Master's thesis

Variational Inference for Continual Learning by using Weight-Perturbation in Adam

(継続学習のためのAdamにおける重み摂動を用いた近似ベイズ推論)

48-166545, Hanna Tseran

Prof. Tatsuya Harada

August 3, 2018

Department of Mechano-Informatics
Graduate School of Information Science and Technology
The University of Tokyo

Variational Inference for Continual Learning by using Weight-Perturbation in Adam
(継続学習のためのAdamにおける重み摂動を用いた近似ベイズ推論)

48-166545 Hanna Tseran, Prof. Tatsuya Harada

Keywords: Continual Learning, Variational Inference, Bayesian Neural Network, Weight Uncertainty, Weight-Perturbation in Adam

1. Introduction

In continual learning a model is trained on several tasks in a sequence. After training on one of the tasks is finished, the training data for it is discarded. However, the model should perform well on all tasks. This problem is important for applications such as robotics [1], where new tasks can appear during the model usage and pre-training on all possible tasks or re-training with all data is infeasible.

The goal of this thesis is to introduce an algorithm for continual learning that performs well, but can be easily implemented and reused. Suggested method is called VadamVCL, and it can be used instead of a usual optimization algorithm to enable continual learning with a model. VadamVCL is similar to Adam [2] algorithm that is extensively used by the community and, therefore, easy to implement based on the Adam implementation. VadamVCL performs well despite its simplicity, and its general form allows to connect it to other methods.

2. Related Work

Approaches to continual learning can be broadly divided into two types: methods based on the external memory and methods based on the modifications to the training algorithm. Suggested method belongs to the second type. Among similar methods are Variational Continual Learning (VCL) [3], Elastic Weight Consolidation (EWC) [4] and Synaptic Intelligence (SI) [5]. VCL is applicable only to Bayesian neural networks (BNNs). Both EWC and SI require to add term to the loss function that penalizes changes to the model parameters based on the separately computed measure of importance. VadamVCL uses similar ideas, however, it does not require to define a BNN or to change a loss function, and the penalties does not require additional computation.

3. Preliminaries

3.1 Variational Continual Learning

Variational continual learning (VCL) [3] is an approach based on the idea that Bayesian inference can be used as a general framework for continual learning. After training on one of the tasks is finished, all model knowledge is incorporated in the posterior. If this posterior is reused as a prior during the training for the next task, this knowledge is incorporated into the model automatically. VCL main idea can be summarized by the recursive formula:

$$p(\boldsymbol{\theta}|\mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{1:T-1})p(\mathcal{D}_T|\boldsymbol{\theta}), \qquad (1)$$

where T is the number of tasks, $\boldsymbol{\theta}$ are model parameters and $\mathcal{D}_t$ is a dataset for the task t.

3.2 Vadam: Variational Inference Using Weight-Perturbation in Adam

Adam [2] is an optimization algorithm that is widely used for training neural networks and implemented in all popular libraries. Vadam [6] is a natural gradient method for Gaussian mean-field variational inference that uses several approximations to make the implementation easier and more similar to Adam. This way Vadam allows to perform variational inference for BNNs through the optimizer. One of the main differences in Vadam algorithm implementation compared to Adam is weight-perturbation - sampling of the model weights before computing the gradients.

4. VadamVCL for Continual Learning

VadamVCL is based on the combination of VCL [3] and Vadam [6]. The algorithm is derived by replacing a prior distribution in the Vadam derivation with a previous task posterior distribution according to the VCL equation 1. The method uses a Bayesian neural network though it does not require a user to define it. Bayesian neural networks allow to use a Bayesian inference framework for continual learning and provide uncertainty estimates for the model parameters. Uncertainty can be interpreted as a measure of how important a given parameter is for a model to make a correct prediction.

4.1 Algorithm

Algorithm 1: VadamVCL algorithm for continual learning. All updates are point-wise. Differences from Adam are highlighted in blue.

for dataset $\mathcal{D}_t$, $N = |\mathcal{D}_t|$ for the task $t, t = \overline{1,T}$ **do**
  $\boldsymbol{m}_0 = 0, \boldsymbol{s}_0 = 0$
  if $t == 1$ **then**
    $\boldsymbol{\mu}_* = 0, \boldsymbol{\sigma}_*^{-2} = \lambda$
  else
    $\boldsymbol{\mu}_* = \boldsymbol{\mu}_{\text{last}}^{(t-1)}, \boldsymbol{\sigma}_*^{-2} = \boldsymbol{s}_{\text{last}}^{(t-1)} + (\boldsymbol{\sigma}_*^{(t-1)})^{-2}$
  for $k = \overline{1,K}$, K - number of epochs **do**
    $\boldsymbol{\theta}_k = \boldsymbol{\mu}_k + \epsilon / \sqrt{\boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}}, \epsilon \sim \mathcal{N}(\epsilon|0,1)$
    $\boldsymbol{m}_{k+1} =$
      $\beta_1 \boldsymbol{m}_k + (1-\beta_1)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\boldsymbol{\sigma}_*^{-2}]$
    $\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)\gamma N^2 \operatorname{diag}[\boldsymbol{H}(\boldsymbol{\theta}_k)]$
    $\hat{\boldsymbol{m}}_{k+1} = \boldsymbol{m}_k / (1-\beta_1^{k+1})$
    $\hat{\boldsymbol{s}}_{k+1} = \boldsymbol{s}_k / (1-\beta_2^{k+1})$
    $\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \eta\, \hat{\boldsymbol{m}}_{k+1} / (\sqrt{\hat{\boldsymbol{s}}_{k+1}} + \boldsymbol{\sigma}_*^{-2})$
  Discard the dataset $\mathcal{D}_t$

Here $\boldsymbol{\theta}$ are model parameters, $\boldsymbol{\mu}, \boldsymbol{\sigma}^2$ are parameters of the weights distribution, $*$ denotes parameters of the previous posterior, $\hat{\boldsymbol{g}}$ is the estimated gradient, $\boldsymbol{H}$ is the Hessian matrix, $\lambda, \beta_1, \beta_2, \eta, \gamma$ are method parameters.

Main differences from Adam are in the initialization of additional parameters (lines 4 and 6), weight-perturbation (line 8), contribution from the previous task posterior (line 9) and diagonal of the Hessian (line 10).

4.2 Approximation to the Hessian and Connection to EWC

The algorithm presented above uses diagonal of the Hessian matrix that is slow to compute and requires approximation. Two types of approximations were considered: gradient magnitude (GM) and Generalized Gauss-Newton (GGN) [7].

GM approximation:

$$\boldsymbol{H}(\mathcal{L}(\boldsymbol{\theta})) \approx \mathbb{E}\Big[\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]\mathbb{E}\Big[\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]^T. \quad (2)$$

GGN approximation:

$$\boldsymbol{H}(\mathcal{L}(\boldsymbol{\theta})) \approx \mathbb{E}\Big[\Big(\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big)^2\Big]. \quad (3)$$

The difference in practice is whether the average over the mini-batch is taken before computing the square (GM) or after (GGN). GM approximation is faster but less accurate than GGN.

If GM approximation is used, then line 10 in the algorithm becomes $\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)\gamma N^2 \hat{\boldsymbol{g}}^2$ and differs from Adam only in γN^2.

If GGN approximation to the Hessian is used with the negative log likelihood loss, then Hessian is in fact approximated with a Fisher information matrix. It makes the method similar to EWC in this case, since EWC works by penalizing parameter changes using a Fisher information matrix [4].

5. Experiments

In this section we will demonstrate on the standard continual learning benchmarks that despite VadamVCL simplicity it performs comparably to other methods.

5.1 Permuted MNIST

During experiments with the MNIST dataset gradient-magnitude approximation to the Hessian was used because of its speed. For experiments with Permuted MNIST the dataset received for each task consists of images of MNIST digits which pixels have undergone a fixed random permutation.

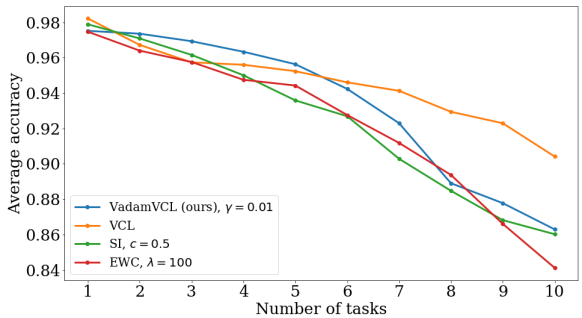


Figure 1: Results on the Permuted MNIST dataset.

VadamVCL performs on the Permuted MNIST comparably to SI and EWC and worse than VCL in the long run. The reason most probably is the uncertainty estimates that are computed differently in VCL and VadamVCL and involve approximation in the case of VadamVCL.

5.2 Split MNIST

For this experiment MNIST digits dataset is split into 5 pairs: 0/1, 2/3, 4/5, 6/7, 8/9, The dataset received for each task is one of these pairs.

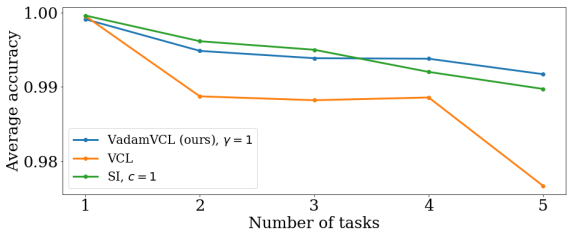


Figure 2: Results on the Split MNIST dataset. EWC results are omitted because of its poor performance.

VadamVCL performs on this experiment better than EWC and VCL and comparable to SI. The likely reason of better performance is the usage of multi-head network. It means that for each new task there are new parameters on which performance on the previous tasks does not depend. VadamVCL penalizes changes to previous parameters stronger than other methods, but new parameters allow to get good performance on new tasks that results in better performance overall.

6. Conclusion

In this thesis we have introduced an algorithm called VadamVCL for continual learning. It is easy to implement and use, and results show that despite the simplicity its performance is comparable to other continual learning methods. The algorithm also shows an interesting connection to EWC. Since the method has a simple expression it allows for various improvements to the method to be done easily. As the future work it is possible to try better approximations to the Hessian and better approximations to the posterior instead of using mean-field variational inference.

Bibliography

[1] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. *arXiv preprint arXiv:1705.03550*, 2017.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[4] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13): 3521–3526, 2017.

[5] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.

[6] Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. *arXiv preprint arXiv:1806.04854*, 2018.

[7] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Abstract

In continual learning a model is trained on several tasks in a sequence, and after training on one of the tasks is finished, the training data for it is discarded. However, the model should perform well on all tasks. In this thesis we introduce an algorithm for continual learning called VadamVCL that makes use of variational inference for continual learning. VadamVCL can be used instead of a usual optimization algorithm to enable continual learning with a model. It is also similar to Adam algorithm that is extensively used by the community and, therefore, easy to implement based on the Adam implementation. VadamVCL performance is comparable to the state of the art despite its simplicity, and its general form allows to connect it to other methods.

Contents

List of Figures

List of Tables

Acknowledgments

This work was made during an internship in RIKEN Center for Advanced Intelligence Project in the Approximate Bayesian Inference team led by Dr. Mohammad Emtiyaz Khan and while I was doing my Master's under the supervision of Prof. Tatsuya Harada. The initial idea was developed during discussions with Dr. Khan and Didrik Nielsen. I am also thankful to Didrik Nielsen, Frederik Kunstner and Aaron Mishkin for helping with the code and useful conversations about Vadam, and to Thang D. Bui for the idea of the mean-estimation experiment.

During my Master's studies I was supported by the scholarship from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

Chapter 1

Introduction

Continual or **lifelong** learning is a setting in which a model is required to **learn several tasks in a sequence**. After training on one of the tasks is finished, **training data for this task is discarded**. It is desired that the model performance on the previously seen tasks does not decrease. The model also should be able to reuse the knowledge both from the previous tasks to ease learning of the new ones, and from the new ones to improve the performance on the previous tasks. The main problem we am concerned with in this thesis is **catastrophic forgetting** - drop in the performance on the previous tasks.

While humans are capable of learning during their lifetime, most of the currently popular machine learning methods are not capable of learning in a lifelong manner. Neural networks that are the most popular approach in machine learning these days are also prone to catastrophic forgetting in the continual learning setting (Goodfellow et al., 2013). If a machine is working in a real-life environment for a long time it will ultimately face many tasks during its operation time. Thus, continual learning is essential to develop systems that can work in such settings. The necessity for continual learning was motivated long time ago in application to robotics (Thrun and Mitchell, 1995), because a robot might face a variety of tasks during its interaction with the physical environment. Other potential applications are intelligent assistants and chat-bots (Chen and Liu, 2016). Continual

learning in general and continual learning with neural networks is described in more detail in the chapter 2.

Bayesian inference provides a general framework for continual learning. (Nguyen et al., 2017) Additionally, uncertainty estimates for the model weights are useful for continual learning because they provide a convenient way to determine the acceptable change rate for the weights. When it comes to neural networks, Bayesian neural networks provide both the uncertainty estimates on the network weights and allow to adapt Bayesian inference approach to continual learning. However, existing solutions based on these ideas cannot be applied directly to a usual neural network. An overview of the related topics is given in the chapter 3. Suggested method is based on the combination of Vadam (Khan et al., 2018) and Variational Continual Learning (VCL) (Nguyen et al., 2017) that are described in the sections 3.3 and 3.4 respectively.

An easy to implement and build on algorithm speeds up the development of new methods. In this thesis we suggest **an optimization algorithm that can make a model work in a continual learning setting simply by replacing the optimization algorithm used**. We call this algorithm **VadamVCL**. It is **similar to Adam** (Kingma and Ba, 2014) algorithm that is extensively used by the community and, therefore, **easy to implement** based on the Adam implementation. *It does not require to define a Bayesian neural network or to change a loss function.*

VadamVCL is derived in the chapter 4. It requires Hessian approximation and method results depend on the type of the approximation used. Explanation of this problem is given in the section 4.3.2. Depending on the approximation it is possible to establish connections to other continual learning methods and in the section 4.3.2 we show connection to the Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017). VadamVCL was developed for neural networks, however since the algorithm in fact uses a Bayesian neural network, the resulting network has the same limitations as modern Bayesian neural networks do. It means that for now we can guarantee that the method is applicable to relatively

small neural networks consisting of fully-connected layers.

Experiment results are given in the chapter 5. VadamVCL was tested on a Gaussian mean estimation, 2D classification, Permuted and Split MNIST experiments. The results show that **the method performs comparably or better than other continual learning methods despite its simplicity**. Conclusion and discussion are given in the chapter 6.

To sum up, the goal of this thesis was to introduce an algorithm for continual learning that **performs well, but can be easily implemented and reused**. As the result, we suggested a new method called VadamVCL that can be used instead of a usual optimization algorithm to enable continual learning with a model. Experiments confirmed that VadamVCL performance is comparable to the state of the art despite its simplicity, and its general form allows to connect it to other methods.

Chapter 2

Continual Learning

There has been a recent rise in the interest in continual learning. Because of this, there is no commonly accepted terminology yet, and the setting described below appears under the names of **continual**, **lifelong** or **continuous** learning in the research. We will use the name "Continual Learning" throughout this thesis.

Continual learning is a setting in which data arrives continuously, and it is not necessarily i.i.d. Tasks may change over time, and new tasks can emerge. It is required that the model improves its performance on all tasks as new data appears (*transfer and reverse transfer learning*), and performance on the old tasks does not decrease (*no catastrophic forgetting*). Majority of the recently suggested methods focuses on alleviating catastrophic forgetting and this is also the focus of the method we describe.

Let us provide the motivation for continual learning. If the machine learning models are to be moved to the real-life settings they need to be capable of adapting the continuously incoming stream of data. In real-life new tasks emerge all the time, but at the same time it is necessary to preserve performance on the previously seen tasks and ideally improve performance on them if new related information is presented. One of the examples is object recognition in a continual learning setting (Lomonaco and Maltoni, 2017). For instance, consider

a domestic robot. Pre-training such a robot beforehand on all possible tasks it might encounter is infeasible. At the same time, re-training it for instance once a day using all data with new data added would be slow and would require a lot of storage for all available information. Simple training with newly arrived data would result in catastrophic forgetting. In such cases continual learning can provide a solution to this problem allowing to train a robot using only new data, but preserving performance on the previously learned tasks.

To avoid the confusion, it is necessary to note that there are various settings that appear to be similar to continual learning. Below is their comparison to continual learning.

On-line learning (Bishop, 2006) refers to the type of learning when data points are considered one at a time and are discarded after the model parameters are updated. Additionally, in on-line learning only one task considered. This is not the case of continual learning when the whole dataset for one of the tasks is available at some point of time before it is discarded, and new tasks may appear.

Another similar setting is *multi-task learning* (Caruana, 1998). However, in multi-task learning the model parameters are optimized jointly for all of the tasks and data for all of them is always available, while in continual learning the model is presented with tasks in the sequential manner and all data for only one of them is available at a given moment in time.

One more related type of learning is *meta learning* (Lemke et al., 2015). Similarly to continual learning it is applied to the set of tasks. However, it assumes existence of two systems: a meta-learning system and a learning subsystem which is not a requirement for continual learning.

Another closely related type of learning is *transfer learning*. Transfer learning can be seen as a part of continual learning, while transfer learning on its own does not result in continual learning because performance on the previous tasks is not important for transfer learning.

2.1 Definition

Since the interest in continual learning has re-emerged recently, there is no commonly accepted strict definition of the term. Thus, we will describe the setting we will use more precisely in this section.

Suppose we want to train a model on T tasks, where T is not known in advance and is used to make the notation more convenient. Each task is characterized by a dataset $\mathcal{D}_t, t = \overline{1,T}$. There is no assumption of data being iid, and tasks might be related or not related to each other. Datasets $\mathcal{D}_1 = \{x_i, y_i\}_{N_1}, \mathcal{D}_2 = \{x_i, y_i\}_{N_2}, ..., \mathcal{D}_T = \{x_i, y_i\}_{N_T}$ arrive in a sequence. After training on one of the datasets is finished, it is discarded, and there is no access to it anymore. The model is to be trained on the tasks one-by-one. It is desired that after training on a new dataset $\mathcal{D}_t, t = \overline{1,T}$ performance on the previous $\mathcal{D}_1, \mathcal{D}_2, ...\mathcal{D}_{t-1}$ is preserved (no catastrophic forgetting) and knowledge from the previous tasks is transferred to $\mathcal{D}_t$. (Ideally, it should be transfered in both directions: from $\mathcal{D}_1, \mathcal{D}_2, ...\mathcal{D}_{t-1}$ to $\mathcal{D}_t$ and from $\mathcal{D}_t$ to $\mathcal{D}_1, \mathcal{D}_2, ...\mathcal{D}_{t-1}$.)

In general tasks might be of a different nature, but we limit the setting and use only the same type of tasks on the same types of data for one experiment, e.g. only image classification problems.

One of the biggest concerns is such a setting is catastrophic forgetting that was already mentioned. Catastrophic forgetting is a situation when a model was trained on tasks $\mathcal{D}_1, ..., \mathcal{D}_{t-1}$, and training on the task $\mathcal{D}_t$ results in the serious drop in performance on the previous tasks. Neural networks, while being the most popular method in machine learning these days, are especially prone to catastrophic forgetting (Goodfellow et al., 2013).

2.2 Related Work

Current approaches to continual learning can be roughly divided into two types. One of them uses some sort of memory either by explicitly saving some data from the previous task, or through generative replay. Another approach

is about modifying the training algorithm to preserve performance on previous tasks. There are also methods that combine both of these approaches. We did not use external memory for the suggested method, but it can be combined with some form of memory.

In the work on the **Elastic weight consolidation (EWC)** (Kirkpatrick et al., 2017) authors add to the loss l_2 constraints weighed by the diagonal of the empirical Fisher information matrix, one for each previous task. Fisher information matrix is treated as a measure of importance of a given parameter. Our method takes a different approach and does not modify the loss, but modifies the training algorithm itself. However, resulting algorithm also penalizes changes to the model parameters but uses uncertainty as a measure of importance.

Synaptic Intelligence (SI) (Zenke et al., 2017) similarly to to EWC uses a measure of "importance" to penalize changes to more important parameters. It adds surrogate loss to the cost function consisting of a sum of l_2 constraints multiplied by the importance estimation based on a loss and parameters changing rate. SI estimates importance of the parameters on-line and along the whole learning trajectory. Our method is different from SI in the same way it is different from EWC.

Gradient Episodic Memory (GEM) (Lopez-Paz et al., 2017) is an example of a memory based method. It makes use of an external memory that stores a subset of samples from observed tasks. GEM uses an update rule that projects gradients for new samples in such a way that they would not increase loss for the stored samples and allows backward transfer thanks to that. VadamVCL takes a different approach and does not use an external memory at all.

Incremental Moment Matching (IMM) (Lee et al., 2017) constructs a new Gaussian posterior for all seen tasks by either averaging the network parameters or doing Laplace approximation for the mixture of Gaussians. This method takes a totally different approach from our method.

Related works based on the usage of variational inference for continual learning will be introduced in the next chapter in the section 3.5.

Chapter 3

Variational Inference and Continual Learning

The notation that is used throughout the thesis including this chapter is described in detail in the chapter 4 in the section 4.1.

3.1 Bayesian Neural Networks

The method was developed with Bayesian Neural Networks (BNNs) (Neal, 2012) in mind. Though the suggested algorithm does not require a user to construct a BNN, since Vadam allows to get distribution parameters through the optimizer, it is necessary to introduce them to make the explanation clearer. Usage of Bayesian Neural Networks allows us to approach continual learning problem in neural networks using Bayesian inference as a general framework for continual learning (Nguyen et al., 2017).

The difference between Bayesian Neural Networks and vanilla neural networks is in the *prior introduced over the weights* in the case of BNNs. This is useful because plain neural networks are prone to over-fitting and does not provide uncertainty estimates over its decisions that are especially useful in some spheres, e.g. in medicine. BNNs can be trained using variational Bayesian learning by

minimizing expected lower bound (ELBO) (Blundell et al., 2015).

3.2 Mean-Field Variational Inference

Variational inference (Murphy, 2012) is one of the approximate inference methods. The basic idea is to pick an approximation $q(\boldsymbol{\theta})$ from some tractable family and to make it as close as possible to the true posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$.

In this work we use a *mean-field approximation* to the posterior (Murphy, 2012). The idea of mean-field variational inference is to approximate posterior with a product of distributions of the simpler form:

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta}) = \prod_i q_i(\boldsymbol{\theta}) \tag{3.1}$$

In the case of a multivariate Gaussian distribution that will be used throughout the thesis it means that approximation is done with $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$. The prior that will be used is $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|0, \frac{1}{\lambda}\boldsymbol{I})$.

3.3 Vadam

Vadam (Khan et al., 2018) is one of the algorithms that was used as a basis of the suggested approach to continual learning. Vadam is not concerned with continual learning on its own. It is a general method for mean-field variational inference that can be used for training neural networks in the same way as other popular optimizers, such as Adam or RMSprop. Vadam is developed based on a natural momentum method applied to the mirror descent algorithm and uses momentum term and bias-corrections like Adam. Since Vadam allows to perform variational inference through the optimizer, it is useful for writing a general optimizer for continual learning that makes use of variational inference.

Adam (Kingma and Ba, 2014) is a popular optimization algorithm that is often used for training neural networks. It is implemented in most of the machine

learning libraries, that means that a method similar to it can be easily implemented and used based on the available solutions. Vadam is similar to Adam in its updates, as it is shown in the table 3.1. (Detailed explanation of the notation is given in the section 4.1.)

Adam	**Vadam**
$\boldsymbol{\theta}_k = \boldsymbol{\mu}_k$	$\boldsymbol{\theta}_k = \boldsymbol{\mu}_k + \frac{\epsilon}{\sqrt{\boldsymbol{s}_k + \lambda}}, \epsilon \sim \mathcal{N}(\epsilon\|0, 1)$
$\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1-\beta_1)[\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]$	$\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1-\beta_1)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + \lambda\boldsymbol{\mu}_k]$
$\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)[\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]^2$	$\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]^2$
$\hat{\boldsymbol{m}}_{k+1} = \frac{\boldsymbol{m}_k}{1-\beta_1^{k+1}}$	$\hat{\boldsymbol{m}}_{k+1} = \frac{\boldsymbol{m}_k}{1-\beta_1^{k+1}}$
$\hat{\boldsymbol{s}}_{k+1} = \frac{\boldsymbol{s}_k}{1-\beta_2^{k+1}}$	$\hat{\boldsymbol{s}}_{k+1} = \frac{\boldsymbol{s}_k}{1-\beta_2^{k+1}}$
$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\eta}{\sqrt{\hat{\boldsymbol{s}}_{k+1}} + \delta}\hat{\boldsymbol{m}}_{k+1}$	$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\eta}{\sqrt{\hat{\boldsymbol{s}}_{k+1}} + \lambda}\hat{\boldsymbol{m}}_{k+1}$

Table 3.1: Adam update versus Vadam update. Differences are highlighted in blue. Main differences are in the weight-perturbation in the first line and in the contribution from the prior distribution in the second line.

3.4 Variational Continual Learning

Variational continual learning (VCL) (Nguyen et al., 2017) is another method that was used as a basis of this thesis. VCL is a general framework for continual learning that combines on-line variational inference and Monte Carlo variational inference for neural networks. It introduces a recursive update rule that reuses a posterior from the previous task as a prior for a new one. The main idea of the VCL is in the following recursive update rule:

$$p(\boldsymbol{\theta}|\mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\mathcal{D}_t|\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{1:T-1}) p(\mathcal{D}_T|\boldsymbol{\theta}), \tag{3.2}$$

where T is the number of tasks, $\boldsymbol{\theta}$ are model parameters and $\mathcal{D}_t$ is a dataset for the task t.

In addition VCL uses coresets - subsets of data points from previous tasks, during training. The method suggested in this work also can be combined with coresets, but since it will not change the method at all, we do not provide any results for it.

3.5 Related Work

Except the work on Variational Continual learning (Nguyen et al., 2017) that is the basis of the suggested method, there are other works that have explored Bayesian on-line learning framework.

Authors of the **Bayesian Gradient Descent (BGD)** (Zeno et al., 2018) similarly to this work argue that knowing uncertainty of weights of a neural network should help to alleviate catastrophic forgetting and provide a general algorithm for continual learning. The authors derive BGD by minimizing KL-divergence for the (n+1) task and reusing q_n as a prior similarly to VCL. They have an SGD-like update while the suggested update is similar to Adam/RMSProp. The authors of BGD additionally perform pruning using signal to noise ratio (SNR) $= \frac{|\mu|}{\sigma}$. BGD uses a different derivation from the suggested method and approximates product of the Hessian and parameters variance σ with $\boldsymbol{H}(\theta)\sigma_i = \frac{\partial^2 \mathcal{L}(\mu)}{\partial \theta_i^2}\sigma_i \approx \mathbb{E}_\epsilon[\frac{\partial \mathcal{L}(\theta)}{\partial \theta_i}\epsilon_i]$, while the suggested method approximates Hessian with $\boldsymbol{H}(\theta) \approx \mathbb{E}\Big[\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]\mathbb{E}\Big[\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]^T$ or with $\boldsymbol{H}(\theta) \approx \mathbb{E}\Big[\Big(\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big)^2\Big]$.

Bayesian incremental learning (Kochurov et al., 2018) uses the same lower bound as VCL and investigates how different posterior approximations behave with it. This work suggests that better approximations to the posterior than mean-field approximation might be beneficial for continual learning.

The work on **on-line structured Laplace approximations** (Ritter et al., 2018) suggests that better approximation to the Hessian that would take into account interaction between the elements should be helpful for continual learning

and uses Kronecker factored approximation to the Hessian.

Two last works suggest that better approximations to the posterior and to the Hessian can be among the ways to improve continual learning in neural networks for methods that use Bayesian inference framework for it. The generality of the method suggested in this thesis should make such experiments especially simple and thus they are considered as the future work.

Chapter 4

Method Derivation and Implementation Details

In this work we propose a general method to perform continual learning. It is an algorithm that can be used in place of the popular optimization algorithms, such as RMSProp (Tieleman and Hinton, 2012) or Adam (Kingma and Ba, 2014). The method is based on the combination of Variational Continual Learning (Nguyen et al., 2017) and Vadam (Khan et al., 2018). In case of a neural network the method uses a Bayesian neural network, but it does not requires a user to define it. The suggest method requires negative log-likelihood as the loss function, and that is the loss function that is usually used for training neural networks.

It is possible to derive two methods for continual learning, one of them based on Vprop and another one based on Vadam. We call these methods VpropVCL and VadamVCL. Further we derive both of them, since VpropVCL is a simpler algorithm and the derivation of the VadamVCL is based on it. It also makes it easier to follow the derivation. The experiments were done using VadamVCL only, since according to the Vadam paper, Vadam performs better than Vprop. The derivation of VPropVCL is given in the section 4.2 and of VadamVCL in the section 4.3.

4.1 Notation

The following notation is used throughout the derivation:

T – total number of tasks

t – task index, $t = \overline{1, T}$

$\mathcal{D}$ – dataset, $\mathcal{D} = \{y_n, x_n\}_{n=\overline{1,N}}$

N – size of the dataset, $N = |\mathcal{D}|$

K – total number of iterations of an optimization algorithm

k – iteration of an optimization algorithm, $k = \overline{1, K}$

$\boldsymbol{\theta}$ – model parameters (network weights in case of a neural network)

$\boldsymbol{\mu}$ – mean of the Gaussian distribution

$\boldsymbol{\Sigma}$ – covariance of the Gaussian distribution

$\boldsymbol{\sigma}^2$ – diagonal of the covariance matrix of the Gaussian distribution
when its covariance matrix is diagonal and given by $\boldsymbol{\sigma}^2 \boldsymbol{I}$

λ – scaling parameter of the covariance
of a prior distribution $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|0, \frac{1}{\lambda}\boldsymbol{I})$

$*$ – parameters of the previous task posterior or of the prior
in case current task is the first one

$\hat{\boldsymbol{g}}$ – estimated gradient

$\boldsymbol{H}$ – Hessian matrix

$\boldsymbol{\mathcal{L}}$ – loss

β_1 – algorithms parameter corresponding to the decay rate of the first moment,
usually set to 0.9

β_2 – algorithms parameter corresponding to the decay rate of the second moment,
usually set to 0.999

η – learning rate

Assume that there are T tasks in total arriving one by one. In general total number of tasks is not known in advance in continual learning, but here it is used to make the notation more convenient. The task $t, t = \overline{1,T}$ is described by its dataset $\mathcal{D}_t$. True posterior distribution for the task t is given by $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$. Using variational inference, true posterior distribution is approximated with $q(\boldsymbol{\theta})$. The method uses a Gaussian distribution $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and derivation uses a full Gaussian distribution for convenience. In the end of the derivation it will be approximated using mean-field variational inference (see section 3.2 for information on mean-field variational inference) with $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \approx \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$. Negative log-likelihood for the task t that is used in the loss is $f^{(t)}(\boldsymbol{\theta}) = -\log\, p(\mathcal{D}_t|\boldsymbol{\theta}) = -\sum_{n=1}^{N_t} \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta})}\Big[\log\, p(\boldsymbol{y}_n^{(t)}|\boldsymbol{\theta}, \boldsymbol{x}_n^{(t)})\Big]$. A prior distribution over the model parameters is given by $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|0, \frac{1}{\lambda}\boldsymbol{I})$.

4.2 VpropVCL

In general in order to do variational inference on a separate dataset $\mathcal{D}_t$ not in a continual learning setting, we would maximize a variational lower bound:

$$\text{ELBO}^{(t)}(q(\boldsymbol{\theta})) = \mathbb{E}_q[\log\, p(\mathcal{D}_t|\boldsymbol{\theta})] - \mathbb{D}_{\text{KL}}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta})]. \tag{4.1}$$

Variational Continual Learning (VCL) (Nguyen et al., 2017) was described in more detail in the section 3.4. It is based on the following recursion:

$$p(\boldsymbol{\theta}|\mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\mathcal{D}_t|\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{1:T-1}) p(\mathcal{D}_T|\boldsymbol{\theta}). \tag{4.2}$$

According to this recursive formula VCL suggests to maximize a lower bound similar to 4.1 with the only difference coming from the prior $p(\boldsymbol{\theta})$ being replaced with a posterior for the previous task $q_*(\boldsymbol{\theta})$:

$$\text{ELBO}_{\text{VCL}}^{(t)}(q(\boldsymbol{\theta})) = \mathbb{E}_q[\log\, p(\mathcal{D}_t|\boldsymbol{\theta})] - \mathbb{D}_{\text{KL}}(q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})), \tag{4.3}$$

the differences are highlighted in blue.

In both Vprop and Vadam the loss is defined as the variational lower bound, i.e. $\mathcal{L}^{(t)} = \text{ELBO}^{(t)}(q(\boldsymbol{\theta}))$. If VCL variational lower bound 4.3 is used in the Vprop derivation then the only difference in the update formulas will come from the changes in the loss, and more specifically from replacing a prior with the previous task posterior. Vprop update in the matrix form before the gradients are calculated (appendix C, equations 40 and 41, (Khan et al., 2018)) is:

$$\boldsymbol{\Sigma}_{k+1}^{-1} = \boldsymbol{\Sigma}_{k}^{-1} - 2\beta_k[\nabla_{\boldsymbol{\Sigma}}\mathcal{L}_k], \tag{4.4}$$

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \beta_k\boldsymbol{\Sigma}_{k+1}[\nabla_{\boldsymbol{\mu}}\mathcal{L}_k], \tag{4.5}$$

where β_k are intermediate constant method parameters.

Since the loss has changed, it is necessary to recompute the gradients. All final differences from the original Vprop derivation in the final formulas will be further highlighted in blue. The gradients of the loss $\mathcal{L}^{(t)} = \text{ELBO}^{(t)}_{\text{VCL}}(q(\boldsymbol{\theta}))$ are:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}}\mathcal{L}^{(t)} &= \nabla_{\mu}\Big[\mathbb{E}_q[-f^{(t)}(\boldsymbol{\theta})] - \mathbb{D}_{\text{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})]\Big] = \\
&= -\nabla_{\mu}\Big[\mathbb{E}_q[f^{(t)}(\boldsymbol{\theta})] + \mathbb{D}_{\text{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})]\Big] = \Bigg\{ \\
&1)\ \nabla_{\mu}\mathbb{E}_q[f^{(t)}(\boldsymbol{\theta})] = [\text{Bonnet and Price theorems, (Rezende et al., 2014)}] = \\
&= \mathbb{E}_q[\nabla_{\boldsymbol{\theta}} f^{(t)}(\boldsymbol{\theta})], \\
&2)\ \mathbb{D}_{\text{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})] = [(\text{Duchi, 2007}), \boldsymbol{\theta} \in \mathbb{R}^l] = \\
&= \frac{1}{2}\Big\{\text{tr}(\boldsymbol{\Sigma}_*^{-1}\boldsymbol{\Sigma}) + (\boldsymbol{\mu}_* - \boldsymbol{\mu})^{\text{T}}\boldsymbol{\Sigma}_*^{-1}(\boldsymbol{\mu}_* - \boldsymbol{\mu}) - l + \log\frac{|\boldsymbol{\Sigma}_*|}{|\boldsymbol{\Sigma}|}\Big\} \Rightarrow \\
&\Rightarrow \nabla_{\mu}\mathbb{D}_{\text{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})] = \boldsymbol{\Sigma}_*^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_*)\Bigg\} = \\
&= -\mathbb{E}_q[\nabla_{\boldsymbol{\theta}} f^{(t)}(\boldsymbol{\theta})] - \boldsymbol{\Sigma}_*^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_*),
\end{aligned} \tag{4.6}$$

$$\nabla_{\boldsymbol{\Sigma}}\mathcal{L}^{(t)} = -\nabla_{\boldsymbol{\Sigma}}\Big[\mathbb{E}_q[f^{(t)}(\boldsymbol{\theta})] + \mathbb{D}_{\mathrm{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})]\Big] = \Bigg\{$$

$$1)\ \nabla_{\boldsymbol{\Sigma}}\mathbb{E}_q[f(\boldsymbol{\theta})] = [\text{Bonnet and Price theorems, (Rezende et al., 2014)}] =$$

$$= \frac{1}{2}\mathbb{E}_q[\nabla^2_{\boldsymbol{\theta\theta}} f^{(t)}(\boldsymbol{\theta})],$$

$$2)\ \nabla_{\boldsymbol{\Sigma}}\mathbb{D}_{\mathrm{KL}}[q(\boldsymbol{\theta})||q_*(\boldsymbol{\theta})] = \frac{1}{2}\boldsymbol{\Sigma}_*^{-1} - \frac{1}{2}\boldsymbol{\Sigma}^{-1}\Bigg\} =$$

$$= -\frac{1}{2}\Big[\mathbb{E}_q[\nabla^2_{\boldsymbol{\theta\theta}} f^{(t)}(\boldsymbol{\theta})] + \boldsymbol{\Sigma}_*^{-1} - \boldsymbol{\Sigma}^{-1}\Big]. \tag{4.7}$$

Resulting update in the matrix form is:

$$\boldsymbol{\Sigma}_{k+1}^{-1} = (1-\beta_k)\boldsymbol{\Sigma}_k^{-1} + \beta_k\Big[\nabla^2_{\theta\theta} f(\boldsymbol{\theta}_k) + \boldsymbol{\Sigma}_*^{-1}\Big]. \tag{4.8}$$

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \beta_k\boldsymbol{\Sigma}_{k+1}[\nabla_\theta f(\boldsymbol{\theta}_k) + \boldsymbol{\Sigma}_*^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)]. \tag{4.9}$$

Further $\boldsymbol{s}_k := \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_*^{-1}$. Both Vprop and Vadam use mean-field approximation, therefore $\boldsymbol{\Sigma}_k^{-1} = \boldsymbol{\sigma}_k^2\boldsymbol{I}, \boldsymbol{\Sigma}_*^{-1} = \boldsymbol{\sigma}_*^2\boldsymbol{I} \Rightarrow \boldsymbol{s}_k := \boldsymbol{\sigma}_k^{-2}\boldsymbol{I} - \boldsymbol{\sigma}_*^{-2}\boldsymbol{I}$. Further Hessian matrix $\nabla^2_{\theta\theta} f(\boldsymbol{\theta}_k)$ is denoted as $\boldsymbol{H}(\boldsymbol{\theta}_k)$, and different step sizes α_k and β_k are used for $\boldsymbol{\mu}$ and $\boldsymbol{s}$ respectively. The point-wise update is:

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\alpha_k}{\boldsymbol{s}_{k+1} + \boldsymbol{\sigma}_*^{-2}}\Big[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k; \mathcal{D}_t) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\boldsymbol{\sigma}_*^{-2}\Big], \tag{4.10}$$

$$\boldsymbol{s}_{k+1} = (1-\beta_k)\boldsymbol{s}_k + \beta_k N[\mathrm{diag}(\boldsymbol{H}(\boldsymbol{\theta}_k; \mathcal{D}_t))]. \tag{4.11}$$

Exact Hessian computation is slow and has to be approximated. This issue is discusses in more detail in the section 4.3.2. The logic described there applies both to VpropVCL and to VadamVCL. Additionally, gradient scaling by N and N^2 is explained in the section 4.3.1. Using a gradient magnitude approximation that makes the method even more similar to RMSprop (Tieleman and Hinton, 2012) and making changes to the notation of the constant parameters for convenience, final point-wise VpropVCL algorithm for one task is shown in the table 4.1, where it is compared to RMSprop.

VpropVCL
$\boldsymbol{\theta}_k = \boldsymbol{\mu}_k + \frac{\epsilon}{\sqrt{\boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}}}, \epsilon \sim \mathcal{N}(\epsilon \vert 0, 1)$
$\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1-\beta_1)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\sigma_*^{-2}]$
$\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2) N^2 [\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]^2$
$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\eta}{\sqrt{\boldsymbol{s}_{k+1}} + \sigma_*^{-2}} \boldsymbol{m}_{k+1}$

Table 4.1: VpropVCL algorithm for training on one task in a continual learning setting. All updates are point-wise. The differences from RMSprop (Tieleman and Hinton, 2012) are highlighted in blue. Main differences are in the weight-perturbation in the first line and in the contribution from the previous task posterior in the second line.

4.3 VadamVCL

VadamVCL is derived in the same way as VpropVCL, but using Vadam as a basis for the derivation. Difference between Vprop and Vadam is in the momentum term added to the mirror descent update in the case of Vadam. While Vprop mirror descent update is

$$\boldsymbol{m}_{k+1} = \underset{\boldsymbol{m}}{\operatorname{argmin}} \langle \boldsymbol{m}, -\nabla_m \mathcal{L}_k \rangle + \frac{1}{\beta_k} \mathbb{D}_{\mathrm{KL}}[q||q_k], \tag{4.12}$$

Vadam adds momentum term to this update (appendix E.2. in (Khan et al., 2018)):

$$\boldsymbol{m}_{k+1} = \underset{\boldsymbol{m}}{\operatorname{argmin}} \langle \boldsymbol{m}, -\nabla_m \mathcal{L}_k \rangle + \frac{1}{\beta_k} \mathbb{D}_{\mathrm{KL}}[q||q_k] - \frac{\alpha_k}{\beta_k} \mathbb{D}_{\mathrm{KL}}[q||q_{k-1}]. \tag{4.13}$$

There is no difference between Vprop and Vadam in terms of presence of a prior distribution. In both updates it appears only in the loss function. Therefore, the only change in Vadam derivation that has to be done to derive VadamVCL, is a replacement of the formulas for gradient in the Vadam update, so that they

would depend on the previous task posterior distribution.

Using results from the equations 4.6, 4.7, the resulting update in the matrix form with $\boldsymbol{S}_k = \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_*^{-1}$ is:

$$\boldsymbol{S}_{k+1} = \Big(1 - \frac{\beta_k}{1-\alpha_k}\Big)\boldsymbol{S}_k + \frac{\beta_k}{1-\alpha_k}\nabla^2_{\boldsymbol{\theta\theta}} f(\boldsymbol{\theta}_k), \tag{4.14}$$

$$\begin{aligned}\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\beta_k}{1-\alpha_k}\Big(\boldsymbol{S}_{k+1} + \boldsymbol{\Sigma}_*^{-1}\Big)^{-1}\Big[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k) + \boldsymbol{\Sigma}_*^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\Big] + \\ + \frac{\alpha_k}{1-\alpha_k}\Big(\boldsymbol{S}_{k+1} + \boldsymbol{\Sigma}_*^{-1}\Big)^{-1}\Big(\boldsymbol{S}_k + \boldsymbol{\Sigma}_*^{-1}\Big)\Big(\boldsymbol{\mu}_k - \boldsymbol{\mu}_{k-1}\Big)\end{aligned} \tag{4.15}$$

The point-wise update is:

$$\boldsymbol{s}_{k+1} = (1-\tilde{\alpha}_k)\boldsymbol{s}_k + \tilde{\alpha}_k[N\boldsymbol{H}(\boldsymbol{\theta}_k)], \tag{4.16}$$

$$\begin{aligned}\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\tilde{\alpha}_k}{\boldsymbol{s}_{k+1} + \boldsymbol{\sigma}_*^{-2}}\Big[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\boldsymbol{\sigma}_*^{-2}\Big] + \\ + \tilde{\gamma}_k\frac{\boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}}{\boldsymbol{s}_{k+1} + \boldsymbol{\sigma}_*^{-2}}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_{k-1})\end{aligned} \tag{4.17}$$

Here $\boldsymbol{\theta_k} \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2), \boldsymbol{\sigma}_k^{-2} = \boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}$. $\boldsymbol{\theta}_k$ is sampled using the re-parametrization trick. The point-wise algorithm for one task without approximation to the Hessian is compared to Adam in the table 4.2.

Prior over the weights is given by $\mathcal{N}(0, \frac{1}{\lambda})$. Parameters β_1, β_2, η are usually set in the same way as in Adam, i.e $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 0.001$. λ was usually set to 1. Initially $\boldsymbol{m}_k$ was set to 0 and $\boldsymbol{s}_k$ to 1. Between different tasks $\boldsymbol{m}_k$ and $\boldsymbol{s}_k$ are reset to their initial values.

Adam	**VadamVCL**
$\boldsymbol{\theta}_k = \boldsymbol{\mu}_k$	$\boldsymbol{\theta}_k = \boldsymbol{\mu}_k + \frac{\epsilon}{\sqrt{\boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}}}, \epsilon \sim \mathcal{N}(\epsilon \vert 0, 1)$
$\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1-\beta_1)[\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]$	$\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + + (1-\beta_1)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\sigma_*^{-2}]$
$\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)[\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]^2$	$\boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2) N \mathrm{diag}[\boldsymbol{H}(\boldsymbol{\theta}_k)]$
$\hat{\boldsymbol{m}}_{k+1} = \frac{\boldsymbol{m}_k}{1-\beta_1^{k+1}}$	$\hat{\boldsymbol{m}}_{k+1} = \frac{\boldsymbol{m}_k}{1-\beta_1^{k+1}}$
$\hat{\boldsymbol{s}}_{k+1} = \frac{\boldsymbol{s}_k}{1-\beta_2^{k+1}}$	$\hat{\boldsymbol{s}}_{k+1} = \frac{\boldsymbol{s}_k}{1-\beta_2^{k+1}}$
$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\eta}{\sqrt{\hat{\boldsymbol{s}}_{k+1}} + \delta} \hat{\boldsymbol{m}}_{k+1}$	$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \frac{\eta}{\hat{\boldsymbol{s}}_{k+1} + \boldsymbol{\sigma}_*^{-2}} \hat{\boldsymbol{m}}_{k+1}$

Table 4.2: VadamVCL algorithm for one task versus Adam algorithm. The differences are highlighted in blue. Main differences are in the weight-perturbation (line 1), contribution from the previous task posterior (line 3) and approximation to the Hessian (line 4).

4.3.1 A Note about Gradients Scaling

Vadam uses $f(\boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{i=1}^{N} p(x_i|\theta) = \sum_{i=1}^{N} f_i(\boldsymbol{\theta}), N = |\mathcal{D}|$ as a loss function and thus the stochastic gradient estimation it needs is $\sum_{i=1}^{N} \nabla_\theta f_i(\boldsymbol{\theta})$. When mini-batch gradient descent is used, machine learning libraries, such as TensorFlow or PyTorch, usually return the mean of the gradient over all points in a mini-batch $\hat{\boldsymbol{g}}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta f_i(\theta)$, where M is a mini-batch size. We can notice that $\sum_{i=1}^{N} \nabla_\theta f_i(\boldsymbol{\theta}) \approx N\mathbb{E}[\frac{1}{M} \sum_{i=1}^{M} \nabla_\theta f_i(\boldsymbol{\theta})] = \frac{N}{M}\mathbb{E}[\sum_{i=1}^{M} \nabla_\theta f_i(\boldsymbol{\theta})] = N\hat{\boldsymbol{g}}(\boldsymbol{\theta})$, with expectation taken over all possible mini-batches. Therefore, to get a correct gradient estimation we can either use $-\frac{N}{M} \sum_{i=1}^{M} p(x_i|\boldsymbol{\theta})$ as a loss function instead of a usual $-\frac{1}{M} \sum_{i=1}^{M} p(x_i|\boldsymbol{\theta})$ or multiply the gradient by N in the optimization algorithm. We used the second option.

4.3.2 Approximation to the Hessian

Equations 4.11 and 4.16 require to compute diagonal of the Hessian matrix. However, as it was already mentioned in the previous sections, computing Hessian is a slow procedure. It is unreasonable to use it for models with many parameters such as neural networks, because its size is a square of the number of model parameters. Thus, it requires an approximation. Two types of approximations were used based on (Khan et al., 2018): gradient magnitude (GM) and Generalized Gauss-Newton (GGN).

Gradient magnitude (GM)

Gradient magnitude approximation is an unconventional name that is used here to denote an approximation to the Hessian of the form:

$$\boldsymbol{H}(\mathcal{L}(\boldsymbol{\theta})) \approx \mathbb{E}\Big[\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]\mathbb{E}\Big[\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big]^T \approx \hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^T. \tag{4.18}$$

This approximation makes methods even more similar to RMSprop and Adam.

Generalized Gauss-Newton (GGN)

Generalized Gauss-Newton (GGN) approximation (Bottou et al., 2016) is an approximation of the form:

$$\boldsymbol{H}(\mathcal{L}(\boldsymbol{\theta})) \approx \mathbb{E}\Big[\Big(\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big)^2\Big]. \tag{4.19}$$

This approximation allows to establish a connection to the Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) method for continual learning.

EWC minimizes the following lower bound:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_B(\boldsymbol{\theta}) + \sum_i \frac{\lambda}{2}\mathcal{F}_i(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{A,i})^2, \tag{4.20}$$

where A, B - indexes of the tasks, A is the first one and B is the second one, i -

index of a model parameter, F - diagonal of the Fisher information matrix, λ - a constant parameter of the method.

If Generalized Gauss-Newton (GGN) approximation to the Hessian is used, then we are in fact approximating Hessian matrix with a Fisher information matrix because

$$\boldsymbol{H}(\mathcal{L}(\boldsymbol{\theta})) \approx \mathbb{E}\Big[\Big(\frac{\partial \mathcal{L}(x_i, y_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big)^2\Big] = \mathbb{E}\Big[\Big(\frac{\partial\{-\log\, p(y_i|x_i, \boldsymbol{\theta})\}}{\partial \boldsymbol{\theta}}\Big)^2\Big] = \mathcal{F}(\boldsymbol{\theta}). \quad (4.21)$$

Taking into account that $\boldsymbol{\sigma}_k^{-2} = \boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}$ and that $\boldsymbol{\mu}_k - \boldsymbol{\mu}_*$ is multiplied by $\boldsymbol{\sigma}_*^{-2}$, one can see the connection to the EWC, because EWC adds difference between current and previous parameters values multiplied by Fisher Information matrix to the loss.

While according to the EWC paper the method required a lot of experimentation to finally decide on the way to modify loss, here it occurs naturally.

Gradient magnitude approximation is faster to compute, but it is a worse approximation than GGN. Since GGN is a better approximation the method should converge in less epochs that might compensate the time algorithm needs to compute it.

Calculating a better approximation once

A better approximation to the Hessian can have two main effects on the suggested method: 1. The method will converge in less epochs for each task; 2. The method will use a better approximation to the covariance matrix of the posterior distribution. While convergence speed for each task is important, better approximation to the posterior distribution covariance should be especially valuable in the case of continual learning because it affects the uncertainty on which depends parameters change rate. While it is expensive to compute a better approximation for each training episode, computing it once in the end of the training for the given task can be done in reasonable time. Thus, effects of a computation of a better approximation once were also explored for some of

the experiments. Results for the Hessian computed once for the mean-fitting experiment are provided in the section 5.1.

4.4 Final Algorithm

Algorithm in the table 4.3 is the full suggested VadamVCL algorithm for continual learning that also takes into account parameter initialization before each new task. The only change required compared to the training in one-task setting is the re-initialization of the parameters. For a new task parameters for moments are set to their default values and parameters that were parameters of a prior distribution in case of Vadam are re-initialized using previous task posterior for all tasks that come after the first one. This can be implemented as a function that is called before the training starts instead of a usual call to initialization and does not put any additional burden on a VadamVCL user.

Final algorithm is presented with the gradient magnitude approximation. This approximation was used for the main continual learning experiments because of its speed. As in the Vadam paper in the case of GM approximation a square root is used in the scaling parameter (line 13), see Appendix E in Vadam paper for the proof. To compensate for this square root Hessian approximation was scaled using N^2 instead of N. Additionally, there is a γ parameter that is used to scale squared gradients. As the experiments have shown Vadam slightly shrinks variance that worsens continual learning results in the long run when we have many tasks, and this parameter was added to compensate for this behavior. Parameter γ was set to 1 in all experiments, unless explicitly said otherwise.

To sum up, there is a set of datasets $\mathcal{D}_t, t = \overline{1, T}$ for T tasks that are arriving sequentially. At one step of an algorithm we have access only to the current dataset and total number of the tasks T is not known in advance. T here is used only to make the notation more convenient.

Table 4.3: Final VadamVCL algorithm for continual learning. All updates are point-wise. All parameters are the parameters related to the current task unless the notation indicates otherwise. Differences from Adam are highlighted in blue.

1 **for** dataset $\mathcal{D}_t$, $N = |\mathcal{D}_t|$ for the task $t, t = \overline{1,T}$ **do**
2 $\quad \boldsymbol{m}_0 = 0, \boldsymbol{s}_0 = 0$
3 $\quad$ **if** $t == 1$ **then**
4 $\quad\quad \boldsymbol{\mu}_* = 0, \boldsymbol{\sigma}_*^{-2} = \lambda$
5 $\quad$ **else**
6 $\quad\quad \boldsymbol{\mu}_* = \boldsymbol{\mu}_{\text{last}}^{(t-1)}, \boldsymbol{\sigma}_*^{-2} = \boldsymbol{s}_{\text{last}}^{(t-1)} + (\boldsymbol{\sigma}_*^{(t-1)})^{-2}$
7 $\quad$ **for** $k = \overline{1,K}$, K - number of epochs **do**
8 $\quad\quad \boldsymbol{\theta}_k = \boldsymbol{\mu}_k + \epsilon / \sqrt{\boldsymbol{s}_k + \boldsymbol{\sigma}_*^{-2}}, \epsilon \sim \mathcal{N}(\epsilon|0,1)$
9 $\quad\quad \boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1-\beta_1)[N\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_*)\boldsymbol{\sigma}_*^{-2}]$
10 $\quad\quad \boldsymbol{s}_{k+1} = \beta_2 \boldsymbol{s}_k + (1-\beta_2)\gamma N^2 [\hat{\boldsymbol{g}}(\boldsymbol{\theta}_k)]^2$
11 $\quad\quad \hat{\boldsymbol{m}}_{k+1} = \boldsymbol{m}_k / (1-\beta_1^{k+1})$
12 $\quad\quad \hat{\boldsymbol{s}}_{k+1} = \boldsymbol{s}_k / (1-\beta_2^{k+1})$
13 $\quad\quad \boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \eta\, \hat{\boldsymbol{m}}_{k+1} / (\sqrt{\hat{\boldsymbol{s}}_{k+1}} + \boldsymbol{\sigma}_*^{-2})$
14 $\quad$ Discard the dataset $\mathcal{D}_t$

Main differences from Adam are in the initialization of additional parameters (lines 4 and 6), weight-perturbation (line 8) and contribution from the previous task posterior (line 9).

Chapter 5

Experiments

In the previous section we have derived VadamVCL algorithm and shown that it has a simple form. In this section we will demonstrate that despite this simplicity VadamVCL produces results comparable to the state of the art for continual learning.

Among the benchmarks used to test continual learning methods are experiments constructed based on the MNIST dataset, see e.g. (Nguyen et al., 2017; Zeno et al., 2018; Lopez-Paz et al., 2017). Further we present results on two smaller experiments that provide intuition into how the method works and on the Permuted MNIST and Split MNIST experiments. One of the small experiments is an experiment about finding a mean of several Gaussian distributions. It is provided in the section 5.1. Another one is a 2D classification experiment in the section 5.2. Results for the Permuted MNIST experiment are presented in the section 5.3 and for Split MNIST in the section 5.4.

Before moving to the experiments it is necessary to introduce terms **single-head** and **multi-head network**. Usually when a neural network is used for classification its last layer has a fixed number of units that is equal to the number of possible classes. During continual learning for classification problems new classes may or may not appear. Based on this there are two main approaches to constructing neural networks. If new classes are expected then additional units

are added to the output layer and only this subset of the units from the last layer is used during training. Such architecture is called a **multi-head network**. We want to emphasize, that we know from which task inputs come from and use only the subset of the units in the output layer that corresponds to this task. Example of an experiment that requires a multi-head network is Split MNIST.

If it is assumed that new classes do not appear for any task then there is no need to add more units to the last layer and all of them are always used. In this case we refer to the network as a **single-head network**. Example of an experiment that uses it, is Permuted MNIST.

For the fair comparison all methods that are used in the experiments do not use additional memory to store any kind of data from previous tasks similarly to VadamVCL. VadamVCL can be augmented with some form of memory, e.g. with coresets that were used in the VCL paper (Nguyen et al., 2017). Basically, a *coreset* is a subset of a dataset that is saved after training on the task is finished and gets mixed into the datasets for the next tasks to help preserve performance on the previous tasks. However, suggested algorithm will not change in any way if memory is used, so experiments did not involve memory.

5.1 Mean Estimation of Several Gaussian Distributions

In this experiment we have T tasks. For each $t = \overline{1,T}$ task we have a data point $\boldsymbol{y}_t$ and a likelihood is given by a Gaussian distribution with an unknown mean $\boldsymbol{\theta}$ and a known variance $\boldsymbol{\Sigma}_t$, $\mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\theta}, \boldsymbol{\Sigma}_t)$. The mean parameter $\boldsymbol{\theta}$ for likelihoods is shared among all tasks and it is the parameter for which we are doing inference. Posteriors are approximated with a diagonal Gaussian $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{m}, \boldsymbol{\sigma}^2\boldsymbol{I})$. The prior for the first task is $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|0, e^{10}\boldsymbol{I})$. The method reuses previous task posterior as a prior for the next task. Therefore, the

model is

$$p(\boldsymbol{\theta}|\boldsymbol{y}_1, ..., \boldsymbol{y}_T) \propto p(\boldsymbol{y}_T|\boldsymbol{\theta})...p(\boldsymbol{y}_1|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{\theta}) \approx q(\boldsymbol{\theta}). \tag{5.1}$$

Experiments were done with $T = 3$ and $T = 5$. The order of tasks appearance is from left to right. Results are provided in the figures 5.1 and 5.2. The results provided use only gradient magnitude approximation or exact Hessian because performance of gradient magnitude and GGN approximation when only one data point is given for each task is the same in this experiment. It can be noticed from these figures that type of the approximation makes a big difference for this small convex problem, however, it is not necessarily so for non-convex problems.

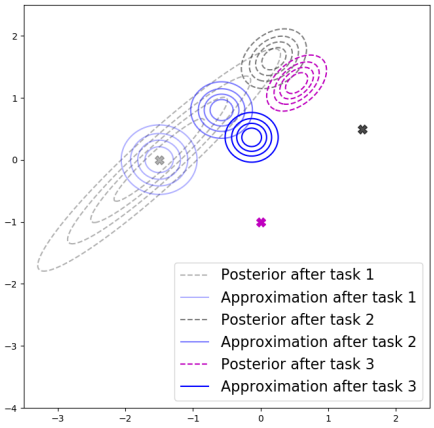
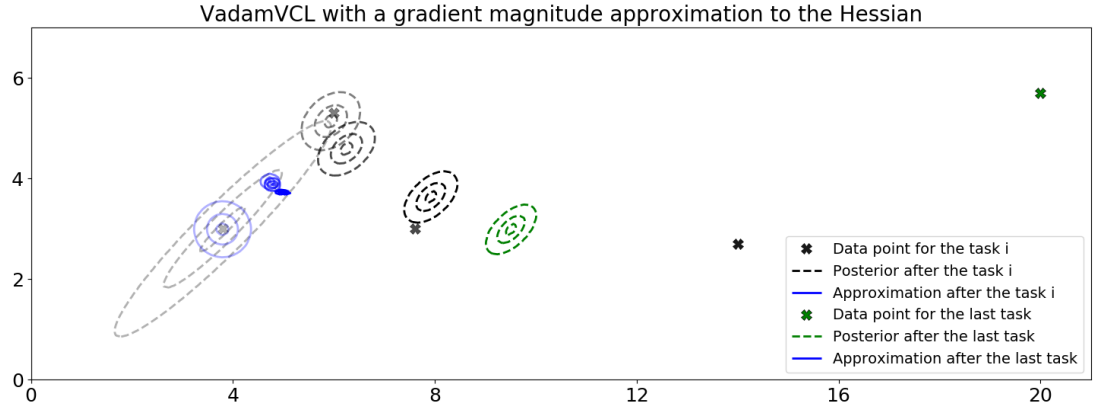
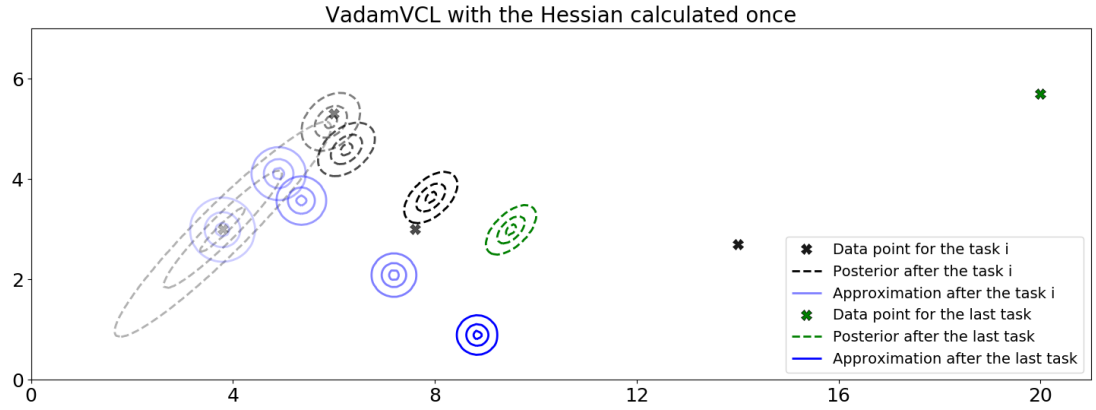


Figure 5.1: Results for the experiment on the mean estimation of 3 Gaussians. Older tasks are more transparent. These are results for VadamVCL with the diagonal of exact Hessian. They are are exactly the same as for VCL, so there is only one plot for both of them.



(a) VadamVCL with a gradient magnitude approximation to the Hessian (default version). The variance is shrinked and the approximation does not get close to the 4th and 5th task.



(b) VadamVCL results with a gradient magnitude approximation to the Hessian used during training as in the default version, but the diagonal of the exact Hessian is calculated in the end of the training on a task, to get posterior parameters. The results looked the same when Hessian was computed for each training step and when VCL was used.

Figure 5.2: Results for the experiment on the mean estimation of several Gaussians. Older tasks are more transparent. The variance is shrinked in the case of VadamVCL because of the gradient magnitude approximation to the Hessian, and it results in poor performance on later tasks. It is unclear from this experiment how this affects performance on non-convex problems, such as neural networks. Interestingly, only one computation of the Hessian, solves the problem in this case.

5.2 2D Classification

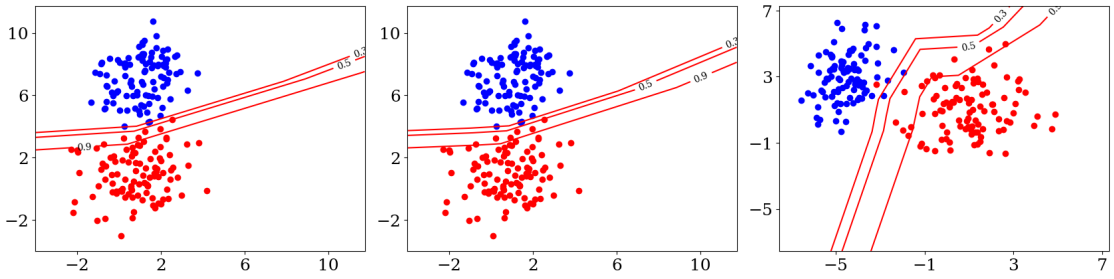
One more simple experiment that provides a way to get intuition into how different methods work is a classification of a 2D dataset with a small neural network. For this experiment we used a one-layer neural network with 20 units. The data for each task was generated using two Gaussian distributions. One of them never changes, while the second Gaussian is rotated around it on the $\frac{2\pi(t-1)}{T}$ radians, $t = \overline{1, T}$, T - total number of tasks. $T = 5$ was used for all experiments.

Multi-head network

For the experiment with a multi-head network data generated for all tasks was considered to be coming from different classes. For the first task the labels were 0 and 1, for the second one 2 and 3, etc. Results for this experiment are provided in the figure 5.3. Results from VCL and VadamVCL are similar in this case. This experiment was done using only the gradient magnitude approximation to the Hessian in VadamVCL because the results are already good and better approximation cannot noticeably improve them.

Single-head network

In order to use a single-head network all classes need to be known in advance. Thus, for all tasks it was assumed that data was coming from the same two classes. Results for VadamVCL with a gradient magnitude approximation to the Hessian are in the figure 5.4. Results with GGN approximation are in the figure 5.5. As you can see, the type of approximation does not change the result much in this experiment. This suggests that GM approximation might be a good choice for bigger experiments because of its speed and good empirical results.



(a) The 1st task after the network was trained on it.

(b) The 1st task after the network was trained on the 5th task.

(c) The 5th task after being trained on it.

Figure 5.3: VadamVCL error-bounds on the 2D classification experiment with a multi-head network. The error-bound on the 1st task changes only slightly after the 5th task, and the error-bound for the 5th task looks correct.

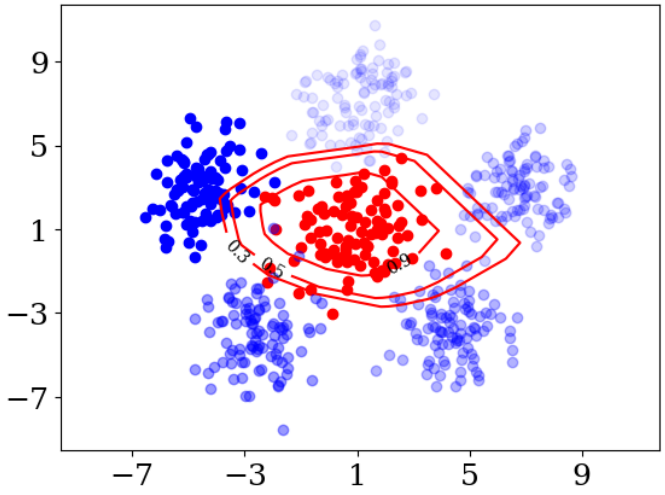


Figure 5.4: VadamVCL error-bound on the 2D classification experiment with a single-head network and a gradient magnitude approximation to the Hessian. Red class remains the same for all tasks, and the blue class changes. More transparent data points belong to the older tasks.

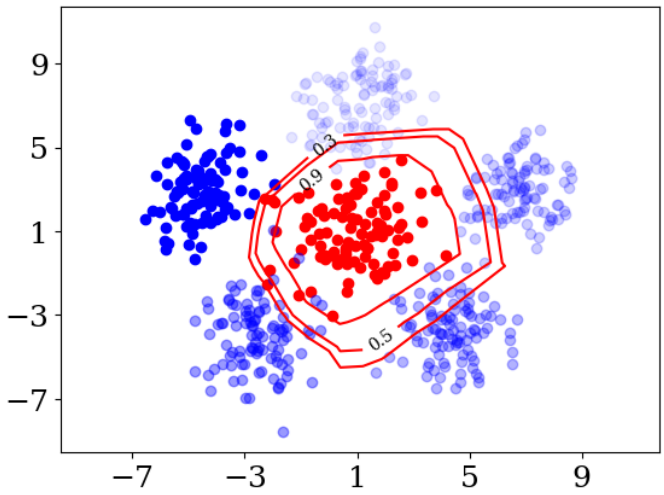


Figure 5.5: VadamVCL error-bound on the 2D classification experiment with a single-head network and a GGN approximation to the Hessian. Red class remains the same for all tasks, and the blue class changes. More transparent data points belong to the older tasks. Results are comparable to the results obtained using a gradient magnitude approximation to the Hessian in the figure 5.4.

5.3 Permuted MNIST

Permuted MNIST experiment is one of the continual learning benchmarks. The dataset received for each task consists of the images of MNIST digits (LeCun, 1998) which pixels have undergone a fixed random permutation. It is used in (Nguyen et al., 2017; Goodfellow et al., 2013; Kirkpatrick et al., 2017; Zenke et al., 2017). One task is generated by applying a fixed permutation to all images in the original MNIST dataset. Tasks generation process is shown in the figure 5.6.

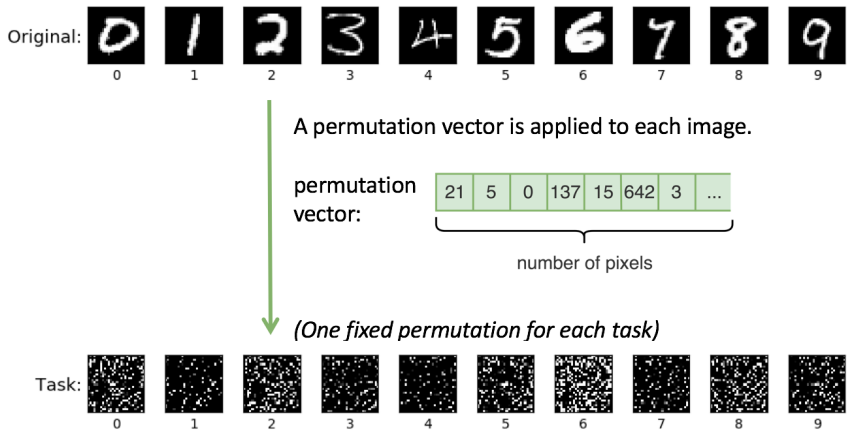


Figure 5.6: Tasks generation for the Permuted MNIST experiment. One permutation vector corresponds to one task, and this vector is applied to each image in the original MNIST dataset to generate images for a given task.

Permuted MNIST experiment was done using 10 tasks in total. All methods were tested using a single-head neural network with 2 layers, 100 units each.

This experiment was performed using a gradient magnitude approximation to the Hessian in VadamVCL because of its speed. Depending on the γ parameter of VadamVCL it is possible to either get better results in the long run with worse performance on the older tasks or get better performance on the previous tasks with worse performance on the new ones and worse performance overall in the long run. With $\gamma = 1$ VadamVCL results in the long run are worse than other

methods, but performance on the earlier tasks is preserved better. Results for $\gamma = 1$ are given in the figures 5.7 and 5.8. Comparable performance to other methods can be achieved with $\gamma = .01$. This results are presented in the figure 5.9.

VadamVCL was run for 500 epochs with $\beta_1 = 0.9, \beta_2 = 0.999, \lambda = 1$ batch size 256 and 10 MC-samples during training and evaluation. For $\gamma = 1$ a learning rate $\eta = 0.0005$ was used, and $\eta = .001$ was used for $\gamma = 0.01$.

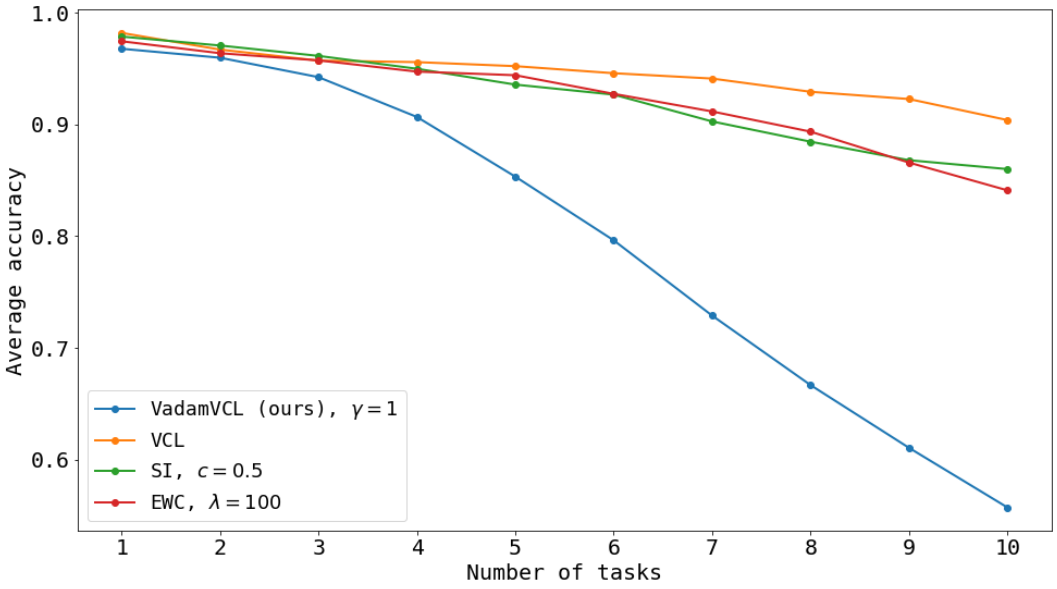


Figure 5.7: Average accuracy results on the Permuted MNIST experiment where VadamVCL uses $\gamma = 1$. Consider this figure together with the figure 5.8.

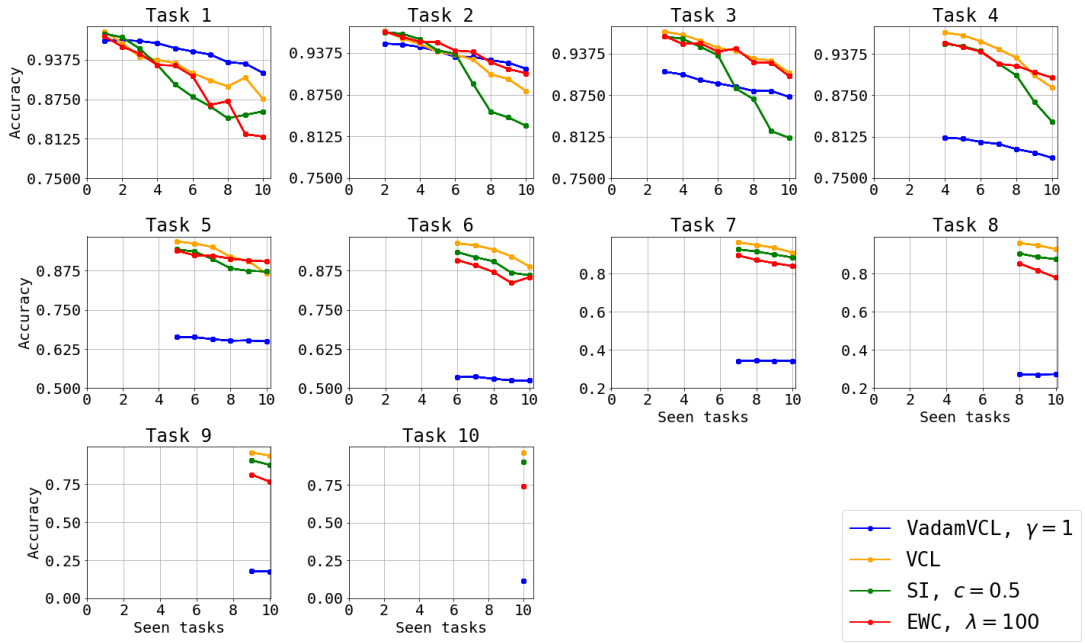


Figure 5.8: Results for individual tasks on the Permuted MNIST experiment. With $\gamma = 1$ VadamVCL average accuracy is worse in the long run that other methods average accuracy. However, the performance on the first tasks remains better and in general there is less forgetting. Even though achieved accuracy on the later tasks is lower, it drops less during further training.

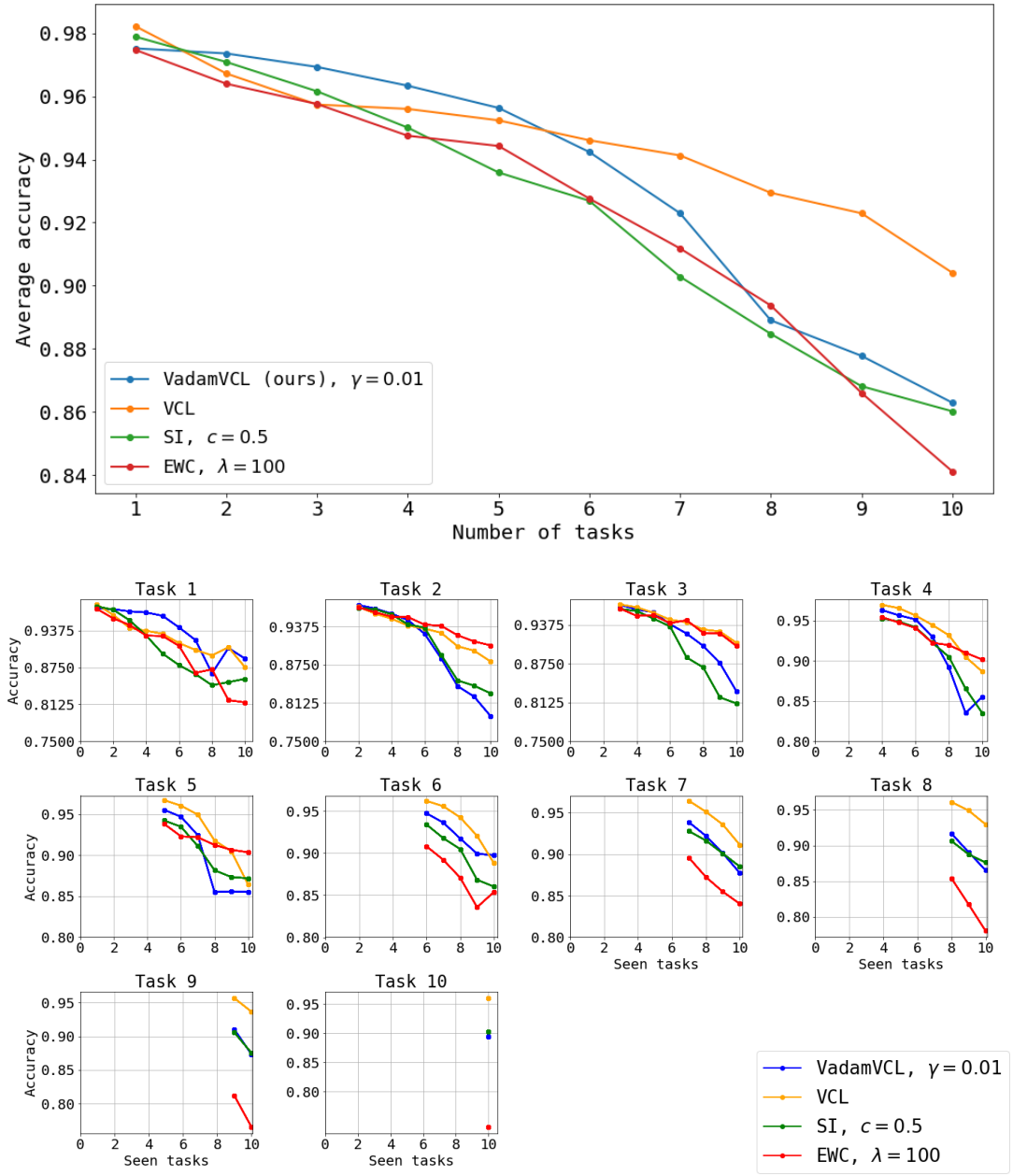


Figure 5.9: Results on the Permuted MNIST experiment. Top plot shows average accuracy and the bottom one accuracy on individual tasks. With $\gamma = 0.01$ VadamVCL average accuracy is comparable to SI and EWC and slightly worse in the long run than VCL. Performance on the individual tasks is also comparable to other methods.

5.4 Split MNIST

Split MNIST is another widely used continual learning benchmark. The experiments are designed by splitting MNIST dataset into 5 pairs: 0/1, 2/3, 4/5, 6/7, 8/9. Each of this subsets is treated as a separate task and is used in a usual continual learning setting. This dataset is used in e.g. (Nguyen et al., 2017; Goodfellow et al., 2013). Tasks generation process is illustrated in the figure 5.10.

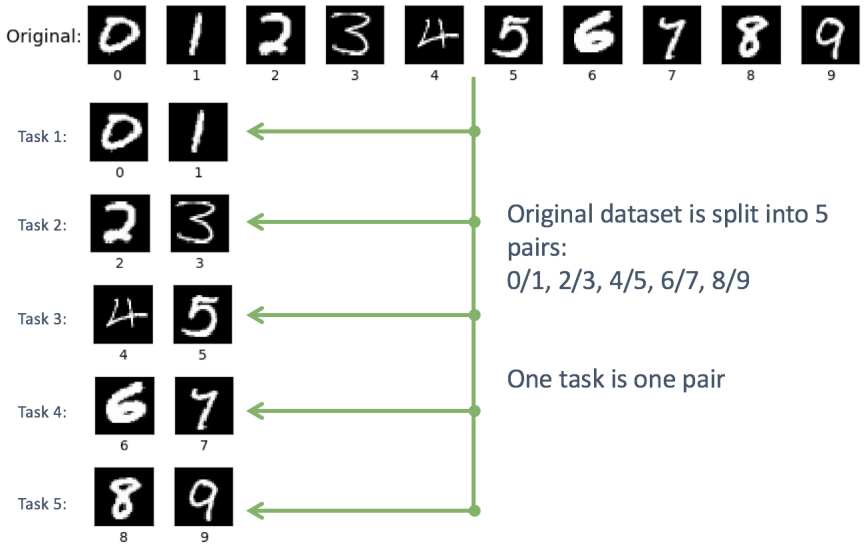


Figure 5.10: Tasks generation for the Split MNIST experiment. MNIST dataset is split into 5 pairs: 0/1, 2/3, 4/5, 6/7, 8/9. Each of this subsets is treated as a separate task.

Results for the Split MNIST experiment are given in the figure 5.11.

For this experiment for all methods was used a multi-head neural network with 2 hidden layers with 256 units each. VadamVCL parameters were set to $\beta_1 = 0.9, \beta_2 = 0.999, \lambda = 1, \gamma = 1, \eta = 0.0006$. Training on each task took 500 epochs.

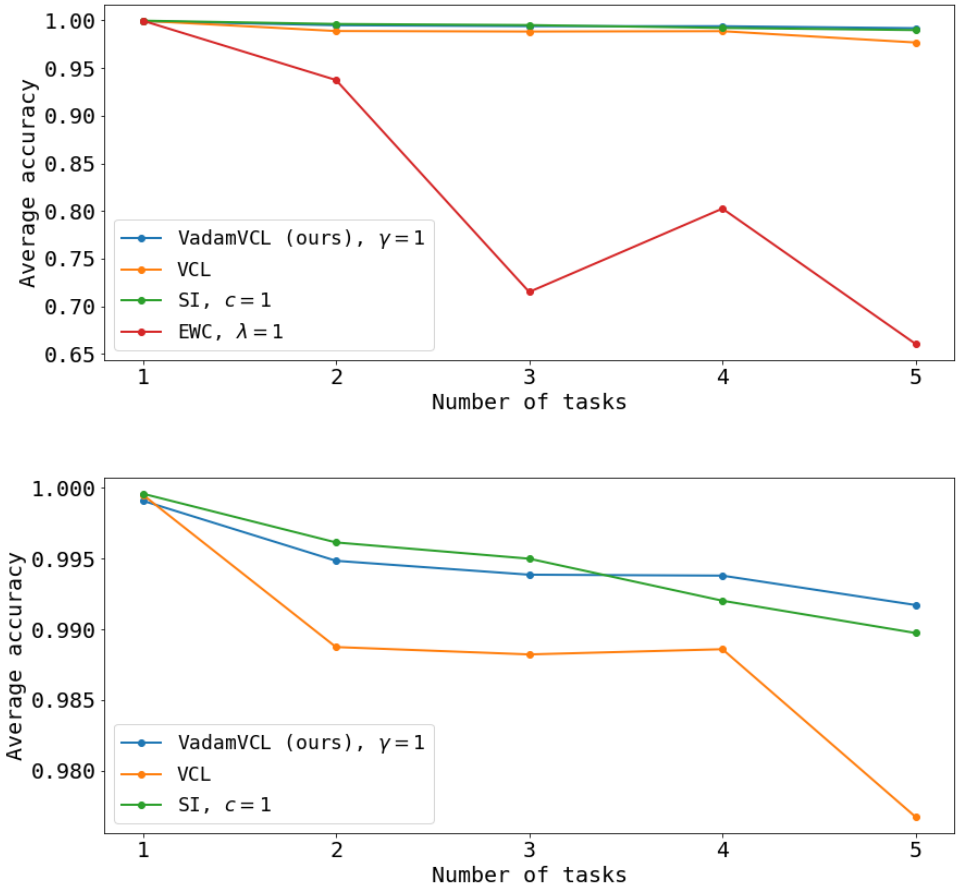


Figure 5.11: Average accuracy on the Split MNIST experiment. In the bottom figure EWC results are omitted to show more detailed results. VadamVCL performs in this experiment better than EWC and VCL and comparable to SI. The reason for such good performance is most probably the fact that VadamVCL slightly shrinks variance of the posterior distribution combined with the usage of a multi-head network. If a multi-head network is used, then for each new task there are new parameters on which performance on the previous tasks does not depend. VadamVCL penalizes changes to previous parameters stronger than other methods because of the shrinked variance, but new parameters allow to get good performance on new tasks, and this results in better performance overall. Since average accuracy is close to 100%, accuracy on individual tasks is not provided.

Chapter 6

Discussion and Conclusion

In this thesis we have introduced an optimization algorithm for continual learning called VadamVCL that is easy to implement and use. VadamVCL makes use of variational inference, and its implementation is similar to Adam algorithm. VadamVCL can be applied to different models and to neural networks in particular, and it does not require a user to define a Bayesian neural network or to modify a loss function. After the algorithm is implemented, it allows to make a model work in a continual learning setting simply by replacing an optimization algorithm used. Results on the small experiments, and Permuted and Split MNIST experiments show that VadamVCL has performance comparable to other continual learning methods. The algorithm also exhibits an interesting connection to Elastic Weight Consolidation method for continual learning.

To sum up, as the result of this study we introduced a new method for continual learning called VadamVCL that

- is easy to implement and use,
- does not use additional memory,
- is applicable to a neural network,
- performs comparably or better than other continual learning methods,

- has connections to other continual learning methods.

Since VadamVCL has a simple expression for its updates it allows for various improvements to the method to be done easily. As the future work it is possible to try better approximations to the Hessian. In the smaller experiments we have compared gradient magnitude approximation to the Hessian, generalized Gauss-Newton approximation to the Hessian and exact Hessian where it was possible. Better approximations improved continual learning results suggesting that this direction is worth exploring more. Another possible direction for the future research is to try a better approximation to the posterior distribution instead of using mean-field variational inference.

Additionally, it would be interesting to see method results on different types of models, especially on bigger networks and other types of networks except the networks that use only fully-connected layers, e.g. on convolutional neural networks. The method also requires experiments with bigger datasets, e.g. Split and Permuted CIFAR and with increased number of tasks.

Bibliography

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*.

Caruana, R. (1998). Multitask learning. In *Learning to learn*, pages 95–133. Springer.

Chen, Z. and Liu, B. (2016). Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145.

Duchi, J. (2007). Derivations for linear algebra and optimization. *Berkeley, California*.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. (2018). Fast and scalable bayesian deep learning by weight-perturbation in adam. *arXiv preprint arXiv:1806.04854*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Kochurov, M., Garipov, T., Podoprikhin, D., Molchanov, D., Ashukha, A., and Vetrov, D. (2018). Bayesian incremental learning for deep neural networks. *arXiv preprint arXiv:1802.07329*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. (2017). Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pages 4655–4665.

Lemke, C., Budka, M., and Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130.

Lomonaco, V. and Maltoni, D. (2017). Core50: a new dataset and benchmark for continuous object recognition. *arXiv preprint arXiv:1705.03550*.

Lopez-Paz, D. et al. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6470–6479.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning. *arXiv preprint arXiv:1710.10628*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Ritter, H., Botev, A., and Barber, D. (2018). Online structured laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995.

Zeno, C., Golan, I., Hoffer, E., and Soudry, D. (2018). Bayesian gradient descent: Online variational bayes learning with increased robustness to catastrophic forgetting and weight pruning. *arXiv preprint arXiv:1803.10123*.