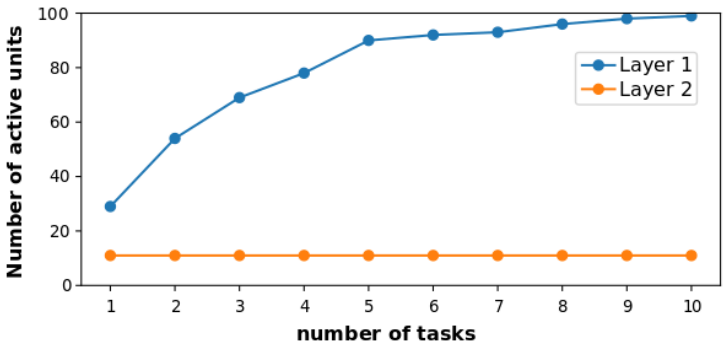

Figure 2: Number of active units per hidden layer after each task in permuted MNIST.

There are more active units in permuted MNIST than were in split MNIST, perhaps due to the more difficult nature of permuted MNIST (classifying between 10 digits, as opposed to between 2). However, only 11 units are used in the 2nd hidden layer, with the remaining 89 units pruned out. Additionally, the output weights on these 11 units do not change between tasks (see Appendix B). This confirms that the hidden layers effectively de-permute the images, allowing the output weights to just classify between the 10 digits.

Beyond re-using the upper level weights, there is not much evidence of forward or backward transfer. We should expect this from permuted MNIST because the network trains on all MNIST digits on the first task itself, hence already learning the 'best' way to classify between MNIST digits. Any subsequent permuted images cannot improve this. Instead, the remaining focus of permuted MNIST seems to be on ensuring we use available model capacity as efficiently as possible. Increasing the model capacity improves results: training a network with 250 units in the lower hidden layer (instead of 100) improves final test accuracy to 95.5% (10 tasks).

4 Conclusions and future work

We reported improved results from earlier Variational Continual Learning work [8] on split MNIST and permuted MNIST, and discussed how the obtained solutions achieved these results. Future work would involve applying VCL to other, tougher benchmarks, where there is more potential for forward and backward transfer. We should also explore the observed pruning effect, a consequence of using mean-field variational Bayesian neural networks.

References

[1] T. Bui, D. Hernandez-Lobato, J. Hernandez-Lobato, Y. Li, and R. Turner. Deep gaussian processes for regression using approximate expectation propagation. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1472–1481, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/bui16.html.

[2] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *CoRR*, abs/1801.10112, 2018. URL http://arxiv.org/abs/1801.10112.

[3] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6211.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[5] D. P. Kingma, T. Salimans, and M. Welling. Variational Dropout and the Local Reparameterization Trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick.pdf.

[6] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 2017*, 2017. URL http://arxiv.org/abs/1612.00796. arXiv: 1612.00796.

[7] M. Mccloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169, 1989.

[8] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkQqq0gRb.

[9] H. Ritter, A. Botev, and D. Barber. Online Structured Laplace Approximations For Overcoming Catastrophic Forgetting. *Preprint*, May 2018. URL http://arxiv.org/abs/1805.07810. arXiv: 1805.07810.

[10] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & Compress: A scalable framework for continual learning. *ICML 2018*, May 2018. URL http://arxiv.org/abs/1805.06370. arXiv: 1805.06370.

[11] B. Trippe and R. Turner. Overpruning in Variational Bayesian Neural Networks. *arXiv:1801.06230 [stat]*, Jan. 2018. URL http://arxiv.org/abs/1801.06230. arXiv: 1801.06230.

[12] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/zenke17a.html.

Appendix A

This appendix has figures showing the means and variances of weights after training on each of the five tasks in split MNIST (MLP with 1 hidden layers, 200 hidden units).

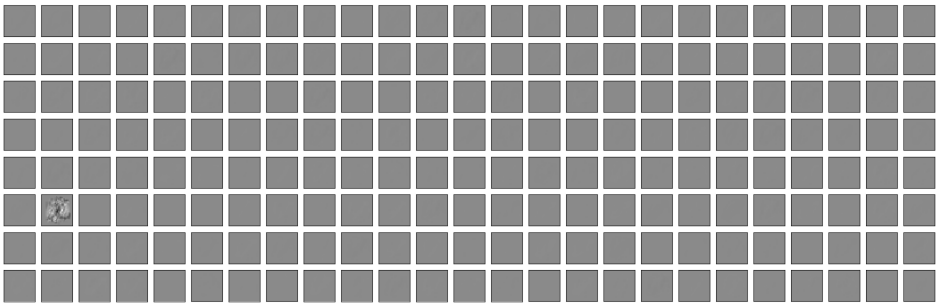

Figure 3a: Hidden layer input means after training on task 1

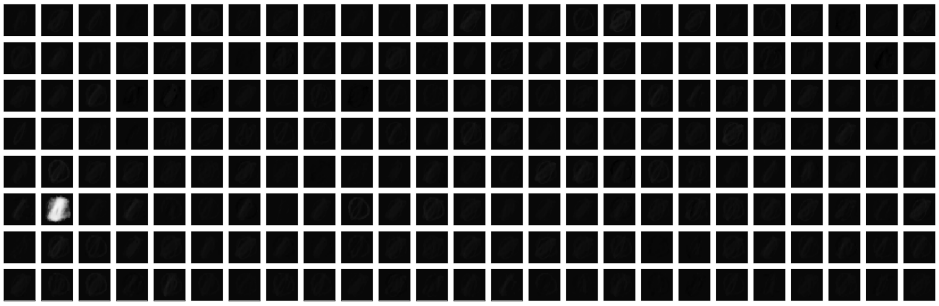

Figure 3b: Hidden layer input variances after training on task 1

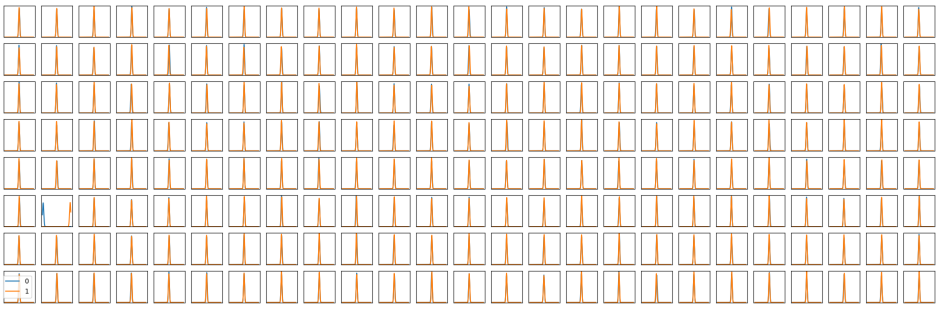

Figure 3c: Hidden layer output weights after training on task 1

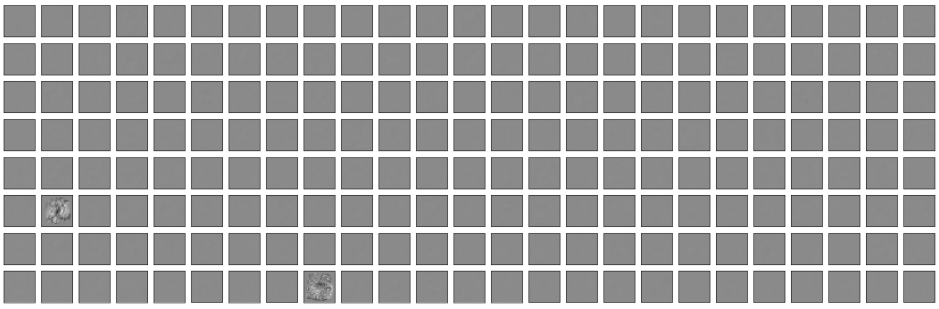

Figure 4a: Hidden layer input means after training on task 2

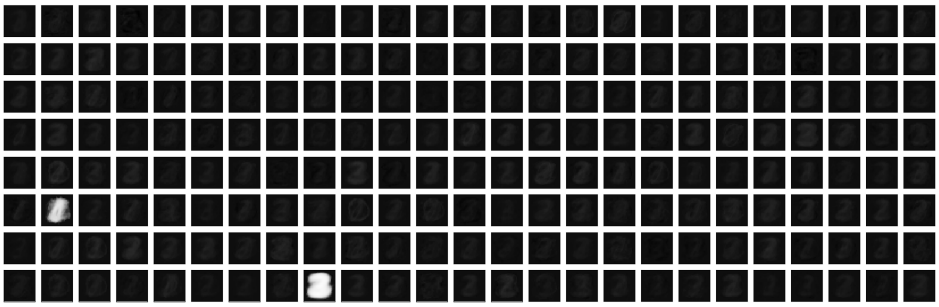

Figure 4b: Hidden layer input variances after training on task 2

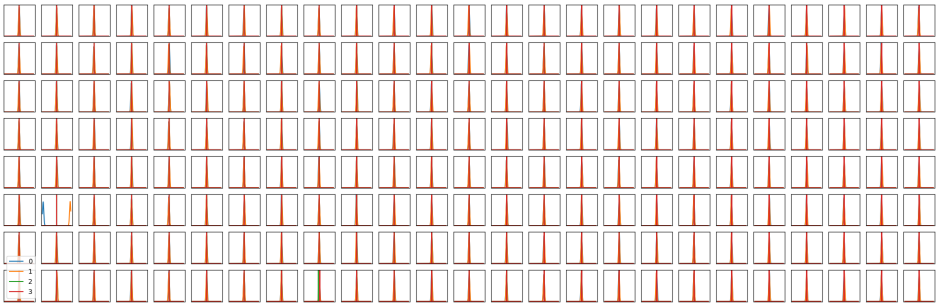

Figure 4c: Hidden layer output weights after training on task 2

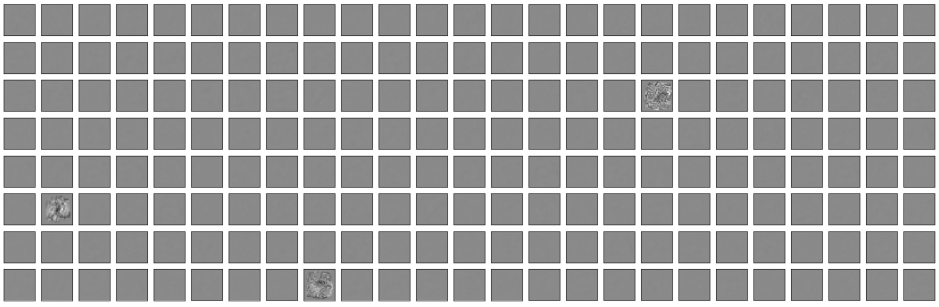

Figure 5a: Hidden layer input means after training on task 3

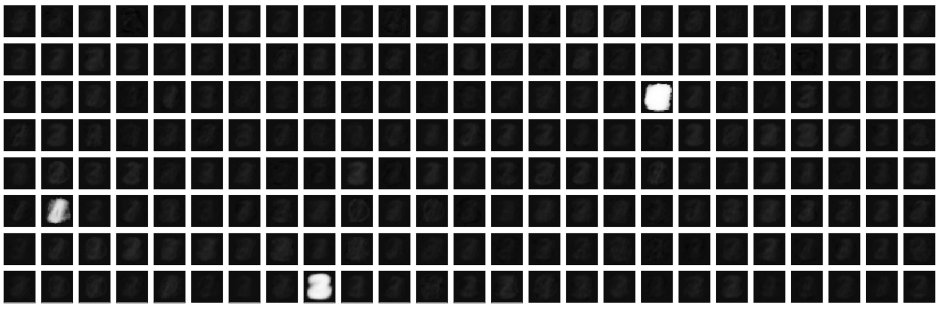

Figure 5b: Hidden layer input variances after training on task 3

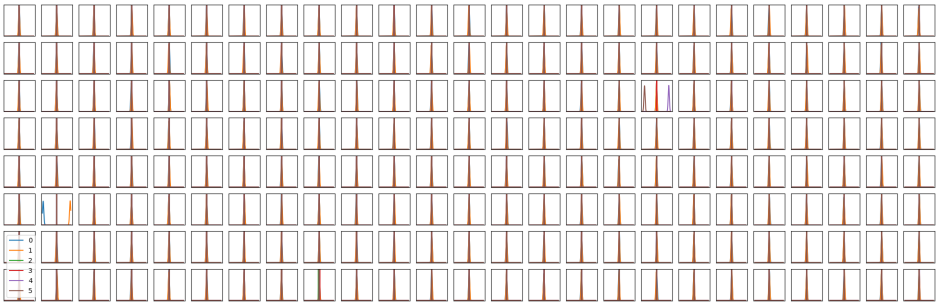

Figure 5c: Hidden layer output weights after training on task 3

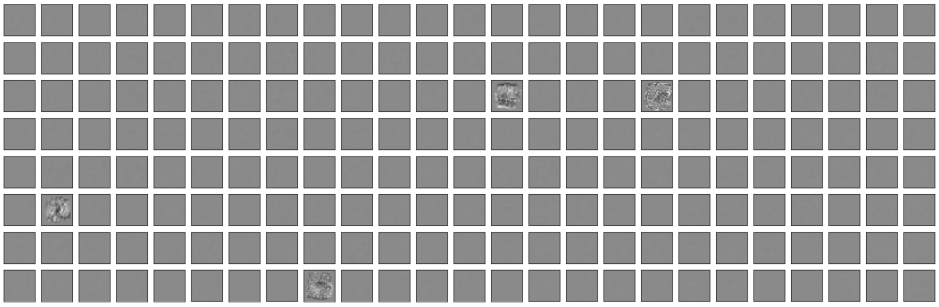

Figure 6a: Hidden layer input means after training on task 4

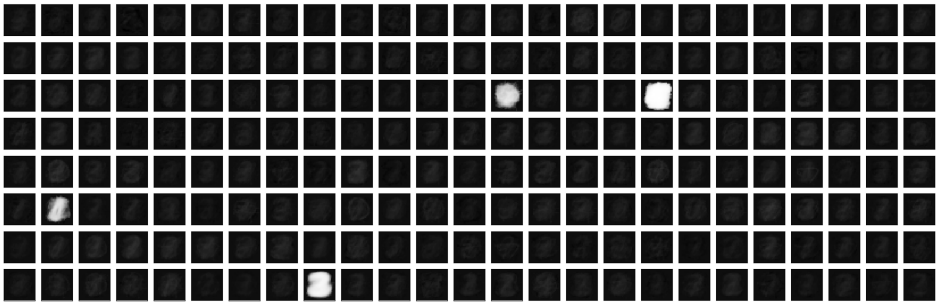

Figure 6b: Hidden layer input variances after training on task 4

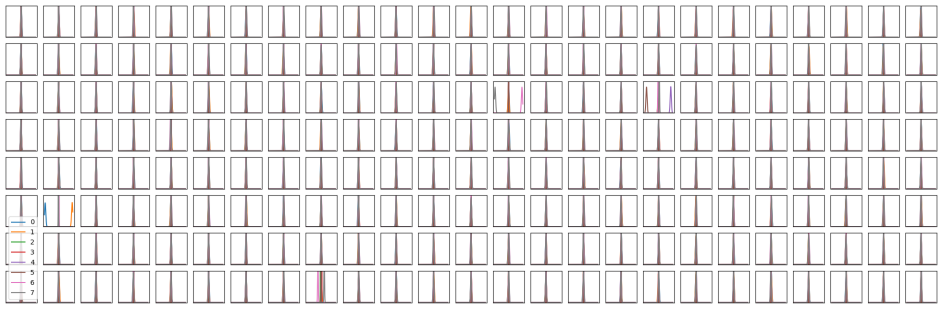

Figure 6c: Hidden layer output weights after training on task 4

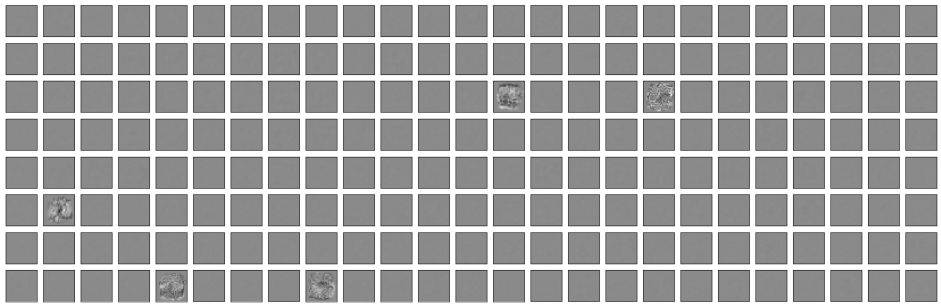

Figure 7a: Hidden layer input means after training on task 5

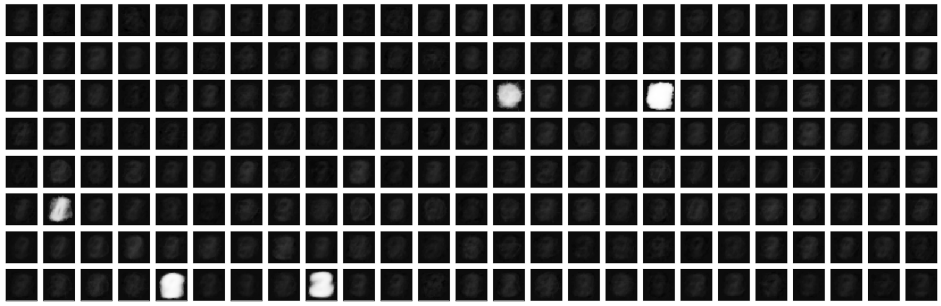

Figure 7b: Hidden layer input variances after training on task 5

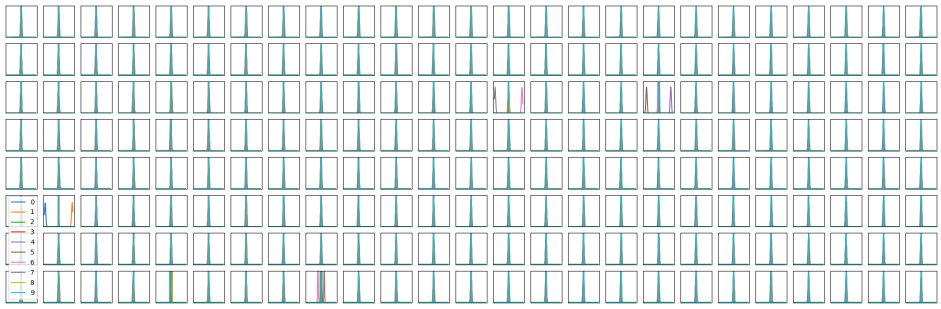

Figure 7c: Hidden layer output weights after training on task 5

Appendix B

This appendix has figures showing the means and variances of weights after training on tasks 1, 5 and 10 on permuted MNIST (MLP with 2 hidden layers, 100 units in each layer, 10 tasks total).

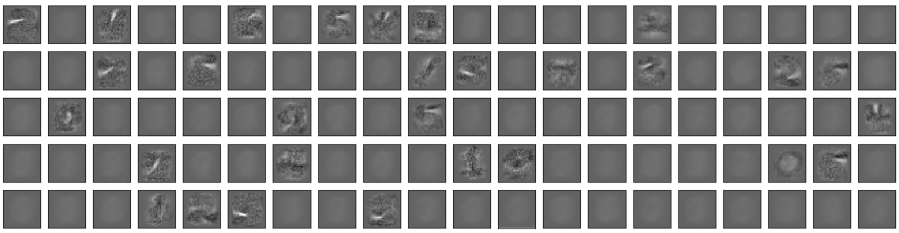

Figure 8a: 1st hidden layer input means after training on task 1

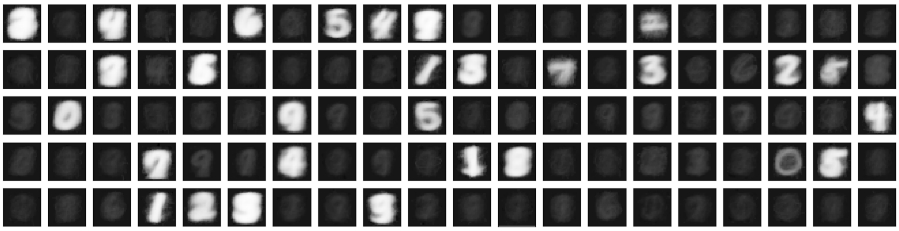

Figure 8b: 1st hidden layer input variances after training on task 1

Figure 8c: 2nd hidden layer input means after training on task 1

Figure 8d: 2nd hidden layer input variances after training on task 1

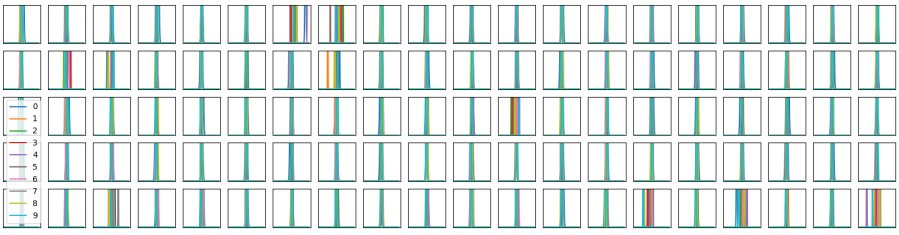

Figure 8e: 2nd hidden layer output weights after training on task 1 (best viewed in colour)

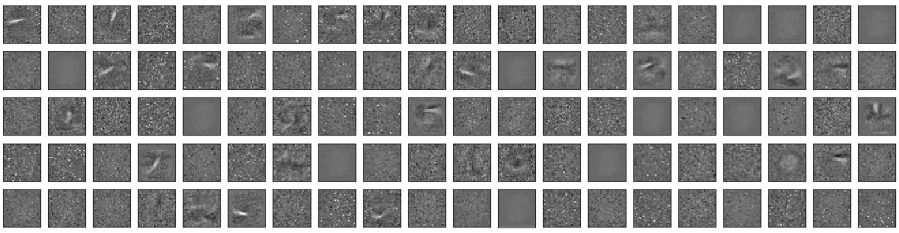

Figure 9a: 1st hidden layer input means after training on task 5

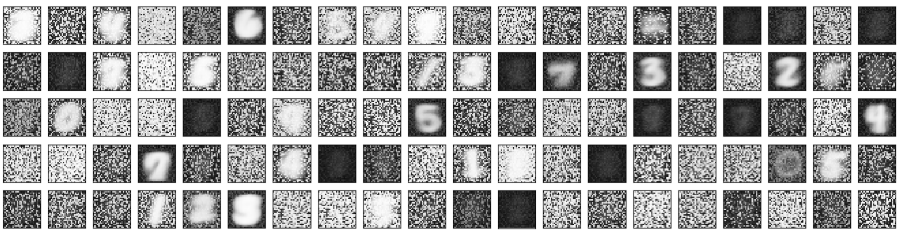

Figure 9b: 1st hidden layer input variances after training on task 5

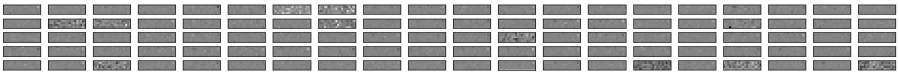

Figure 9c: 2nd hidden layer input means after training on task 5

Figure 9d: 2nd hidden layer input variances after training on task 5

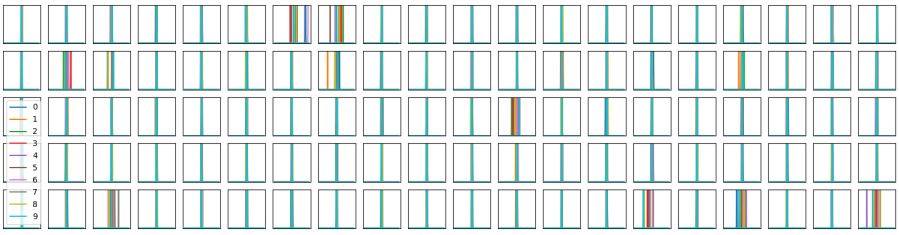

Figure 9e: 2nd hidden layer output weights after training on task 5 (best viewed in colour)

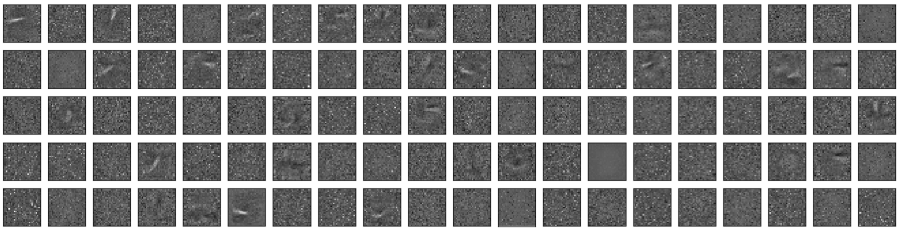

Figure 10a: 1st hidden layer input means after training on task 10

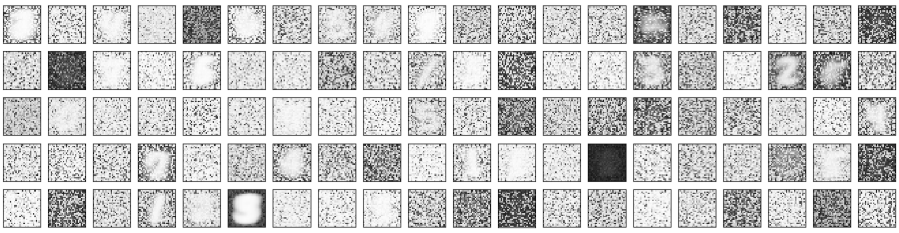

Figure 10b: 1st hidden layer input variances after training on task 10

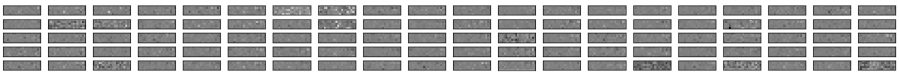

Figure 10c: 2nd hidden layer input means after training on task 10

Figure 10d: 2nd hidden layer input variances after training on task 10

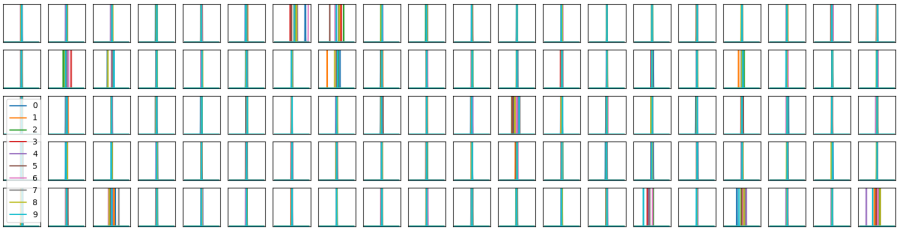

Figure 10e: 2nd hidden layer output weights after training on task 10 (best viewed in colour)