Master's Thesis

Deep Bayesian Neural Networks for Continual Learning

지속학습을 위한 심층 베이지안 신경망

February 2018

Graduate School of Seoul National University
Department of Computer Science and Engineering

Seongho Son

Deep Bayesian Neural Networks for Continual Learning

지속학습을 위한 심층 베이지안 신경망

지도교수 장 병 탁

이 논문을 공학석사 학위논문으로 제출함

2017 년 10 월

서울대학교 대학원

컴퓨터공학부

손 성 호

손 성 호의 공학석사 학위논문을 인준함

2017 년 12 월

위 원 장	강 유	(인)
부위원장	장 병 탁	(인)
위 원	김 건 희	(인)

Abstract

In the problem of *continual learning*, tasks are given in sequence and the goal of the machine is to learn new tasks while retaining previously learned tasks' performances. Even though deep neural networks have become prevalent among machine learning techniques recently, they may fail to deal with continual learning environment. As the model learns new tasks, parameters contributing largely to previous tasks' performances may change and result in poor performance. This phenomenon is called *catastrophic forgetting*, and researchers tackled this problem using regularizers and structure optimization techniques.

We aim to show that applying Bayesian modelling to deep neural network is beneficial for continual learning. Not only does Bayesian framework provide systematic way of performing online learning, it also provides uncertainty estimates for measuring each parameter's contribution to previously learned tasks' performances. By rescaling gradients applied to parameters with large performance contribution, the model can retain performances to previous tasks longer. This thesis shows how deep Bayesian neural networks utilize model uncertainty, which leads to alleviation of catastrophic forgetting in continual learning environment.

Keywords: deep learning, bayesian neural network, continual learning, catastrophic forgetting, uncertainty estimation

Student Number: 2016-21213

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order to develop machines which are truly intelligent, machines need to be equipped with learning ability. Since the external environment is always changing and thus cannot be predicted perfectly with previously obtained knowledge, it is crucial for machines to continuously acquire new knowledge about the world, while utilizing them to refine its reaction toward the environment. This setting of machine learning is often called *continual learning* or *lifelong learning*(J. Lee et al., 2017; Thrun, 1994), where tasks arrive in sequence and the goal of the machine is to learn new tasks while retaining previously learned tasks' performances.

Among techniques developed in the field of machine learning, deep neural network has become prevalent in recent years, thanks to huge improvement in computational power of graphical processing unit(GPU)s. Various models including convolutional neural network and recurrent neural network have drastically outperformed other machine learning techniques in fields like computer

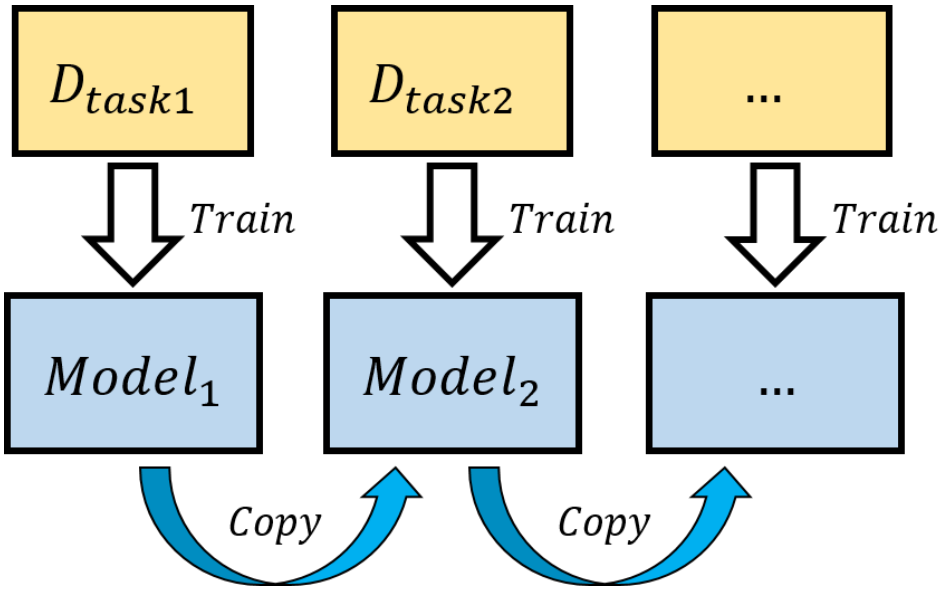

Figure 1.1 Continual learning environment

vision, voice recognition, and machine translation(Kalchbrenner and Blunsom, 2013; Krizhevsky, Sutskever, and Hinton, 2012; Mnih et al., 2013). Variants of deep neural network are now used in real-life applications, while many others are expected to be used in the future.

When it comes to continual learning, deep neural network bears possibility of failure. Since tasks are only given in sequence, parameters of the model which largely contributed to performance on previous tasks could change afterwards and result in poor performance. This phenomenon is called *catastrophic forgetting*, and researchers tackled this problem using regularizers and structure optimization techniques(Kirkpatrick et al., 2017; S. W. Lee et al., 2017; J. Lee et al., 2017; Zenke, Poole, and Ganguli, 2017).

One of the ways to tackle continual learning problem is applying Bayesian modeling to deep neural networks. Not only does Bayesian framework provide

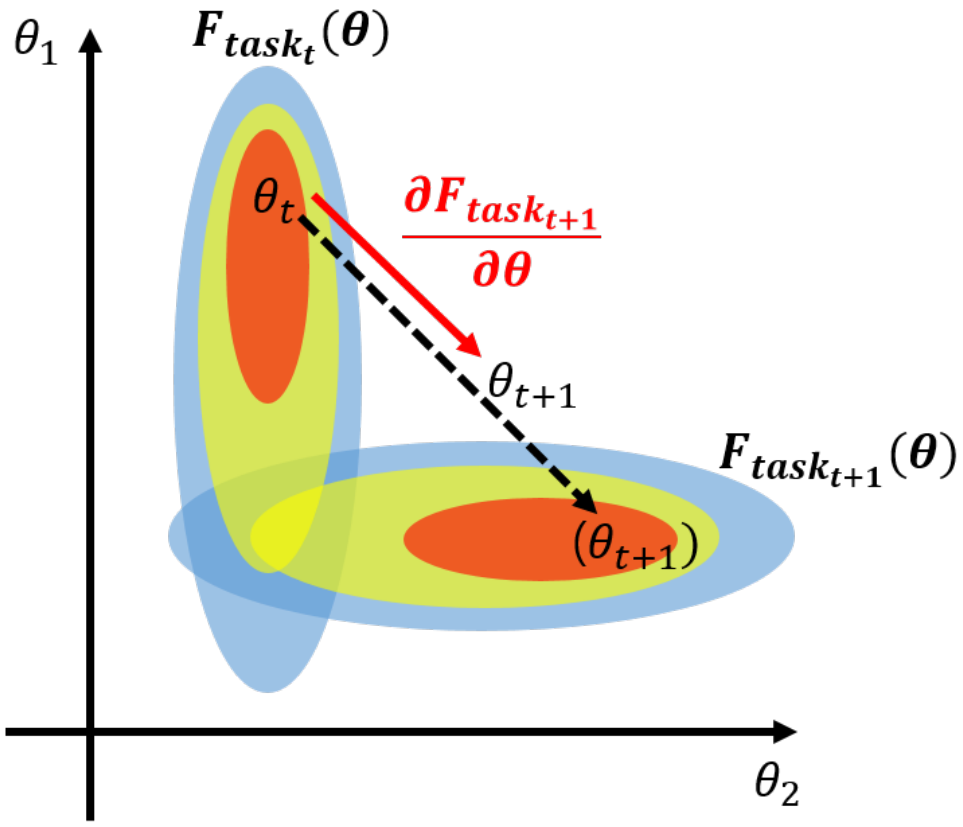

Figure 1.2 Catastrophic forgetting

systematic way of performing online learning, it also provides uncertainty estimates for various applications(Son, Kim, and Zhang, 2016b). With uncertainty estimates available, deep learning models' performance could be improved in fields including active learning and exploration optimization(Gal, 2016; Gal, Islam, and Ghahramani, 2017; Son, Kim, and Zhang, 2016a). Uncertainty estimates can be useful in continual learning setting as well, because they can be used to identify parameters with significant contribution to previously learned tasks' performances, which can lead to prevention of catastrophic

forgetting(Son, Kim, and Zhang, 2017).

We aim to show that utilizing the uncertainty information obtained from deep Bayesian neural network is beneficial for alleviation of catastrophic forgetting in continual learning problem. Through rescaling gradients of variational posterior means with uncertainty information of prior distribution, parameters with large contribution to previous tasks can be selectively protected from drastic changes. Principles lying behind deep Bayesian neural network models we rely on will be first introduced, followed by extension to online and continual learning problem using recursive Bayesian estimation. Our approach to catastrophic forgetting with rescaled gradients will then be introduced. Based on experiments conducted on disjoint MNIST and permuted MNIST datasets, we will show how our proposed method greatly improves the deep Bayesian neural network's continual learning capability. Benefits and drawbacks of the proposed algorithm will be analyzed as well.

Chapter 2

Preliminaries

2.1 Approaches on Catastrophic Forgetting

This chapter introduces how prior works approached continual learning problem. According to (Zenke, Poole, and Ganguli, 2017), studies can be roughly classified into three categories: architectural, functional, and structural approaches.

Architectural approaches change the architecture of neural networks, while leaving the objective function intact. This is done to reduce interference between parameters relating to different tasks. (Razavian et al., 2014) proposed an algorithm in which selected weights are completely prevented from changing their values. Copying the entire network after each task learning with feature augmentation for new tasks was another example of architectural approach(Rusu et al., 2016). Sparsifying gradients by injecting noise through application of dropout was also known to improve performances(Goodfellow et al., 2013).

Functional approaches try to penalize changes in the input-output mapping

of the neural network, by adding regularization term to loss functions. (Li and Hoiem, 2016) proposed *Learning without Forgetting*(LwF), in which the network tries to preserve the prediction outputs on previous task data. By generating pseudo-output label with current task data before the training for the new task begins, Learning without Forgetting approximates training on both old task data and current task data. (Jung et al., 2016) focuses on L2 distance between the final hidden activation values, rather than trying to preserve output values of the network.

Structural regularization methods are the techniques which this thesis mainly focuses on. Structural regularization methods add penalty terms based on the distance between the current parameter values and the values at the ends of training on previous tasks. *Elastic Weight Consolidation*(EWC) proposed by (Kirkpatrick et al., 2017) put a regularization term with a squared distance between parameter values of previous and current tasks. Diagonal terms of the Fisher information matrix are multiplied to corresponding squared distances, making each weight have different penalty value. Diagonal terms of Fisher information matrix are easily approximated by computing square of gradients with previous task's data. The loss function of the Elastic Weight Consolidation is given below:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2 \tag{2.1}$$

$\mathcal{L}_B$ denotes the loss for the current task B, $\theta^*_{A,i}$ denotes the i th parameter of the network after the training on the preceding task A, and F_i denotes the diagonal term of the Fisher information matrix corresponding to the i th parameter.

(S. W. Lee et al., 2017) proposed *Incremental Moment Matching*(IMM), in which optimal parameters $\mu^*_{1:K}$ and $\Sigma^*_{1:K}$ for K sequentially given tasks are

calculated. *Mode-IMM*, one of the Incremental Moment Matching technique, approximates mixture of Gaussian distributions using Laplacian approximation:

$$\log q_{1:K} \approx \sum_{k}^{K} \alpha_k \log q_k + C = -\frac{1}{2}\theta^T (\sum_{k}^{K} \alpha_k \Sigma_k^{-1})\theta + (\sum_{k}^{K} \alpha_k \Sigma_k^{-1}\mu_k)\theta + C' \tag{2.2}$$

$$\mu_{1:K}^{*} = \Sigma_{1:K}^{*} \cdot (\sum_{k}^{K} \alpha_k \Sigma_k^{-1}\mu_k) \tag{2.3}$$

$$\Sigma_{1:K}^{*} = (\sum_{k}^{K} \alpha_k \Sigma_k^{-1})^{-1} \tag{2.4}$$

Incremental Moment Matching also uses the inverse of the Fisher information matrix like (Kirkpatrick et al., 2017, Pascanu and Bengio, 2013). The Fisher information matrix for the kth task is calculated as below:

$$F_k = \mathbb{E}\left[\frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k) \cdot \frac{\partial}{\partial \mu_k} \ln p(\tilde{y}|x, \mu_k)^T\right] \tag{2.5}$$

where $x \sim \pi_k$ and $\tilde{y} \sim p(y|x, \mu_k)$. π_k denotes the empirical distribution of X_k, which is the input data of kth task.

2.2 Deep Bayesian Neural Networks

The history of applying Bayesian inference to neural network dates back to 1993(Hinton and Camp, 1993), which added Gaussian noise to neural network's weights in order to control the amount of information the weight is containing. Based on Hinton's work, (Graves, 2011) introduced a stochastic variational method that can be applied to most neural networks. (Kingma and Welling, 2013) introduced *Stochastic Gradient Variational Bayes*(SGVB) estimator, which is an unbiased estimator of the variational lower bound for efficient approximate posterior inference based on reparameterization technique.

It was also proved that stochastic regularization techniques including dropout are equivalent to casting Bayesian framework to deep neural networks(Gal and Ghahramani, 2015; Gal and Ghahramani, 2016b). Gal's works showed that by using dropout method, we can exploit the benefits of Bayesian inference with low computational complexity and smaller number of parameters(Gal and Ghahramani, 2016a; Gal, Islam, and Ghahramani, 2017).

The deep Bayesian neural networks mainly experimented in our work are based on the work of (Blundell et al., 2015), which uses an algorithm called *Bayes by Backprop* to make the model compatible with backpropagation. Each weight of the neural network is sampled from a unit Gaussian distribution with different mean and variance, and the cost function consists of a KL divergence between the variational posterior $q(\mathbf{w})$ and the true Bayesian posterior $P(\mathcal{D}|\mathbf{w})$. Since computing the exact value is intractable, Bayes by Backprop relies on Monte Carlo sampling to evaluate approximate expectation.

Chapter 3

Bayes by Backprop

This chapter summarizes how *Bayes by Backprop* works, upon which our work is built on. Bayes by Backprop is a backpropagation-compatible algorithm for learning a probability distribution on the weights of a deep neural network(Blundell et al., 2015). Instead of being represented as a single fixed value, every weight in the model is represented by a probability distribution. This makes the network sample each weight every time an inference is done, which results in training ensemble of networks using unbiased Monte Carlo estimates of the gradients.

3.1 Variational Approximation to the Posterior

Bayesian inference for neural networks requires calculating posterior distribution, which is then used for taking expectations to obtain predictions. Posterior distribution of the model's weights is represented as $P(\mathbf{w}|\mathcal{D})$, where $\mathcal{D}$ represents the training data. The predictive probability of an unknown label $\hat{y}$ corresponding to a given test input $\hat{x}$ is $P(\hat{y}|\hat{x}) = \mathbb{E}_{P(\mathbf{w}|\mathcal{D})}[P(\hat{y}|\hat{x}, \mathbf{w})]$. Since taking an expectation under the posterior distribution is equivalent to averaging over infinite number of sampled neural networks, the calculation is intractable.

Bayes by Backprop is based on the work of (Graves, 2011), in which a variational approximation to the Bayesian posterior distribution is suggested. Variational learning finds the parameters θ of a distribution on the weights $q(\mathbf{w}|\theta)$, which minimizes the Kullback-Leibler(KL) divergence between $q(\mathbf{w}|\theta)$ and the true Bayesian posterior on the weights:

$$\begin{aligned}
\theta^* &= \arg\min_{\theta} KL[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \\
&= \arg\min_{\theta} \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})} d\mathbf{w} \\
&= \arg\min_{\theta} KL[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] \qquad (3.1)
\end{aligned}$$

The cost function derived above is known as *variational free energy* or *expected lower bound*(Neal and Hinton, 1998). From now on, this cost function will be simplified as

$$\mathcal{F}(\mathcal{D}, \theta) = KL[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]. \qquad (3.2)$$

The cost function contains a data-dependent part, which includes log likelihood, and a prior-dependent part, which penalizes $q(\mathbf{w}|\theta)$ shifting away from $P(\mathbf{w})$.

3.2 Monte Carlo Gradients

The cost function obtained from the previous section cannot be exactly calculated. Hence gradient descent and Monte Carlo approximation are needed, in order to apply Bayesian learning to neural networks:

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D}|\mathbf{w}^{(i)}) \tag{3.3}$$

$\mathbf{w}^{(i)}$ in the equation above denotes the sample drawn from the variational posterior $q(\mathbf{w}|\theta)$. Each additive term in the approximated cost function above is using the same weight sample, so the gradient applied to each weight is only affected by the parts characterized by the weight samples.

3.3 Gaussian Variational Posterior

Variational posterior used here is supposed to be a diagonal Gaussian distribution. This enables the weight sample $\mathbf{w}^{(i)}$ to be obtained by drawing samples from a unit Gaussian distribution, shifting it by a mean μ and scaling it by a standard deviation σ. Standard deviation of the weight is parameterized as $\sigma = \log(1+\exp(\rho))$ so that σ is always non-negative. This makes the parameters of the variational posterior be $\theta = (\mu, \rho)$. Reparameterization trick suggested by (Kingma and Welling, 2013) gives the equation below:

$$\mathbf{w} = t(\theta, \epsilon) = \mu + \log(1 + \exp(\rho)) \circ \epsilon \tag{3.4}$$

$t(\theta, \epsilon)$ in the equation above denotes a deterministic function which makes the uncertainty of $\mathbf{w}$ rely on a sample of parameter-free noise. This enables expressing the derivative of an expectation as the expectation of a deriva-

tive(Blundell et al., 2015):

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathbf{w}|\theta)}[\mathcal{F}(\mathbf{w},\theta)] = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial\mathcal{F}(\mathbf{w},\theta)}{\partial\mathbf{w}}\frac{\partial\mathbf{w}}{\partial\theta} + \frac{\partial\mathcal{F}(\mathbf{w},\theta)}{\partial\theta}\right] \tag{3.5}$$

Since $\mathbf{w}$ in (3.4) can represent a matrix containing multiple values, $\circ$ in the equation denotes pointwise multiplication. This means that each value in ϵ is sampled independently from a unit Gaussian distribution $\mathcal{N}(0, I)$. With this setting of weights, **Algorithm 1** shows how the optimization works:

It was reported that allowing the prior distribution $P(\mathbf{w})$ to be shifted during training results in poorer performance(Blundell et al., 2015). Parameters constituting the prior distribution are allowed to quickly learn the approximated distribution of early training, which does not provide good information about the entire data. Due to this, parameters of the prior distribution $\theta^{prior} = (\mu^{prior}, \rho^{prior})$ were fixed to the initialized values during the entire training.

Algorithm 1: Bayes by Backprop(BbB)

Input: minibatches of training data $\{(X_1, Y_1), ..., (X_K, Y_K)\}$,

learning rate α,

number of Monte Carlo samples M,

number of training epochs N

Result: trained parameters $\theta^{posterior} = (\mu^{posterior}, \rho^{posterior})$;

for $n \in \{1, 2, ..., N\}$ *th epoch* **do**

  for $j \in \{1, 2, ..., K\}$ *th minibatch* **do**

    Sample $\epsilon^{(i)} \sim \mathcal{N}(0, I) \quad (i \in \{1, 2, ..., M\})$;

    $\mathbf{w}^{(i)} = \mu^{posterior} + log(1 + exp(\rho^{posterior})) \circ \epsilon^{(i)}$;

    Calculate $\mathcal{F}_{MC}(\mathbf{w}, \theta^{posterior}) =$

    $\sum_i \log q(\mathbf{w}^{(i)}|\theta^{posterior}) - \log P(\mathbf{w}^{(i)}) P(\mathcal{D}_j|\mathbf{w}^{(i)})$;

    $\Delta_{\mu^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}} + \frac{\partial \mathcal{F}_{MC}}{\partial \mu^{posterior}}$;

    $\Delta_{\rho^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}} \frac{\epsilon}{1+\exp(-\rho^{posterior})} + \frac{\partial \mathcal{F}_{MC}}{\partial \rho^{posterior}}$;

    $\mu^{posterior} \leftarrow \mu^{posterior} - \alpha \Delta_{\mu^{posterior}}$;

    $\rho^{posterior} \leftarrow \rho^{posterior} - \alpha \Delta_{\rho^{posterior}}$;

  end

end

Chapter 4

Proposed Method

This chapter introduces algorithms extending from Bayes by Backprop for continual learning environment. Recursive Bayesian estimation is used as a key method to online learning, in which the model updates its prior parameters θ^{prior} with its posterior parameters $\theta^{posterior}$ after every time step. *Rescaled gradient* method, which utilizes uncertainty information ρ^{prior} for scaling gradient $\frac{\partial \mathcal{F}}{\partial \mu^{posterior}}$, again extends from recursive Bayesian estimation to make the model compatible with continual learning. Resetting uncertainty $\rho^{posterior}$ of variational posterior is also suggested for better performance.

4.1 Bayes by Backprop for Online Learning

4.1.1 Recursive Bayesian Estimation

Bayes by Backprop may work in learning environments with fixed datasets, but its performance could severely degenerate in online learning setting. In order to make the model more compatible to online learning, we suggest applying *recursive Bayesian estimation* (Bayesian filtering) scheme to the model discussed so far. Recursive Bayesian estimation predicts the state at a certain time step x_t $t \in 0, 1, ...$ as the states change according to Markov process(Bergman, 1999). With prior distribution of the initial state given as $P(x_0)$, transition probability as $P(x_{t+1}|x_t)$, likelihood as $P(y_t|x_t; \mathcal{D})$, and the vector of measurements collected up to time step t as Y_t, the equation below is satisfied:

$$P(x_t|Y_t; \mathcal{D}) = \frac{P(y_t|x_t; \mathcal{D})P(x_t|Y_{t-1}; \mathcal{D})}{P(y_t|Y_{t-1})} \tag{4.1}$$

$$P(x_{t+1}|Y_t; \mathcal{D}) = \int_{\mathbb{R}^n} P(x_{t+1}|x_t)P(x_t|Y_t; \mathcal{D})dx_t \tag{4.2}$$

In the equation 4.1, we can see that the posterior distribution calculated at the last time step $t-1$ is used as the prior at the current time step t. This procedure enables continuous update of its estimation of the unknown target density function.

Using the same method, we can estimate the distribution of deep Bayesian neural network's parameters θ in online learning environment, where training data $\mathcal{D}_i$ ($i \in \{1, 2, ...\}$) is given sequentially and previously learned data cannot be used for later training(Son, Kim, and Zhang, 2016b). Parameter values of the prior distribution $\theta_i^{prior} = (\mu_i^{prior}, \rho_i^{prior})$ are fixed while the neural network is optimizing parameters $\theta_i^{posterior} = (\mu_i^{posterior}, \rho_i^{posterior})$ based on the current data $\mathcal{D}_i$. After the model is done with training of the variational posterior

$q(\mathbf{w}|\theta^{posterior})$, θ_{prior} is replaced with values of $\theta^{posterior}$ before the model begins training with the following data $\mathcal{D}_{i+1}$.

4.1.2 Mode-fitting with Posterior Uncertainty Reset

In the setting of *Bayes by Backprop* with recursive Bayesian estimation, parameters of the variational posterior $\theta^{posterior} = (\mu^{posterior}, \rho^{posterior})$ retain its values after the end of learning each task. This may be considered natural since the variational posterior $q(\mathbf{w}|\theta^{posterior})$ can be optimized again with the data of the new task. However, with the arrival of the new task's data with a different distribution, this setting may not always result in good performance.

Consider the problem of fitting a Gaussian distribution to a loss curve with complex shape. If the objective is to minimize the KL divergence between two distributions, the resulting shape of the Gaussian distribution may differ based on the value of initial variance. If the value of the initial variance was large, the Gaussian distribution will try to fit the global trend of the loss curve. On the contrary, if the initial value of the distribution's variance was small, the Gaussian distribution will try to fit to one of the loss curve's modes.

Same situation happens in learning environment using models based on

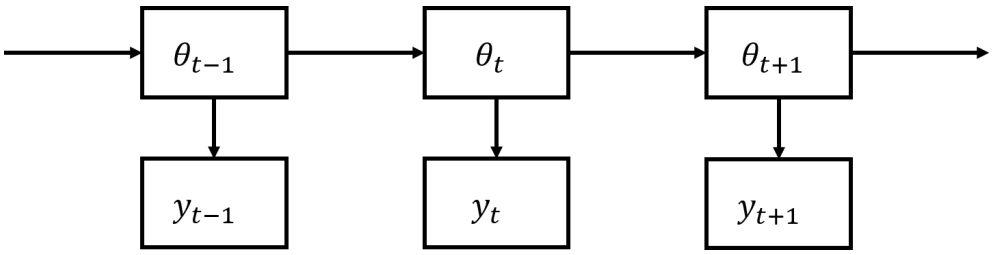

Figure 4.1 Recursive Bayesian estimation

Bayes by Backprop. Since each weight of the Bayesian neural network is sampled from a unit Gaussian distribution, we can think of the training procedure of the model as fitting each weight's correspondent Gaussian distribution to the loss surface with respect to the weight's dimension. Loss surface of the certain task's data $\mathcal{D}_i$ could differ drastically from another task's data $\mathcal{D}_j$, so depending on the variance value of each weight, its fitting tendency may differ like the way described at the previous paragraph. In continual learning environment, we cannot correctly assume the loss surface optimal to all the tasks considered together. Hence fitting to the mode of the current task's loss surface may be a reasonable solution.

We adopted the strategy of resetting the value of variational posterior $q(\mathbf{w}|\theta^{posterior})$'s $\rho^{posterior}$ to its initial small value before the model begins to start training on the new task's data. As it will be shown in the experiments later, this resetting of $\rho^{posterior}$ to small values generally results in better performance. The algorithm of performing online learning with resetting uncertainty of variational posterior is given below.

Algorithm 2: Bayes by Backprop with Recursive Bayesian Estimation(BbB-RBE)

Input: Sequence of training data $\{(X_1, Y_1), ..., (X_K, Y_K)\}$,

learning rate α,

number of Monte Carlo samples M,

number of training epochs N,

uncertainty reset value ρ^{init}

Result: trained parameters $\theta = (\mu, \rho)$;

for $j \in \{1, 2, ..., K\}$ *th data* **do**

$\rho^{posterior} \leftarrow \rho^{init}$;

for $n \in \{1, 2, ..., N\}$ *th epoch* **do**

Sample $\epsilon^{(i)} \sim \mathcal{N}(0, I) \quad (i \in \{1, 2, ..., M\})$;

$(\mathbf{w}_j^{posterior})^{(i)} = \mu_j^{posterior} + log(1 + exp(\rho_j^{posterior})) \circ \epsilon^{(i)}$;

Calculate

$\mathcal{F}_{MC}(\mathbf{w}_j^{posterior}, \theta_j^{posterior}) = \sum_i \log q((\mathbf{w}_j^{posterior})^{(i)} | \theta_j^{posterior}) -$

$\log P((\mathbf{w}_j^{posterior})^{(i)}) P(\mathcal{D}_j | (\mathbf{w}_j^{posterior})^{(i)})$;

$\Delta_{\mu_j^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}_j^{posterior}} + \frac{\partial \mathcal{F}_{MC}}{\partial \mu_j^{posterior}}$;

$\Delta_{\rho_j^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}_j^{posterior}} \frac{\epsilon}{1+\exp(-\rho_j^{posterior})} + \frac{\partial \mathcal{F}_{MC}}{\partial \rho_j^{posterior}}$;

$\mu_j^{posterior} \leftarrow \mu_j^{posterior} - \alpha \Delta_{\mu_j^{posterior}}$;

$\rho_j^{posterior} \leftarrow \rho_j^{posterior} - \alpha \Delta_{\rho_j^{posterior}}$;

end

$\mu_{j+1}^{prior} \leftarrow \mu_j^{posterior}$;

$\rho_{j+1}^{prior} \leftarrow \rho_j^{posterior}$;

end

4.2 Rescaled Gradient Method

Uncertainty of model parameters can be used as approximate measure of each parameter's contribution to previous tasks' performances. Using this information, we can rescale changes to parameters which are assumed to be important to previous tasks. This can alleviate catastrophic forgetting, as will be shown in the experiments mentioned later.

Reparameterized weight $\mathbf{w}$ contains two parameters: μ and ρ. While μ corresponds to the mean of the Gaussian distribution, $\log(1+exp(\rho))\circ\epsilon$ corresponds to the standard deviation of the distribution. It means that the smaller the value of $\log(1+exp(\rho))\circ\epsilon$ is, the smaller the uncertainty of the weight sampled from the corresponding probability distribution becomes. Since the resulted parameters of learning the previous task are denoted by θ^{prior} and currently learning parameters are denoted by $\theta^{posterior}$, the size of $\log(1+exp(\rho^{prior}))\circ\epsilon$ can be considered to be inversely proportional to the importance of μ^{prior} to the previous task. Hence we can rescale the change of parameters $\mu^{posterior}$ by multiplying *rescaling value* to the gradient $\frac{\partial\mathcal{F}}{\partial\mu^{posterior}}$:

$$s_i = (\sigma_i^{prior})^{-a} = (\log(1+exp(\rho_i^{prior})))^{-a} \qquad (a\in[1,\infty)) \tag{4.3}$$

As the value of *rescaling controller* a gets bigger, the polarization applied to the gradient values becomes stronger. It makes the Gaussian distributions corresponding to weights with small prior uncertainty (ones with standard deviations smaller than 1) harder to shift their means, while making the distributions corresponding to weights with big prior uncertainty (ones with standard deviations bigger than 1) easier to shift their means. This enables the model to retain its knowledge on the previously learned tasks, while making the weights

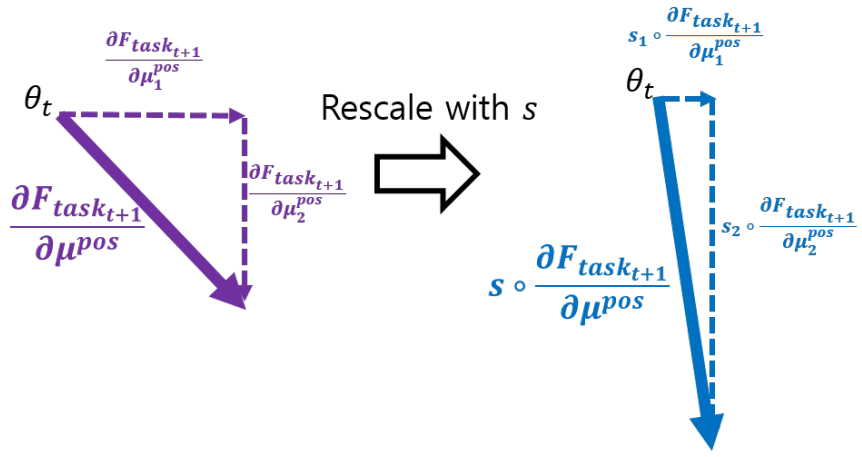

Figure 4.2 Rescaled gradient method

with bigger uncertainty contribute more on the learning of the current task.

Algorithm 3: Rescaled Gradient Method(ResGrad) for Continual Learning

Input: Sequence of training data $\{(X_1, Y_1), ..., (X_K, Y_K)\}$,

learning rate α,

number of Monte Carlo samples M,

number of training epochs N,

rescaling controller $a \in [1, \infty)$,

uncertainty reset value ρ^{init}

Result: trained parameters $\theta^{posterior} = (\mu^{posterior}, \rho^{posterior})$;

for $j \in \{1, 2, ..., K\}$ *th task data* **do**

$\rho^{posterior} \leftarrow \rho^{init}$;

for $n \in \{1, 2, ..., N\}$ *th epoch* **do**

Sample $\epsilon^{(i)} \sim \mathcal{N}(0, I) \quad (i \in \{1, 2, ..., M\})$;

$(\mathbf{w}_j^{posterior})^{(i)} = \mu_j^{posterior} + log(1 + exp(\rho_j^{posterior})) \circ \epsilon^{(i)}$;

Calculate

$\mathcal{F}_{MC}(\mathbf{w}_j^{posterior}, \theta_j^{posterior}) = \sum_i \log q((\mathbf{w}_j^{posterior})^{(i)} | \theta_j^{posterior}) -$

$\log P((\mathbf{w}_j^{posterior})^{(i)}) P(\mathcal{D}_j | (\mathbf{w}_j^{posterior})^{(i)})$;

$s_j = (\log(1 + exp(\rho_j^{prior})))^{-a}$;

$\Delta_{\mu_j^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}_j^{posterior}} + \frac{\partial \mathcal{F}_{MC}}{\partial \mu_j^{posterior}}$;

$\Delta_{\rho_j^{posterior}} = \frac{\partial \mathcal{F}_{MC}}{\partial \mathbf{w}_j^{posterior}} \frac{\epsilon}{1+\exp(-\rho_j^{posterior})} + \frac{\partial \mathcal{F}_{MC}}{\partial \rho_j^{posterior}}$;

$\mu_j^{posterior} \leftarrow \mu_j^{posterior} - \alpha s_j \circ \Delta_{\mu_j^{posterior}}$;

$\rho_j^{posterior} \leftarrow \rho_j^{posterior} - \alpha \Delta_{\rho_j^{posterior}}$;

end

$\mu_{j+1}^{prior} \leftarrow \mu_j^{posterior}$;

$\rho_{j+1}^{prior} \leftarrow \rho_j^{posterior}$;

end

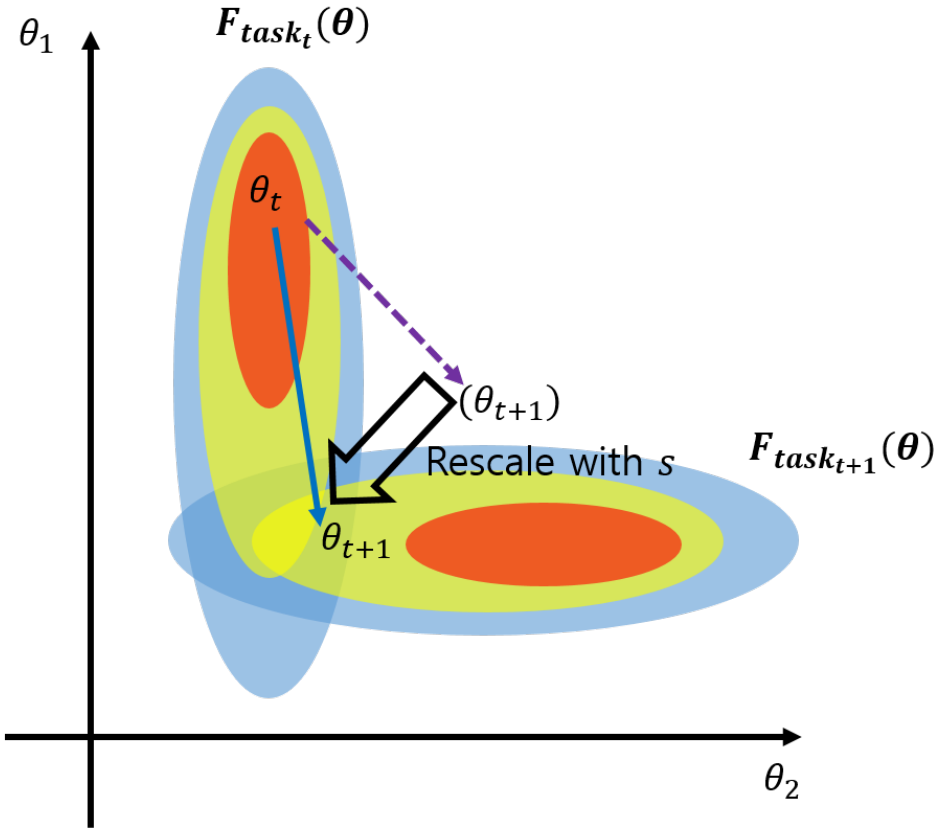

Figure 4.3 Result of applying rescaled gradient

Chapter 5

Experiments

5.1 Configuration

We compared conventional deep learning models with our model. Deep neural networks with stochastic gradient descent method(DNN), L2 regularization(L2), L2 regularization with respect to the weight values learned in the previous task(L2-Transfer), and Elastic Weight Consolidation proposed by (Kirkpatrick et al., 2017) were experimented. Model based on Bayes by Backprop with recursive Bayesian estimation (BbB-RBE) was also experimented to compare with our proposed algorithm, rescaled gradient method (ResGrad). Parameters $\rho^{posterior}$ of the deep Bayesian neural networks used in this experiment were reset to its initial values after training on each task is done, for mode-fitting on the loss surface. This is denoted by uncertainty resetting(UR) in the results shown below, while deep Bayesian neural networks that did not reset their $\rho^{posterior}$ values were marked NUR. The value of ρ^{init} was set to 10^{-4} for deep Bayesian neural networks implemented in this thesis, if not particularly noted.

Every deep Bayesian neural network model used for the experiments sampled 40 times for each input. Regularization coefficients used for L2 regularization and Elastic Weight Consolidation were denoted by λ, while the rescaling controller used for rescaled gradient method is denoted by a in the tables showing experimental results. Every model was trained with stochastic gradient descent method, in which data to learn is shuffled after every epoch and split into several minibatches.

Comparisons were done among the models with the same number of hidden layers and the same number of hidden nodes, while initial learning rates α were chosen among $\{10^{-4}, 10^{-5}\}$ depending on models for better results. Since every problem discussed below belongs to the format of online multi-task learning, test performances were calculated by averaging the test performances of data from all tasks. Some of the validation performances of each task data during the training procedure will also be given, in order to show how the performance on each task data changes. Performances show the percentage of correctly classified test data.

5.2 Disjoint MNIST Problem

We simulate continual learning environment with disjoint MNIST environment(S. W. Lee et al., 2017). Original MNIST dataset is divided into several tasks, each of which contains images from different labels. **Figure 5.1** shows how digit images are split into different tasks from the original MNIST dataset. Training models with this dataset simulates an environment where only part of neural network's prediction values are used for a certain task. This also makes the model decide from which task the current input is coming, making the

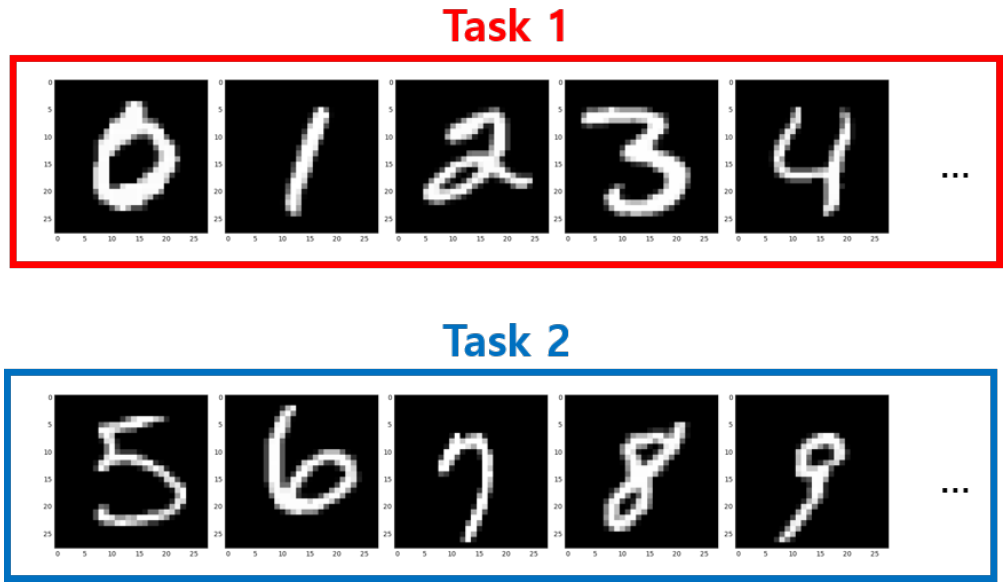

Figure 5.1 Disjoint MNIST dataset

problem more difficult(S. W. Lee et al., 2017).

During training, the model gets images belonging to one of the tasks, and when its learning on the current task is done, the model learns images from another task consisting of different digits. Tasks are given sequentially, and once the training on one of the tasks is done, the same data is not used again for later training.

Table 5.1 shows the test performances of algorithms on disjoint MNIST dataset. Our method, rescaled gradient, shows the best test performance compared to other models with 1 hidden layer. Among the models with rescaled gradient, those with uncertainty resetting and rescaling controller's value set to 2 showed better performance compared to others. However, the model with Incremental Moment Matching from the work of (S. W. Lee et al., 2017) showed

far better performance than deep Bayesian neural network with rescaled gradient.

Digits in MNIST dataset share many overlapping pixels that contribute to the classification. Hence the weights of neural network, which produces first hidden layer's node values from given input data, has many weights that are important to classifying several digits. However, when digits are learned sequentially, these weights are naturally exposed to catastrophic forgetting. The reason only the models from (S. W. Lee et al., 2017) were successful on disjoint MNIST dataset would be that in incremental moment matching, moments of posterior distributions incorporating all the previously learned tasks' information is calculated in a systematic way. Considering the test performance of Elastic Weight Consolidation, only penalizing the change of weights whose contributions to previous tasks are large is not enough. Rescaled gradient method also seems to yield finding variational posterior parameters $\theta^{posterior}$ that can contribute to both of the tasks, though its performance is not as good as that of incremental moment matching.

Used Algorithm	Model Setting	Number of Hidden Nodes	Task-averaged Performance
SGD		100	47.3
L2	$\lambda = 0.01$	100	47.5
L2-Transfer	$\lambda = 1$	100	47.8
EWC	$\lambda = 1000$	100	51.1
ResGrad	NUR, $a = 1$	100	53.3
ResGrad	UR, $a = 1$	100	54.9
ResGrad	NUR, $a = 2$	100	60.1
ResGrad	UR, $a = 2$	100	**64.5**
SGD		800, 800	47.7
L2	$\lambda = 0.01$	800, 800	47.7
L2-Transfer	$\lambda = 0.01$	800, 800	47.7
EWC	$\lambda = 1000$	800, 800	50.9
ResGrad	NUR, $a = 1$	800, 800	59.2
ResGrad	UR, $a = 1$	800, 800	61.6
ResGrad	NUR, $a = 2$	800, 800	61.0
ResGrad	UR, $a = 2$	800, 800	63.6
L2, Drop-transfer + Mode-IMM (S. W. Lee et al., 2017)	$\lambda = 10^{-3}$, $p = 0.5$, $\alpha_2 = 0.45$	800, 800	**94.12**

Table 5.1 Disjoint MNIST Experiment: task-averaged performances

5.3 Permuted MNIST Problem

This section shows experimental results with permuted MNIST environment (Kirkpatrick et al., 2017; S. W. Lee et al., 2017), consisting of seven sequential tasks. In this experiment, tasks are distinguished by different permutations. Data for the first task contains original MNIST dataset(LeCun, Cortes, and Burges, 2010), while data for other tasks contain images with the same permutations in the same task. Permutation is applied to every pixel of the image. In **Figure 5.2**, the images in the first task's data are the images of the original dataset, while the others in task 2 and 3 are the same images with different permutations. The way the model gets task data is the same with that of disjoint MNIST experiment, except that tasks are distinguished by different permutations applied to images. Experiments with the same dataset were done in (Kirkpatrick et al., 2017) and (S. W. Lee et al., 2017), but since the number of tasks used for the dataset and model configuration details are different from our work, so their results are omitted here.

Table 5.2 shows the test performances of algorithms after the training on permuted MNIST dataset. The model with our method, rescaled gradient, shows the best test performance compared to other models. Those with uncertainty resetting and $a = 2$ showed better performance, which is the same with the results of disjoint MNIST experiment. It is also notable that deep Bayesian neural network models without rescaled gradient method showed worse results than all the other algorithms, even compared to deep neural network with only stochastic gradient descent applied. When rescaled gradient method is applied, its performance is improved by almost twenty percent. It shows that even though Bayes by Backprop with recursive Bayesian estimation may give sub-

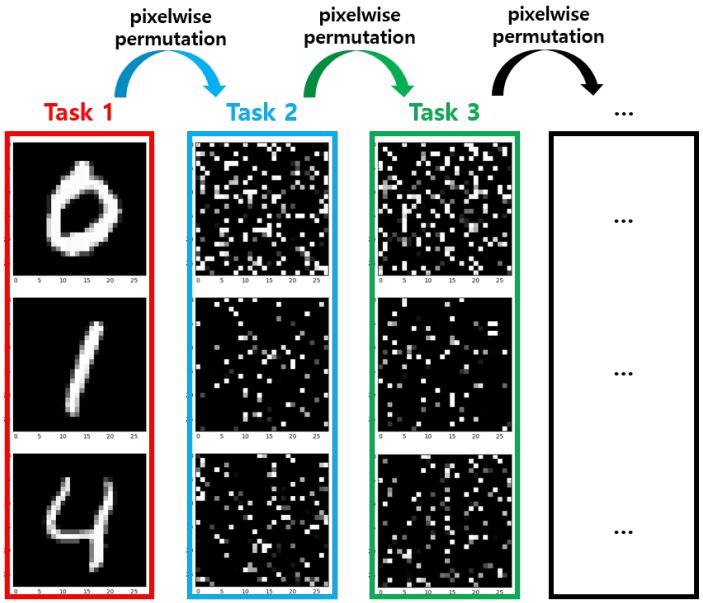

Figure 5.2 Permuted MNIST dataset

tle regularization effect, it easily suffers from catastrophic forgetting. Through rescaling the gradient $\frac{\partial \mathcal{F}}{\partial \mu^{posterior}}$ with uncertainty information of prior distribution ρ^{prior}, weights with small prior uncertainty were kept from drastic shift, while others with bigger prior uncertainty were easily changed for learning the current task.

Used Algorithm	Model Setting	Number of Hidden Nodes	Task-averaged Performance
SGD		100	83.9
L2	$\lambda = 0.1$	100	83.1
L2-Transfer	$\lambda = 1$	100	83.3
EWC	$\lambda = 100$	100	86.0
BbB-RBE	NUR	100	60.0
BbB-RBE	UR	100	65.1
ResGrad	NUR, $a = 1$	100	80.9
ResGrad	UR, $a = 1$	100	84.6
ResGrad	NUR, $a = 2$	100	85.1
ResGrad	UR, $a = 2$	100	**87.6**
SGD		800, 800	87.3
L2	$\lambda = 0.01$	800, 800	82.9
L2-Transfer	$\lambda = 0.1$	800, 800	83.6
EWC	$\lambda = 100$	800, 800	89.2
ResGrad	UR, $a = 2$, $\rho^{init} = 10^{-3}$	800, 800	**92.5**

Table 5.2 Permuted MNIST Experiment: task-averaged performances

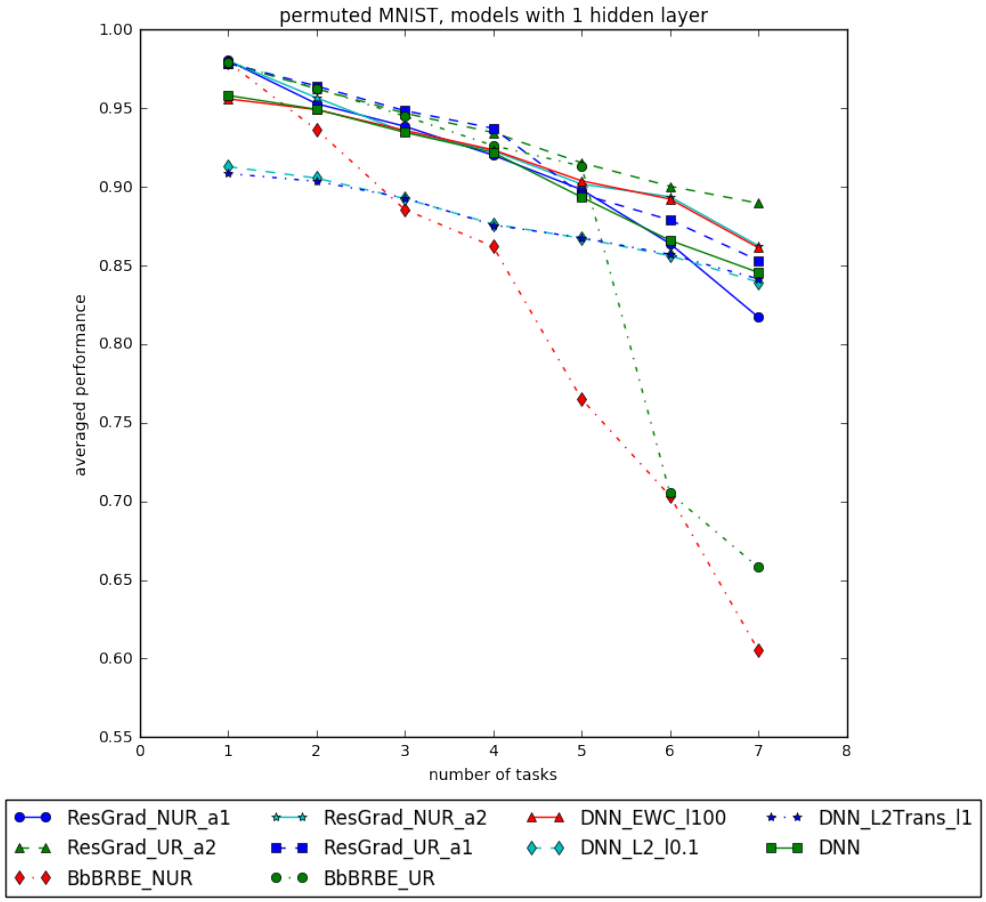

Figure 5.3 Permuted MNIST Experiment: validation performances of models with 1 hidden layer

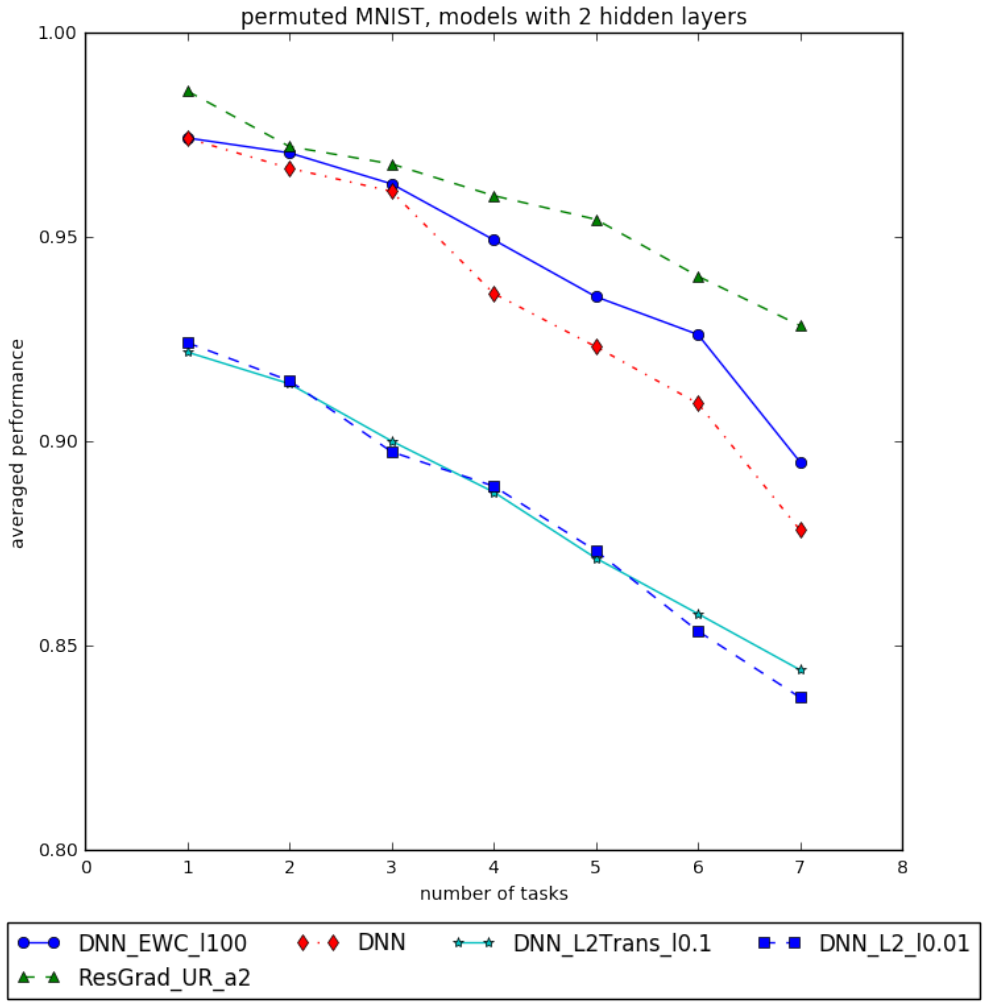

Figure 5.4 Permuted MNIST Experiment: validation performances of models with 2 hidden layers

Three graphs given below show how the validation performances of different tasks change as the learning proceeds with different models. In the case of deep neural network trained without any regularization method, the rate of performance decrease between different tasks are similar. Model with Elastic Weight Consolidation alleviated the overall trend of performance decrease, but the performance of each task drops faster as more tasks are trained. On the contrary, deep Bayesian neural network with rescaled gradient method applied shows far less performance drop. The model also preserves the first task's validation performance high for longer period, while the tasks trained later showed lower highest validation performance compared to the first one. This shows that rescaled gradient method prevented posterior mean values $\mu^{posterior}$ that are assumed to be important from easily changing their values, only allowing other parameters to adjust to the new tasks.

Compared to the experiments conducted on disjoint image datasets, the performances of algorithms used in permuted MNIST experiment are generally higher, even though the number of tasks is more than three times larger. This is because once the pixels are permuted, parameters $\mu^{posterior}$ that are crucial to current task's performance barely overlap with those of previous tasks. Hence retaining the performance on previous tasks can be easily achieved by only penalizing the change of important weights.

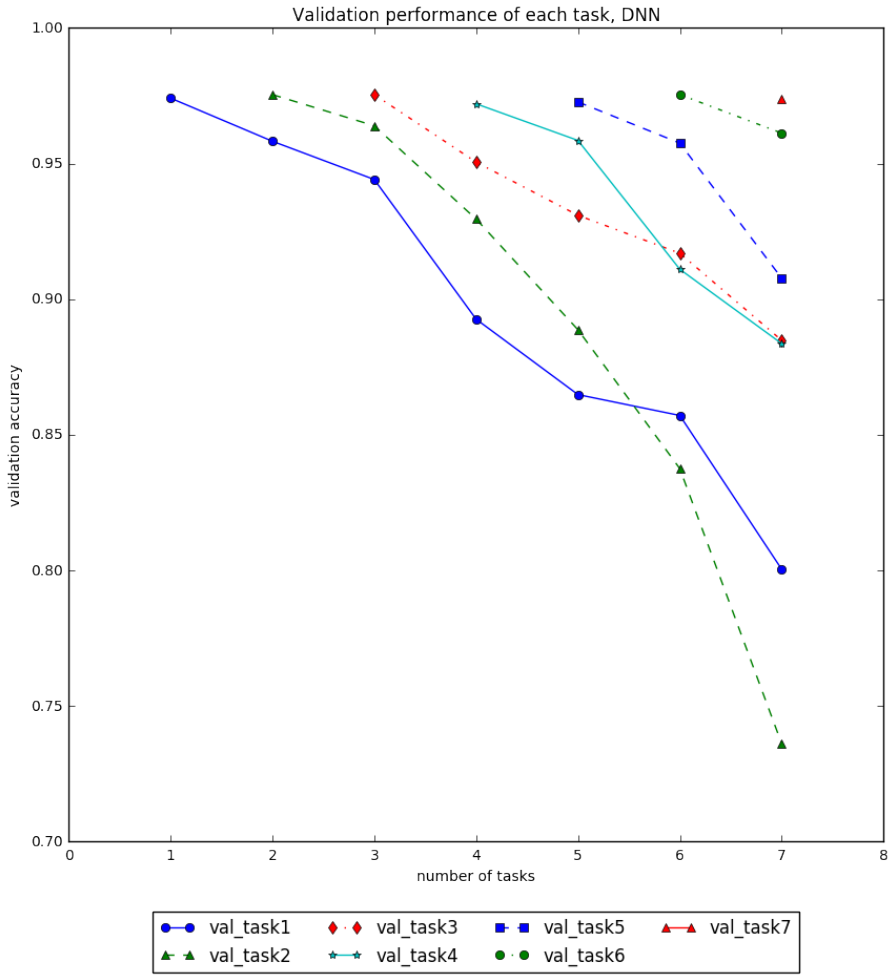

Figure 5.5 Permuted MNIST Experiment: per-task performances of deep neural network

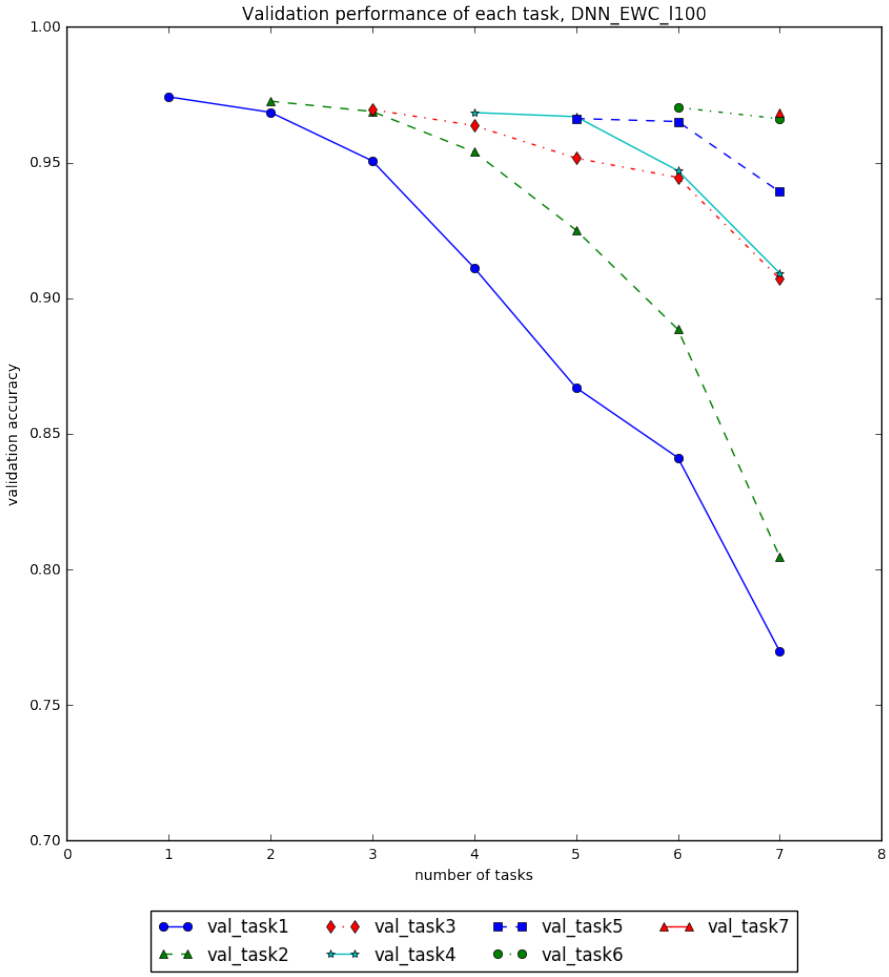

Figure 5.6 Permuted MNIST Experiment: per-task performances of deep neural network with Elastic Weight Consolidation

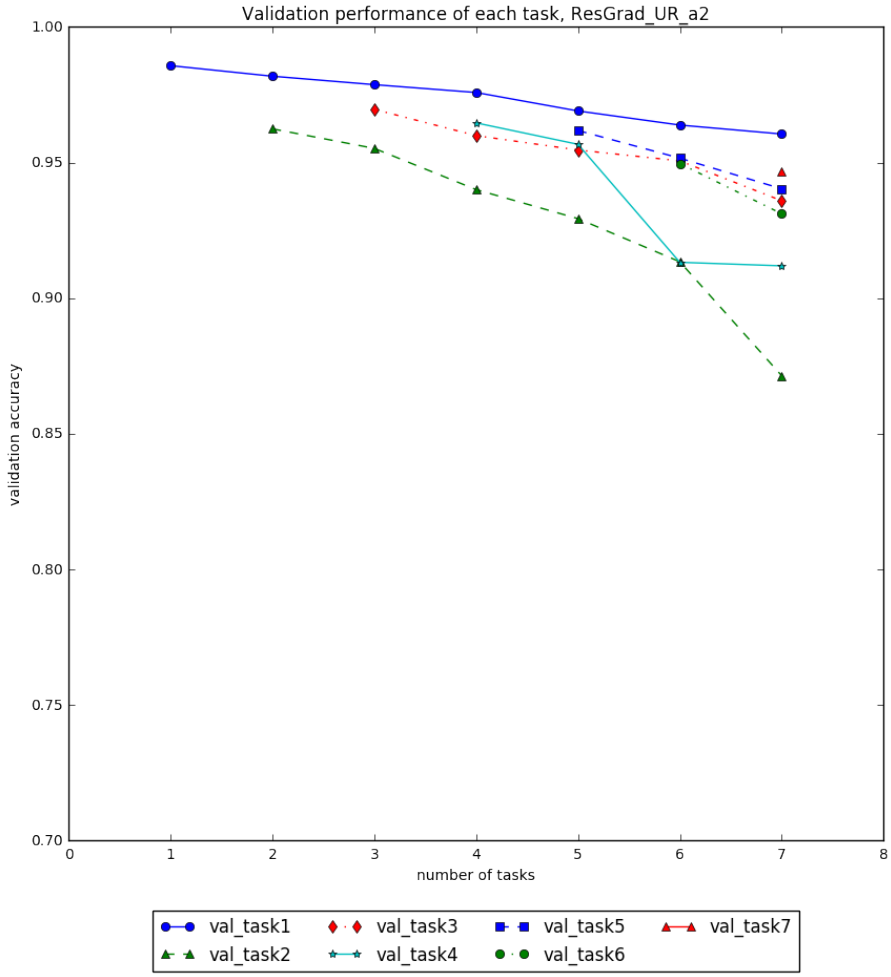

Figure 5.7 Permuted MNIST Experiment: per-task performances of deep Bayesian neural network with rescaled gradients

Figure 5.8 and **Figure 5.9** show the plotted parameters of various models' first layers used for permuted MNIST experiment. Each model consisted of 1 hidden layer with 100 hidden nodes, which means that there are 78400 parameters in the first layer. these parameters were reshaped into 100 images with 784 pixels, both height and width of each image being 28 pixels. Images in the left column show the parameters after the training of the first task was done, while images in the right show the parameters after the training of the seventh task. Brighter dots represent higher values, while darker dots represent lower values.

Plotted images of the deep Bayesian neural network's parameters $\mu^{posterior}$ in **Figure 5.8** show less noise compared to other non-Bayesian models. This seems to be caused by the regularization effect, which reduces the variance of parameter values $\mu^{posterior}$ that are less important to current task. Noise-like patterns in the images in the right column show that models learned permuted MNIST images following the first task data. Compared to non-Bayesian models, less noise-like patterns are shown in the images of the Bayesian model. This shows that Bayesian framework with rescaled gradient method helped retain the parameter values important to previous tasks, while still utilizing the remaining parameters for the training of the current task .

Figure 5.9 shows plotted values of $\rho^{posterior}$ of deep Bayesian neural network. The top image shows the values after the first task's training, while the image below shows the values after the seventh task's training. Since the first task data consisted of original MNIST images, parameters with low values form digit-like shapes after the training. Since lower values of $\rho^{posterior}$ mean smaller uncertainty of the corresponding $\mu^{posterior}$, this image shows that rescaled gradient method is going to preserve $\mu^{posterior}$ values in the darker area. After

the training of the seventh task, values of $\rho^{posterior}$ with bigger importance to the current task are shown darker, while digit-like shapes are not seen at all. This is due to uncertainty reset technique used for mode-fitting, and also implies that relatively better performance was obtained in permuted MNIST problem because each task had different set of important weights. Hence continual learning environments like disjoint MNIST problem, where important parameters for different tasks largely overlap, may require different techniques other than rescaled gradient method for better performance.

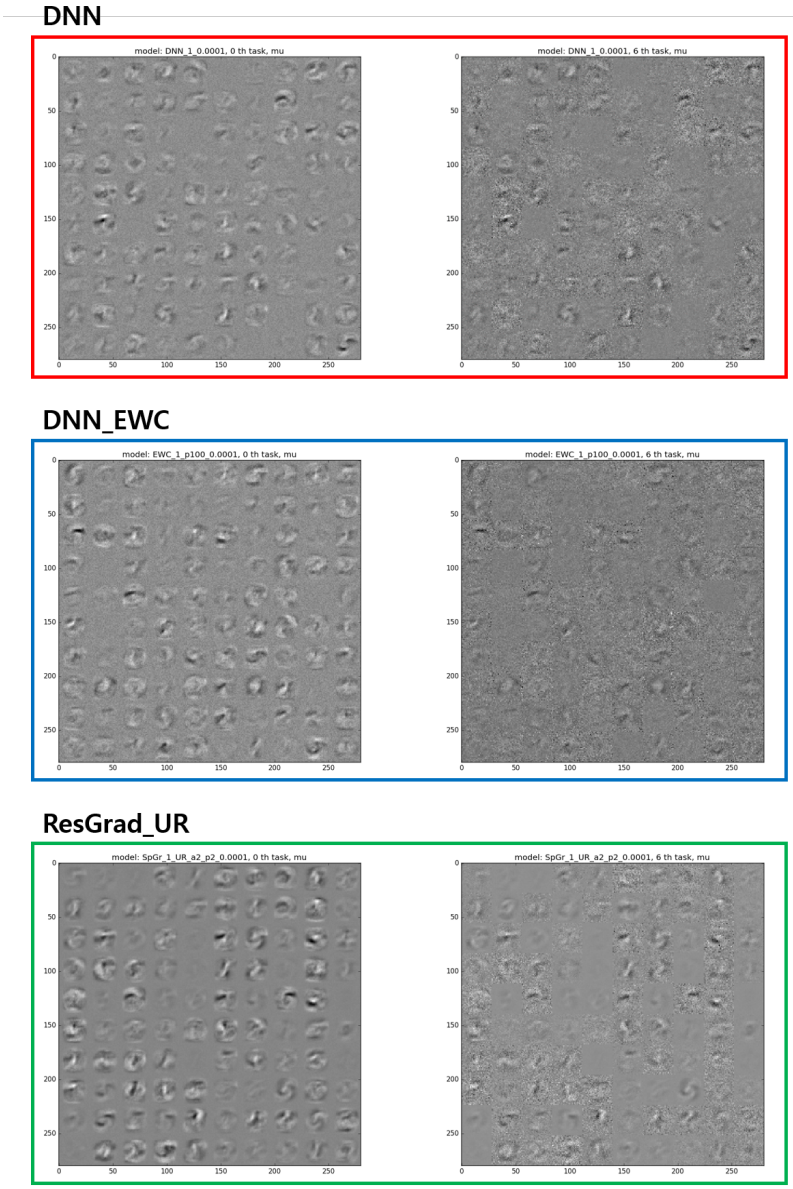

Figure 5.8 Permuted MNIST Experiment: plotted parameters($\mathbf{w}, \mu^{posterior}$) of various models' first layers

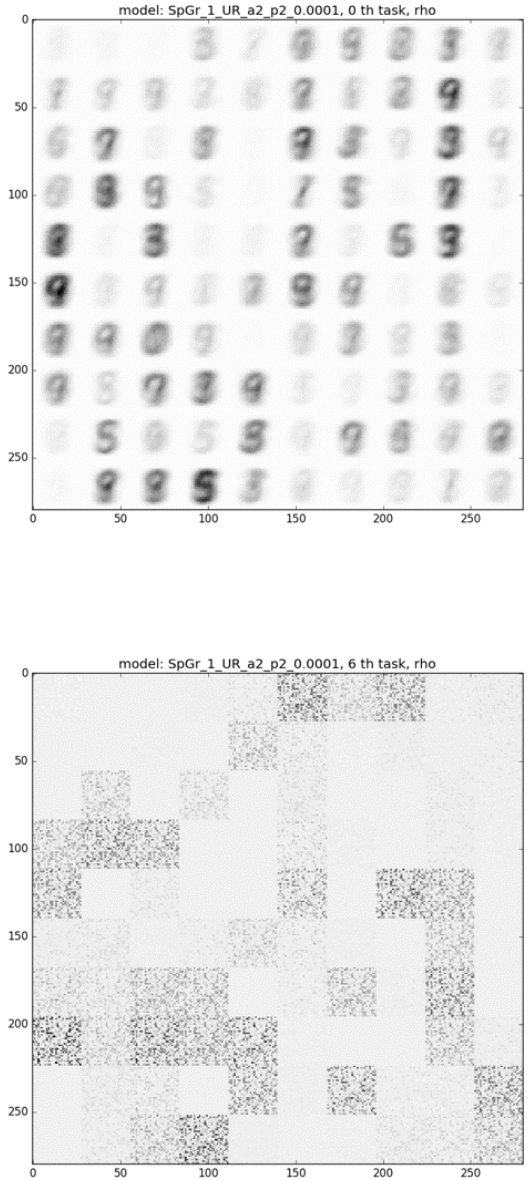

Figure 5.9 Permuted MNIST Experiment: plotted $\rho^{posterior}$ of a deep Bayesian neural network's first layer

Chapter 6

Conclusion

In this thesis, we proposed an algorithm using deep Bayesian neural network for continual learning, which can alleviate catastrophic forgetting problem. Using variational inference and Monte Carlo sampling, Bayesian neural networks can approximate true Bayesian posterior and achieve regularization effect(Blundell et al., 2015). Since Bayesian neural networks can obtain information on the distribution of parameters, it can utilize uncertainty information in various applications including active learning and continual learning(Gal and Ghahramani, 2016a; Gal, Islam, and Ghahramani, 2017; Son, Kim, and Zhang, 2016a).

We first applied recursive Bayesian estimation to the previously proposed Bayesian neural network for online learning setting, then rescaled gradients applied to the means of variational posterior distribution with uncertainty information obtained from prior distribution. This alleviates catastrophic forgetting, because the values of parameters ρ^{prior} approximate the importance of corresponding means on the previously learned tasks. We also showed through ex-

periments that the stronger the rescaling becomes, the better the model retains the previously acquired information. Experiments showed that deep Bayesian neural networks with rescaled gradients performed better than most of the conventional methods including Elastic Weight Consolidation from (Kirkpatrick et al., 2017), though its performance was not as good as incremental moment matching(S. W. Lee et al., 2017) in disjoint MNIST experiment.

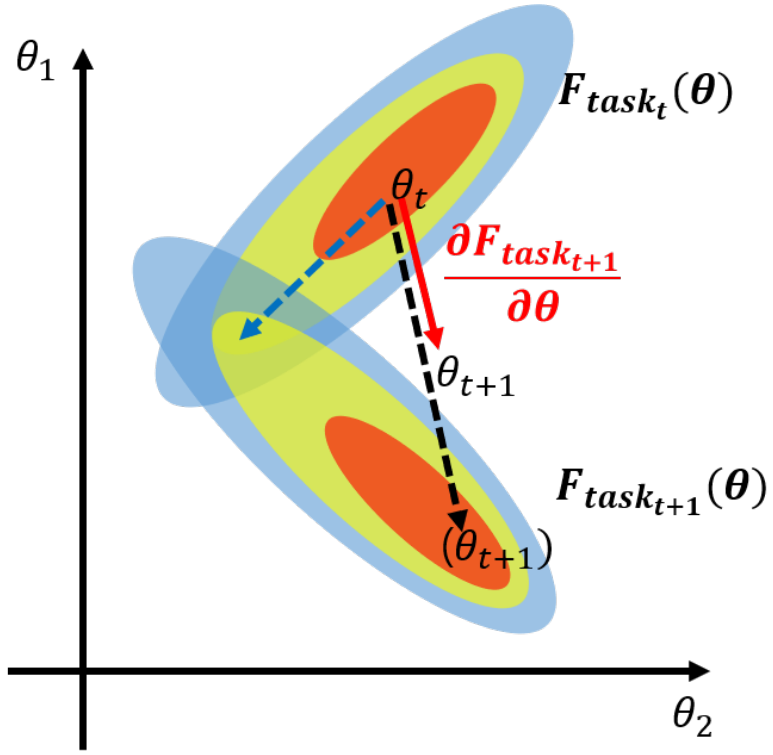

Figure 6.1 An example in which rescaled gradient method cannot contribute to performance improvement

Though high performances were achieved in the experiments, there are situations in which rescaled gradient method may not work well. In situations like **Figure 6.1**, higher task-averaged performance can be achieved when gradients

are set considering the entire landscape of loss surfaces(Blue dashed direction in the figure). However, rescaled gradient method relies on local approximations of the loss surface near the parameter values of prior distribution μ^{prior}, while only nonnegative values are used to rescale gradient values. Other regularization methods including (Kirkpatrick et al., 2017) and (Zenke, Poole, and Ganguli, 2017) share the same limitations, since they also rely on only penalizing the changes of weights that are considered important.

Deep Bayesian neural networks based on Bayes by Backprop share inherent limitations. Models contain twice as more parameters(μ and ρ for each weight) compared to other non-Bayesian neural networks with the same structure. They are slower proportional to the number of Monte Carlo samples, so parallel computing is required for faster inference and learning. Due to these limitations, using deep Bayesian neural networks may not be an optimal option for environments where small computational complexity and model size are necessary. However, since there are many benefits that can be exploited from uncertainty information which can only be acquired from Bayesian framework, future researches to circumvent the current limitations will be valuable. Dropout method, which was proved to be bearing Bayesian characteristics by (Gal and Ghahramani, 2015; Gal and Ghahramani, 2016b), can be one of the alternative options of performing Bayesian deep learning without suffering from limitations mentioned above.

There are many other possible researches that could extend from our work. Applying Bayesian framework to other types of neural networks including recurrent neural networks and convolutional neural networks would be one of the interesting examples, as similar works were already done by (Gal and Ghahra-

mani, 2016a; Gal, 2016). Since the performance of rescaled gradient method was not satisfactory in disjoint MNIST experiment, incorporating the principle of incremental moment matching in our algorithm could also be a plausible option for better continual learning performance. Using rescaled gradient method in active learning or exploration in reinforcement learning environment would be a straightforward application, as these fields are good examples of utilizing uncertainty information of Bayesian models.

Bibliography

[1] N. Bergman. "Recursive bayesian estimation". PhD thesis. Department of Electrical Engineering, Linköping University, Linköping Studies in Science and Technology, 1999.

[2] C. Blundell et al. "Weight uncertainty in neural networks". In: *Proceedings of the 32nd International Conference on Machine Learning(ICML-15)* (2015), pp. 1613–1622.

[3] Y. Gal. "Uncertainty in deep learning". PhD thesis. Department of Engineering, University of Cambridge, 2016.

[4] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian approximation: Insights and applications". In: *Deep Learning Workshop, ICML* (2015).

[5] Y. Gal and Z. Ghahramani. "Bayesian convolutional neural networks with Bernoulli approximate variational inference". In: *ICLR workshop track* (2016).

[6] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". In: *ICML* (2016).

[7] Y. Gal, R. Islam, and Z. Ghahramani. "Deep Bayesian Active Learning with Image Data". In: *arXiv preprint arXiv:1703.02910* (2017).

[8] I. J. Goodfellow et al. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211* (2013).

[9] A. Graves. "Pratical variational inference for neural networks". In: *Advances in Neural Information Processing Systems* (2011), pp. 2348–2356.

[10] G. E. Hinton and D. V. Camp. "Keeping the neural networks simple by minimizing the description length of the weights". In: *Proceedings of the 16th Annual Conference On Learning Theory(COLT)* (1993), pp. 5–13.

[11] H. Jung et al. "Less-forgetting Learning in Deep Neural Networks". In: *arXiv preprint arXiv:1607.00122* (2016).

[12] N. Kalchbrenner and P. Blunsom. "Recurrent Continuous Translation Models". In: *EMNLP* 3.39 (2013), p. 413.

[13] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[14] J. Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 201611835 (2017).

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* (2012), pp. 1097–1105.

[16] Y. LeCun, C. Cortes, and C. J. Burges. *MNIST handwritten digit database*. 2010. URL: `http://yann.lecun.com/exdb/mnist`.

[17] J. Lee et al. “Lifelong Learning with Dynamically Expandable Networks”. In: *arXiv preprint arXiv:1708.01547* (2017).

[18] S. W. Lee et al. “Overcoming Catastrophic Forgetting by Incremental Moment Matching”. In: *arXiv preprint arXiv:1703.08475* (2017).

[19] Z. Li and D. Hoiem. “Learning without forgetting”. In: *European Conference on Computer Vision* (2016), pp. 614–629.

[20] V. Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[21] R. M. Neal and G. E. Hinton. “A view of the EM algorithm that justifies incremental, sparse, and other variants”. In: *Learning in graphical models* (1998), pp. 355–368.

[22] R. Pascanu and Y. Bengio. “Revisiting natural gradient for deep networks”. In: *arXiv preprint arXiv:1301.3584* (2013).

[23] A. S. Razavian et al. “Cnn features off-the-shelf: an astounding baseline for recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2014), pp. 806–813.

[24] A. A. Rusu et al. “Progressive neural networks”. In: *arXiv preprint arXiv:1606.04671* (2016).

[25] S. H. Son, J. S. Kim, and B. T. Zhang. “Active Image Learning of Household Robots Using Bayesian Neural Network”. In: *Proceedings of The 43rd Winter Conference of Korean Institute of Information Scientists and Engineers* (2016), pp. 690–602.

[26] S. H. Son, J. S. Kim, and B. T. Zhang. “Online Image Recognition Using Bayesian Neural Network for Sequential Estimation”. In: *Proceedings of Korea Computer Congress 2016* (2016), pp. 1033–1035.

[27] S. H. Son, J. S. Kim, and B. T. Zhang. “Sequential Multitask Learning Optimization Using Bayesian Neural Network”. In: *Proceedings of Korea Computer Congress 2017* (2017), pp. 943–945.

[28] S. Thrun. “A lifelong learning perspective for for mobile robot control”. In: *Intelligent Robots and Systems'94. 'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on* 1 (1994), pp. 23–30.

[29] F. Zenke, B. Poole, and S. Ganguli. “Continual learning through synaptic intelligence”. In: *International Conference on Machine Learning* (2017), pp. 3987–3995.

국문초록

지속학습을 위한 심층 베이지안 신경망

손 성 호
컴퓨터공학부
서울대학교 대학원

지속학습 문제에서 개별 작업은 순차적으로 주어지며, 학습 모델의 목표는 기존에 학습한 작업들에 대한 성능을 유지하면서 새로운 작업을 학습하는 것이 된다. 심층 신경망은 최근 기계학습 기법으로서 각광받고 있으나, 지속학습 환경에서는 제대로 작동하지 못할 수 있다. 모델이 새로운 작업을 배우는 동안 기존의 작업에 대한 성능에 기여도가 큰 매개변수가 변화하여 좋지 않은 성능을 보일 수 있기 때문이다. 이 현상은 *파멸적 망각(catastrophic forgetting)*이라 하며, 기존 연구에서는 구조 최적화나 정규화 기법 등을 이용하여 이 문제에 접근하였다.

본 논문에서는 심층신경망에 베이지안 모델링을 적용하는 것이 지속학습에 도움을 준다는 것을 보이고자 한다. 베이지안 프레임워크는 신경망이 온라인 학습을 수행할 수 있는 체계적인 방법을 제공할뿐만 아니라 모델의 각 매개변수가 이전에 학습한 작업들의 성능에 어느 정도 기여하는지를 불확실성 계량을 통해 알 수 있게 해준다. 기존 작업에 대한 성능 기여도가 높은 매개변수의 변화를 억제함으로써, 학습 모델은 기존 작업에 대한 작업 성능을 더 오래 유지할 수 있게 된다. 본 논문에서는 심층 베이지안 신경망이 어떻게 모델의 불확실성을 활용하며, 이것이 어떻게 지속학습 환경에서 파멸적 망각 현상을 완화하게 되는지를 보인다.

주요어: 심층신경망, 베이지안 신경망, 지속학습, 파멸적 망각, 불확실성 추정
학번: 2016-21213