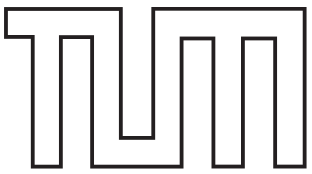


Technische Universität München
Chair of Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

Master Thesis

Uncertainty-Based Improvement of a Visual Classification System

Author:	Jianxiang Feng
Matriculation Number:	03679926
Address:	Connollystr. 03/W06 80809 Munich
Advisor:	Dr. Zoltán-Csaba Márton Maximilian Durner
Begin:	07.05.2018
End:	01.04.2019

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, April 1, 2019 _____

Place, Date Signature

München, April 1, 2019 _____

Place, Date Signature

Abstract

Classifiers based on deep neural networks have achieved tremendous success in different kinds of vision tasks. However, at the same time, this kind of classifier could not provide reliable uncertainty estimation about their predictions, which can easily lead to seriously overconfident predictions and sub-optimal decision in the following steps. From a systematic perspective, this is undesirable and even hazardous in wide range of safety-critical applications such as robotic deployment, medical diagnosis and so on. On the other hand, in object recognition task, some predictions are uncertain because they have similar appearances and thus ambiguous although their relationship with contextual objects are distinguishable. For example a marker has similar appearance with a toothbrush, when a calculator is recognized nearby, a prediction of marker should have higher probability. This kind of uncertainty from data ambiguity can be considered in order to obtain a more robust classifier.

In this work, regarding the first issue, I investigate applying dropout variational inference in Bayesian neural network to obtain reliable model uncertainty estimation on object recognition task. Predictions with low uncertainty can be used to label observed data automatically, which can decrease manual effort on training a more accurate classifier when robot is confronted with objects in real life, which has a slight domain gap with objects in training set. When it comes to the second issue, object co-occurrence statistics is introduced as scene contextual information via conditional random field to handle the data ambiguity and improve the exiting results. We have conducted experiments to show that reliable uncertainty estimates can be obtained via dropout inference and even improved with proposed variants of dropout inference and its ensemble. Additionally, those improved uncertainty estimation can be used with semantic statistic to train a more accurate domain specific classifier.

Contents

Chapter 1

Introduction

1.1 What is uncertainty?

This question seems a little philosophical and difficult to answer. However, in the context of modeling in machine learning, I find one answer reasonable and illustrative, that is "Uncertainty arises because of limitations in our ability to observe the world, limitations in our ability to model it, and possibly even because of innate nondeterminism."[KFB09]. In spite of innate nondeterminism, uncertainty can be further categorized into two main types :**epistemic uncertainty** and **aleatoric uncertainty** [DKD09][SBD+14] [KG17].

The former one, epistemic or model uncertainty illustrates the uncertainty in modeling, which is associated with **imperfect models** of the real world because of insufficient or imperfect knowledge of reality, which corresponds to our ability of modeling, and thus refers to uncertainty caused by a lack of knowledge, i.e., it refers to the epistemic state of the decision maker. Therefore it may be reduced through collecting more knowledge. Examples for this kind of uncertainty can be uncertainty of model parameter, or uncertainty of model structure depending on which kind of model and so on. One simple example to illustrate this kind of uncertainty would be linear regression with limited observed points, where we can use different curves to explain those points and then predict unobserved points with those curves. If we have observed enough points in some intervals, the models or curves that are able to explain the data are more restricted and if there are not enough points, there would be more curves that can explain these points, which indicates high uncertainty. In reality, it's quite normal to have inadequate data, and thus difficult to model the underlying distribution quite well and precisely. Therefore we could explain the data with several or many different models, where the model uncertainty comes in.

The latter one, aleatoric or data uncertainty, accounts for the **inherent randomness** of underlying phenomenon which is expressed as variability in observed data and meanwhile associated with the limitation of our observing ability. It's treated as non-reducible within

our ability to observe the data. Examples for this kind of uncertainty could be noise induced by limited resolution of sensor or ambiguity from content of data which are unrelated to the model we use to represent these data. One example can be classification of images captured by cameras with different resolutions, images with lower resolutions would have higher uncertainty because they are more ambiguous and less illustrative compared with those with higher resolutions.

In other words, epistemic uncertainty refers to the reducible part of the (total) uncertainty, whereas aleatoric uncertainty refers to the non-reducible part. Model uncertainty and aleatoric uncertainty are combined and expressed in the final prediction, which is so called predictive uncertainty.

1.2 Why do we need uncertainty?

Aforementioned model uncertainty represents the belief of model about its predictions and thus bring us more useful information about predictions related with model and observed data instead of just a crispy prediction, whose predictor, is always overconfident when advanced neural network architectures are employed[GPSW17]. As far as I am concerned, this kind of uncertainty plays an important role in two following scenarios.

The first one are those requiring **high safety guarantee** or **optimal decision-making**, in which false predictions can cause hazardous consequences, which is so called AI safety[AOS+16]. With reliable model uncertainty estimation, we are able to know when the model is uncertain about its prediction and then people or operator are aware of that and can take corresponding counter measures in order to avoid unnecessary accidents. With more and more wide-spread deep learning algorithms in real life applications, concrete examples for this case include steering control in self-driving car[MGK+17], disease diagnosis in medical domain[LAA+17], or even tasks in aerospace or other domains requiring higher precision and stronger robustness and so on. Although someone would argue that if the trained model can perform perfectly, that means that it is able to achieve 100% accuracy, then we do not need this kind of uncertainty information. However, as mentioned in [DL91], it's hard or even impossible to have enough training data to define a precise model, which is in coincidence with reality because it's hard to define a task very precisely and get access to enough samples drawn from its real distribution. Uncertainty information would be more valuable and important when algorithms are confronted with multiple tasks[KGC]. Other relevant tasks such as misclassified and out-of-distribution data detection[HG16] and adversarial attacks [KGB16][FCSG17] have also attracted more and more research interests in uncertainty estimation in order to address these challenges.

The second one are scenarios requiring **interaction between algorithms and people or environments**. Model uncertainty can build a bridge between machine learning algorithms and human beings which can construct a more robust and more data-efficient

machine learning system. This scenario involves different kinds of tasks such as active learning[GIG17], reinforcement learning[BCKW15][OBPVR16][GMR], automatically labeling, industrial components inspection and so on. The idea behind that is, with reliable model uncertainty estimation we know which data sample or prediction the model is unfamiliar with. In active learning or reinforcement learning, those unfamiliar data samples can be used to train our model more quickly and efficiently. In tasks involving human robot interaction, we can know when the model requires help from human experts based on uncertainty estimation. For example, in industrial component inspection task, we firstly train a model which can perform quite well on some evaluation datasets and then deploy them in real applications. At this moment, we assume that all predictions made by the model are true all the time. However, there is no guarantee that this model has learned to represent the real distribution of data perfectly and is robust against slight or even large domain transfer. If the distributions of the real world data vary because of unexpected factors such equipments aging or changes of weather condition. The model would fail silently and lead to unexpected accidents. With reliable model uncertainty estimation, this kind of weakness can be mitigated or even eliminated.

On the other hand, aleatoric uncertainty mainly assist in **improving model performance** by quantifying and considering this kind of information by representing noisy level of predictions and making good use of data points with less noise. By taking this information into account, the model can be trained more optimally and yield better performance, some examples of image data and Lidar data can be referred to [KC16][FRD18]. Instead of representing noise, data uncertainty also includes uncertainty from semantic view point, which we can also incorporate into training and yield more robust model. This kind of uncertainty is not quantified explicitly but can be incorporated conceptually with help of specifically designed model and high level statistics cues or features to **disambiguate prediction** directly. One example of this can be modeling contextual information among pixels with conditional random field in semantic segmentation task[KK11][SM$^+$12][LSVDHR16]. The idea behind that is simple and conditional random field provides a principle way to achieve this idea. Since uncertainty caused by appearance ambiguity represents strong correlation between different categories. Conditional random field can model correlation or dependency between random variables by taking into account high order potentials. In semantic segmentation task, because of the reason of tractable computation, at most second order potentials are taken into account and can yield better performance compared results with just first order potentials.

1.3 How can we obtain and handle uncertainty in deep learning?

As we know, deep learning is a powerful black box model while it has achieved tremendous success in different tasks. Therefore to obtain and handle uncertainty in deep learning model and data can help us to get more understanding about this black box model and make deep learning based application more robust and safer.

Regarding model uncertainty or quality measure that acts similar to model uncertainty, there are many different ways to obtain this kind of quality measure of prediction. On the one hand, because of intractability of real model uncertainty in deep neural networks, there are some **ad-hoc approaches** that try to obtain such a quality measure on specific tasks which are solved usually with model uncertainty such as misclassified and out-of-distribution(OOD) detection task. [LLS17] combines temperature scaling and input preprocessing which is called adversarial training together and yields a simple but effective out-of-distribution sample detector, but this approach requires additional out-of-distribution dataset for training which is hard to collect in reality. [DT18] modifies the loss function to train a classifier against OOD data, however this approach is hard to generalize in case of slight domain transfer. [LLLS17] treats uniform distribution as ground truth distribution of OOD data and show that OOD data can be generated via generative adversarial network(GAN). This approach seems practical, but training GAN effectively does not always work for images of large size because of high dimension of data distribution.

On the other hand, there are some more theoretically sound approaches which employ **Bayesian neural network**[Mac92][Nea12], **bootstrap**[OBPVR16] or **ensemble** method[LPB17] or even ensemble of Bayesian neural network[SG18]. Although Bayesian neural network provides a principal way to obtain model uncertainty by putting randomness on model parameters, it's difficult to infer the exact posterior distribution over model parameters of deep neural network because of its complex architecture and large scale dataset. Therefore approximate inference plays an important role in addressing this issue. The golden standard for approximate inference is Hamiltonian Monte Carlo[Nea12], which requires dealing with whole batch training data and storing the samples of posterior distribution. While the former issue is addressed partially by stochastic gradient Langevin Dynamics[WT11], the latter one attracts many research interests like [BRMW15][WVL$^+$18]. However, since it's a Markov Chain Monte Carlo(MCMC) method, to assess the convergence is still non-trivial.

There are some alternatives to MCMC, one popular one is variational inference(VI)[HVC93]. In VI, this inference problem is cast into optimization problem by minimizing the Kullbach-Leibler divergence between the approximate posterior distribution and real posterior distribution. Therefore approximate distribution family needs to be chosen and thus has a large impact on the result, which may impede the performance if the chosen distribution family is not flexible enough for the problem. [Gra11]

firstly employed fully factorized Gaussian with a biased gradient estimator on a practical problem. Later [BCKW15] improved the result with the same approximate distribution but different estimator with help of "reparameterization trick"[KW13]. HLA15 also used the same approximate distribution but with expectation propagation[Min01] instead of variational inference. Since the fully factorization assumption may impose an restriction on the flexibility of the approximate distribution between the correlations between weights could not be learned and expressed. By taking this into account, there are many attempts trying to use more expressive approximate distribution such as Bernoulli (treated as mixture of two Gaussian with small variance)[GG16] and Gaussian [KSW15] which capture variances of rows in weight matrix. Another more expressive one is matrix variate Gaussian distribution[LW16][SCC17][ZSDG17], which capture both rows and columns correlation in weight matrix. Additionally inspired by the idea of normalizing flow in latent variable models [LW17]applied normalizing flows to auxiliary latent variables to produce more flexible approximate posteriors.

Another alternative is Laplace approximation[Mac92], which put a Gaussian centered at MAP estimate of model parameters with a covariance determined by the inverse of Hessian of the log-likelihood. The main issue of this approach for the large neural network is the large size of Hessian which is quadratic of the number of parameters and inversion operation on it whose complexity is cubic over number of elements of Hessian matrix. Recently [RBB18] working out this problem by approximating the Hessian or the Fisher matrix of log-likelihood with kronecked-factorization which reduces the storage size and complexity of inversion operation. The resulting posterior can also be treated as matrix variate Gaussian.

However, most of those ideas are verified with a relatively simple architecture and on toy or small-scale datasets and need to be modified a lot in order to serve the existing advanced architectures, which is undesirable in our work and many practical applications. Therefore dropout inference[GG16] looks promising considering this restriction and thus is adopted to obtain model uncertainty in our work. On the other hand, scalable Laplace approximation [RBB18] has the potential resolving this issue, because it requires only one MAP point estimate of model parameters, so there is no need to modify the training phase which could save a lot of computation effort and unnecessary modifications for many existing architectures.

When it comes to aleatoric uncertainty, as noise of data itself, it can be modeled by adding another head on top of the network [KG17]. However, the uncertainty caused by appearance ambiguity is not trivial to include in deep neural network training. Therefore another model, **conditional random field**[LMP01], is adopted based on the predictions of discriminative neural network classifier. As is mentioned in previous section, choice of features of second order potentials is key factor in this problem. In this work, because we work on scene object recognition task, therefore the correlation between objects in specific scene are of importance. The correlation can be represented by co-occurrence matrix co-occurrence[LRKT10][RV09][GRB08][RVG+07].

1.4 Contributions and Structure

This work can be divided into two main parts, both focusing on uncertainty-based improvement for a deep learning based classifier on object recognition task.

The first part is to employ dropout inference to obtain reliable and calibrated uncertainty from classifier with ResNet50[HZRS16] as backbone. We evaluate the prediction and calibration performance of this probabilistic classifier and then based on uncertainty estimation, we select predictions with low uncertainty/high confidence as automatically labeled data which is used for training a more accurate classifier while the performance of original one declines a lot in case of slight domain transfer which is caused by possibly by light condition changes, appearance changes and even recording equipment changes. This is proof-of-concept idea which tries to proof that improved uncertainty estimation can assist in automatic labeling and reducing manual effort in fine-tuning a new classifier in case of slight domain transfer during deployment of robot. In this part, we consider advanced version of dropout inference and its variants for improving the results both in terms of accuracy and calibration.

The second part is to handle the uncertainty caused by appearance ambiguity. To note that again, this kind of uncertainty is not modeled directly, but the concept is taken into account in choosing model and features. To achieve this goal, we choose conditional random field to model the correlation between objects with similar appearance but different semantics. As key factor of the model, we employ co-occurrence statics of categories as features of second order potentials.

There are several main contributions of this work:

- to show that dropout inference and Laplace approximation can obtain better uncertainty information on multi-class classification problem with 51 classes, which can be further improved by learning dropout rates during training and its ensemble.
- to show that manual effort in data collection in slight domain transfer learning task can be reduced by making use of reliable uncertainty estimation.
- to show that uncertainty caused by appearance ambiguity can be resolved by integrating contextual information via condition random field.

The structure of this work is as follows, in order to give reader a clear theoretical background of models employed in this work, I will briefly review the basic theory and ideas of Bayesian neural network as well as dropout inference and Laplace approximation in chapter 2, where detailed adaptations and modifications of original model are described. In chapter 3, similar to previous chapter, the theory of conditional random field is reviewed firstly and the adaptations are described. In chapter 4, different experiments and results about evaluation of different variants of dropout inference are given and discussed firstly. In the following, experiments and results on scene recognition task resolved by conditional random field with co-occurrence statics as second order features are also showed and discussed.

Then conclusion and possible future work come in the chapter 5.

Chapter 2

Bayesian neural network

As is mentioned in the introduction chapter, we investigate obtaining model uncertainty via Bayesian neural network. Therefore in this chapter, a brief review on theory of Bayesian neural network is given. Since capturing the exact posterior distribution over the parameters of neural network is intractable, we need to have techniques to perform approximate inference. Those techniques could be categorized into two main types, variational inference(VI) and Markov Chain Monte Carlo(MCMC). Brief history and basic idea of these two types of method are introduced. However, since larger dataset and more complex neural network are applied nowadays, traditional approximate inference techniques are difficult to scale to large dataset and complex advanced architectures. Therefore modern approximate inference needs to adopt to obtain model uncertainty. We will introduce and evaluate two modern variational inference methods for neural network, which are dropout inference[GG16] and scalable Laplace approximation[RBB18]. Additionally, we investigate improving uncertainty estimation by learning dropout rate from data via concrete dropout [GHK17], which is introduced following section of dropout inference. Finally, with help of concrete dropout, a new variants of dropout inference is introduced here, which has more flexible approximation distribution family.

2.1 Introduction

Let $\mathcal{D}$ denote an observable dataset consisting of a set of input and output pairs, that is $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\} = \{(x_i, y_i)_{i=1}^N\}$, where $x_i \in \mathbb{R}^D$ and $y_i \in \mathbb{Y}$, $\mathbb{Y}$ is set of labels, and $f^{\boldsymbol{\omega}}$ denote one parametric model which is in our case neural network with $\boldsymbol{\omega}$ its parameters which are weights $W_{1:L}$ and biases $b_{1:L}$ for L layers.

In supervised learning, our goal is to learn a probabilistic model of conditional distribution of output given input, $p(\mathbf{y}|\mathbf{x})$ that can explain the underlying data distribution instead of just the observables $\mathcal{D}$ very well based on the observables $\mathcal{D}$ which are samples drawn from the underlying data distribution. Then this learned model can be used to make predictions on unobservable data samples under the same data distribution. In classification where output is label represented by discrete integer, the likelihood function is defined based on parametric model, which is softmax score:

$$p(y = d|\mathbf{x}, \boldsymbol{\omega}) = \frac{exp(f_d^{\boldsymbol{\omega}})}{\sum_{d'} exp(f_{d'}^{\boldsymbol{\omega}})} \tag{2.1}$$

In regression case, the likelihood is Gaussian:

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega}) = \mathcal{N}(\mathbf{x}; f^{\boldsymbol{\omega}}(\mathbf{x}), \tau^{-1}\mathbf{I}) \tag{2.2}$$

with model precision τ, which represents the inverse of noise level of the outputs.

As mentioned above, learning means to find model(s) which is parameterized by a set of parameters that can explain the data well, which means the likelihood should be maximized w.r.t. model parameters over the observables. On the other hand, some prior constraints are imposed on the model via prior distribution of the model parameters $p(\boldsymbol{\omega})$. The probability distribution of model parameters is updated from prior distribution into posterior distribution after observing training dataset $\mathcal{D}$ via Bayes' theorem:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})} \tag{2.3}$$

where $p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})d\boldsymbol{\omega}$ is so called model evidence or marginal likelihood, whose integration is always intractable.

After obtaining posterior distribution over model parameters, we can make predictions by marginalizing the likelihood of unseen input points such as $x^\star$ over model parameters, which leads to predictive distribution over output:

$$p(y^\star|x^\star, \mathcal{D}) = \int p(y^\star|x^\star, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega} \tag{2.4}$$

To point out the difference between normal deterministic neural network and Bayesian neural network can help understanding the mechanism of Bayesian neural network better. In figure 2.1, we use graphical model to express these two kinds of neural network, in which solid point denotes deterministic variable, and circle denotes random variable, while shaded circle denote observed random variable which represents training set. Plate notation denotes N observed data pairs. As we can see in the figure, parameters $\boldsymbol{\omega}$ in deterministic is normal variable which is a point estimate done by Maximum a posterior method:

$$\boldsymbol{\omega}^{\star} = argmax_{\boldsymbol{\omega}}\{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})\} \tag{2.5}$$

On the other hand, in Bayesian neural network, the model parameter $\boldsymbol{\omega}$ is random variable which obey the distribution parameterized by θ. To note that for explanation of concept here, we do not make any assumption on the function parameterized by θ, which means it can be arbitrary function. This distribution over model parameters can be inferred based on Bayes' rule in equation 2.3. However, for model with large number of parameters, it's hard to calculate the integral tractably. Therefore many approximation methods mentioned in the introduction chapter are applied to solve this problem. It's worth to note that if we choose the approximate distribution as delta function, $q_{\theta}(\boldsymbol{\omega}) = \delta(\boldsymbol{\omega} - \theta)$, then we can recover Bayesian neural network into deterministic neural network. Because the computation of model evidence $p(\mathbf{Y}|\mathbf{X})$ is always intractable in complex model and large dataset, we need to resort to approximate inference, which will briefly be introduced in next section.

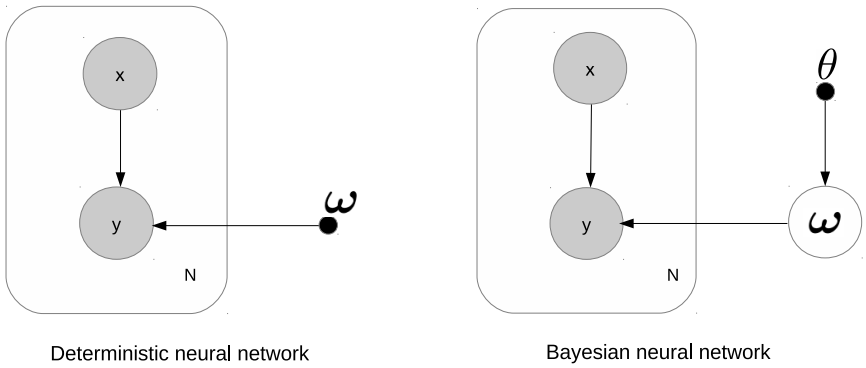


Figure 2.1: Difference between parameter estimation of deterministic neural network and Bayesian neural network.

2.2 Variational inference

2.2.1 Introduction

As is mentioned above, variational inference cast inference into optimization by minimizing the Kullbach-Leibler divergence between approximate posterior distribution and the real posterior distribution. However, there is no analytical definition of this KL divergence between the real posterior distribution is unkown. We can derive a lower bound which is also called evidence lower bound($ELBO$) which bounds the log marginal likelihood with Jensen's inequality. And from that we know marginal likelihood is the sum of $ELBO$ and KL divergence between approximate posterior and real posterior. The derivation is given in the following:

$$\begin{aligned}
\log(p(\boldsymbol{Y}|\boldsymbol{X})) &= \log(\int p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})d\boldsymbol{\omega}) \\
&= \log(\int q_\theta(\boldsymbol{\omega})\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})}d\boldsymbol{\omega}) \\
&\geq \int q_\theta(\boldsymbol{\omega})\log(\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})})d\boldsymbol{\omega} \\
&= \mathbb{E}_{q_\theta(\boldsymbol{\omega})}[\log(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega}))] - KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))) \\
&= ELBO
\end{aligned} \tag{2.6}$$

where $p(\boldsymbol{Y}|\boldsymbol{X})$ is the likelihood, $p(\boldsymbol{\omega})$ is the prior distribution over model parameters, $q_\theta(\boldsymbol{\omega})$ is the approximate posterior distribution over parameters which is parameterized by θ.

We can get $\log(p(\boldsymbol{Y})|\boldsymbol{X})$ by adding $ELBO$ and KL divergence between approximate posterior $q_\theta(\boldsymbol{\omega})$ and real posterior $p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})$:

$$\begin{aligned}
&ELBO + KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})) \\
&= \int q_\theta(\boldsymbol{\omega})\log(\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})})d\boldsymbol{\omega} + \int q_\theta(\boldsymbol{\omega})\log(\frac{q_\theta(\boldsymbol{\omega})}{p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})})d\boldsymbol{\omega} \\
&= \int q_\theta(\boldsymbol{\omega})\log(p(\boldsymbol{Y}|\boldsymbol{X}))d\boldsymbol{\omega} \\
&= log(p(\boldsymbol{Y}|\boldsymbol{X}))
\end{aligned} \tag{2.7}$$

When we maximize the $ELBO$ w.r.t the parameters of approximate posterior θ, it's equivalent to minimizing the KL divergence because the log marginal likelihood is not a function of θ. Then we have a well-defined objective which is the $ELBO$, in which the first term is called expected log likelihood which ensures the model can explain the data well and the second term is called regularization term which ensures the approximate posterior does not deviate too far from the prior distribution. Now we have cast an inference problem into an optimization problem:

$$\theta^{\star} = argmin_{\theta}[KL(q_{\theta}(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y}))] \tag{2.8}$$

which is equivalent to

$$\theta^{\star} = argmax_{\theta}[ELBO] \tag{2.9}$$

However, there are still some difficulties if we want to solve this optimization with this objective. The first one is to deal with large size dataset, which induces large computation in the expected log likelihood term. [Gra11] shows that this can solved by data subsampling which is also called stochastic optimization. Another one is that we need to obtain the derivatives of $ELBO$ w.r.t. approximate parameter θ. Since the model parameters are samples from approximate distribution, good estimator for the derivatives is required which will be introduced in next subsection.

2.2.2 Dropout variational inference

Dropout

Dropout[SHK+14] is originally introduced as regularization approach in training deep neural network which can improve the generalization performance. Although the author said this idea is inspired from human beings sexual reproduction, there are different interpretations trying to explain why it can work such as ensemble perspective and Bayesian perspective. In this subsection, the Bayesian interpretation of dropout is introduced and used for improving the uncertainty estimation of neural network.

The mechanism of dropout is simple, each units of specific layer is multiplied by a random variable under Bernoulli distribution with $1 - p$ as its parameter, where p is dropout rate. In each iteration of training, dropout is turned on, which means each unit is multiplied by the sample drawn from Bernoulli distribution in forward propagation which is kept in derivatives back propagtion during current iteration. In testing, the sampling phase is turned off, only one forward propagation is needed to obtain predictions. Normally, in order to avoid rescaling weights in testing, which is used to keep the output magnitudes in the same scale when dropout is off, rescaling of the output of dropout is always done in training. In figure 2.2, there are two figures about that dropout is on and off.

Bayesian interpretation of dropout

As is mentioned in the last subsection, in variational inference, we want to minimize the KL divergence between approximate distribution and the real posterior distribution over the model parameters, which turns out to be maximization of evidence lower bound($ELBO$).

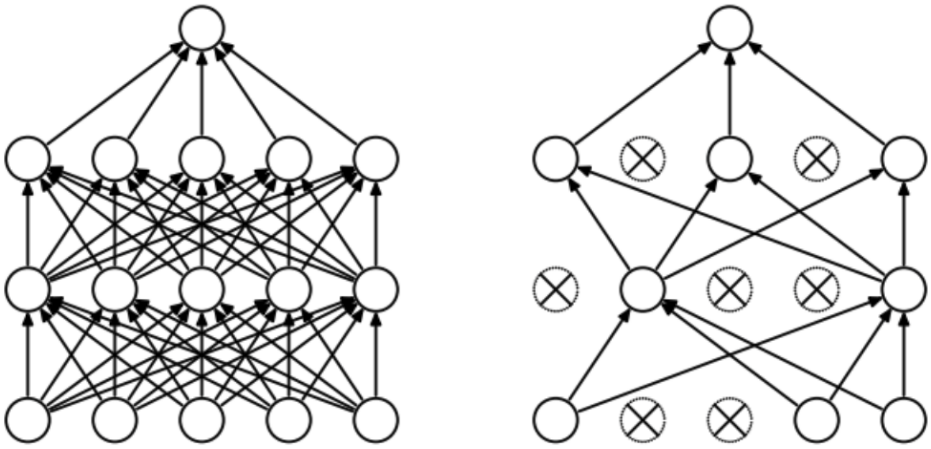


Figure 2.2: How dropout works[SHK+14].

When dropout is interpreted in Bayesian way, the distribution over hidden units is reformulated as distribution over weight matrices. The training objectives of neural network with dropout is proved to be similar as the *ELBO* of Bayesian neural network with Bernoulli distribution factorized over the input dimension of weight matrix. In the following, we will explain this interpretation including key factors such as **approximate distribution**, **training objective**, **marginalization in testing** by using one simple example in classification case in 2.3, in which we define the hidden layer as the first layer and output layer as the second layer and assume that we use L2 regularization and put a prior distribution of fully factorized Gaussian over weights initially, which can be generalized to multi-layer neural network easily.

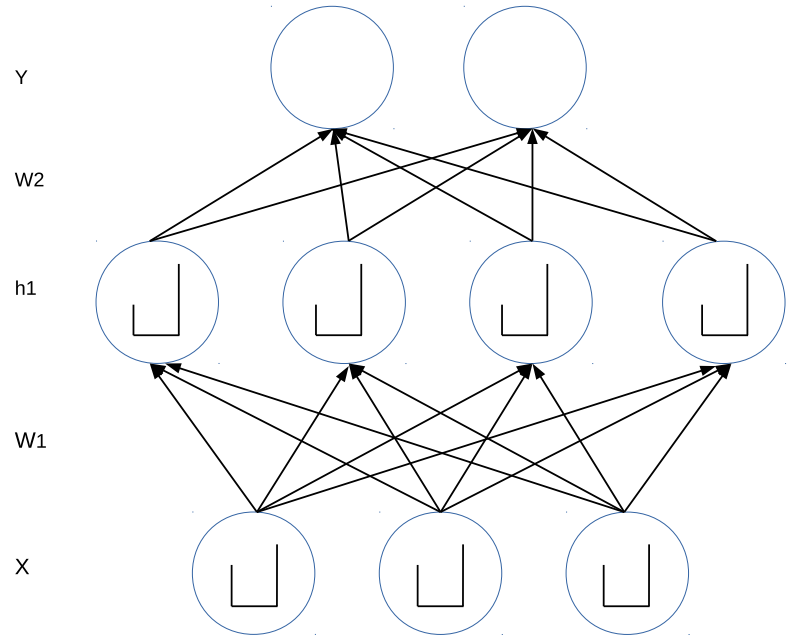


Figure 2.3: A two layer neural network example of dropout inference, a Bernoulli random variable is imposed on each unit of input layer and hidden layer.

Approximate distribution Let's denote $\boldsymbol{y} \in \mathbb{R}^{m \times D_2}$ as output, $\boldsymbol{x} \in \mathbb{R}^{m \times D_0}$ as input, $\boldsymbol{h}_1 \in \mathbb{R}^{D_1}$ as the response of hidden layer, where m represents number of data instances, D_i is dimensionality of i-th layer, where 0-th layer represents input layer and $i \in \{1, .., L\}$, $L = 2$ in this example. Further we define $\boldsymbol{\omega} = \{(\boldsymbol{W}_i)_{i=1}^L\}$ as model parameters, and $\boldsymbol{\epsilon}_i \in \mathbb{R}^{D_{i-1}}$ as the Bernoulli distributed random vector parameterized by $\boldsymbol{p}_i \in \mathbb{R}^{D_{i-1}}$, for i-th layer. In normal dropout, elements in vector $\boldsymbol{p}_i$ have same values p_i, which means that there is one keep rate or (1−dropout rate) for each layer. In the following, we will use $\boldsymbol{p}_i$ and p_i interchangeably depending on the context if there is no special specification. To note that since weight matrix is treated as random variable, we use $\boldsymbol{M}_i \in \mathbb{R}^{D_{i-1} \times D_i}$ to denote position of non-zero element in Bernoulli distribution for $\boldsymbol{W}_i$. To note that bias $\boldsymbol{b}_i \in \mathbb{R}^{D_i}$ is absorbed into W_i by appending a new row at the end of weight matrix and 1 at the end of each data input vector, which is also called homogeneous coordinate. We also assume that approximate weight distribution is factorized over layer, which yields

$$q_\theta(\boldsymbol{\omega}) = \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i).$$

To start with the formulation of dropout, we model likelihood of output conditioned on

input with softmax scores of neural network in classification case:

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\omega}) &= \sigma((\boldsymbol{h}_1 \odot \boldsymbol{\epsilon}_2)\boldsymbol{M}_2) \\ &= \sigma(\boldsymbol{h}_1(diag(\boldsymbol{\epsilon}_2)\boldsymbol{M}_2)) \\ &= \sigma(\boldsymbol{h}_1\boldsymbol{W}_2) \\ &= \sigma(\phi(\boldsymbol{x}(diag(\boldsymbol{\epsilon}_1)\boldsymbol{M}_1) + \boldsymbol{b}_1)\boldsymbol{W}_2) \\ &= \sigma(\phi(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2) \end{aligned} \tag{2.10}$$

where $\odot$ is Hadamard product(element-wise product), $\sigma(a_j) = \frac{exp(a_j)}{\sum_k exp(a^k)}$ is softmax function, $\phi(\cdot)$ is element-wise non-linear activation function such as rectified unit function.

From the equation above, we have

$$\begin{aligned} &\boldsymbol{W}_i = g(\boldsymbol{M}_i, \boldsymbol{\epsilon}_i) = diag(\boldsymbol{\epsilon}_i)\boldsymbol{M}_i \\ &\text{with } \boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon}_i) = Bernoulli(\boldsymbol{p}_i) \end{aligned}$$

which means weight matrix $\boldsymbol{W}_i$ is a random variable whose probability density function is parameterized by $\boldsymbol{p}_i$ and $\boldsymbol{M}_i$, which are denoted by $\theta = \{(\boldsymbol{M}_i, \boldsymbol{p}_i)_{i=1}^L\}$ in equation 2.8, where $i = 1, .., L$ denotes i-th layer of the network, and $L = 2$ in this example. The expression of approximate posterior distribution is not obvious, but we can define the its form as

$$\begin{aligned} q_\theta(\boldsymbol{\omega}) &= \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i) \\ &= \prod_{i=1}^{L} \int q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}_i)p(\boldsymbol{\epsilon}_i)d\boldsymbol{\epsilon}_i \end{aligned} \tag{2.11}$$

with

$$\begin{aligned} q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}) &= \delta(\boldsymbol{W}_i - g(\boldsymbol{M}_i, \boldsymbol{\epsilon}_i)) \\ &= \delta(\boldsymbol{W}_i - diag(\boldsymbol{\epsilon}_i)\boldsymbol{M}_i) \end{aligned} \tag{2.12}$$

As we can see, the approximate posterior distribution over the parameter matrix puts a same Bernoulli distribution over the input dimension of parameter matrix, which is the row dimension in this example. Meanwhile each element of the same row is multiplied with same realization of the random variable but with different non-zero position, which is corresponding to different expectation of each row element and thus induces correlations between row elements. To make this definition more clear, based on the computation of expectation and variance of Bernoulli distribution, we can write down the first and second moment of the approximate distributed random variables in the following:

$$\mathbb{E}_{q_\theta}(\boldsymbol{W}_i) = \boldsymbol{M}_i \odot \boldsymbol{P}_i \tag{2.13}$$

where $\boldsymbol{P}_i = [\boldsymbol{p}_i, ..., \boldsymbol{p}_i] \in \mathbb{R}^{D_{i-1} \times D_i}$.

Covariance matrix of parameter matrix is:

$$\big[Cov_{q_\theta}(vec(\boldsymbol{W}_i))\big]_{jk} = \mathbb{1}\big[l = m\big] m_{lq}^i * m_{mn}^i * p_i * (1 - p_i) \tag{2.14}$$

where

$$j = l * D_{i-1} + q$$

,

$$k = m * D_{i-1} + n$$

, which is the linear mapping between element index before and after matrix vectorization. m_{lq}^i is the element of l-th row and q-th column in matrix $\boldsymbol{M}_i$, which applies to m_{mn}^i as well. $\mathbb{1}\big[i = j\big]$ denotes indicator function, which is equal to 1 only when i is equal to j and otherwise 0. Because $vec(\cdot)$ operation converts matrix $\boldsymbol{W}_i \in \mathbb{R}^{D_{i-1} \times D_i}$ into column vector $vec(\boldsymbol{W}_i) \in \mathbb{R}^{D_{i-1}D_i \times 1}$ by stacking the columns of matrix on top of one another. Then it's easy to see that covariance matrix $Cov_{q_\theta}(vec(\boldsymbol{W}_i)) \in \mathbb{R}^{D_{i-1}D_i \times D_{i-1}D_i}$.

From equation 2.14, we can see that the entire covariance matrix is consisting of $D_i * D_i$ diagonal sub-matrices whose dimensionality are $D_{i-1} \times D_{i-1}$. When observing covariance matrix of parameter matrix $\boldsymbol{W}_i$ w.r.t. approximate posterior distribution $q_\theta(\boldsymbol{W}_i)$, we know that samples of row vectors in weight matrix are drawn independently and thus covariance between rows are zero. On the other hand, samples for different weights in the same row are drawn at the same time because they are multiplied by the same realization of ϵ_i, from which covariances between weights within the same row are induced. Therefore, by fitting this approximate distribution to the real posterior distribution weights within the same row can be learned. This means that the approximate distribution family have more flexibility to approximate the real posterior distribution when compared with other common approximate posterior distribution family that assumes distribution for each parameter is independent such as fully factorized Gaussian.

Training objective Up to now, we have only analyzed the approximate distribution of dropout inference. As is mentioned in introduction section of this chapter, we know that we perform optimization of $ELBO$ w.r.t. the approximate distribution parameters, in order to obtain a good approximation to the true posterior distribution over parameters, which we can use in testing to obtain more reliable uncertainty estimation. In the following, we will show that training a neural network with dropout is equivalent to optimizing the $ELBO$ w.r.t. approximate distribution parameters.

At first, let's define the training objective of neural network with dropout, which is cross entropy between predictive distribution and target distribution plus L2 regularization:

$$L_{dropout} = \sum_{i=1}^{N} \big[- log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega})) \big] + \lambda(\sum_{i=1}^{L} ||\boldsymbol{W}_i||^2) \tag{2.15}$$

where N represents the size of entire dataset, λ is L2 regularization coefficient. We want to maximize the likelihood over the entire dataset w.r.t. the model parameter $\boldsymbol{\omega}$, which is known as maximum likelihood estimation. Equipped with L2 regularization and Gaussian prior over parameters, we can obtain a max-a-posterior estimation by minimizing equation 2.15.

Normally we use gradient descent to tune our parameters in training, which requires first derivative of objective w.r.t. model parameters $\boldsymbol{\omega}$. Since nowadays large size of dataset is ubiquitous, which means N in equation 2.15 is too large, we could not obtain exact gradient of entire batch with efficient computation. Therefore we use data of mini-batch to estimate gradient, which is so called stochastic gradient descent(SGD). Apart from making computation tractable, noise of gradient estimation in each mini-batch is helpful for optimization procedure to escape the poor local minimum. The expression of gradients required in each iteration of SGD is given in the following:

$$\frac{\partial L_{dropout}}{\partial \boldsymbol{\omega}} = \frac{N}{K} \sum_{i \in S} \big[- \frac{\partial log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega}))}{\partial \boldsymbol{\omega}} \big] + \lambda \frac{\partial (\sum_{i=1}^{L} ||\boldsymbol{W}_i||^2)}{\partial \boldsymbol{\omega}} \tag{2.16}$$

where S is one random subset of entire dataset, and $|S| = K$.

On the other hand, let's have a look at $ELBO$ in equation 2.6, if we want to maximize $ELBO$, which is equivalent to minimize negative $ELBO$ w.r.t. the approximate distribution parameters θ, with SGD. The gradients are computed with the following expression:

$$\frac{\partial(-ELBO)}{\partial \theta} = \frac{N}{K} \sum_{i \in S} \big[- \frac{\partial \mathbb{E}_{q_\theta(\boldsymbol{\omega})} \big[log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega})) \big]}{\partial \theta} \big] + \frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial \theta} \tag{2.17}$$

In order to calculate two terms in equation 2.17, we need to introduce one technique called re-parameterization trick[KW13] for the first term and one condition called KL condition[Gal] for the second term in the following.

Re-parameterization trick: when we take a close look at the first term in equation 2.17, we know that we need to compute the gradients of expectation w.r.t. the parameters of distribution to which this expectation is subject. That means, we need to estimate the gradients of those parameters because our objective is generated from these parameters randomly. Fortunately, there are different approaches to estimate the gradients of this kind of parameters such as score function or REINFORCE estimator[Wil92], re-parameterization

trick [KW13] and so on. As is stated in [KW13], re-parameterization trick has lower variance than score function estimator and is also unbiased. Let's have a quick recap of this technique and see it's already a built-in part in neural network with dropout.

To identify the problem, we can write a more general form of calculus we want to solve in the following:

$$I(\theta) = \frac{\partial}{\partial\theta}\mathbb{E}_{p_\theta(x)}\big[f(x)\big] = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \tag{2.18}$$

Assume that $p_\theta(x)$ can be re-parameterized as $p(\epsilon)$ which is a parameter-free distribution such that random variable x can be generated from a deterministic differentiable function with θ and ϵ as arguments, that is

$$x = g(\theta, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

Then we can derive the estimator of the gradients w.r.t. distribution parameters θ with $p(x|\epsilon) = \delta(x - g(\theta, \epsilon))$:

$$\begin{aligned}
I'(\theta) &= \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \\
&= \frac{\partial}{\partial\theta}\int f(x)(\int p_\theta(x|\epsilon)p(\epsilon)d\epsilon)dx \\
&= \frac{\partial}{\partial\theta}\int (\int f(x)p_\theta(x|\epsilon)dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int (\int f(x)\delta(x - g(\theta,\epsilon))dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int \frac{\partial}{\partial\theta} f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)p(\epsilon)d\epsilon \\
&= \mathbb{E}_{p(\epsilon)}\big[f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)\big]
\end{aligned} \tag{2.19}$$

From practical point of view, the expectation of last line in expression 2.19 can be approximated with Monte Carlo integration. In dropout inference, we know that if we fix the dropout rate which is equal to $1 - p_i$ in equation 2.12 in training. Then $\boldsymbol{\epsilon}_i$ in equation 2.12 is a parameter free random variable and $g(\cdot)$ is a differentiable function w.r.t. parameter $\boldsymbol{M}_i$. And the approximate distribution parameter θ only contains $\{(\boldsymbol{M}_i)_{i=1}^L\}$, which are exactly the weights to be learned in training neural network with dropout. As a result, if we estimate the gradient of $ELBO$ w.r.t. approximate distribution parameters $\{(\boldsymbol{M}_i)_{i=1}^L\}$,

that is the first term in 2.17, it's equivalent to calculate the gradient of dropout loss w.r.t. model parameters $\boldsymbol{\omega}$ which is the first term in equation 2.16.

KL condition: The second term in equation 2.17 is proved to be equivalent to the second term in equation 2.16 for a large enough number of hidden units when we specify the model prior to be a product of independent Gaussian distributions over each weight with prior length scale l, that is:

$$p(\boldsymbol{\omega}) = \prod_{i=1}^{L}(p(\boldsymbol{W}_i))$$

with

$$p(vec(\boldsymbol{W}_i)) = \mathcal{N}(0, l^{-2}\boldsymbol{I}_{D_{i-1}D_i})$$

To establish this condition, we need to make a approximation of the approximate posterior distribution $q_\theta(\boldsymbol{\omega})$ in equation 2.11, where we approximate $q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}_i)$ in equation 2.12 as a narrow Gaussian with a very small standard deviation. As we know, $q_\theta(\boldsymbol{W}_i)$ factorizes over each row of the weight matrix. Then that means $q_\theta(\boldsymbol{\omega})$ is a mixture of two Gaussians with small standard deviations, and one component fixed at zero:

$$\begin{aligned} q_\theta(\boldsymbol{\omega}) &= \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i) \\ &= \prod_{i=1}^{L}\prod_{j=1}^{D_{i-1}} q_\theta(\mathbf{w}_{i,j}) \\ &= \prod_{i=1}^{L}\prod_{j=1}^{D_{i-1}} \left[p_i\mathcal{N}(\mathbf{m}_{i,j}, \sigma^2\boldsymbol{I}_{D_i}) + (1-p_i)\mathcal{N}(\mathbf{0}, \sigma^2\boldsymbol{I}_{D_i})\right] \end{aligned} \tag{2.20}$$

where $\mathbf{w}_{i,j} \in \mathbb{R}^{D_i}$ is the j-th row of weight matrix $\boldsymbol{W}_i$ and p_i is the parameter of Bernoulli distributed random variable of i-th layer. With this, the KL divergence between approximate posterior and prior is KL divergence between mixture of Gaussian and a single Gaussian. In order to keep this report as self-contained as possible, we attach the derivation of KL divergence between mixture of Gaussian and single Gaussian in appendix A.1. Then we have KL condition in the following:

$$KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \approx \sum_{i=1}^{L}\sum_{j=1}^{D_{i-1}} \left[\frac{p_i}{2}(l^2\boldsymbol{m}_{i,j}^T\boldsymbol{m}_{i,j} + D_i(\sigma^2 - \log(\sigma^2) - 2\log l - 1) - \mathcal{H}(\boldsymbol{p}_i)\right] \tag{2.21}$$

with

$$\mathcal{H}(\boldsymbol{p}_i) = D_{i-1}(-p_i\log p_i - (1-p_i)\log(1-p_i))$$

for large enough D_i. If we fix dropout rate in training and compute the gradients of KL divergence w.r.t. model parameters $\theta = \{(\boldsymbol{M}_i)_{i=1}^L\}$. We can see that, if we choose $\lambda = \frac{l^2 p_i}{2}$, then it's equivalent to the gradients of regularization term in dropout loss function(cf. 2.16):

$$\frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial \theta} \approx \frac{\partial \sum_{i=1}^L \lambda ||\boldsymbol{M}_i||^2}{\partial \theta} \tag{2.22}$$

With aforementioned re-parameterization trick and KL condition, we know that the computation of gradients of objective function w.r.t. model parameters in equation 2.16 is equivalent to those of $ELBO$ w.r.t. approximate distribution parameters in equation 2.17. That means if we train a neural network with dropout, we can obtain the approximate posterior distribution over model parameters defined in equation 2.11.

Marginalization in testing After we have obtain the parameters θ of approximate posterior distribution over model parameters $q_\theta(\boldsymbol{\omega})$, we can marginalize all possible parameters over approximate posterior to get final predictive distribution of output, similar to equation 2.4 but with approximate posterior. Because the integration is hard to evaluate analytically. In practice, we always use Monte Carlo integration:

$$\begin{aligned} p(\boldsymbol{y}^\star|\boldsymbol{x}^\star, \mathcal{D}) &= \int p(\boldsymbol{y}^\star|\boldsymbol{x}^\star, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathcal{D}) d\boldsymbol{\omega} \\ &\approx \int p(\boldsymbol{y}^\star|\boldsymbol{x}^\star, \boldsymbol{\omega}) q_\theta(\boldsymbol{\omega}) d\boldsymbol{\omega} \\ &\approx \frac{1}{K} \sum_{i=1}^K p(\boldsymbol{y}^\star|\boldsymbol{x}^\star, \hat{\boldsymbol{\omega}}_{\boldsymbol{i}}) \end{aligned} \tag{2.23}$$

where $\boldsymbol{\omega} \sim q_\theta(\boldsymbol{\omega})$ and $\hat{\boldsymbol{\omega}}_{\boldsymbol{i}}$ is one of K realizations of $\boldsymbol{\omega}$, which is equivalent to turning on dropout in testing time. This is also called **MC-dropout** in the literatures.

2.2.3 Concrete dropout and Multi-Drop

Concrete dropout Based on the aforementioned description of dropout inference, we can see that if we fix dropout rate in training, then parameters of approximate distribution θ only contains $\{(\boldsymbol{M}_i)_{i=1}^L\}$, which is equivalent to $\boldsymbol{\omega} = \{(\boldsymbol{M}_i)_{i=1}^L\}$ in neural network with dropout. Therefore optimising any neural network with dropout is equivalent to a form of approximate inference in a probabilistic interpretation of the model. This means that the optimal weights found through the optimization of a neural network with dropout are the same as the optimal variational parameters in a Bayesian neural network with the same structure.

However, fixing dropout rate in train is not a trivial task for several reasons. Firstly, as is shown in [SHK+14], different dropout rates have different impact on model capacity and thus model performance. To choose an optimal dropout rate manually requires repeating tedious experiments and thus waste of time and computation effort. Secondly, if we want our model not only to achieve satisfied performance but also to possess reliable uncertainty estimation, the dropout rate matters a lot because it belongs to variational parameter θ and further influences the flexibility of approximate distribution family from the perspective of Bayesian interpretation.

Accordingly, one direct counter measure is to learn dropout rate from the data [GHK17]. One major difficulty to learn dropout rate from the data in gradient-based optimization is that it's not trivial to estimate gradients of expectation w.r.t. parameters of the distribution when this distribution is discrete. Before in case of continuous distribution, we estimate this gradient with help of re-parameterization trick. As introduced in the last section, re-parameterization trick requires re-parameterizing model parameters by a differential function with variational parameters and a parameter-free random variable as input arguments. For continuous distribution, this function can be found easily if they have tractable inverse cumulative density function or functional form like Gaussian[KW13]. For most of discrete distributions such as Bernoulli distribution or categorical distribution, they lack useful reparameterizations due to the discontinuous nature of discrete states [MMT16].

Faced with this issue, [JGP16] and [MMT16] come up with one approach that replaces "max" operation in Gumbel-Max trick with softmax function, which can yields a practical re-parameterization for discrete random variable. With this method, gradients w.r.t. parameters of discrete distribution can be computed and used in gradient-based optimization. In this subsection, a quick recap of this method is given in the following.

Re-parameterization of Bernoulli distribution Firstly, Gumbel-Max trick[MTM14] is introduced in figure 2.2.3, which is used for drawing samples from a discrete distribution which is parameterized by set of unnormalized probability $\{\alpha_i\}_{i=1}^n$ via inverse cumulative distribution function of Gumbel distribution, where $\alpha_i \in \mathbb{R}_{>0}$ and n denotes the number of class. Assuming that we use one-hot encoding vector for the class representation, that is $\boldsymbol{d} \in \{0,1\}^n$ and $\sum_{i=1}^n d_i = 1$. The Gumbel-Max trick proceeds as follows(cf. figure 2.2.3):

- sample $G_i \sim \text{Gumbel}(0,1) = -\log(-\log(\text{Uniform}(0,1)))$, for $i = 1,..,n$
- compute $x_i = \log(\alpha_i) + G_i$, for $i = 1,..,n$
- set $d_k = 1$, where $k = argmax_i\{x_i\}_{i=1,..n}$, and $d_i = 0$ for $i \neq k$

Then we obtain one sample from this discrete distribution and the probability for specific class.

As observed in Gumbel-Max trick, the sampling step is re-paramterized by one function

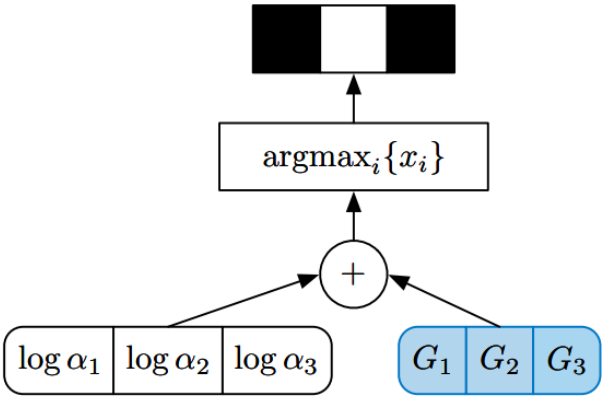


Figure 2.4: Example of Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\alpha_i\}_{i=1,2,3}$ as class parameters representing the possibility of occurrence of that class. $\{G_i\}_{i=1,2,3}$ are i.i.d Gumbel$(0, 1)$ [MMT16].

with distribution parameters and parameter-free random variable as arguments. This function is componentwise addition of two input arguments followed by $argmax$ operation. However, $argmax$ operation is not differential w.r.t. distribution parameter α_i. Fortunately, this operation can be approximated by a continuous function, $softmax$, which is also differential w.r.t. distribution parameters. With this replacement, we can obtain a continuous approximation to $\boldsymbol{d}$ on the simplex $\triangle^{n-1} = \{\boldsymbol{x} \in \mathbb{R}^n | x_k \in [0, 1], \sum_{i=1}^n x_i = 1\}$:

$$x_k = \frac{\exp((\log\alpha_k + G_k)/\lambda)}{\sum_{i=1}^n \exp((\log\alpha_i + G_k)/\lambda)} \tag{2.24}$$

The sampling steps are similar to Gumbel-Max trick, but with smoothed $softmax$ operation instead of $argmax$(cf. figure 2.5). When $\lambda \to 0$, the $softmax$ function is recovered to $argmax$ operation, when $\lambda \to \infty$, this operation generates uniform vector instead of one-hot encoded vector.

When it comes to Bernoulli discrete distribution in our case, it becomes similar because there are only two states in this distribution and samples live in two dimensional simplex. On the other hand, the difference of two Gumbels distributed random variable is similar to a logistic distributed random variable. With the probability of state 1 which can be expressed by:

$$\begin{aligned}\mathbb{P}(d_1 = 1) &= \mathbb{P}(\log\alpha_1 + G_1 > \log\alpha_2 + G_2) \\ &= \mathbb{P}(\log\alpha_1 - \log\alpha_2 + G_1 - G_2 > 0)\end{aligned}$$

where $G_1 - G_2 = \log(\text{Uniform}(0, 1)) - \log(1 - \text{Uniform}(0, 1))$, which is the inverse cumulative density function of Logistic(0,1). Then we can infer the re-parameterization of Bernoulli

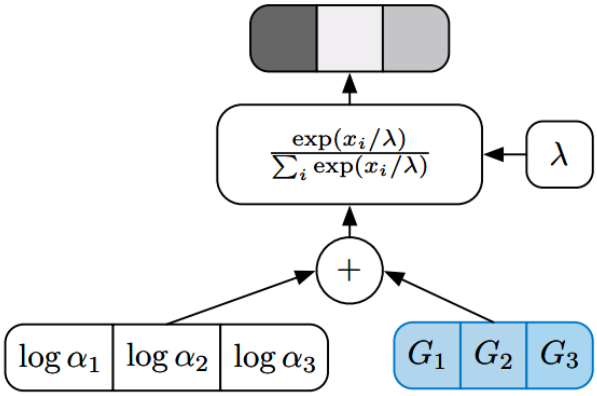


Figure 2.5: Example of continuous approximation to Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\alpha_i\}_{i=1,2,3}$ as class parameters representing the possibility of occurrence of that class. $\{G_i\}_{i=1,2,3}$ are i.i.d Gumbel$(0, 1)$[MMT16].

distributed variable with p as parameter as follows:

$$x_1 = \text{sigmoid}\big(\frac{1}{\lambda}(\text{log}p - \log(1-p) + \text{u} - \log(1-u))\big) \tag{2.25}$$

where $u \sim \text{Uniform}(0, 1)$.

In order to make explanation more clear and validate this re-parameterization, there is one plot in the following (figure 2.6) showing the relationship between average values of 100 samples drawn from this continuous approximation of Bernoulli distribution with different probability. We can see that the samples drawn from equation 2.25 distributed similar to the Bernoulli distribution, while most of samples lie on the boundary of range $[0, 1]$ and only few of them lie in the interior.

Dropout regularization Because keep rate of dropout is optimized now, the parameters of approximate distribution θ contains both $\{(\boldsymbol{M}_i)_{i=1}^L\}$ and $\{(\boldsymbol{p}_i)_{i=1}^L\}$. In order to compute gradients of ELBO in equation 2.17, we employ categorical re-parameterization to estimate the gradients w.r.t. Bernoulli distribution parameter $\boldsymbol{p}_i$ in the first term, for the second term, we could not ignore the term with $\boldsymbol{p}_i$ in KL condition 2.21, which is the entropy term. Consequently, unlike equation 2.22, the KL divergence term should be:

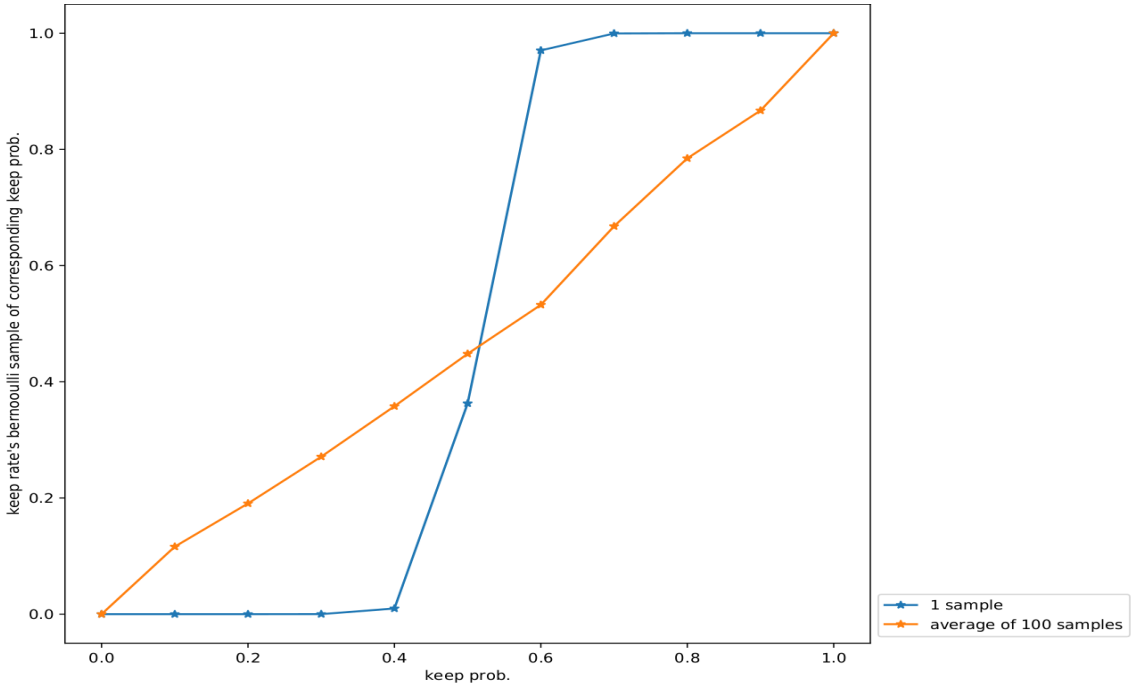


Figure 2.6: One sample and average value of 100 samples drawn from continuous approximation of Bernoulli distribution with parameter $p = [0.1, 0.2, ..., 1.0]$ and temperature $\lambda = 0.1$.

$$\begin{aligned} \frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial \theta} &\approx \frac{\partial \sum_{i=1}^{L} \lambda ||\boldsymbol{M}_i||^2 - \mathcal{H}(\boldsymbol{p}_i)}{\partial \theta} \\ &= \frac{\partial}{\partial \theta}\Big(\sum_{i=1}^{L} \lambda ||\boldsymbol{M}_i||^2 - D_{i-1}(-p_i \log p_i - (1-p_i)\log(1-p_i))\Big) \end{aligned} \tag{2.26}$$

From the equation above, we can see that this term maximize the entropy of Bernoulli distribution, which means this term pushes $\boldsymbol{p}_i$ to 0.5. The coefficient of the dropout regularisation term means that large models will push the dropout probability towards 0.5 much more than smaller models, but as the amount of data N increases the dropout probability will be pushed towards 1 because of the expected log likelihood in the first term. One of reasons behind could be pushing $\boldsymbol{p}_i$ to 0.5 would decrease the capacity of the model which will decrease the expected log-likelihood.

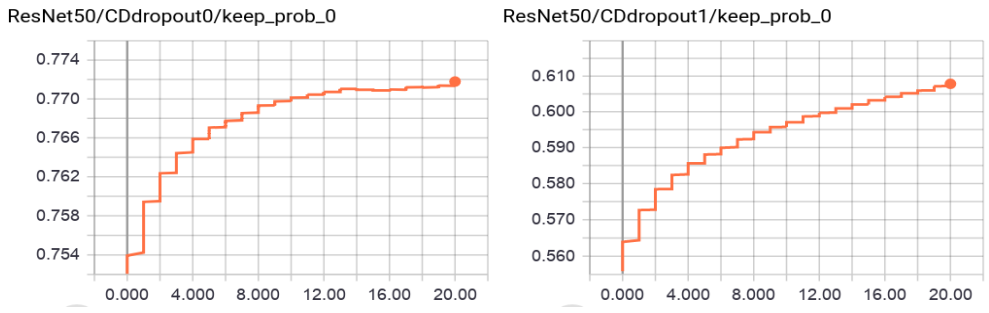
Multi-Drop From figure 2.2 and expression of approximate distribution in equation 2.11, we choose only one probability of Bernoulli distributed random variable for each layer, therefore random vector $\boldsymbol{p}_i = [p_i]^{D_{i-1}}$ for i-th layer, which stacks same value into a vector. While the dropout regularization term pushes the probability of Bernoulli to

0.5 to maximize its entropy, the expected likelihood term tries to increase the probability because decreasing probability will lead to a different model with lower capacity and thus low performance. An equilibrium state between them should be achieved in training. With concrete dropout introduced above, we could extend dropout for each hidden units instead of each layer(cf. figure 2.8), which means random vector $\boldsymbol{p}_i = [p_i^k]_{k=1}^{D_{i-1}}$. While the first term in gradients computation stays the same, the second term should be modified to:

$$\begin{aligned}\frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial\theta} &\approx \frac{\partial \sum_{i=1}^{L} \lambda||\boldsymbol{M}_i||^2 - \mathcal{H}(\boldsymbol{p}_i)}{\partial\theta} \\ &= \frac{\partial}{\partial\theta}\Big(\sum_{i=1}^{L}\lambda||\boldsymbol{M}_i||^2 - \sum_{k=1}^{D_{i-1}}(-p_i^k \log p_i^k - (1-p_i^k)\log(1-p_i^k))\Big)\end{aligned} \tag{2.27}$$

The reasons behind multi-drop are as following:

- to increase flexibility in tuning variational parameters. The tunability of parameter of Bernoulli random variable in the likelihood term is low because there is only single parameter controlling the entire layer. As is observed in the experiments(cf.figure 2.2.3), these parameters are always increased for each layer. The reason for this is probably that reducing it would lead to low capacity and thus low likelihood.
- the solution space of concrete dropout should be a subset of the solution space of multi-drop if all of them are reachable. Because if it's optimal to assign same probability for each hidden units, this can be recovered in training with multi-drop. Otherwise, other optimal solutions of assigning different probabilities to different hidden units could be considered.
- last but not least, multi-drop can extend the flexibility and diversity of the dropout approximate distribution family by adding more parameters. Hence the truth posterior can be approximated by the approximate posterior better.



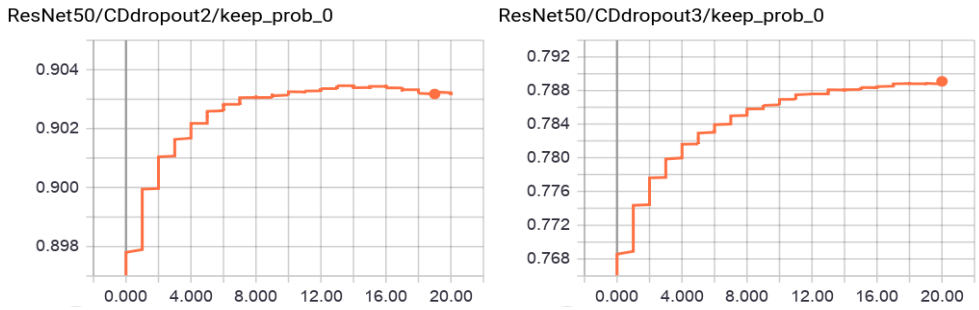


Figure 2.7: Changes along epochs of keep probability in training network with concrete dropout layer.

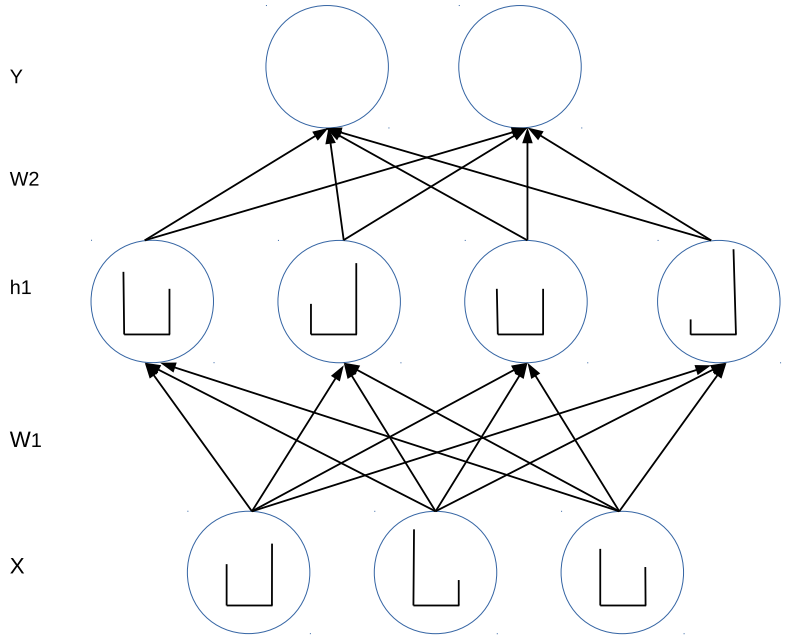


Figure 2.8: Different dropout rates for different hidden units in multi-drop.

2.2.4 Modified network architecture

After introducing dropout and concrete dropout variational inference, we will describe the modified network architecture in this subsection. The fundamental task in this work is object classification. We choose ResNet50[HZRS16] pre-trained on ImageNet as backbone for fine-tuning because of its strong ability to learn powerful representation for images. However, there is no dropout in original version of ResNet50. If we want to employ dropout variational inference to obtain reliable uncertainty estimation, dropout should be inserted into the network. In this work, we add three fully connected layer with 1024 hidden units ,

which are initialized from scratch, before the output layer whose dimension needs to be set to the number of classes. Then we add concrete dropout at flatten layer, and these three new added fully connected layer, respectively. There are three reasons why we modify the network in this way:

- Because we initialize major parts of network with pre-trained weights, inserting dropout in layers initialized with pre-trained weights would destroy pre-trained features. This would lead to significant drop of performance after fine-tuning.
- According to the suggestions from [SHK+14], insertion of dropout reduces the capacity of the model and thus a model with dropout should have larger capacity than one without dropout. Therefore we add three more fully connected layers to make sure that our model possesses large enough model capacity.
- As we have introduced in previous sub-sections, weights are major part of variational parameters. Therefore to have more weights can enhance the flexibility and capacity of approximate distribution family, which improves the quality of approximation.

From figure 2.9, we can see the sketch of our modified network architecture. More concretely, we can interpret major parts of network, which do not have dropout inserted, as a deterministic feature extractor. On the other hand, for layers with dropout inserted work as a probabilistic classifier based on aforementioned Bayesian interpretation of dropout. In training, these two parts are trained jointly in order to achieve a better balance and more optimal results. In testing, we need to marginalize possible parameters according to posterior distribution. Layers with dropout should be sampled and run multiple times to produce a predictive distribution of output class.

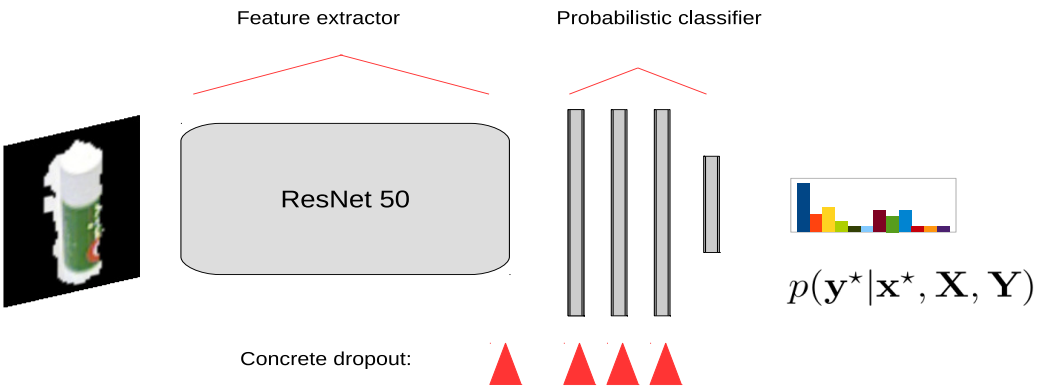


Figure 2.9: Modified network architecture.

2.3 Laplace approximation

2.3.1 Introduction

In this section, we introduce another method to approximate the real posterior distribution over weights of deep neural nets. This method is called Laplace approximation[Bis06], which is one deterministic approximation because only point estimation of model parameters instead of the learned variational parameters is required. The idea behind that is to put a Gaussian approximation on the maximum a posterior point estimate of the model parameters, which is one mode of posterior distribution. The reasons why we consider this method are as following:

- it has easy compatibility to existing network. To perform Laplace approximation, we only need one point estimation of model parameters which is already available for most of architectures. That also means, we do not need modify the training phase and we can have approximation of the true posterior for all trained models.
- it can capture relationship between model parameters. As is mentioned in the first chapter, most of approximation approaches make assumption that parameters are independent to each other for simplicity and computation burden. That could be a quite strong restrictions on the approximation leading to bad performance.

In the following, we introduce the basic idea of Laplace approximation and further introduce its scalable version for deep neural network based on Kronecker factor approximation.

Assume we have a point estimate of model parameters via maximum a posterior estimation:

$$\begin{aligned}\boldsymbol{\omega}^{\star} &= argmax_{\boldsymbol{\omega}}\{p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})\} \\ &= argmax_{\boldsymbol{\omega}}\{\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\boldsymbol{Y}|\boldsymbol{X})}\} \\ &= argmax_{\boldsymbol{\omega}}\{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})\}\end{aligned} \tag{2.28}$$

After taking a second order Taylor expansion of the logarithm of of posterior distribution $p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})$ around its mode $\boldsymbol{\omega}^{\star}$, we have:

$$\begin{aligned}\log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X}) \approx \log p(\boldsymbol{\omega}^{\star}|\boldsymbol{Y},\boldsymbol{X}) + \\ (\boldsymbol{\omega}-\boldsymbol{\omega}^{\star})^T\frac{\partial \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}} + \\ \frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^{\star})^T\frac{\partial^2 \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}^2}(\boldsymbol{\omega}-\boldsymbol{\omega}^{\star}) \\ = \log p(\boldsymbol{\omega}^{\star}|\boldsymbol{Y},\boldsymbol{X}) - \frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^{\star})^T\boldsymbol{H}(\boldsymbol{\omega}-\boldsymbol{\omega}^{\star})\end{aligned} \tag{2.29}$$

where

$$\begin{aligned}\boldsymbol{H} &= -\frac{\partial^2 \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}^2} \\ &= -\frac{\partial^2 \log\big(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2} - \frac{\partial^2 p(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}^2}\end{aligned}$$

which is negative Hessian of the log posterior. The first order term in equation 2.29 vanishes because we expand the function around a local maximum $\boldsymbol{\omega}^\star$, where the first derivative is zero. If we exponentiate this equation, we can get the following Gaussian-like functional form:

$$p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X}) \propto p(\boldsymbol{\omega}^\star|\boldsymbol{Y},\boldsymbol{X}) \exp\{-\frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star)^T \boldsymbol{H} (\boldsymbol{\omega}-\boldsymbol{\omega}^\star)\} \tag{2.30}$$

which means $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\omega}^\star, \boldsymbol{H}^{-1})$. Therefore, we can obtain a Gaussian approximate distribution with local maximum $\boldsymbol{\omega}^\star$ as mean and inverse of negative Hessian as covariance matrix.

We assume a Gaussian prior for weights. Then it's easy to know that the second order derivative of prior distribution term $\frac{\partial^2 p(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}^2}$ is a identity matrix multiplied by regularization coefficient. And the non-trivial part is the first term, second derivatives of log likelihood. To make the explanation uncluttered, we define the negative Hessian of log likelihood with $\hat{\boldsymbol{H}}$ instead:

$$\hat{\boldsymbol{H}} = -\frac{\partial^2 \log\big(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2}$$

However, we should note that for a large training set, it's always infeasible to analyze the gradients or Hessian exactly. To resolve this, normally we estimate the expectation of gradients or Hessian in each mini-batch. That means we need to estimate Hessian with empirical average Hessian computed in mini-batches:

$$\hat{\boldsymbol{H}} \approx N \mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}[\hat{\boldsymbol{H}}]$$

where N is size of training samples, and

$$\mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}[\hat{\boldsymbol{H}}] \approx -\frac{1}{K}\sum_k \big[\frac{1}{M}\sum_i \frac{\partial^2 \log\big(p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2}\big] \tag{2.31}$$

where K is total number of mini-batch, and $(y_{ik}, \boldsymbol{x}_{ik})$ is the i-th training data sample in k-th mini-batch.

Fisher information matrix is equivalent to the expected negative Hessian of exponential family log probability:

$$\hat{\boldsymbol{F}} = \mathbb{E}_{p(\boldsymbol{X},\boldsymbol{Y})}[\hat{\boldsymbol{H}}]$$

Therefore we can use Fisher information matrix as replacement of expected Hessian for the log likelihood term. The derivation of equivalence between expected Hessian and Fisher matrix is straightforward. We define Fisher matrix $\boldsymbol{F}$ in the following:

$$\begin{aligned}\hat{\boldsymbol{F}} &= \mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}\big[\frac{\partial}{\partial \boldsymbol{\omega}}\log p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\frac{\partial}{\partial \boldsymbol{\omega}}\log p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})^T\big] \\ &\approx \frac{1}{K}\sum_k \big[\frac{1}{M}\sum_i \big(\frac{\partial}{\partial \boldsymbol{\omega}}\log p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})\frac{\partial}{\partial \boldsymbol{\omega}}\log p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})^T\big)\big]\end{aligned} \tag{2.32}$$

We include the derivation of equivalence between negative expected Hessian and Fisher matrix for case that parameter is scalar in appendix **??**, which can be generalized to case for vector. Therefore we can obtain

$$\hat{\boldsymbol{H}} \approx N\hat{\boldsymbol{F}} \tag{2.33}$$

and

$$\boldsymbol{H} \approx N\hat{\boldsymbol{F}} + \tau\boldsymbol{I} \tag{2.34}$$

Actually, to compute outer product of gradients is feasible. But to compute the empirical average of outer product we need to save values with amount quadratic to the number of model parameters which induces large storage overhead. More than that, to inverse this matrix, the computation complexity is cubic of dimensionality of this matrix which is the number of model parameters which is also infeasible because the number of parameters in deep neural nets can reach million easily. Therefore an efficient approximation for Fisher matrix is required.

2.3.2 Scalable Laplace approximation for neural network

In order to mitigate the computational burden in Laplace approximation above, [RBB18] proposes to use Kronecker-factored approximation curve(KFAC) in [MG15] to approximate the Fisher information matrix $\boldsymbol{F}$ and perform Laplace approximation for neural network. This approximation is derived by approximating the large blocks matrix of Fisher(corresponding to each layer) by Kronecker product of two much smaller matrices. While several times more expensive to compute this, the issues of storage and inversion in Laplace approximation can be mitigated. Additionally, because we need this Fisher

information matrix after training, this little expensive computation is required to perform once.

With the notations used in previous subsection, where dropout variational inference is introduced, we will derive the approximation in the following. For clarity, we repeat the definitions of existing notations and new defined ones here. A neural network transforms its input $a_0 = \boldsymbol{x}$ to an output $a_L = p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\omega}) = \sigma(\boldsymbol{x}, \boldsymbol{\omega})$ through a series of L layers. The units each receive as input a weighted sum of outputs of units from the previous layer and compute their output via a non-linear activation function $\phi(\cdot)$. We denote by $\boldsymbol{s}_i$ the vector of these weighted sums for i-th layer, and by $\boldsymbol{a}_i$ the vector output of non-linear activation function. Computation performed for each layer $i \in 1, ..., L$ is given as follows:

$$\boldsymbol{s}_i = \boldsymbol{a}_i \boldsymbol{W}_i \text{ and } \boldsymbol{a}_i = \phi(\boldsymbol{s}_i) \tag{2.35}$$

We define $\tilde{\boldsymbol{w}}$ to be the vector consisting of all of the network's parameters concatenated together, i.e. $\tilde{\boldsymbol{w}} = \left[vec(\boldsymbol{W}_1)^T \, vec(\boldsymbol{W}_2)^T \, ... \, vec(\boldsymbol{W}_L)^T\right]^T$, where vec is the operator which vectorizes matrices by stacking their columns together. For simplicity we first define the following notations:

$$\partial v = -\frac{\partial \log p(\boldsymbol{y}, |\boldsymbol{x}\boldsymbol{\omega})}{\partial v} \text{ and } \boldsymbol{g}_i = \partial \boldsymbol{s}_i$$

The first derivatives of objective w.r.t. $\tilde{\boldsymbol{w}}$ is defined as follows:

$$\partial \tilde{\boldsymbol{w}} = \left[vec(\partial \boldsymbol{W}_1)^T \, vec(\partial \boldsymbol{W}_2)^T \, ... \, vec(\partial \boldsymbol{W}_L)^T\right]^T \tag{2.36}$$

From equation 2.32, we know $\hat{\boldsymbol{F}} = \mathbb{E}[\partial \hat{\boldsymbol{w}} \partial \hat{\boldsymbol{w}}^T]$ which can viewed as L by L block matrix, with the (i, j)-th block $\hat{\boldsymbol{F}}_{i,j}$ given by $\hat{\boldsymbol{F}}_{i,j} = \mathbb{E}[vec(\partial \boldsymbol{W}_i) vec(\partial \boldsymbol{W}_j)^T]$.

Noting that $\partial \boldsymbol{W}_i = \boldsymbol{a}_{i-1} \boldsymbol{g}_i^T$ and $vec(\boldsymbol{u}\boldsymbol{v}^T) = \boldsymbol{v} \otimes \boldsymbol{u}$, then we have

$$vec(\partial \boldsymbol{W}_i) = vec(\boldsymbol{a}_{i-1} \boldsymbol{g}_i^T) = \boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1}$$

Then we can rewrite

$$\begin{aligned}
\hat{\boldsymbol{F}}_{i,j} &= \mathbb{E}[vec(\partial \boldsymbol{W}_i) vec(\partial \boldsymbol{W}_j)^T] \\
&= \mathbb{E}[(\boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1})(\boldsymbol{g}_j \otimes \boldsymbol{a}_{j-1})^T] \\
&= \mathbb{E}[(\boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1})(\boldsymbol{g}_j^T \otimes \boldsymbol{a}_{j-1}^T)] \\
&= \mathbb{E}[(\boldsymbol{g}_i \boldsymbol{g}_j^T) \otimes (\boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T)]
\end{aligned}$$

Here we need to define $\tilde{\boldsymbol{F}}_{i,j}$ as approximation to $\hat{\boldsymbol{F}}_{i,j}$ in the following:

$$\begin{aligned}
\hat{\boldsymbol{F}}_{i,j} &\approx \tilde{\boldsymbol{F}}_{i,j} \\
&= \mathbb{E}[\boldsymbol{g}_i \boldsymbol{g}_j^T] \otimes \mathbb{E}[\boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T] \\
&= \mathbb{E}[\boldsymbol{G}_{i,j}] \otimes \mathbb{E}[\boldsymbol{A}_{i,j}]
\end{aligned} \tag{2.37}$$

where $\boldsymbol{G}_{i,j} = \boldsymbol{g}_i \boldsymbol{g}_j^T$ and $\boldsymbol{A}_{i,j} = \boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T$. The expectation of a Kronecker product is, in general, not equal to the Kronecker product of expectations, and so this is indeed a major

approximation to make, and one which likely won't become exact under any realistic set of assumptions, or as a limiting case in some kind of asymptotic analysis. Nevertheless, it seems to be fairly accurate in practice, and is able to successfully capture the "coarse structure" of the Fisher information Matrix. In this work, we only consider the case that layers are independent to others. Therefore the Fisher information matrix of the entire network is a diagonal block matrix, which means $\tilde{\boldsymbol{F}}_{i,j}$ is non-zero only for $i = j$. For i-th layer, we can compute the Fisher information matrix with equation 2.37. The two factors in Kronecker are outer product of gradients of pre-activation $\boldsymbol{G}_i = \boldsymbol{g}_i \boldsymbol{g}_j^T$ and outer product of activation from previous layer $\boldsymbol{A}_i = \boldsymbol{a}_{i-1} \boldsymbol{a}_{i-1}^T$, respectively:

$$\tilde{\boldsymbol{F}}_i = \boldsymbol{G}_i \otimes \boldsymbol{A}_i$$

If we treat them as covariances of resulting Gaussian. That means each Gaussian has a Kronecker factored covariance, corresponding to a matrix normal distribution[GN99], which considers the two Kronecker factors of the covariance to be the covariances of the rows and columns of a matrix. The two factors are much smaller than the full covariance and allow for significantly more efficient inversion and sampling. The final posterior of Laplace approximation for i-th layer is:

$$\boldsymbol{W}_i \sim \mathcal{MN}(\boldsymbol{W}_i^\star, \mathbb{E}[\boldsymbol{A}_i]^{-1}, \mathbb{E}[\boldsymbol{G}_i]^{-1}) \tag{2.38}$$

f If we consider Gaussian prior and scale of Fisher information matrix(cf.equation2.34), then the resulting posterior can be written in the following form:

$$\boldsymbol{W}_i \sim \mathcal{MN}(\boldsymbol{W}_i^\star, (\sqrt{N}\mathbb{E}[\boldsymbol{A}_i] + \sqrt{\tau}\boldsymbol{I})^{-1}, (\sqrt{N}\mathbb{E}[\boldsymbol{G}_i] + \sqrt{\tau}\boldsymbol{I})^{-1}) \tag{2.39}$$

where N is the size of training set and τ is the standard deviation of Gaussian prior. However, we find that in practice, to treat them as hyper-parameters and optimize them w.r.t. the predictive performance on a validation set can yield better performance. The possible reasons for them are two fold. Firstly, nowadays the scale of dataset is really large which can exceed tens of thousand easily. The large size of data could be probably redundant and if we take the redundant data into account and thus increase N. This will lead to unreasonably high precision for Gaussian and underestimate the uncertainty. Second, Laplace approximation requires positive definite Hessian and τ always serves as damp factors to fulfill this condition, therefore it needs to be chosen and tuned carefully.

Chapter 3

Conditional random field

3.1 Introduction

i

3.2 Learning

T

3.3 Inference

Infer

Chapter 4

Experiments and Discussion

4.1 Introduction

In this chapter, we perform different experiments for the following purposes:

- to evaluate the performance of uncertainty estimation of dropout variational inference and its variants as well as Laplace approximation. Comparisons and analysis towards the results of these two kinds of approach are given.
- to evaluate performance of training a domain specific classifier, which is performing more accurately for the objects that are encountered by the robot, with different strategies for collecting training data including manual labeling and automatic labeling, where the latter one is chosen based on uncertainty estimation.
- to evaluate performance of classifier including context information via CRF.

Before looking into the results, we need to specify the details of experiments such as the dataset, evaluation metrics. Besides, the detailed parameters of the model will be reported in section of each experiment to avoid confusions. Models were implemented in Tensorflow [**TODO CITATION**], and optimization was performed using RMSprop with initial learning rate of $1e^{-5}$ and $l2$ regularization with coefficient of $3.5e^{-6}$. We used early stopping with all methods, where the amount of epochs to run was determined based on performance on a validation set. We set the number of maximum epoch as 20.

4.1.1 Dataset

WRGBD[LBRF11] This dataset is a large-scale dataset of 300 household objects captured from multi-viewpoint. The objects are organized into 51 categories, some of categories and their subtrees are shown in figure 4.1. To note that category level recognition

and detection involves classifying previously unseen objects as belonging in the same category as objects that have previously been seen(e.g., coffee mug). Instance level recognition and detection is identifying whether an object is physically the same object that has previously been seen. Therefore the overlapping of features in category classification is larger than that in instance classification. The setup of dataset is, that each object was placed on a turn table and captured from a systematically sampled view hemisphere with 15° step in elevation (from 30°to 60°) and 2° step in azimuth (from 0° to 360°) (cf. figure 4.2). The total size of entire dataset is 160.9×10^3.

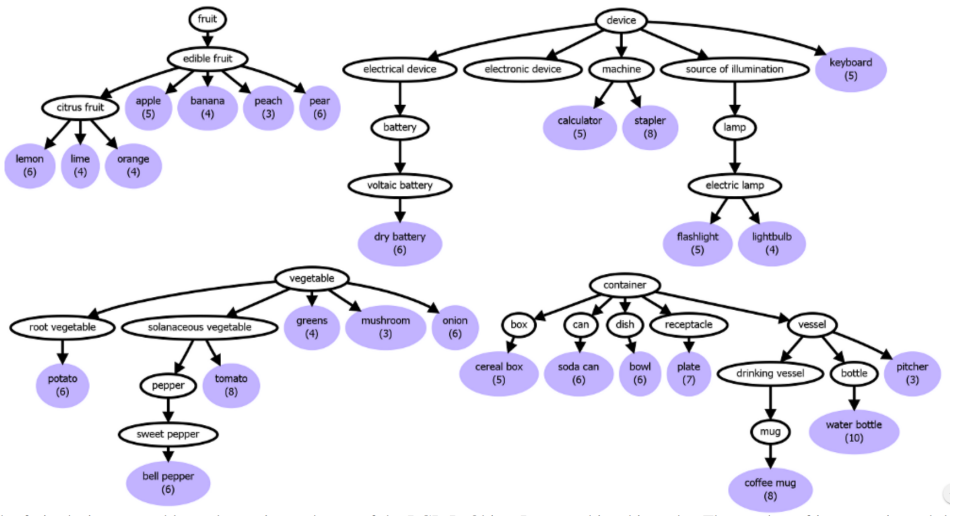


Figure 4.1: The fruit, device, vegetable, and container subtrees of the RGB-D Object Dataset object hierarchy, the shaded ellipse represent categories and number inside parenthesis denotes the number of instances within this category[LBRF11].

UniHB To simulate application-specific situation in robotic deployment, we recorded a similar dataset by putting objects on a turn table in front of the robot and recording partial views of objects this way. We follow the same methodology as [LBRF11] suggests for WRGBD, but with only one object instance for each category in the dataset. This analogue of WRGBD dataset is called the IAI-ODU dataset of UniHB. The size of data with elevation 45° is 8.6×10^3 and that of data with elevation 30° and 60° is 17.1×10^3.

While the UniHB dataset's setup strives to mimic the WRGBD one, the differing capturing conditions such as different equipment and light conditions even appearances of objects, result in obvious changes in feature space, thus yielding a significant drop on classification accuracy. In addition to the existing 51 categories, there are 28 novel objects(cf. figure 4.3) that do not belong to the 51 categories and these objects are treated as out-of-distribution(OOD) data for testing uncertainty estimation of the model. To note that these novel objects are recorded in a slight different way, that is, their view points are

sampled with 15° step in elevation (from 30°to 60°) and 5° step in azimuth (from 0° to 360°). The size of OOD dataset is around 6.0×10^3.

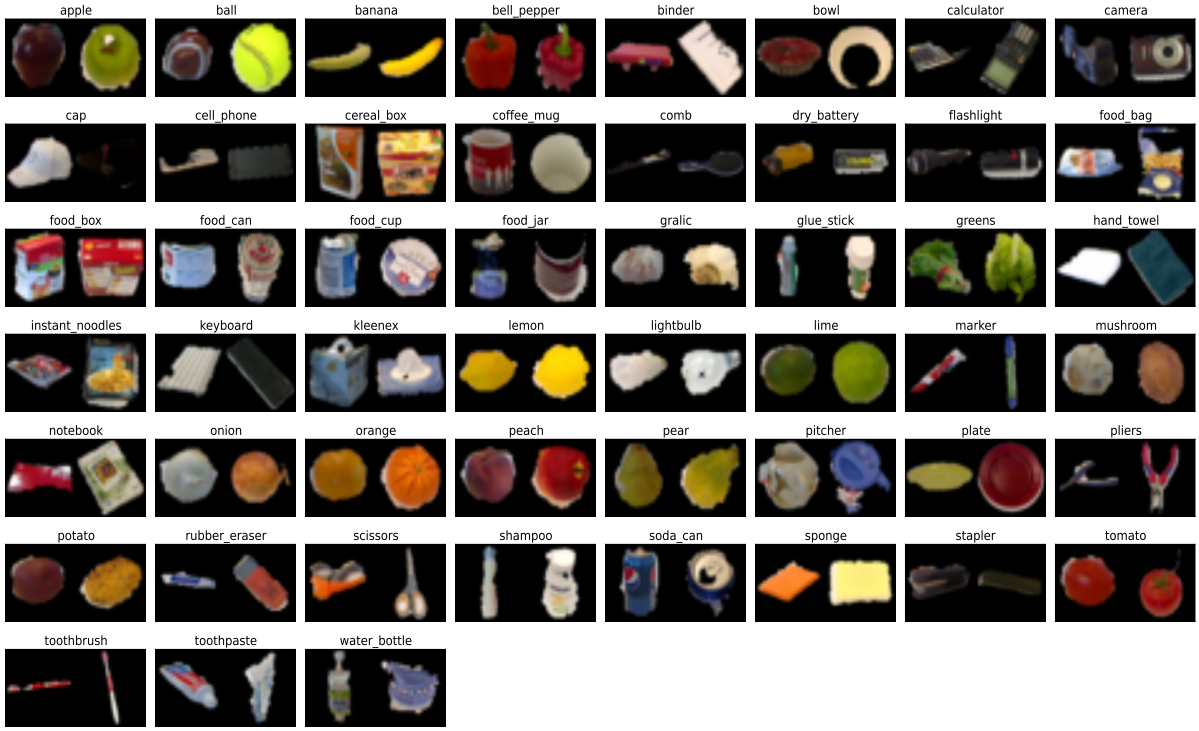


Figure 4.2: Example of masked images of objects from 51 categories in WRGBD and UniHB dataset. In each category, the left is from WRGBD and the right is from UniHB. We randomly pick one instance for the objects of WRGBD

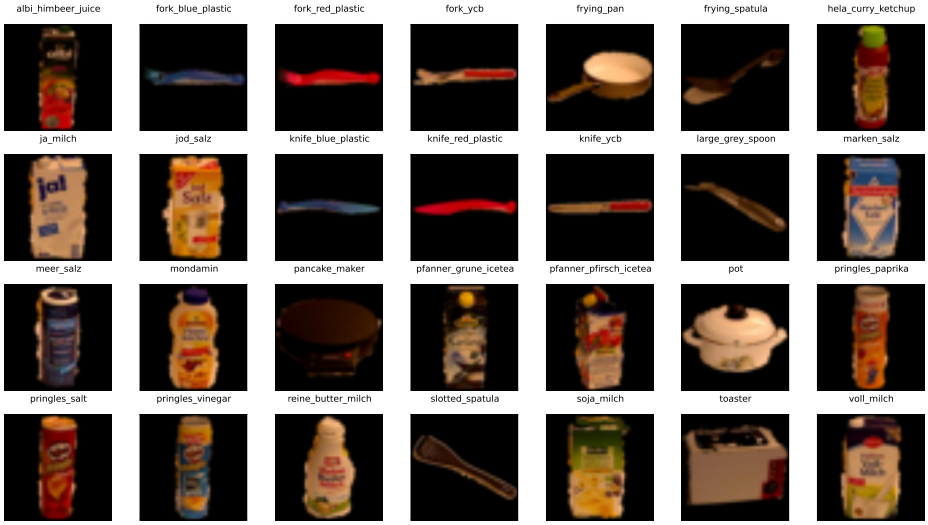


Figure 4.3: Example of masked images of objects from 28 categories which are not belonging to WRGBD categories, which are treated as OOD data samples.

T-LESS[HHO$^+$17]

4.1.2 Uncertainty measures

For each prediction, we can obtain one predictive probability distribution from our model with equation 2.23. In order to quantify uncertainty of the prediction,there are different metrics to measure the uncertainty of prediction, which are introduced in the following. We define $x^\star$ as one test data sample and $\mathcal{D}$ as our training set.

Confidence is defined as the maximum probability of the output predictive distribution, whose index is the class prediction.

$$conf = \max_c \left[p(y = c|x^\star, \mathcal{D})\right] \tag{4.1}$$

where $conf \in [0, 1]$, $c \in \mathcal{P} = \{0, ..., |\mathcal{P}| - 1\}$, and $\mathcal{P}$ represents output space in which label is expressed as number index which is transformed into one-hot representation in computation of objective function. The larger this quantity is, the less uncertainty the prediction is.

Predictive entropy is quantity that captures the average amount information contained in predictive distribution[Sha48]:

$$\mathcal{H}[p(y|x^\star, \mathcal{D})] = -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \tag{4.2}$$

where c is the possible class y can take, $\mathcal{H}(\cdot) \in [0, \log|\mathcal{P}|]$, the larger this quantity is, the more uncertainty the prediction is.

Mutual information Mutual information between the prediction y and the weights posterior offers a different uncertainty measure. This measure is widely used in active learning tasks[HHGL11], which is also called Bayesian active learning disagreement(BALD).

$$\begin{aligned}
\mathcal{I}[y, \boldsymbol{\omega}|x^\star, \mathcal{D}] &= \mathcal{H}[y|x^\star, \mathcal{D}] - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D})}\big[\mathcal{H}[y|x^\star, \boldsymbol{\omega}]\big] \\
&= -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \\
&\quad + \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D})}[-\sum_c p(y = c|x^\star, \boldsymbol{\omega}) \log p(y = c|x^\star, \boldsymbol{\omega})] \\
&\approx -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \\
&\quad + \mathbb{E}_{q(\boldsymbol{\omega})}[-\sum_c p(y = c|x^\star, \boldsymbol{\omega}) \log p(y = c|x^\star, \boldsymbol{\omega})]
\end{aligned} \tag{4.3}$$

where $\mathcal{I}(\cdot) \in [0, \log|\mathcal{P}|]$. This quantity consider the effect of approximate posterior distribution directly. Compared with aforementioned uncertainty measures, this one should be able to capture the model uncertainty more accurately. To think about it intuitively, this quantity measure the information gain between the entropy of predictive output distribution and the expected entropy of output distribution marginalized over weights posterior. It will be low only if the predictive distribution agrees with most of possible models(weights realizations), which means that the model is sure about its prediction. Otherwise, it will be high because most of possible models do not agree with each other and thus the predictive distribution will be more uniform.

4.1.3 Evaluation metrics

As is stated in [GBR07], the goal of probabilistic prediction is to maximize the sharpness of predictive prediction subject to calibration. Calibration refers to the statistical consistency between the predictive probability and the occurrence of observations. Therefore we employ different metrics including both the accuracy and other quantities related to calibration as well as summary of accuracy and calibration. Additionally, we also employ histogram and diagram to express the results visually. While the visual one can show us the results more intuitively, the quantitative one can allow us to evaluate the results more objectively. The

comparison between them can also help us to examine if visual metrics are corresponding to the numerical metrics and thus provide more insights in evaluating uncertainty estimation.

Uncertainty histogram is a intuitive visual representation for analyzing the statistics of the uncertainty estimation. Compared with normal histogram, there is one difference to stress on. In order to make visual effect more clear, **normalizer** of each type of prediction is the total number of this type of predictions instead of the entire dataset. More concretely, we have three types of predictions when plotting the histogram, which are:

- **correct prediction**
- **miss-classification**
- **out-of-distribution(OOD)**

The range of y axis is $[0, 1]$ and range of x axis depends on the type of uncertainty.

Reliability diagram(Calibration curve) is a visual representation of model **calibration**[GPSW17], which plot the frequency of success(accuracy of predictions in specific bin) as a function of confidence. If the model is perfectly calibrated, this function should be overlapping with the diagonal line. In order to have this plot, we firstly group the predictions into M interval bins w.r.t. confidence and then calculate the accuracy of predictions in each bin. We use M=20 in this work.

- **Expectation calibration error(ECE)**:Furthermore, in order to obtain a more objective measure of calibration quality, we can compute the **expected calibration error** by computing the weighted average of difference between accuracy and confidence:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{4.4}$$

  where n is the number of samples, and $acc(B_m)$ represents the accuracy of samples, $conf(B_m)$ the average predicted confidence of samples in m-th bin. We can see that this metric measures the inconsistency between the statistics and predictive distribution. To note that this metric only considers the calibration quality instead of accuracy. On the other hand, because the weights are computed based on number of samples in each bin, the metric would bias if most of predictions are clustered in few bins.

- **Maximal calibration error(MCE)**: Additionally, in high risk applications where reliable confidence measures are absolutely necessary, we may wish to take the worst case into account. Based on these two points, **maximum calibration error** should also be evaluated, whose expression is defined as follows:

$$MCE = \max_m |acc(B_m) - conf(B_m)| \tag{4.5}$$

Proper scoring rules: Scoring rules provides a **summary** measures for the evaluation of probabilistic forecasts by assigning a numerical score based on the predictive distribution and real distribution of event we want to predict. Because we want to make predictions for the future and also have a suitable measures of the uncertainty associated with them(see [GR07] for a review). Let's define the scoring rule as a function $\mathcal{S}(p(y|\boldsymbol{x}), (y|\boldsymbol{x}))$ that evaluates the quality of predictive distribution $p(y|\boldsymbol{x})$ relative to an event $y|\boldsymbol{x} \sim q(y|\boldsymbol{x})$ where $q(y|\boldsymbol{x})$ represents the true distribution over $(y|\boldsymbol{x})$. Consequently, the expected scoring rule is:

$$\mathcal{S}(p, q) = \int q(y|\boldsymbol{x})\mathcal{S}(p, (y|\boldsymbol{x}))dy \tag{4.6}$$

$\mathcal{S}(p, q)$ is proper if $\mathcal{S}(p, q) \leq S(q, q)$, with equality holds if and only if $p(y|\boldsymbol{x}) = q(y|\boldsymbol{x})$. Here we adopt two simple and famous proper scoring rules in which the less it is, the better the performance is:

- **negative log likelihood(NLL)**:NLL is a popular metric for evaluating predictive uncertainty [QCRS$^+$05]. This metric considers aforementioned confidence as likelihood. The smaller this metric is, the better the predictive distribution is.

$$NLL = -\frac{1}{|\mathcal{D}_{test}|} \sum_{i=1}^{|\mathcal{D}_{test}|} \log(p(y_i = c_i|\boldsymbol{x}_i)) \tag{4.7}$$

  where c_i is the ground truth label for $\boldsymbol{x}_i$.

- **Brier score** is the mean squared error between the target distribution(one-hot encoding label) and predictive distribution:

$$BS = -\frac{1}{|\mathcal{D}_{test}|} \sum_{i=1}^{|\mathcal{D}_{test}|} (\boldsymbol{y}_i^{gt} - p(\boldsymbol{y}_i|\boldsymbol{x}_i))^2 \tag{4.8}$$

  where $\boldsymbol{y}_i^{gt}$ is the one-hot encoding ground truth label for $\boldsymbol{x}_i$.

Separability metrics: In addition to the summary metrics of predictive probability. We are also interested in the separability between different types of predictions, which can assist the down-stream tasks by separating them(e.g. separate correct predictions and false predictions or out-of-distribution data).

- **Area under Receiver Operating Characteristic curve(AUROC)**:Since one of our goals is to choose automatically labeled data based on uncertainty estimation, it's better to evaluate the separability between correct predictions and false predictions or out-of distribution samples. ROC curve describes the relationship between true positive rate($tpr = \frac{tp}{tp+fn}$) and false positive rate($fpr = \frac{fp}{fp+tn}$). Moreover, AUROC can be interpreted as the probability that a positive samples has a greater score than a negative samples
- **Area under Precision Recall curve(AUPR)**: Because the normalizers of two kinds of rate in ROC curve are unrelated to each other. When these two normalizers differs too much. AUROC can provide misleading conclusion. For example, if the number of negative predictions is much higher than positive one, the AUROC could still achieve a relatively high value. But in fact there are many false positives which could induce undesired effects. Therefore another evaluation metric is employed, that is AUPR. AUPR describes the relationship between precision($pr = \frac{tp}{tp+fp}$) and recall($tpr = \frac{tp}{tp+fn}$), which resolves the problem of different base number. In the previously mentioned case, though the AUROC is high, the AUPR will be low because the precision is low. To note that, we treat the correct predictions as positive samples in these two kinds of curve.

4.2 Uncertainty estimation experiments

4.2.1 Experiments I

In this experiments, we want to evaluate our model on a relatively easy task at first, which means that the appearances of different categories as well as the out-of-distribution categories are highly discriminable. Therefore we separate WRGBD dataset into two subsets based on the instance label. Subset I contains objects with instance label from 0 to 199 (assuming that here we use index to denote the instance label) and Subset II contains objects with instance label from 200 to 299.

We train our model with objects captured in elevation 30° and 60° of Subset I (in which we split off 20% of training set as validation set for model selection in training). The size of training set and validation set is around 71.0×10^3. Then we test the model on objects captured in elevation 45° of both Subset I and Subset II. In this experiment, the objects in Subset II serve as out-of-distribution samples because they are not present in training. The size of test set is 35.6×10^3 and that of OOD dataset is 17.8×10^3.

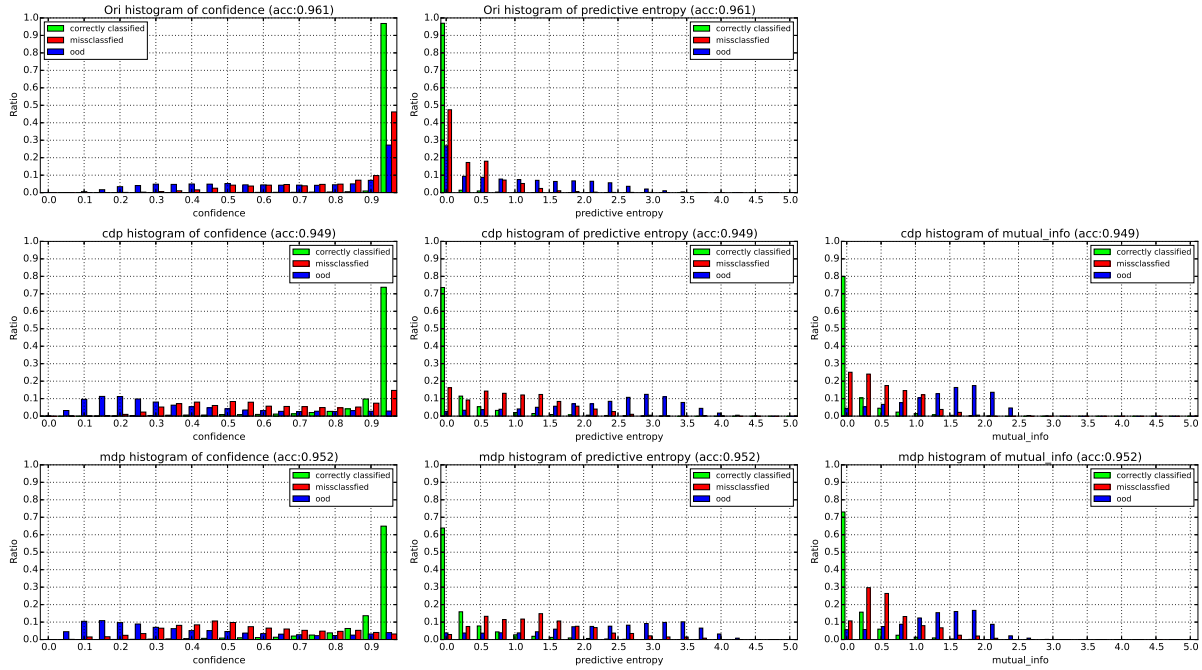


Figure 4.4: Uncertainty(confidence, predictive entropy, mutual information) histograms of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout.

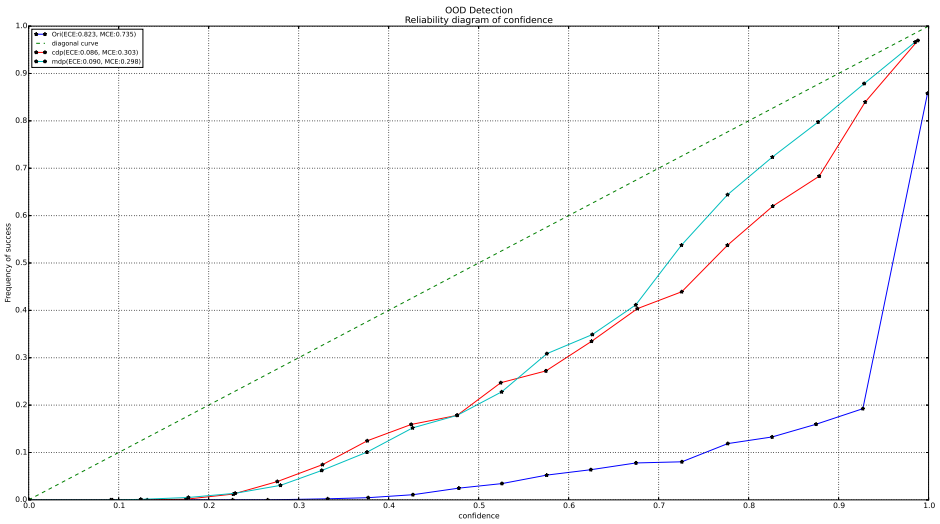


Figure 4.5: Calibration curves of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout.

4.2.2 Experiments II

In this experiment, we evaluate our model on category recognition task. The difficulties are expressed in two aspects, firstly, the task is category recognition instead of instance recognition, which means that the model is confronted with classes with more overlaps and thus more abstract concept needed to be learned. Second, since we want to simulate situation of deploying robot in real world scenario, the UniHB dataset with domain gap(cf. figure4.2) to WRGBD is used to achieve this goal. It means that the uncertainty estimation should not only be able to perform well on dataset with same distribution, but also to generalize well to dataset with domain gap which may decrease the accuracy significantly.

Accordingly, we use the entire WRGBD dataset including all view points, whose size is around 160.9×10^3, to train our model (in which we split off 20% of training set as validation set for model selection during training). When it comes to UniHB dataset, we treat objects captured in elevation 30° and 60° in this dataset as **adaptation set**, whose size is around 17.1×10^3, on which we test performance of uncertainty estimation. The images of 45°, whose size is around 8.6×10^3 , are used for final evaluation after we fine-tune the model with subset of adaptation set to obtain a domain specific model, which will be experimented in next section. Besides, we also evaluate the uncertainty estimation on OOD dataset(cf. figure 4.3) whose size is around 6.0×10^3.

There are some protocols of different metrics needed to be clarified, which can help understanding the plots and extract useful information more easily and quickly.

- The following visual metrics are chosen in one of three runs of different random seeds. The average quantitative results are given in table 4.1.
- The calibration curve with tile "Classification" on the left is plotted only on test dataset. The one with tile "OOD detection" on the right is plotted on both OOD dataset and test dataset.
- The ROC curve and PR curve measure the separability between two types of prediction. The ones with title "classification" on the left measure the separability between **miss-classification** and **correct prediction**. The ones with title "OOD detection" on the right measure the separability between **OOD prediction** and **correct prediction**.
- As is shown in uncertainty histogram, we have used three uncertainty measures. Each one has its own ROC curve and PR curve of correct prediction versus miss-classification or OOD prediction. In the following results of ROC curve and PR curve, we only show the one of uncertainty with highest area under curve. In case of correct prediction versus miss-classification, **confidence** is chosen. In case of correct prediction versus OOD prediction, **mutual information** is chosen.

4.2.3 Comparison with Ensemble

Uncertainty histogram This is uncertainty hitsogram

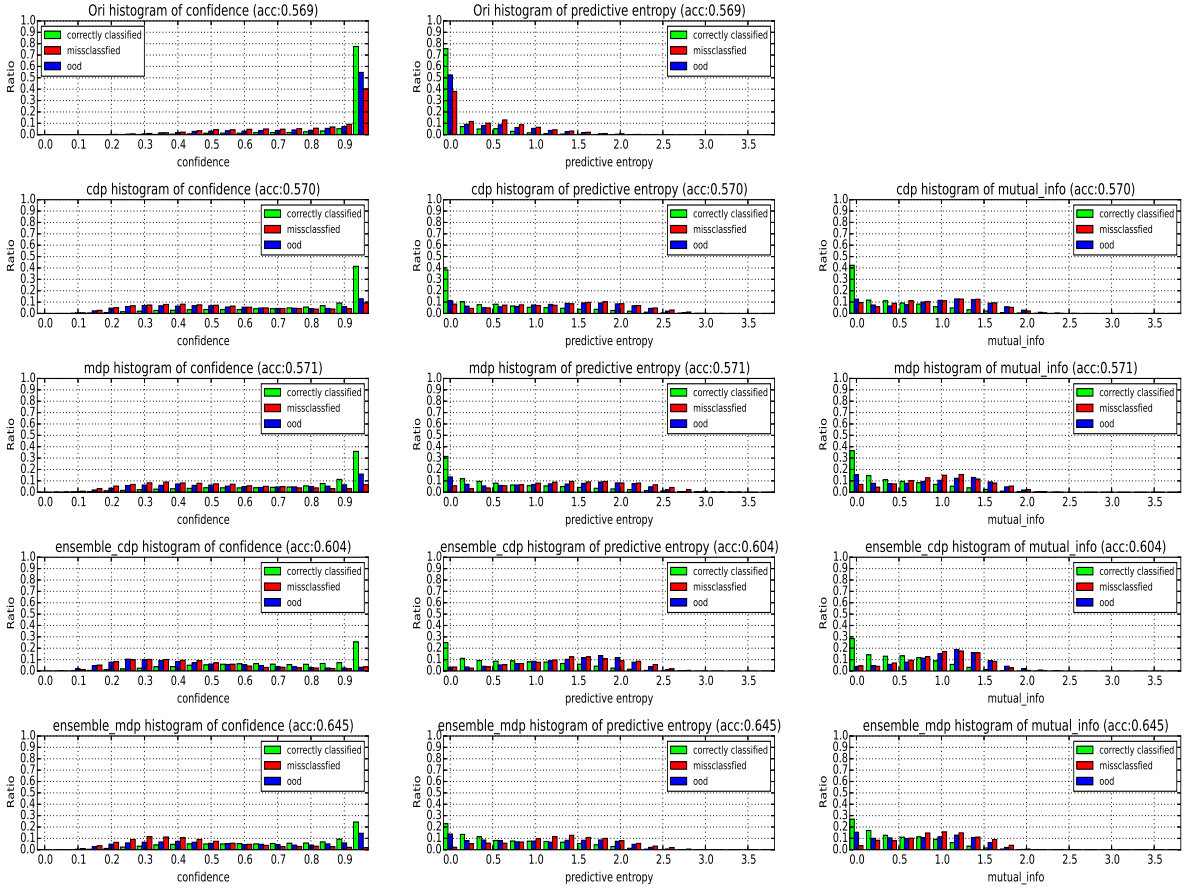


Figure 4.6: Uncertainty histogram of one of three runs.

Reliability diagram This is Reliability diagram

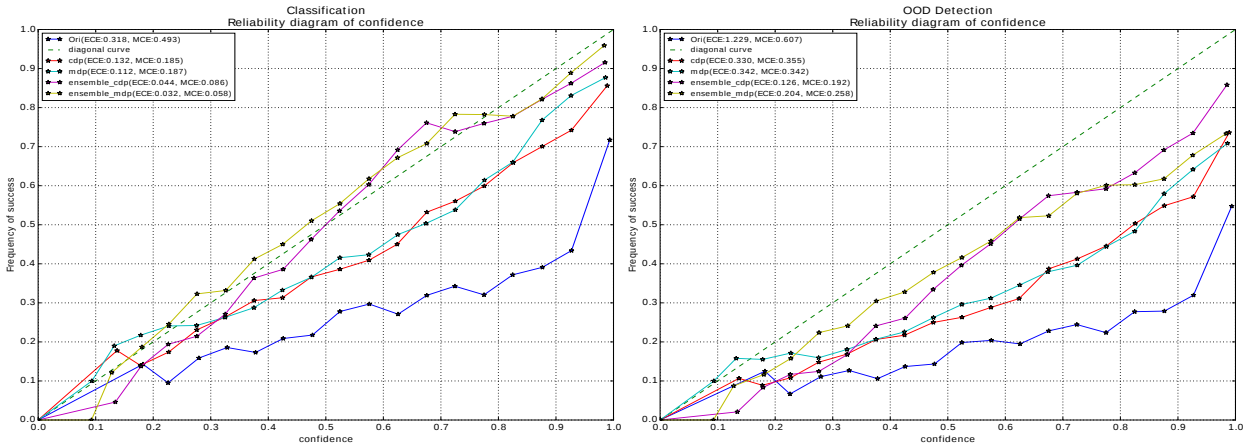


Figure 4.7: Calibration curve of one of three runs.

ROC curve and PR curve This is ROC and PR curve

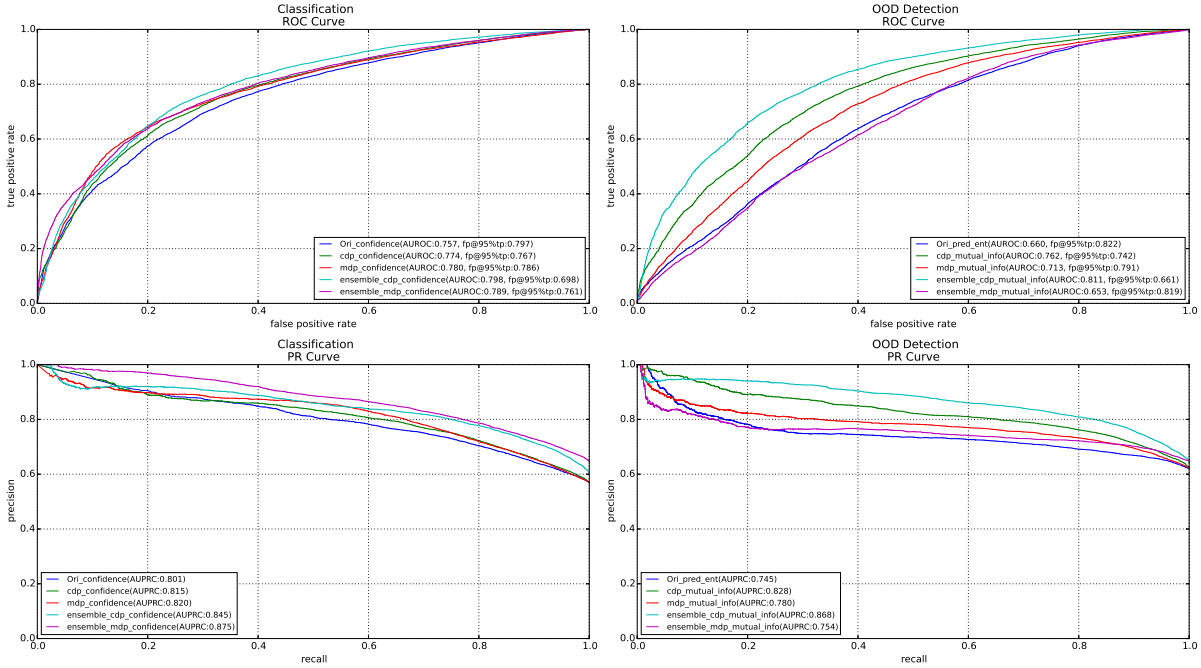


Figure 4.8: ROC and PR curve of one of three runs.

Table 4.1: results of acc, bs, nll, ece, mce, auroc, aupr

	accuracy↑	brier_score ↓	negative log likelihood ↓
ori	0.568±0.008	0.722±0.019	3.242±0.340
cdp	0.577±0.008	0.594±0.013	2.088±0.181
mdp	0.599±0.023	0.566±0.020	1.940±0.064
emsemble_cdp	0.604	0.534	1.452
emsemble_mdp	0.645	0.496	1.389

	expected calibration error(w/o. OOD/ w. OOD)↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
ori	0.304±0.016 / 0.633±0.065	0.461±0.027 / 0.362±0.025	0.750±0.007 / 0.664±0.011	0.802±0.008 / 0.751±0.018
cdp	0.124±0.023/ 0.288±0.048	0.206±0.015/ 0.374±0.018	0.775±0.008/ 0.783±0.022	0.825±0.007/ 0.850±0.022
mdp	0.114±0.012/ 0.383±0.046	0.199±0.016/ 0.367±0.023	0.780±0.011/ 0.709±0.004	0.838±0.013/ 0.788±0.006
emsemble_cdp	0.044/ 0.042	0.086/ 0.093	0.798/ 0.811	0.845/ 0.868
emsemble_mdp	0.032/ 0.170	0.058/ 0.227	0.789/ 0.653	0.875/ 0.754

4.2.4 Comparison with Laplace approximation

Because Laplace approximation requires only MAP point estimate of parameter, re-training the model is not needed. Therefore we take the already trained model of different approaches as our MAP point estimate. We set the scale parameter of kronecker factors as 1 and dump factor as 15 based on grid search on the validation set. his is uncertainty hitsogram

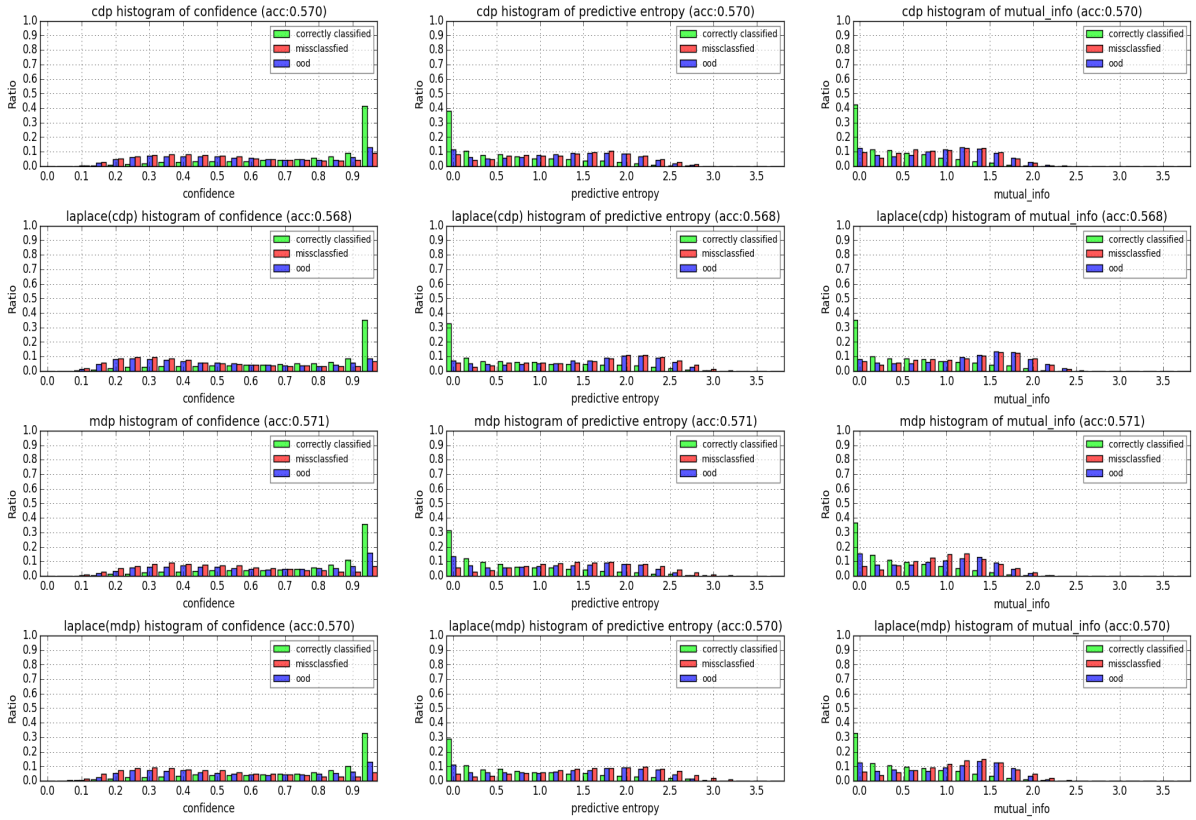


Figure 4.9: Uncertainty histogram of one of three runs.

This is Reliability diagram

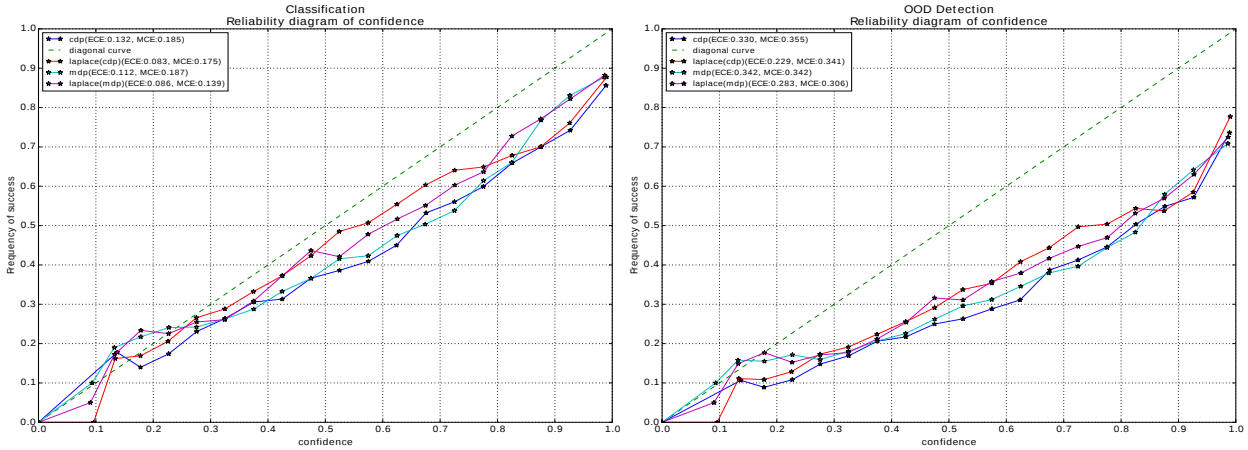


Figure 4.10: Calibration curve of one of three runs.

This is ROC and PR curve

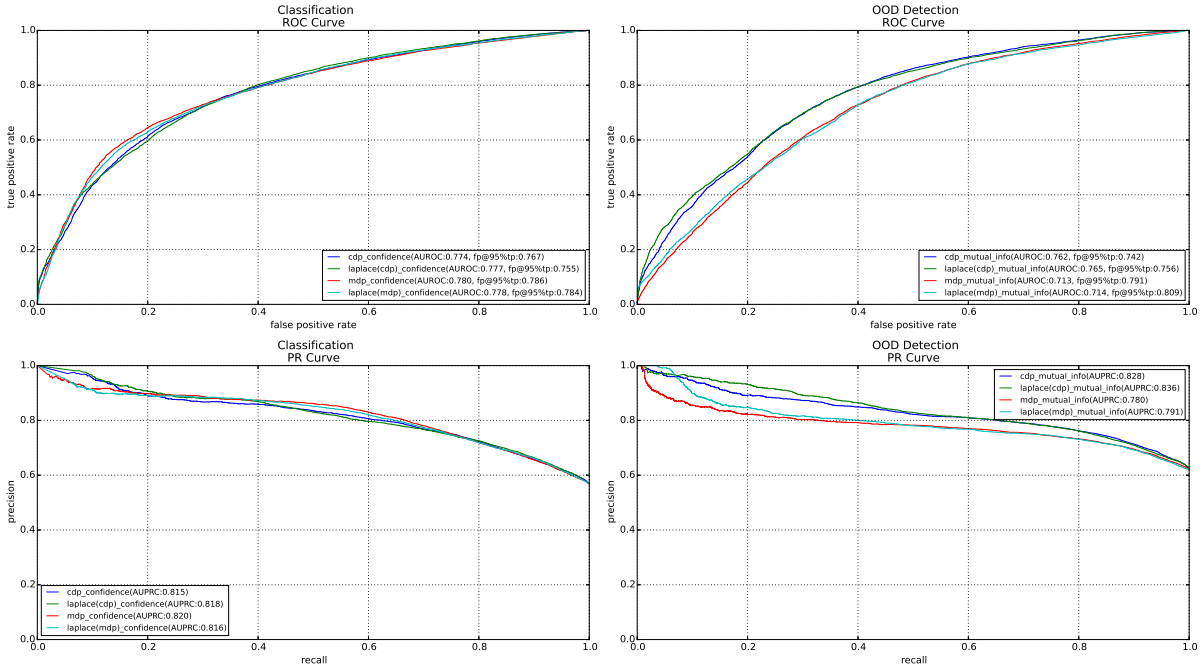


Figure 4.11: ROC and PR curve of one of three runs.

	accuracy ↑	brier score ↓	negative log likelihood ↓
cdp	0.577±0.008	0.594±0.013	2.088±0.181
laplace_cdp	0.576±0.009	0.602±0.011	2.322±0.350
mdp	0.599±0.023	0.566±0.020	1.940±0.064
laplace_mdp	0.598±0.024	0.567±0.018	1.970±0.117

	expected calibration error(w/o. OOD/ w. OOD) ↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
cdp	0.124±0.023/ 0.288±0.048	0.206±0.015/ 0.374±0.018	0.775±0.008/ 0.783±0.022	0.825±0.007/ 0.850±0.022
laplace_cdp	0.129±0.058/ 0.341±0.157	0.235±0.073/ 0.406±0.070	0.779±0.004/ 0.782±0.017	0.826±0.007/ 0.849±0.016
mdp	0.114±0.012/ 0.383±0.046	0.199±0.016/ 0.367±0.023	0.780±0.011/ 0.709±0.004	0.838±0.013/ 0.788±0.006
laplace_mdp	0.104±0.018/ 0.352±0.061	0.179±0.029/ 0.352±0.038	0.776±0.012/ 0.711±0.005	0.837±0.015/ 0.798±0.005

4.2.5 Ablation study

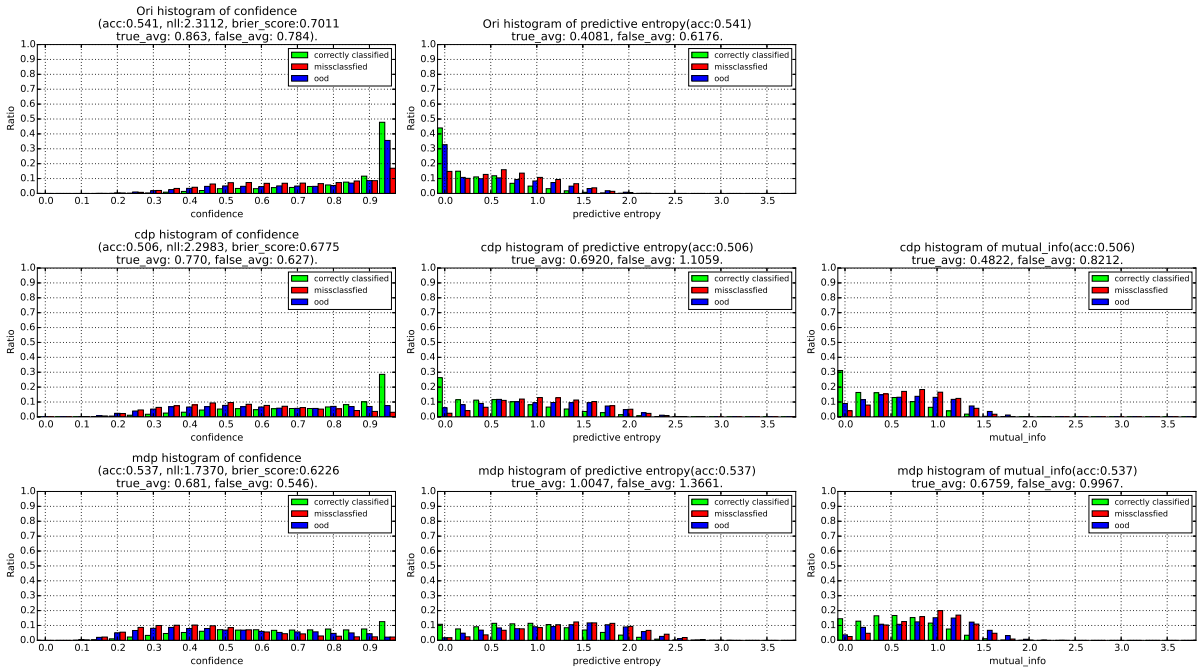


Figure 4.12: Uncertainty histogram of one of three runs.

This is Reliability diagram

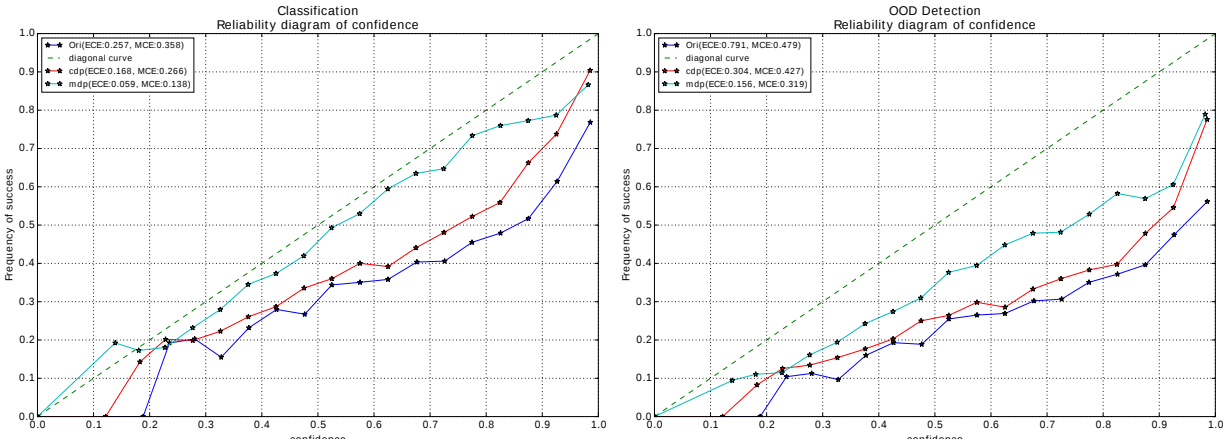


Figure 4.13: Calibration curve of one of three runs.

This is ROC and PR curve

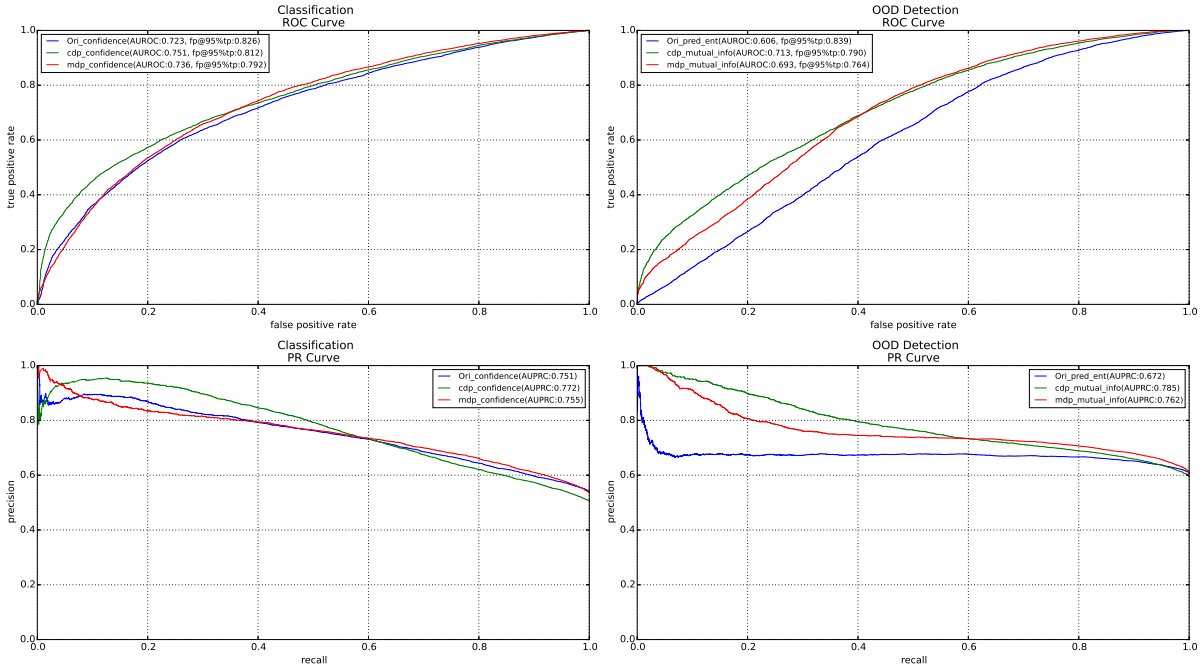


Figure 4.14: ROC and PR curve of one of three runs.

Table 4.2: results of acc, bs, nll, ece, mce, auroc, aupr

	accuracy ↑	brier_score ↓	negative log likelihood ↓
ori	0.532±0.015	0.717±0.023	2.383±0.053
cdp	0.521±0.011	0.655±0.017	2.190±0.149
mdp	0.520±0.012	0.641±0.014	1.804±0.073

	expected calibration error(w/o. OOD/ w. OOD)↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
ori	0.262±0.011/ 0.793±0.012	0.365±0.010/ 0.491±0.013	0.716±0.005/ 0.602±0.014	0.724±0.022/ 0.681±0.007
cdp	0.141±0.022/ 0.270±0.037	0.203±0.053/ 0.359±0.056	0.748±0.002/ 0.712±0.006	0.770±0.002/ 0.782±0.004
mdp	0.068±0.009/ 0.158±0.006	0.134±0.018/ 0.311±0.008	0.730±0.009/ 0.674±0.014	0.749±0.009/ 0.748±0.010

4.3 Automatic labeling experiments

4.3.1 Experiments settings

4.3.2 Results

4.3.3 Analysis

4.4 Context-based improvement experiments

4.4.1 Experiments settings

4.4.2 Results

4.4.3 Analysis

Chapter 5

Conclusion

Am Schluß werden noch einmal alle wesentlichen Ergebnisse zusammengefaßt. Hier können auch gemachte Erfahrungen beschrieben werden. Am Ende der Zusammenfassung kann auch ein Ausblick folgen, der die zukünftige Entwicklung der behandelten Thematik aus der Sicht des Autors darstellt.

Appendix A

Appendix

A.1 KL condition

Proposition 1. Fix K, $L \in \mathbb{N}$, a probability vector $\boldsymbol{p} = (p_1, ..., p_L)$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{K \times K}$ diagonal positive definite for $i = 1, ..., L$, with the elements of each $\boldsymbol{\Sigma}_i$ not dependent on K. Let

$$q(\boldsymbol{x}) = \sum_{i=1}^{L} p_i \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

be a mixture of Gaussians with L components and $\boldsymbol{\mu}_i \in \mathbb{R}^K$, let $p(\boldsymbol{x}) = \mathcal{N}(0, \boldsymbol{I}_K)$, and further assume that $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j \sim \mathcal{N}(0, \boldsymbol{I})$ for all i and j.

The KL divergence between $q(\boldsymbol{x})$ and $p(\boldsymbol{x})$ can be approximated as:

$$KL(q(\boldsymbol{x})||p(\boldsymbol{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2} (\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + tr(\boldsymbol{\Sigma}_i) - K(1 + \log 2\pi) - \log(\det(\boldsymbol{\Sigma}_i))) - \mathcal{H}(\boldsymbol{p})$$

with

$$\mathcal{H}(\boldsymbol{p}) = -\sum_{i=1}^{L} p_i \log p_i$$

for large enough K.

Proof. We have

$$\begin{aligned} KL(q(\boldsymbol{x})||p(\boldsymbol{x})) &= \int q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x} \\ &= \int q(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} - \int q(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} \\ &= -\mathcal{H}(q(\boldsymbol{x})) - \int q(\boldsymbol{x}) \log\, p(\boldsymbol{x}) d\boldsymbol{x} \end{aligned}$$

which is sum of entropy of $q(\boldsymbol{x})$ and the expected log probability of $\boldsymbol{x}$. The expected log probability can be evaluated analytically, but the entropy term has to be approximated.

We begin by approximating the entropy term. We write

$$\begin{aligned}\mathcal{H}(q(\boldsymbol{x})) &= -\sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \log\, q(\boldsymbol{x}) d\boldsymbol{x} \\ &= -\sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \boldsymbol{I}) \log\, q(\boldsymbol{\mu}_i + \boldsymbol{L}_i\boldsymbol{\epsilon}_i) d\boldsymbol{\epsilon}_i\end{aligned}$$

with a re-parameterization of $\boldsymbol{x}$, that is $\boldsymbol{x} = \boldsymbol{\mu}_i + \boldsymbol{L}_i\boldsymbol{\epsilon}_i$ with $\boldsymbol{L}_i\boldsymbol{L}_i^T = \boldsymbol{\Sigma}_i$ and $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, I)$.

Now the term inside logarithm can be written as

$$\begin{aligned}q(\boldsymbol{\mu}_i + \boldsymbol{L}_i\boldsymbol{\epsilon}_i) &= \sum_{j=1}^{L} p_j \mathcal{N}(\boldsymbol{\mu}_i + \boldsymbol{L}_i\boldsymbol{\epsilon}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \\ &= \sum_{j=1}^{L} p_j (2\pi)^{-\frac{K}{2}} \det(\boldsymbol{\Sigma}_j)^{-\frac{1}{2}} \exp\, \{-\frac{1}{2}||\boldsymbol{\mu}_j - \boldsymbol{\mu}_i - \boldsymbol{L}_i\boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_j}\}\end{aligned}$$

where $||\cdot||_{\boldsymbol{\Sigma}}$ is Mahalanobis distance with $\boldsymbol{\Sigma}$ as covariance matrix. Since $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j$ is assumed to be normally distributed, the quantity $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j - \boldsymbol{L}_i\boldsymbol{\epsilon}_i$ is also normally distributed. Since the expectation of a generalized $\mathcal{X}^2$ distribution with K degrees of freedom increases with K, we have that $K \gg 0$ implies that $||\boldsymbol{\mu}_i - \boldsymbol{\mu}_j - \boldsymbol{L}_i\boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_j} \gg 0$ for $i \neq j$(since the elements of $\boldsymbol{\Sigma}_j$ does not depend on K). Finally, we have for $i = j$ that $||\boldsymbol{\mu}_i - \boldsymbol{\mu}_j - \boldsymbol{L}_i\boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_j} = \boldsymbol{\epsilon}_i^T\boldsymbol{L}_i^T\boldsymbol{L}_i^{-T}\boldsymbol{L}_i^{-1}\boldsymbol{L}_i\boldsymbol{\epsilon}_i = \boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i$. Therefore the last equation can be approximated as

$$q(\boldsymbol{\mu}_i + \boldsymbol{L}_i\boldsymbol{\epsilon}_i) \approx p_i (2\pi)^{-\frac{K}{2}} \det(\boldsymbol{\Sigma}_i)^{-\frac{1}{2}} \exp\, \{-\frac{1}{2}\boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i\}$$

I.e. in high dimensions the mixture components will not overlap.This gives us

$$\begin{aligned}\mathcal{H}(q(\boldsymbol{x})) &\approx -\sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \boldsymbol{I}) \log\, \big(p_i (2\pi)^{-\frac{K}{2}} \det(\boldsymbol{\Sigma}_i)^{-\frac{1}{2}} \exp\, \{-\frac{1}{2}\boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i\}\big) d\boldsymbol{\epsilon}_i \\ &= -\sum_{i=1}^{L} p_i \big[\log\, p_i - \frac{K}{2}\log\, 2\pi - \frac{1}{2}\log\, \det(\boldsymbol{\Sigma}_i) - \frac{1}{2}\int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \boldsymbol{I}) \boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i d\boldsymbol{\epsilon}_i\big] \\ &= \sum_{i=1}^{L} \frac{p_i}{2}\big(K\log\, 2\pi + \log\, \det(\boldsymbol{\Sigma}_i) + \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \boldsymbol{I}) \boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i d\boldsymbol{\epsilon}_i\big) + \mathcal{H}(\boldsymbol{p})\end{aligned}$$

Since $\boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_i$ distributes according to a $\mathcal{X}^2$ distribution , its expectation is K, and in the end the entropy term can be approximated as

$$\mathcal{H}(q(\boldsymbol{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2}\big(K(\log\, 2\pi + 1) + \log\, \det(\boldsymbol{\Sigma}_i)\big) + \mathcal{H}(\boldsymbol{p})$$

Next, we can evaluate the expected log probability term, we get

$$\int q(\boldsymbol{x})\log\, p(\boldsymbol{x})d\boldsymbol{x} = \sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\log\, p(\boldsymbol{x})d\boldsymbol{x}$$

for $p(\boldsymbol{x}) = \mathcal{N}(0, \boldsymbol{I}_K)$, it is easy to show that

$$\begin{aligned}\int q(\boldsymbol{x})\log\, p(\boldsymbol{x})d\boldsymbol{x} &= \sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\log\big[(2\pi)^{-\frac{K}{2}}\det(\boldsymbol{I}_K)^{-\frac{1}{2}}\exp\{-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\}\big]d\boldsymbol{x} \\ &= \sum_{i=1}^{L} p_i\big[-\frac{K}{2}\log 2\pi - \frac{1}{2}\int \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\boldsymbol{x}^T\boldsymbol{x}d\boldsymbol{x}\big] \\ &= -\frac{1}{2}\sum_{i=1}^{L} p_i(\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + tr(\boldsymbol{\Sigma}_i) + K\log 2\pi)\end{aligned}$$

Finally, combining the equations above, we have

$$KL(q(\boldsymbol{x})||p(\boldsymbol{x})) \approx \sum_{i=1}^{L}\frac{p_i}{2}(\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + tr(\boldsymbol{\Sigma}_i) - K - \log(\det(\boldsymbol{\Sigma}_i))) - \mathcal{H}(\boldsymbol{p})$$

as required to show.

□

A.2 Figures

Beispiel für eine Tabelle:

Table A.1: Beispiel für eine Beschriftung. Tabellenbeschriftungen sind üblicherweise über der Tabelle platziert.

left	center	right
entry	entry	entry
entry	entry	entry
entry	entry	entry

A.3 Implementation Details

List of Figures

List of Tables

Bibliography

[AOS+16] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[BCKW15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BRMW15] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

[DKD09] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

[DL91] John S Denker and Yann Lecun. Transforming neural-net output levels to probability distributions. In *Advances in neural information processing systems*, pages 853–859, 1991.

[DT18] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[FCSG17] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[FRD18] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. *arXiv preprint arXiv:1804.05132*, 2018.

[Gal] Yarin Gal. *Uncertainty in deep learning*. PhD thesis.

[GBR07] Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic

forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[GHK17] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[GIG17] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

[GMR] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models.

[GN99] AK Gupta and DK Nagar. *Matrix Variate Distributions*, volume 104. CRC Press, 1999.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

[GR07] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[Gra11] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[GRB08] Carolina Galleguillos, Andrew Rabinovich, and Serge Belongie. Object categorization using co-occurrence, location and appearance. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[HG16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[HHGL11] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[HHO+17] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[HLA15] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backprop-

agation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[HVC93] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[KC16] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE international conference on Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016.

[KFB09] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. 2009.

[KG17] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[KGC] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.

[KK11] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

[KSW15] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LAA+17] Christian Leibig, Vaneeda Allken, Murat Seçkin Ayhan, Philipp Berens, and Siegfried Wahl. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports*, 7(1):17816, 2017.

[LBRF11] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchi-

cal multi-view rgb-d object dataset. In *2011 IEEE international conference on robotics and automation*, pages 1817–1824. IEEE, 2011.

[LLLS17] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*, 2017.

[LLS17] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[LMP01] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[LRKT10] Lubor Ladicky, Chris Russell, Pushmeet Kohli, and Philip HS Torr. Graph cut based inference with co-occurrence statistics. In *European Conference on Computer Vision*, pages 239–253. Springer, 2010.

[LSVDHR16] Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3194–3203, 2016.

[LW16] Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

[LW17] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

[Mac92] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[MG15] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[MGK+17] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

[Min01] Thomas P Minka. Expectation propagation for approximate bayesian infer-

ence. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[MMT16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[MTM14] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094, 2014.

[Nea12] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[OBPVR16] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[QCRS+05] Joaquin Quinonero-Candela, Carl Edward Rasmussen, Fabian Sinz, Olivier Bousquet, and Bernhard Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges Workshop*, pages 1–27. Springer, 2005.

[RBB18] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. 2018.

[RV09] Nikhil Rasiwasia and Nuno Vasconcelos. Holistic context modeling using semantic co-occurrences. 2009.

[RVG+07] Andrew Rabinovich, Andrea Vedaldi, Carolina Galleguillos, Eric Wiewiora, and Serge Belongie. Objects in context. In *Computer vision, 2007. ICCV 2007. IEEE 11th international conference on*, pages 1–8. IEEE, 2007.

[SBD+14] Robin Senge, Stefan Bösner, Krzysztof Dembczyński, Jörg Haasenritter, Oliver Hirsch, Norbert Donner-Banzhoff, and Eyke Hüllermeier. Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Information Sciences*, 255:16–29, 2014.

[SCC17] Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pages 1283–1292, 2017.

[SG18] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks

from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SM+12] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WT11] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[WVL+18] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*, 2018.

[ZSDG17] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2017.