Project 2 - Continual Learning (CL) for Robotic Perception Report

Due on April 5th, 11:59 pm

Professor Pantelis Monogioudis

Zhu, Haoran - hz1922

Li, Qi - ql1045

Zhao, Jingyi - jz2216

Project Goal

According to class website's description:

CORe50 Option	Rotated MNIST Option
You will use this dataset and evaluate your method for New Class (NC) scenario.	Use the dataset provided here based on this paper
Object recognition (classification).	Object recognition (classification).

We have two project options while we have done experiments on both settings.

For small dataset, instead of testing rotated MNIST, we use MNIST and FASHION MNIST as datasets for two separate tasks: classifying digit numbers and classifying fashion items. (Professor said this setting will be ok in the slack) We'll test whether the network can learn new tasks without forgetting the previous knowledge.

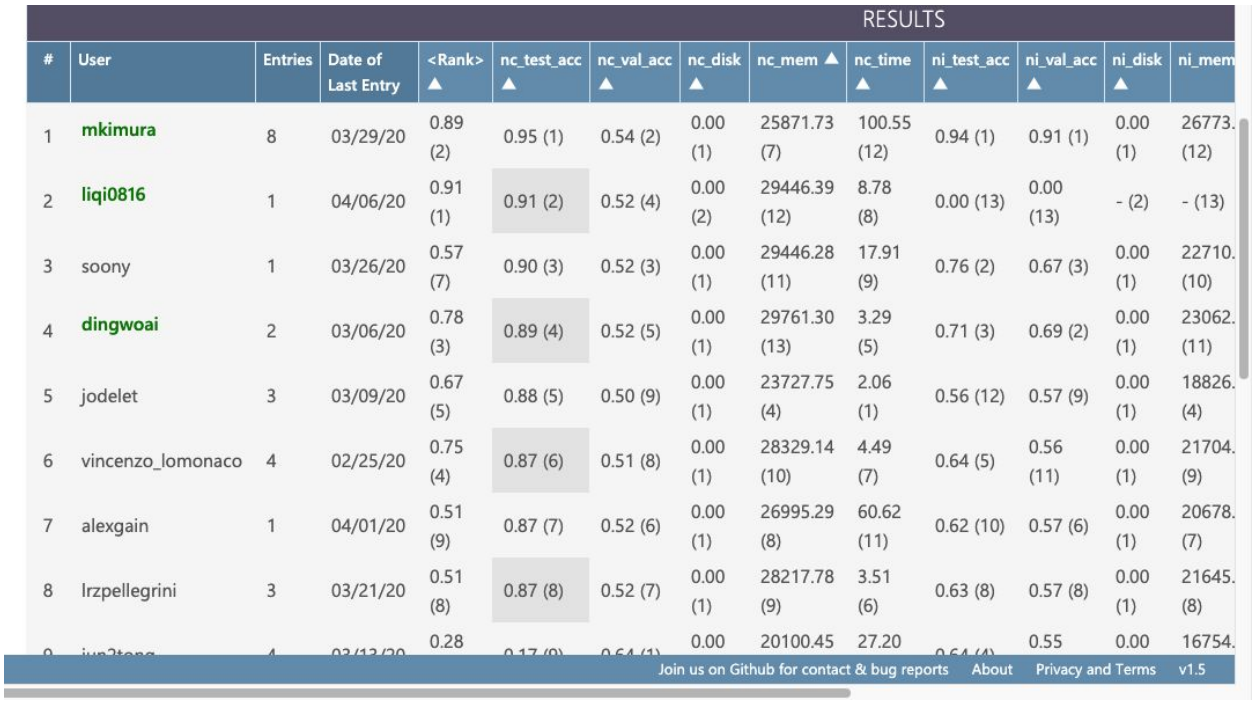
For real-life dataset, we use CoRe50 dataset and do the task for the New Class(NC) scenario.

The report includes goal of this project, brief introductions and implementation details of the selected algorithm, experiments results of the selected algorithm, conclusion and an appendix for all useful formula derivations.

All the code is available at the public github repository:

https://github.com/liqi0816/6613-proj2

We currently rank 2nd in the New Classes Task in this competition!!!! !



RESULTS

#	User	Entries	Date of Last Entry	<Rank> ▲	nc_test_acc ▲	nc_val_acc ▲	nc_disk ▲	nc_mem ▲	nc_time ▲	ni_test_acc ▲	ni_val_acc ▲	ni_disk ▲	ni_mem
1	mkimura	8	03/29/20	0.89 (2)	0.95 (1)	0.54 (2)	0.00 (1)	25871.73 (7)	100.55 (12)	0.94 (1)	0.91 (1)	0.00 (1)	26773. (12)
2	liqi0816	1	04/06/20	0.91 (1)	0.91 (2)	0.52 (4)	0.00 (2)	29446.39 (12)	8.78 (8)	0.00 (13)	0.00 (13)	- (2)	- (13)
3	soony	1	03/26/20	0.57 (7)	0.90 (3)	0.52 (3)	0.00 (1)	29446.28 (11)	17.91 (9)	0.76 (2)	0.67 (3)	0.00 (1)	22710. (10)
4	dingwoai	2	03/06/20	0.78 (3)	0.89 (4)	0.52 (5)	0.00 (1)	29761.30 (13)	3.29 (5)	0.71 (3)	0.69 (2)	0.00 (1)	23062. (11)
5	jodelet	3	03/09/20	0.67 (5)	0.88 (5)	0.50 (9)	0.00 (1)	23727.75 (4)	2.06 (1)	0.56 (12)	0.57 (9)	0.00 (1)	18826. (4)
6	vincenzo_lomonaco	4	02/25/20	0.75 (4)	0.87 (6)	0.51 (8)	0.00 (1)	28329.14 (10)	4.49 (7)	0.64 (5)	0.56 (11)	0.00 (1)	21704. (9)
7	alexgain	1	04/01/20	0.51 (9)	0.87 (7)	0.52 (6)	0.00 (1)	26995.29 (8)	60.62 (11)	0.62 (10)	0.57 (6)	0.00 (1)	20678. (7)
8	lrzpellegrini	3	03/21/20	0.51 (8)	0.87 (8)	0.52 (7)	0.00 (1)	28217.78 (9)	3.51 (6)	0.63 (8)	0.57 (8)	0.00 (1)	21645. (8)
9	jun2tong	4	03/13/20	0.28	0.17 (9)	0.64 (1)	0.00	20100.45	27.20	0.64 (4)	0.55	0.00	16754.

Join us on Github for contact & bug reports About Privacy and Terms v1.5

Algorithm we used - Elastic Weight Consolidation(EWC)

Algorithm brief summary

For project 2, we followed the work from paper: *Overcoming catastrophic forgetting in neural networks (EWC)*, as well as learnt the CL knowledge from paper: Continual Lifelong Learning with Neural Networks: A Review.

Thus, in this project we use EWC(the algorithm from the paper) for CL. We start by addressing the problem of whether elastic weight consolidation could allow deep neural networks to learn a set of complex tasks without catastrophic forgetting.

Here are 3 parts: idea of EWC, applications of EWC in supervised learning, and reinforcement learning.

Continual Learning (CL): The ability to continually learn over time by accommodating new knowledge while retaining previously learned experiences is referred to as continual or lifelong learning.

Elastic weight consolidation (EWC): an algorithm analogous to *synaptic consolidation* for artificial neural networks, which we refer to as elastic weight consolidation (EWC for short). This algorithm slows down learning on certain weights based on how important they are to multitask learning. EWC can be used in supervised learning and reinforcement learning problems to train several tasks sequentially without forgetting older ones, in marked contrast to previous deep-learning techniques.

Implementation - Baysesian perspective to new task and old tasks

In brains, synaptic consolidation enables continual learning by reducing the plasticity of synapses that are vital to previously learned tasks. We implement an algorithm that performs a similar operation in artificial neural networks by constraining important parameters to stay close to their old values.

For two sequentially happened tasks `A` and `B`. While learning task B, EWC therefore protects the performance in task A by constraining the parameters to stay in a region of low error for task A.

From a probabilistic perspective: optimizing the parameters is tantamount to finding their most probable values given some data D. We can compute this conditional probability p(θ|D) from the prior probability of the parameters p(θ) and the probability of the data p(D|θ) by using Bayes' rule:

log p(θ|D) = log p(D|θ) + log p(θ) − log p(D)

Here is applying Bayes' rule on two following tasks A & B

log p(θ|D) = log p(DB|θ) + log p(θ|DA) − log p(DB)

log p(DB|θ) is training loss for task B, we can optimize this by gradient descent.

log p(DB) is a constant, we don't need to care about it when optimizing for the overall goal.

log p(θ|DA) 's real value is intractable and will be approximated laplace approximation. The approximated value can further be the derivation regarding approximated mean value and its corresponding Hessian matrix. (You can see appendix's derivations for more mathematical details.)

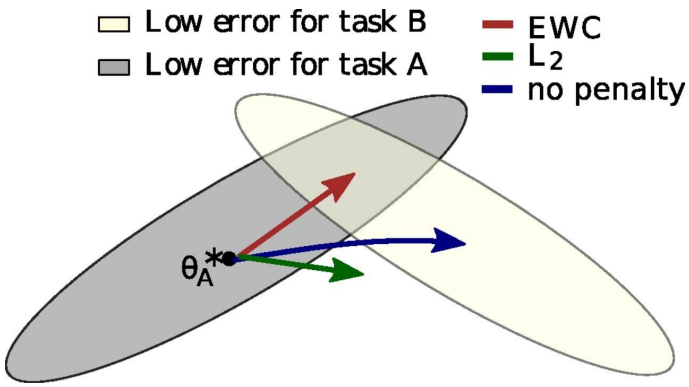
Implementation - Laplace approximation for knowledge of previous task

`p(θ|DA)`: All the information about task A must therefore have been absorbed into the posterior distribution p(θ|DA). we approximate the posterior as a Gaussian distribution with mean given by the parameters `θA*` In order to further simplify the computation, Hessian matrix is simplified into a diagonal precision given by the diagonal of the Fisher information matrix `F`.

The function L that we minimize in EWC is:

$$L(\theta) = L_B(\theta) + \sum_i^n \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

This picture shows the ideal goal of EWC:



Implementation - Train network with EWC

We trained a fully connected multilayer neural network on several supervised learning tasks in sequence. Within each task, we trained the neural network in the traditional way, namely by shuffling the data and processing it in small batches. After a fixed amount of training on each task, however, we allowed no further training on that task's dataset. Using EWC, and thus taking into account how important each weight is to task A, the network can learn task B well without forgetting task A.

At this point, total loss can now be approximated by model loss for the current task and loss regarding previous tasks. We can now use our favorite gradient descent to optimize the loss until the loss convenges into a small value.

Performance

MNIST and Fashion MNIST

We do experiments on small datasets(MNIST and Fashion MNIST). The network will face two tasks. Task1 is classifying digit numbers in MNIST, task 2 is classifying fashion items in Fashion MNIST.

I set hyperparameters learning rate = 0.01, optimizer for network is Adam, loss function for training process is cross entropy loss, epoch=14. EWC's importance of old tasks to new tasks ratio=1, which means old tasks are not very important to new tasks.

Results:
At time 1:
Network is trained for task1, testing accuracy for task1 is: 97.13%
At time 2:
Network is trained for task2, testing accuracy for task2 is: 80.15%
While we recompute performance on network after training for task2, testing accuracy for task 1 is: 34.12%
We can see that under this setting where networks do not care much about preserving knowledge for old tasks, it will perform fairly well on new task but forget old tasks catastrophically.

Then I reset EWC's importance of old tasks to new tasks ratio=10000, which means old tasks are very important to new tasks.
Results:
At time 1:
Network is trained for task1, testing accuracy for task1 is: 97..05%
At time 2:
Network is trained for task2, testing accuracy for task2 is: 78.75%
While we recompute performance on network after training for task2, testing accuracy for task 1 is: 94.90%

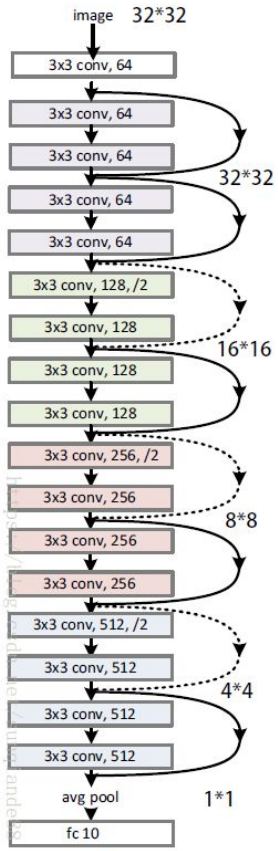
We can see that by choosing appropriate hyperparameters, neural networks can learn new tasks without totally forgetting ability for previous tasks on small datasets.

CoRe50

The CORe50 dataset is a resource demanding one and cannot fit in a regular personal computer. Therefore, we employed a google cloud compute engine with 8 vCPU, 54GB RAM and a Nvidia T4 GPU to carry out this part of the experiment. The implementation used the Torch framework, and source code can be found online[1].

We used the image preprocessing part in starter pack as-is. The preprocessing function serves to satisfy requirements assumed by torchvision pretrained models[2]. It pads mini-batches of images to at least 224x224 pixels, normalizes images with `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`. At the end it also shuffles the batch.

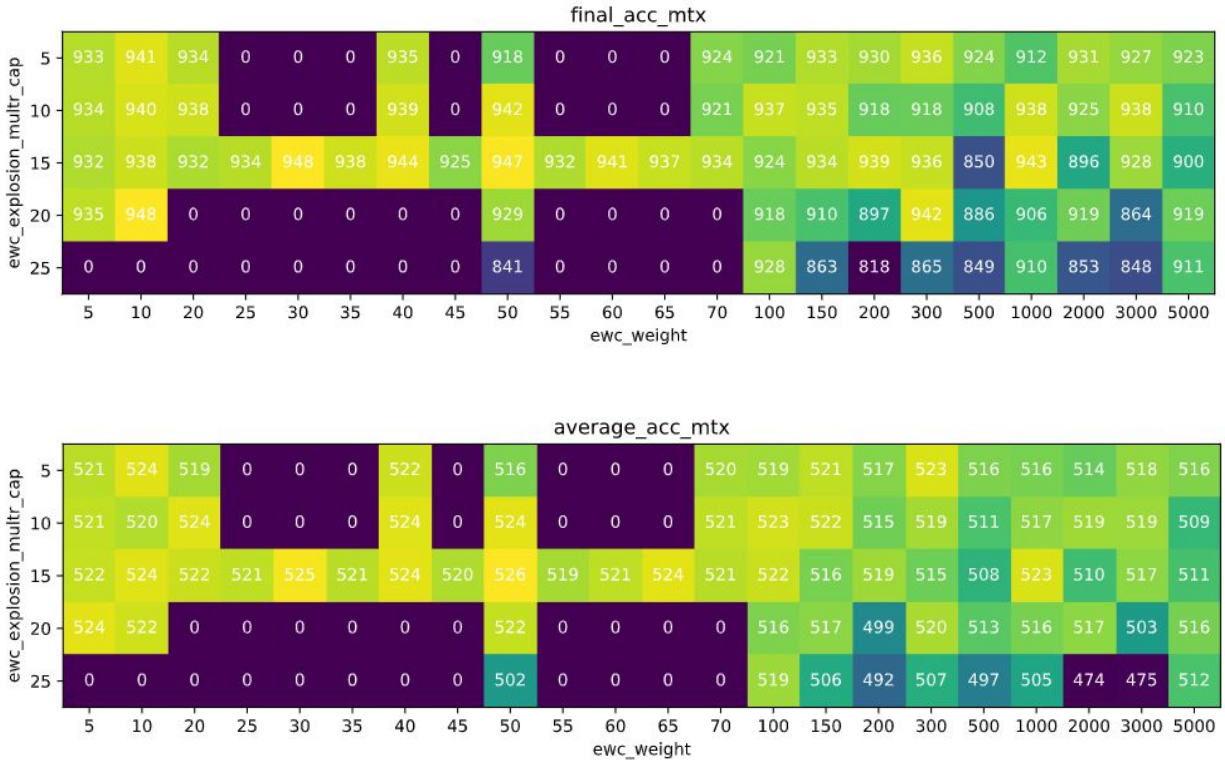
The base model is namely ResNet-18, pre-trained by pytorch. It is the most basic ResNet structure, with 17 convolutional layers and 1 fully connected layer. Experiments showed that the baseline model from CVPR competition starter pack could already achieve an accuracy of 1.0 within each subtask. For a better comparison, we kept the same base model, on the top of which added EWC mechanism. The EWC module is shared between MINST experiment and CORe50 experiment.



We trained the network with the data loader bundled in the starter pack. The loader will split 50 classes into 9 tasks/batches, and feed them into the network successively. A portion of the dataset will become validation set, and the same validation set is kept between tasks.

There are two major hyper-parameter in our model, `ewc_explosion_multr_cap`, which controls how smooth EWC loss should be, and `ewc_weight`, which controls how important EWC loss is. In experiment We frequently observe gradient explosion and loss going to `NaN`. We borrowed the idea of gradient clipping[3], setting a maximum for EWC loss. To further smoothen the loss curve, we use as the clipping function. We empirically chose `ewc_explosion_multr_cap` between 5-25 and `ewc_weight` between 5-5000 in our experiment.

The experiment result are shown as in the following two figures:



(due to limit of computation resources, some cells may contain 0 which indicates missing experiment)

As shown above, the best result is obtained with `ewc_explosion_multr_cap=15` and `ewc_weight=50`. The experiment log is attach as appendix. The final result is

```
----------------------------------------
Avg. acc: [0.9471604938271605]
----------------------------------------
Training Time: 7.673416260878245m
Average Accuracy Over Time on the Validation Set: 0.5267215363511659
Total Training/Test time: 7.673416260878245 Minutes
Average RAM Usage: 21473.329210069445 MB
Max RAM Usage: 29456.984375 MB
Experiment completed.
```

Note that while "`Avg. acc:`" is actually the final accuracy after all tasks/batches averaged *over all classes*.

The baseline result is

```
----------------------------------------
Avg. acc: [0.922716049382716]
----------------------------------------
Training Time: 7.656683707237244m
Average Accuracy Over Time on the Validation Set: 0.5145679012345679
Total Training/Test time: 7.656683707237244 Minutes
Average RAM Usage: 21424.665364583332 MB
Max RAM Usage: 29409.3671875 MB
Experiment completed.
```

which shows that our model performs 2% better after *all* tasks/batches.

Conclusion

EWC allows knowledge of previous tasks to be protected during new learning, thereby avoiding catastrophic forgetting of old abilities. It does so by selectively decreasing the plasticity of weights, and thus has parallels with neurobiological models of synaptic consolidation.

EWC has a run time which is linear in both the number of parameters and the number of training examples. Most notably by approximating the posterior distribution of the parameters on a task (i.e. the weight uncertainties) by a factorized Gaussian. With EWC, three values have to be stored for each synapse: the weight itself, its variance and its mean.

The ability to learn tasks in succession without forgetting is a core component of biological and artificial intelligence.

Thanks for the help from our professor, TAs, teammates and the paper.

Appendix 1 - all useful formula derivations

Bayesian Perspective of learning new tasks and old tasks

$$\log p(\theta|D) = \log p(D|\theta) + \log p(\theta) - \log p(D)$$

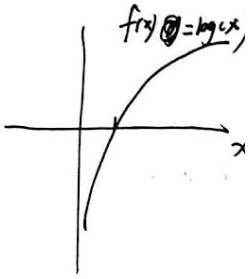
D includes D_A and D_B

$$\log p(\theta|D) = \log p(D_A|\theta) p(D_B|\theta) + \log p(\theta) - \log p(D_A) p(D_B)$$

$$\log p(\theta|D) = \log p(D_B|\theta) + \underbrace{\log p(D_A|\theta) + \log p(\theta) - \log p(D_A)}_{\log p(\theta|D_A)} - \underbrace{\log p(D_B)}_{\text{constant}}$$

$$\log p(\theta|D) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B)$$

posterior probability — negative loss of task B — task A info can't compute directly. Laplace Approximation — constant

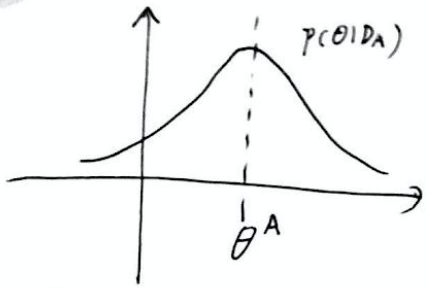


To maximize posterior probability is to minimize θ^*_A

F Fisher Information Matrix

$$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^*_{A,i})^2 \longrightarrow \text{Loss we need to minimi in EWC}$$

EWC deviation 1



$\log p(\theta|D_A)$

$$\log p(\theta|D) = \underbrace{\log p(D_B|\theta)}_{\text{negative loss of task B}} + \underbrace{\log p(\theta|D_A)}_{\text{tricky to compute But can use Laplace Approximation}} - \text{const}$$

Step 1:

To compute $\log p(\theta|D_A)$, Use Laplace Approximation

For function $h(\theta) = \log q(\theta)$

$$\approx \underbrace{h(\hat{\theta})}_{\text{const}} + \underbrace{(\theta-\hat{\theta})h'(\hat{\theta})}_{=0\ (h'(\hat{\theta})=0)} + \frac{1}{2}(\theta-\hat{\theta})^2 h''(\hat{\theta})$$

$$= \text{const} + \frac{1}{2}(\theta-\hat{\theta})^2 h''(\hat{\theta}) \quad \text{where } h'' \text{ is Hessian Matrix of } h(\theta)$$

$$\propto N(\hat{\theta}, -1/h''(\hat{\theta}))$$

Step2

Use Fisher Information Matrix to simplify computation of Hessian Matrix

Property: $F_\theta^X = -E_X[H_\theta]$

Proof: $F_\theta^X = E_X[\nabla_\theta \log P_\theta(X) \nabla \log P_\theta(X)^T]$

Hessian $H_\theta = \nabla_\theta^2 \log P_\theta(X) = \frac{\nabla^2 P_\theta(X)}{P_\theta(X)} - \frac{\nabla P_\theta(X) \nabla P_\theta(X)^T}{P_\theta(X) P_\theta(X)}$

Take Expertaction:

$$E_X[\nabla^2 \log P_\theta(X)] = \underbrace{E_X[\nabla^2 P_\theta(X)/P_\theta(X)]}_{=0} - \underbrace{E_X[\nabla_\theta \log P_\theta(X) \nabla \log P_\theta(X)^T]}_{F_\theta^X}$$

$$\Rightarrow F_\theta^X = -E_X[H_\theta]$$

EWC deviations

Step3 Use step2's conclusion to compute step1

$$\log p(\theta|D) \approx \log p(D_B|\theta) + \frac{1}{2}(\theta - \theta_A^*)^2 h''(\hat{\theta})$$

$$= \log p(D_B|\theta) - \underbrace{\frac{1}{2}(\theta - \theta_A^*)^2 F_{ii}}_{\text{scalar}}$$

$$= -L_B(\theta) - \frac{1}{2}\sum_i F_{ii}(\theta_i - \theta_{A,i}^*)^2$$

Thus the loss we want to minimize in EWC is:

$$L(\theta) = L_B(\theta) + \frac{\lambda}{2}\sum F_{ii}(\theta_i - \theta_{A,i}^*)^2$$

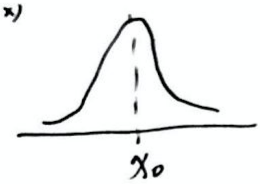
λ sets how important the old task is compared to the new one

EWC
derivations
3

Laplace Approximation

Laplace Approximation

$g(x)$



$\int g(x)^2 dx < \infty$

$\int g(x)\,dx \approx g(x_0)\,\varepsilon$ → use step function to approximate

$$\int_a^b g(x)\,dx = \int_a^b \exp(h(x))\,dx, \qquad h(x) = \log(g(x))$$

Taylor series at x_0 $\approx \int_a^b \exp\left(h(x_0) + h'(x_0)(x-x_0) + \frac{1}{2}h''(x_0)(x-x_0)^2\right)dx$

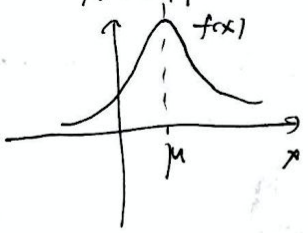
$h'(x_0) = 0$

$$= \int_a^b \exp\left(h(x_0) + \frac{1}{2}h''(x_0)(x-x_0)^2\right)dx$$

$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$ $= \exp(h(x_0)) \int_a^b \exp\left(-\frac{1}{2}\frac{(x-x_0)^2}{-h''(x_0)^{-1}}\right)dx$

($h(x_0)$: const)

$= \mathcal{N}(x \mid \mu, \sigma^2)$ $= \exp(h(x_0))\sqrt{\frac{2\pi}{-h''(x_0)}} \int_a^b \varphi(x \mid x_0, -h''(x_0)^{-1})\,dx$



$\simeq \exp(h(x_0))\sqrt{\frac{2\pi}{-h''(x_0)}}\left[\Phi(b \mid x_0, -h''(x_0)^{-1}) - \Phi(a \mid x_0, -h''(x_0)^{-1})\right]$

where φ is pdf, Φ is cdf

Laplace Approximation

Hessian Matrix