Uncertainty in Deep Learning

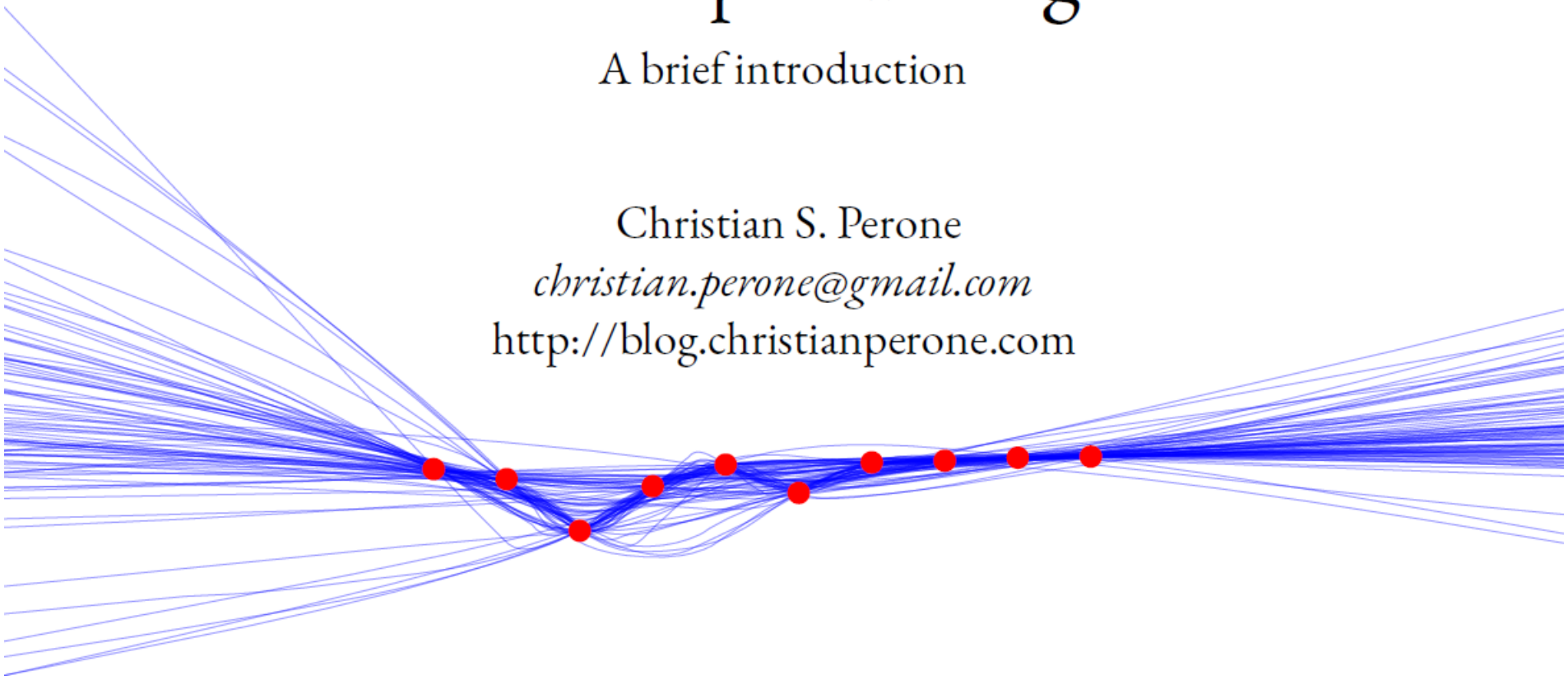
Yarin Gal

2018.7.29

Uncertainty Estimation in Deep Learning

A brief introduction

Christian S. Perone
christian.perone@gmail.com
http://blog.christianperone.com



Different Uncertainties

Two main types of uncertainty, often confused by practitioners, but very different quantities:

Aleatoric uncertainty

Information data cannot explain, also called data uncertainty, or irreducible uncertainty. More data might not reduce it;
Ex: increasing measurement precision can reduce it.

Epistemic uncertainty

Uncertainty in the model itself, also called model uncertainty, or reducible uncertainty;
Ex: can be explained away by increasing training size

Importance of Uncertainty

- Autonomous vehicles (what's the uncertainty this object is a tree ?);
- Active Learning (which sample should be labeled ?);
- Explore/exploit dilemma in reinforcement learning;
- Model understanding/dataset understanding;
- ⋮

A simple frequentist regression

In a frequentist linear regression, we have a point estimate for the parameters of our model.

First, we define our model:

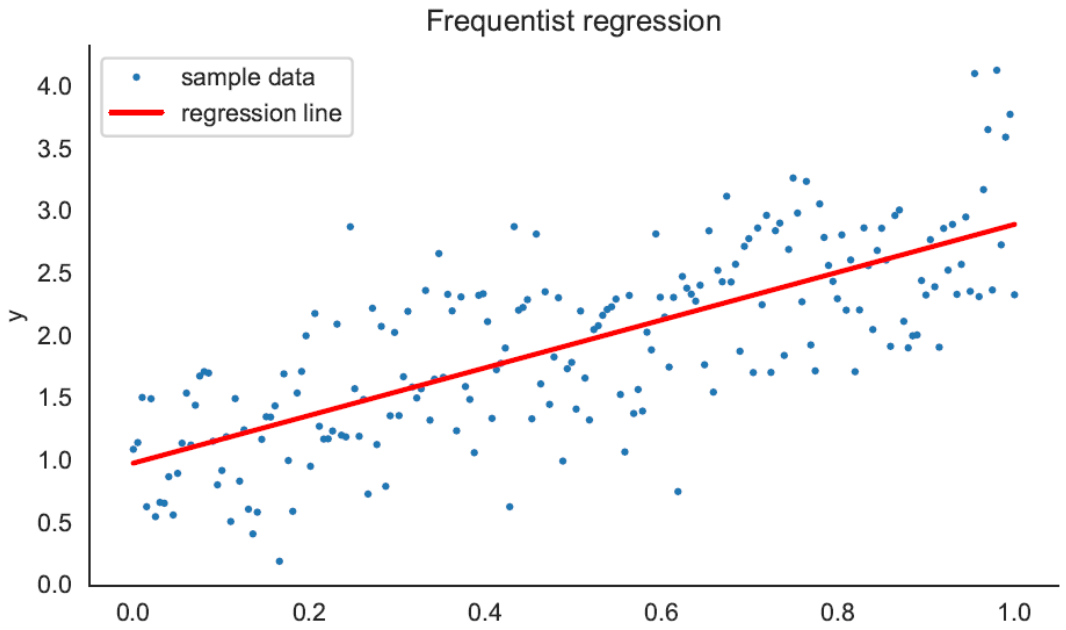
$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots = \overbrace{\mathbf{x}^\intercal \beta}^{\text{Vectorial notation}}$$

Later, we denote a loss such as the MSE (mean squared error):

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2$$

Finally, we optimize it:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(f(\mathbf{x}), y)$$



$$\mathbf{X} = \{x_1, ..., x_n\} \quad \mathbf{Y} = \{y_1, ..., y_n\} \qquad y = f(x)$$

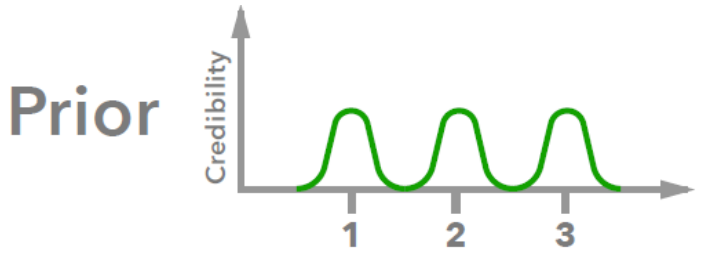
Since W is not deterministic, the output of neural network is also a random variable. Prediction for a new input X can be formed by integrating with respect to the posterior distribution of W as follows:

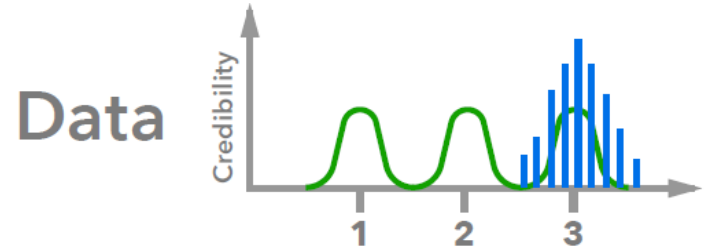
$$p(y|x, \mathbf{X}, \mathbf{Y}) = \int p(y|x, W)p(W|\mathbf{X}, \mathbf{Y})\, dW$$

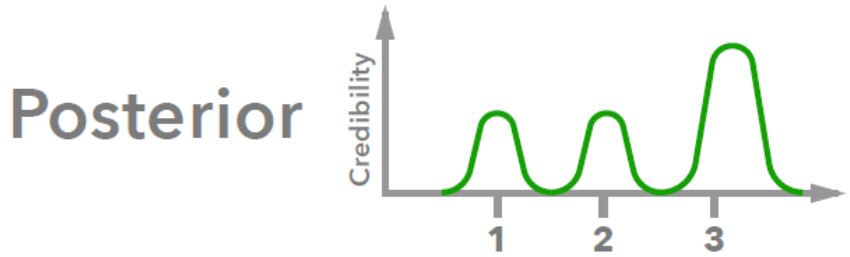
Bayesian approaches represent the uncertainty using **a distribution over parameters**. Instead of a **point estimate**, we have an entire posterior.

- To formulate our bayesian regression, we first select a likelihood
- After that, we select priors over parameters;
- Then we compute or approximate (sampling) the posterior of our model and data.

$$\overbrace{p(\theta|\mathbf{X})}^{\text{Posterior}} \propto \underbrace{p(\mathbf{X}|\theta)}_{\text{Likelihood}} \overbrace{\pi(\theta)}^{\text{Prior}}$$







Bayesian regression

We will use a simple Gaussian distribution for our observations, defined as:

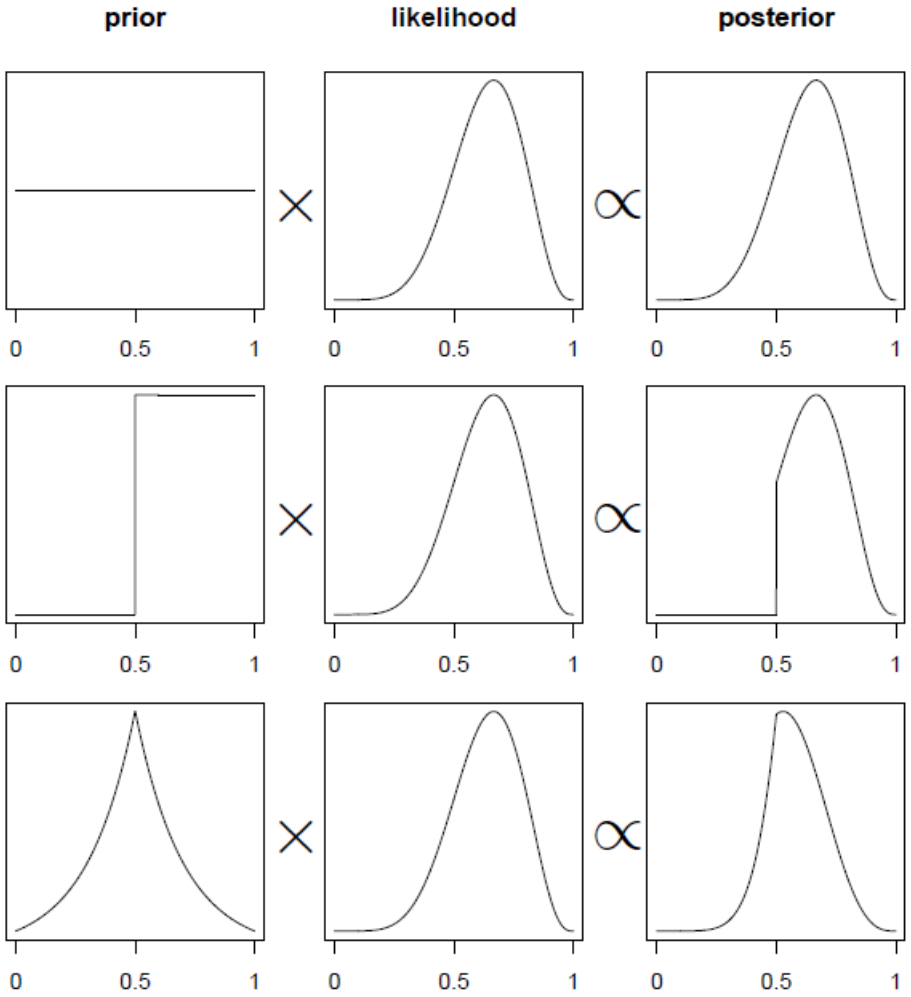
$$Y \sim \mathcal{N}(\mu, \sigma^2)$$

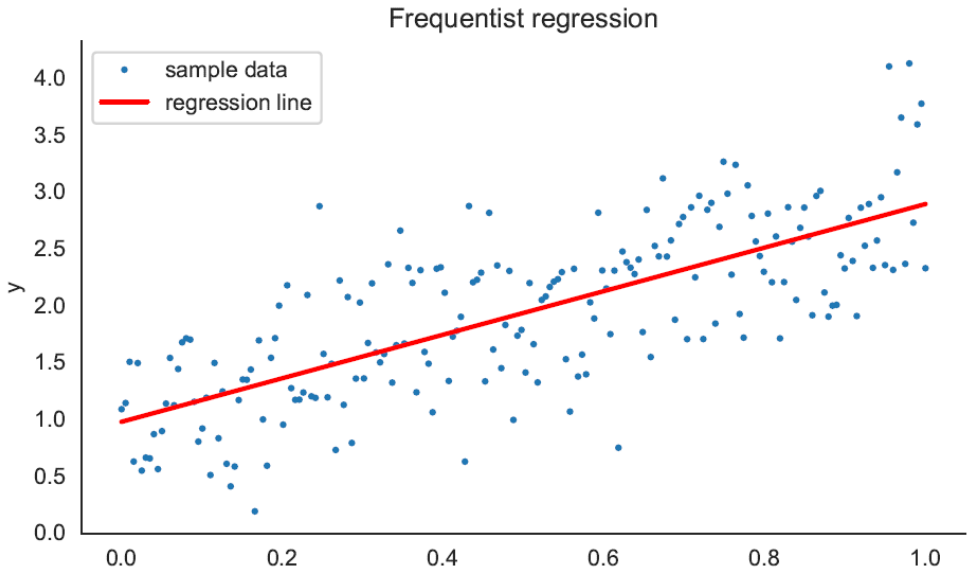
We plug our regression of the μ:

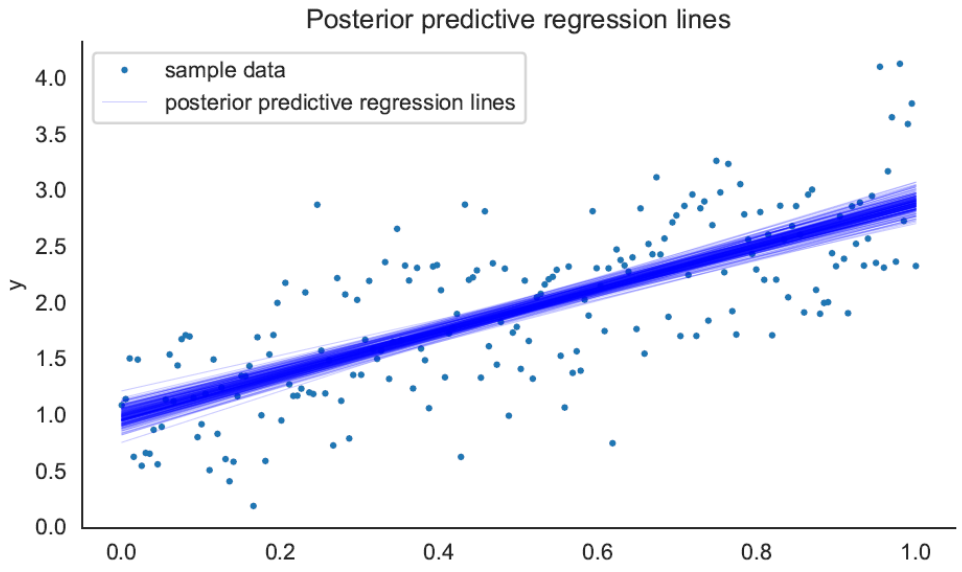
$$Y \sim \mathcal{N}(\underbrace{\alpha + \beta x}_{\text{Linear model}}, \sigma^2)$$

And define the priors:

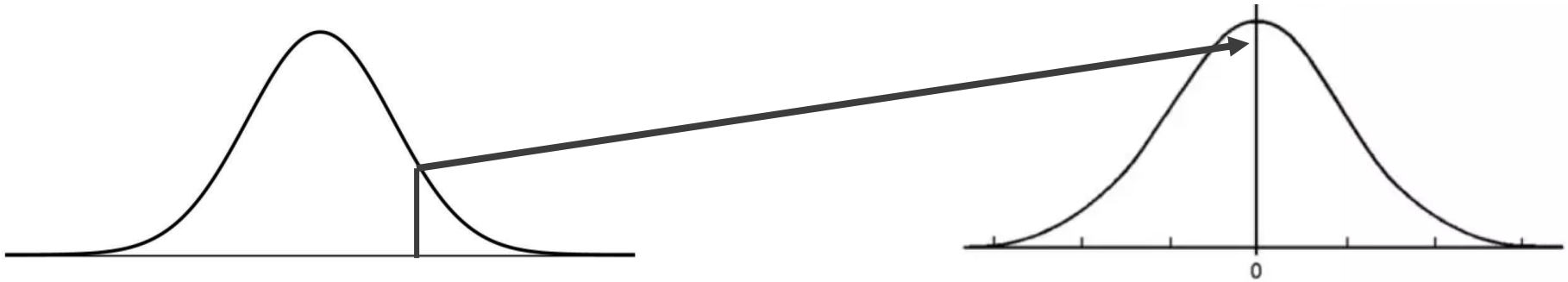
$$\alpha \sim \mathcal{N}(0, 20)$$
$$\beta \sim \mathcal{N}(0, 20)$$
$$\sigma \sim \mathcal{U}(0, 5)$$







$$p(y|x, \mathbf{X}, \mathbf{Y}) = \int p(y|x, W)p(W|\mathbf{X}, \mathbf{Y})\, dW$$



model

uncertainty of x_i

Bayesian methods

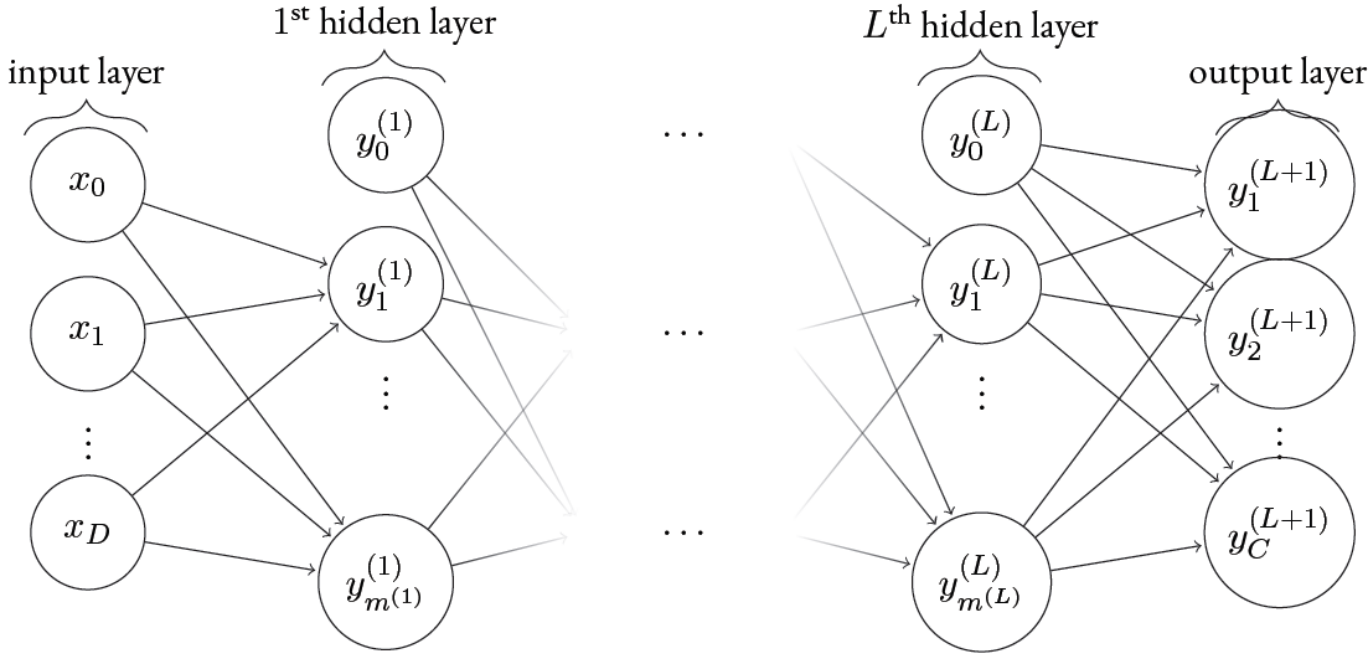
- Bayesian methods can give us a full posterior to reason about;
- Explicit priors;
- Uncertainty;
- They’re on the side of algorithms, not models

However

Intractable posterior for many practical cases and large datasets;

Tuning and using MCMC algorithms can be tricky.

$$p(\theta|\mathbf{X}) = \frac{p(\mathbf{X}|\theta)\pi(\theta)}{p(\mathbf{X})}$$



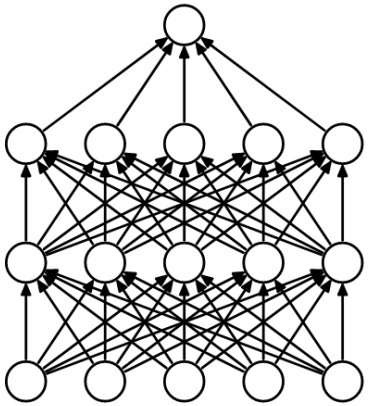
- Parametrized models with composition of functions;
- Trained using backpropagation and SGD;
- Learned usually by maximizing the log likelihood;

In modern Deep Neural Networks, however, we have some challenges:

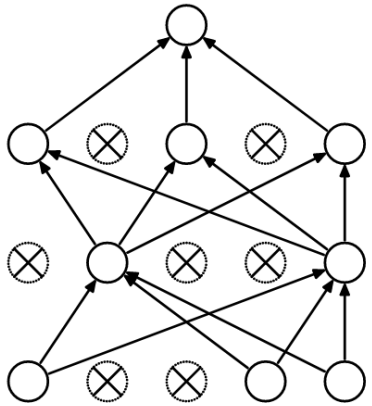
- A lot of data;
- High-dimensionality in data;
- Millions of parameters;
- Highly non-convex surfaces;

This makes these models very difficult for Bayesian methods, therefore an approximation is required:

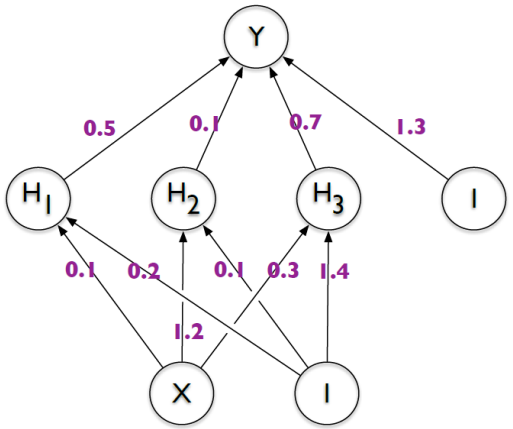
Variational Inference

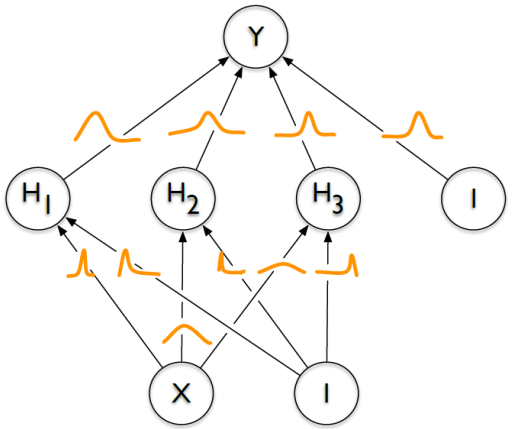


(a) Standard Neural Net



(b) After applying dropout.





Dropout as a Bayesian Approximation: Insights and Applications ICML2015

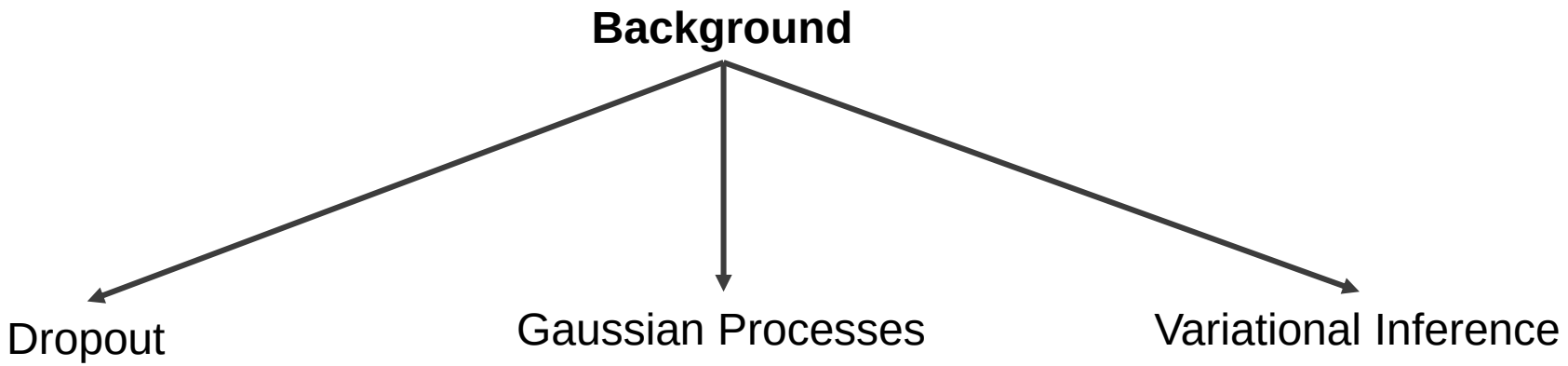
Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning ICML2016

BAYESIAN CONVOLUTIONAL NEURAL NETWORKS WITH BERNOULLI APPROXIMATE VARIATIONAL INFERENCE ICLR2016

Deep Bayesian Active Learning with Image Data NIPS2017

BatchBALD: Ecient and Diverse Batch Acquisition for Deep Bayesian Active Learning arXiv2019.6

A deep neural network (NN) with arbitrary depth and non-linearities, with dropout applied before every weight layer, is mathematically equivalent to an approximation to the probabilistic deep Gaussian process model



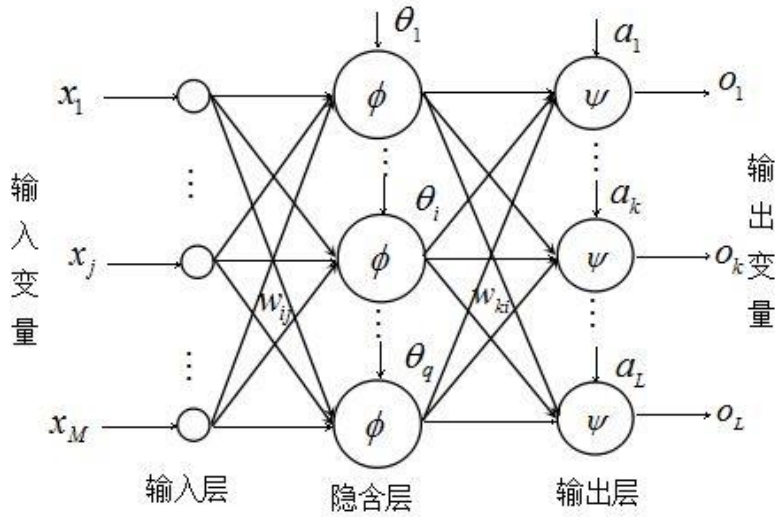
Dropout

We review the dropout NN model for the case of a single hidden layer NN.

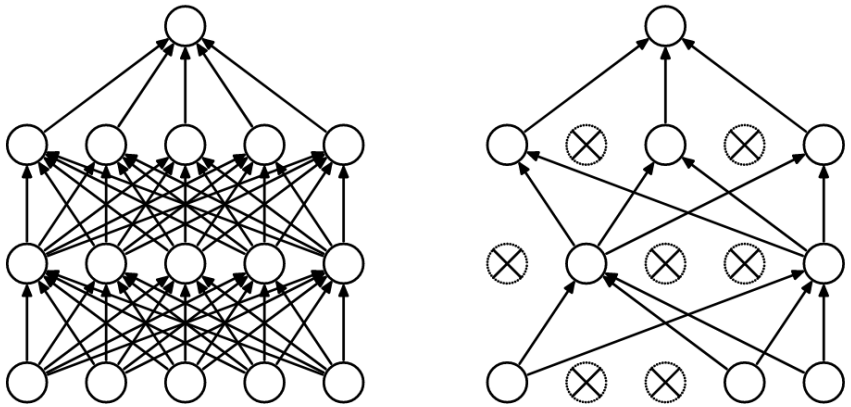
$$\mathbf{W}_1 : Q \times K, \mathbf{W}_2 : K \times D$$

$$\widehat{\mathbf{y}} = \sigma(\mathbf{x}\mathbf{W}_1 + \mathbf{b})\mathbf{W}_2$$

$\mathbf{b}$ is a K dimensional vector.



sampling two binary vectors $\mathbf{z}_1, \mathbf{z}_2$ of dimensions Q and K respectively.



(a) Standard Neural Net

(b) After applying dropout.

$$\mathbf{z}_{1,q} \sim \text{Bernoulli}(p_1) \text{ for } q = 1, ..., Q$$

$$\mathbf{z}_{2,k} \sim \text{Bernoulli}(p_2) \text{ for } k = 1, ..., K.$$

The output of the first layer is given by

$$\sigma((\mathbf{x} \circ \mathbf{z}_1)\mathbf{W}_1 + \mathbf{b}) \circ \mathbf{z}_2$$

Which is linearly transformed to give the dropout model's output

$$\widehat{\mathbf{y}} = \big((\sigma((\mathbf{x} \circ \mathbf{z}_1)\mathbf{W}_1 + \mathbf{b})) \circ \mathbf{z}_2\big)\mathbf{W}_2$$

To use the NN model for regression, we might use the Euclidean loss

$$E = \frac{1}{2N} \sum_{n=1}^{N} ||\mathbf{y}_n - \widehat{\mathbf{y}}_n||_2^2$$

To use the model for classification

$$\widehat{p}_{nd} = \exp(\widehat{y}_{nd}) / \left(\sum_{d'} \exp(\widehat{y}_{nd'})\right) \qquad E = -\frac{1}{N} \sum_{n=1}^{N} \log(\widehat{p}_{n,c_n})$$

During optimisation a regularisation term is often added.

$$\mathcal{L}_{\text{dropout}} := E + \lambda_1 ||\mathbf{W}_1||_2^2 + \lambda_2 ||\mathbf{W}_2||_2^2 + \lambda_3 ||\mathbf{b}||_2^2.$$

We sample new realisations for the binary vectors $\mathbf{z}_i$ for every input point and every forward pass thorough the model (evaluating the model's output), and use the same values in the backward pass (propagating the derivatives to the parameters).

The dropped weights $\mathbf{z}_1 \mathbf{W}_1$ and $\mathbf{z}_2 \mathbf{W}_2$ are often scaled by $\frac{1}{p_i}$ to maintain constant output magnitude. At test time no sampling takes place. This is equivalent to initialising the weights $\mathbf{W}_i$ with scale $\frac{1}{p_i}$ with no further scaling at training time, and at test time scaling the weights $\mathbf{W}_i$ by p_i.

Gaussian Processes

Given a training dataset consisting of N inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and their corresponding outputs $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, we would like to estimate a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ that is likely to have generated our observations. We denote the inputs $\mathbf{X} \in \mathbb{R}^{N \times Q}$ and the outputs $\mathbf{Y} \in \mathbb{R}^{N \times D}$.

Following the Bayesian approach we would put some prior distribution over the space of functions $p(\mathbf{f})$

This distribution represents our prior belief as to which functions are more likely and which are less likely to have generated our data.

We then look for the posterior distribution over the space of functions

$$p(\mathbf{f}|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \mathbf{f})p(\mathbf{f})$$

We place a joint Gaussian distribution over all function values

$$\mathbf{F} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})) \qquad \mathbf{Y} \mid \mathbf{F} \sim \mathcal{N}(\mathbf{F}, \tau^{-1}\mathbf{I}_N)$$

For classification we sample from a categorical distribution with probabilities given by passing $\mathbf{Y}$ through an element-wise softmax,

$$\mathbf{F} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})) \qquad \mathbf{Y} \mid \mathbf{F} \sim \mathcal{N}(\mathbf{F}, 0 \cdot \mathbf{I}_N)$$

$$c_n \mid \mathbf{Y} \sim \text{Categorical}\left(\exp(y_{nd}) / \left(\sum_{d'} \exp(y_{nd'})\right)\right)$$

Evaluating the Gaussian distribution above involves an inversion of an N by N matrix, an operation that requires $\mathcal{O}(N^3)$ time complexity. Many approximations to the Gaussian process result in a manageable time complexity. Variational inference can be used for such, and will be explained next.

Variational Inference

Consider a probabilistic model in which we collectively denote all of the observed variables by $\mathbf{X}$ and all of the hidden variables by $\mathbf{Z}$. The joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is governed by a set of parameters denoted $\boldsymbol{\theta}$. Our goal is to maximize the likelihood function that is given by

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

make use of the product rule of probability

$$\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \ln p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) + \ln p(\mathbf{X}|\boldsymbol{\theta})$$

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \ln \left\{ \frac{p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})} \right\} \qquad \mathrm{KL}(q \,\|\, p) = -\sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \ln \left\{ \frac{p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta})}{q(\boldsymbol{Z})} \right\}$$

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q\|p)$$

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \qquad \mathrm{KL}(q\|p) = -\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}$$

Given training inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and their corresponding outputs $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$

we would like to estimate a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ that is *likely to have generated our outputs*.

What is a function that is likely to have generated our data?

Following the Bayesian approach we would put some prior distribution over the space of functions $p(\mathbf{f})$

This distribution represents our prior belief as to which functions are likely to have generated our data.

We define a likelihood $p(\mathbf{Y}|\mathbf{f}, \mathbf{X})$ to capture the process in which observations are generated given a specific function.

We then look for the posterior distribution over the space of functions given our dataset: $p(\mathbf{f}|\mathbf{X}, \mathbf{Y})$

This distribution captures the most likely functions given our observed data.

predict an output for a new input point $\mathbf{x}^*$ by integrating over all possible functions $\mathbf{f}$

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\mathrm{d}\mathbf{f}^*$$

To approximate it we could condition the model on a finite set of random variables $\boldsymbol{\omega}$

We make a modelling assumption and assume that the model depends on these variables alone, making them into sufficient statistics in our approximate model.

The predictive distribution for a new input point $\mathbf{x}^*$ is then given by

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\ \mathrm{d}\mathbf{f}^*\mathrm{d}\boldsymbol{\omega}.$$

The distribution $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ cannot usually be evaluated analytically as well. Instead we define an approximating *variational* distribution $q(\boldsymbol{\omega})$

We thus minimise the Kullback–Leibler (KL) divergence $\mathrm{KL}(q(\boldsymbol{\omega}) \,||\, p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}))$

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\, \mathrm{d}\mathbf{f}^*\mathrm{d}\boldsymbol{\omega}.$$

$$q(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \boldsymbol{\omega})q(\boldsymbol{\omega})\mathrm{d}\mathbf{f}^*\mathrm{d}\boldsymbol{\omega}.$$

Minimising the Kullback–Leibler divergence is equivalent to maximising the *log evidence lower bound*,

$$\mathcal{L}_{\mathrm{VI}} := \int q(\boldsymbol{\omega})p(\mathbf{F}|\mathbf{X}, \boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{F})\mathrm{d}\mathbf{F}\mathrm{d}\boldsymbol{\omega} - \mathrm{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

This is known as variational inference, a standard technique in Bayesian modelling.

BAYESIAN NEURAL NETWORKS

One defines a Bayesian NN by placing a prior distribution over a NN's weights. Given weight matrices $\mathbf{W}_i$ and bias vectors $\mathbf{b}_i$ for layer i, we often place standard matrix Gaussian prior distributions over the weight matrices, $p(\mathbf{W}_i)$:

$$\mathbf{W}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

denote the random output of a NN with weight random variables $(\mathbf{W}_i)_{i=1}^L$ on input $\mathbf{x}$ by $\widehat{\mathbf{f}}\left(\mathbf{x}, (\mathbf{W}_i)_{i=1}^L\right)$

$$p\left(y|\mathbf{x}, (\mathbf{W}_i)_{i=1}^L\right) = \text{Categorical}\left(\exp(\widehat{\mathbf{f}}) / \sum_{d'} \exp(\widehat{f}_{d'})\right) \qquad \widehat{\mathbf{f}} = \widehat{\mathbf{f}}\left(\mathbf{x}, (\mathbf{W}_i)_{i=1}^L\right)$$

Even though Bayesian NNs seem simple, calculating model posterior is a hard task.

To relate the approximate inference in our Bayesian NN to dropout training, we define our approximating variational distribution $q(\mathbf{W}_i)$ for every layer i as

$$\mathbf{W}_i = \mathbf{M}_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$$
$$\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i) \text{ for } i = 1, ..., L,\ j = 1, ..., K_{i-1}.$$

$\mathbf{M}_i$ are variational parameters to be optimised.

$$\mathcal{L}_{\text{VI}} := \int q(\boldsymbol{\omega}) p(\mathbf{F}|\mathbf{X}, \boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{F}) \text{d}\mathbf{F} \text{d}\boldsymbol{\omega} - \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

The integral is intractable and cannot be evaluated analytically. Instead, we approximate the integral with Monte Carlo integration over $\boldsymbol{\omega}$.

This results in an unbiased estimator

$$\widehat{\mathcal{L}}_{\text{VI}} := \sum_{i=1}^{N} E\big(\mathbf{y}_i, \widehat{\mathbf{f}}(\mathbf{x}_i, \widehat{\boldsymbol{\omega}}_i)\big) - \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

$$\widehat{\mathcal{L}}_{\text{VI}} := \sum_{i=1}^{N} E\left(\mathbf{y}_i, \widehat{\mathbf{f}}(\mathbf{x}_i, \widehat{\boldsymbol{\omega}}_i)\right) - \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

sampling from $q(\mathbf{W}_i)$ is identical to performing dropout on layer i in a network whose weights are $(\mathbf{M}_i)_{i=1}^{L}$

$$\mathcal{L}_{\text{dropout}} := E + \lambda_1 ||\mathbf{W}_1||_2^2 + \lambda_2 ||\mathbf{W}_2||_2^2 + \lambda_3 ||\mathbf{b}||_2^2.$$

The second term in $\mathcal{L}_{\text{VI}} := \int q(\boldsymbol{\omega}) p(\mathbf{F}|\mathbf{X}, \boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{F}) \text{d}\mathbf{F} \text{d}\boldsymbol{\omega} - \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$

can be approximated resulting in the objective

$$\mathcal{L}_{\text{dropout}} := \frac{1}{N} \sum_{i=1}^{N} E(\mathbf{y}_i, \widehat{\mathbf{y}}_i) + \lambda \sum_{i=1}^{L} \left(||\mathbf{W}_i||_2^2 + ||\mathbf{b}_i||_2^2\right)$$

Dropout and Bayesian NNs, in effect, result in the same model parameters that best explain the data.

Predictions in this model follow equation

$$q(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \boldsymbol{\omega})q(\boldsymbol{\omega})\mathrm{d}\mathbf{f}^*\mathrm{d}\boldsymbol{\omega}.$$

replacing the posterior $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ with the approximate posterior $q(\boldsymbol{\omega})$

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \int p(y^*|\mathbf{x}^*, \boldsymbol{\omega})q(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|\mathbf{x}^*, \widehat{\boldsymbol{\omega}}_t) \qquad \boldsymbol{\omega} = \{\mathbf{W}_i\}_{i=1}^{L}$$

Obtaining Model Uncertainty

we sample T sets of vectors of realisations from the Bernoulli distribution $\{\mathbf{z}_1^t, ..., \mathbf{z}_L^t\}_{t=1}^T$ with $\mathbf{z}_i^t = [\mathbf{z}_{i,j}^t]_{j=1}^{K_i}$, giving $\{\mathbf{W}_1^t, ..., \mathbf{W}_L^t\}_{t=1}^T$.

$$\text{We estimate } \mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}(\mathbf{y}^*) \approx \frac{1}{T}\sum_{t=1}^{T} \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, ..., \mathbf{W}_L^t)$$

We refer to this Monte Carlo estimate as **MC dropout**.

In practice this is equivalent to performing T stochastic forward passes through the network and averaging the results.

Proposition 3. *Given weights matrices* $\mathbf{M}_i$ *of dimensions* $K_i \times K_{i-1}$, *bias vectors* $\mathbf{m}_i$ *of dimensions* K_i, *and binary vectors* $\mathbf{z}_i$ *of dimensions* K_{i-1} *for each layer* $i = 1, ..., L$, *as well as the approximating variational distribution*

$$q(\mathbf{y}^*|\mathbf{x}^*) := \mathcal{N}\big(\mathbf{y}^*; \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L), \tau^{-1}\mathbf{I}_D\big) Bern(\mathbf{z}_1) \cdots Bern(\mathbf{z}_L)$$

for some $\tau > 0$, *with*

$$\widehat{\mathbf{y}}^* = \sqrt{\frac{1}{K_L}}(\mathbf{M}_L\mathbf{z}_L)\sigma\left(...\sqrt{\frac{1}{K_1}}(\mathbf{M}_2\mathbf{z}_2)\sigma\big((\mathbf{M}_1\mathbf{z}_1)\mathbf{x}^* + \mathbf{m}_1\big)...\right),$$

we have

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}(\mathbf{y}^*) \approx \frac{1}{T}\sum_{t=1}^{T} \widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})$$

with

$$\widehat{\mathbf{z}}_{i,t} \sim Bern(p_i).$$

Proof.

$$
\begin{aligned}
\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}(\mathbf{y}^*) &= \int \mathbf{y}^* q(\mathbf{y}^*|\mathbf{x}^*)\mathrm{d}\mathbf{y}^* \\
&= \int \mathbf{y}^* \mathcal{N}\big(\mathbf{y}^*; \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L), \tau^{-1}\mathbf{I}_D\big)\mathrm{Bern}(\mathbf{z}_1)\cdots\mathrm{Bern}(\mathbf{z}_L)\mathrm{d}\mathbf{z}_1\cdots\mathrm{d}\mathbf{z}_L\mathrm{d}\mathbf{y}^* \\
&= \int \left(\int \mathbf{y}^* \mathcal{N}\big(\mathbf{y}^*; \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L), \tau^{-1}\mathbf{I}_D\big)\mathrm{d}\mathbf{y}^* \right)\mathrm{Bern}(\mathbf{z}_1)\cdots\mathrm{Bern}(\mathbf{z}_L)\mathrm{d}\mathbf{z}_1\cdots\mathrm{d}\mathbf{z}_L\mathrm{d}\mathbf{y}^* \\
&= \int \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L)\mathrm{Bern}(\mathbf{z}_1)\cdots\mathrm{Bern}(\mathbf{z}_L)\mathrm{d}\mathbf{z}_1\cdots\mathrm{d}\mathbf{z}_L \\
&\approx \frac{1}{T}\sum_{t=1}^{T} \widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t}).
\end{aligned}
$$

We estimate the second raw moment in the same way:

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}\left((\mathbf{y}^*)^T(\mathbf{y}^*)\right) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, ..., \mathbf{W}_L^t)^T\widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, ..., \mathbf{W}_L^t)$$

To obtain the model's predictive variance we have:

$$\mathrm{Var}_{q(\mathbf{y}^*|\mathbf{x}^*)}\left(\mathbf{y}^*\right) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, ..., \mathbf{W}_L^t)^T\widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, ..., \mathbf{W}_L^t) - \mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}(\mathbf{y}^*)^T\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}(\mathbf{y}^*)$$

Proposition 4. *Given weights matrices* $\mathbf{M}_i$ *of dimensions* $K_i \times K_{i-1}$*, bias vectors* $\mathbf{m}_i$ *of dimensions* K_i*, and binary vectors* $\mathbf{z}_i$ *of dimensions* K_{i-1} *for each layer* $i = 1, ..., L$*, as well as the approximating variational distribution*

$$q(\mathbf{y}^*|\mathbf{x}^*) := p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})q(\boldsymbol{\omega})$$
$$q(\boldsymbol{\omega}) = Bern(\mathbf{z}_1) \cdots Bern(\mathbf{z}_L)$$
$$p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) = \mathcal{N}\big(\mathbf{y}^*; \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L), \tau^{-1}\mathbf{I}_D\big)$$

for some $\tau > 0$*, with*

$$\widehat{\mathbf{y}}^* = \sqrt{\frac{1}{K_L}}(\mathbf{M}_L\mathbf{z}_L)\sigma\bigg(...\sqrt{\frac{1}{K_1}}(\mathbf{M}_2\mathbf{z}_2)\sigma\big((\mathbf{M}_1\mathbf{z}_1)\mathbf{x}^* + \mathbf{m}_1\big)...\bigg),$$

we have

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}\big((\mathbf{y}^*)^T(\mathbf{y}^*)\big) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})^T\widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})$$

with

$$\widehat{\mathbf{z}}_{i,t} \sim Bern(p_i).$$

Proof.

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}\left((\mathbf{y}^*)^T(\mathbf{y}^*)\right) \\
&= \int \left(\int (\mathbf{y}^*)^T(\mathbf{y}^*) p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) \mathrm{d}\mathbf{y}^* \right) q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\mathrm{Cov}_{p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})}(\mathbf{y}^*) + \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})}(\mathbf{y}^*)^T \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})}(\mathbf{y}^*) \right) q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\tau^{-1}\mathbf{I}_D + \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L)^T \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L) \right) \mathrm{Bern}(\mathbf{z}_1) \cdots \mathrm{Bern}(\mathbf{z}_L) \mathrm{d}\mathbf{z}_1 \cdots \mathrm{d}\mathbf{z}_L \\
&\approx \tau^{-1}\mathbf{I}_D + \frac{1}{T} \sum_{t=1}^{T} \widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})^T \widehat{\mathbf{y}}^*(\mathbf{x}^*, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})
\end{aligned}
$$

since $p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) = \mathcal{N}\left(\mathbf{y}^*; \widehat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{z}_1, ..., \mathbf{z}_L), \tau^{-1}\mathbf{I}_D\right)$.

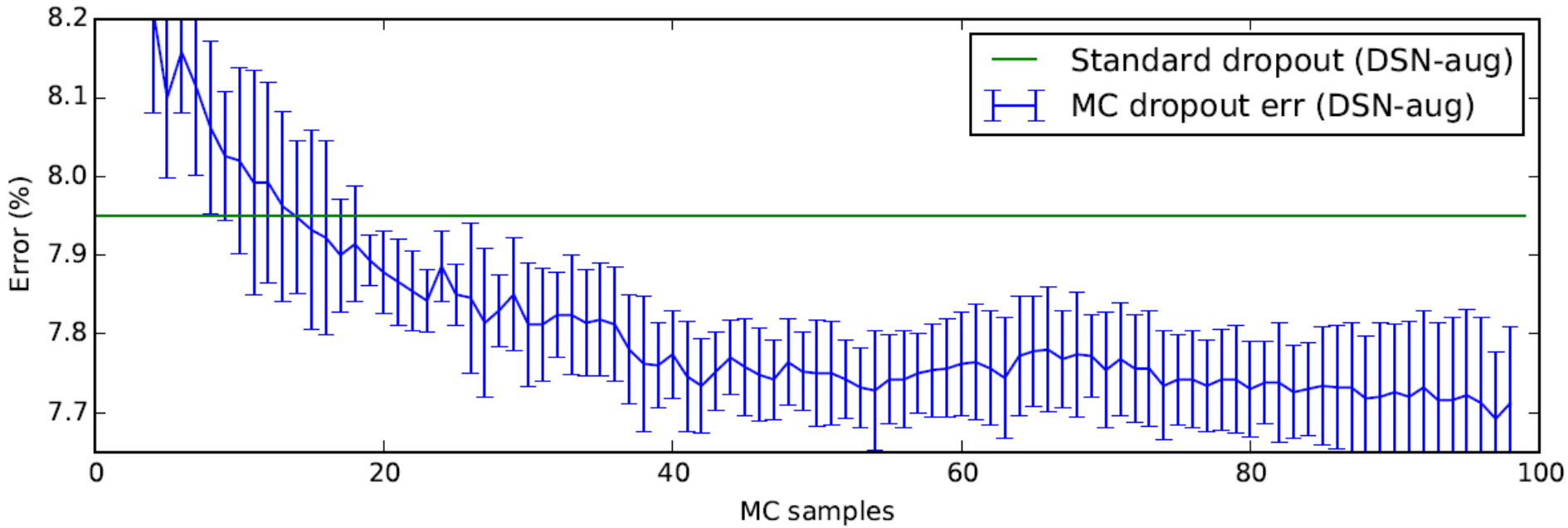


Figure 3: **Augmented-DSN test error for different number of averaged forward passes in MC dropout** (blue) averaged with 5 repetitions, shown with 1 standard deviation. In green is test error with Standard dropout. MC dropout achieves a significant improvement (more than 1 standard deviation) after 20 samples.

Deep Bayesian Active Learning with Image Data NIPS2017

BALD

Choose pool points that are expected to maximise the information gained about the model parameters, i.e. maximise the **mutual information between predictions and model posterior**:

Points that maximize this acquisition function are points on which the model is uncertain on average, but there exist model parameters that produce disagreeing predictions with high certainty.
This is equivalent to points with high variance in the input to the softmax layer (the logits) thus each stochastic forward pass through the model would have the highest probability assigned to a different class.

$$\mathbb{I}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}] = \mathbb{H}[y | \mathbf{x}, \mathcal{D}_{\text{train}}] - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})} \left[\mathbb{H}[y | \mathbf{x}, \boldsymbol{\omega}] \right]$$

$$= -\sum_c p(y = c | \mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c | \mathbf{x}, \mathcal{D}_{\text{train}}) \; + \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})} \left[\sum_c p(y = c | \mathbf{x}, \boldsymbol{\omega}) \log p(y = c | \mathbf{x}, \boldsymbol{\omega}) \right]$$

$\mathbb{I}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}]$ can be approximated in our setting using the identity

$$p(y = c | \mathbf{x}, \mathcal{D}_{\text{train}}) = \int p(y = c | \mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega} | \mathcal{D}_{\text{train}}) \mathrm{d}\boldsymbol{\omega}$$

$$\mathbb{I}[y, \boldsymbol{\omega}|\mathbf{x}, \mathcal{D}_{\text{train}}] = -\sum_c \int p(y=c|\mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}}) \mathrm{d}\boldsymbol{\omega} \cdot \log \int p(y=c|\mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}}) \mathrm{d}\boldsymbol{\omega}$$

$$+ \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})} \left[\sum_c p(y=c|\mathbf{x}, \boldsymbol{\omega}) \log p(y=c|\mathbf{x}, \boldsymbol{\omega}) \right]$$

Swapping the posterior $p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})$ with our approximate posterior $q_\theta^*(\boldsymbol{\omega})$, and through MC sampling, we then have:

$$\approx -\sum_c \int p(y=c|\mathbf{x}, \boldsymbol{\omega}) q_\theta^*(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \cdot \log \int p(y=c|\mathbf{x}, \boldsymbol{\omega}) q_\theta^*(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}$$

$$+ \mathbb{E}_{q_\theta^*(\boldsymbol{\omega})} \left[\sum_c p(y=c|\mathbf{x}, \boldsymbol{\omega}) \log p(y=c|\mathbf{x}, \boldsymbol{\omega}) \right]$$

$$\approx -\sum_c \left(\frac{1}{T} \sum_t \widehat{p}_c^t \right) \log \left(\frac{1}{T} \sum_t \widehat{p}_c^t \right) + \frac{1}{T} \sum_{c,t} \widehat{p}_c^t \log \widehat{p}_c^t =: \widehat{\mathbb{I}}[y, \boldsymbol{\omega}|\mathbf{x}, \mathcal{D}_{\text{train}}]$$

defining our approximation, with $\widehat{p}_c^t$ the probability of input $\mathbf{x}$ with model parameters $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$ to take class c:

$$\widehat{\mathbf{p}}^t = [\widehat{p}_1^t, ..., \widehat{p}_C^t] = \text{softmax}(\mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}))$$

$$\widehat{\mathbb{I}}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}] \xrightarrow[T \to \infty]{} \mathbb{H}[y | \mathbf{x}, q_\theta^*] - \mathbb{E}_{q_\theta^*(\boldsymbol{\omega})}\left[\mathbb{H}[y | \mathbf{x}, \boldsymbol{\omega}]\right] \approx \mathbb{I}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}]$$

BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning arXiv2019.6

a batch of data points $\{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_b^*\}$ is selected using an acquisition function a which scores a candidate batch of unlabelled data points $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_b\} \subseteq \mathcal{D}_{\textbf{pool}}$ using the current model parameters $\text{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})$:

$$\{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_b^*\} = \underset{\{\boldsymbol{x}_1, ..., \boldsymbol{x}_b\} \subseteq \mathcal{D}_{\textbf{pool}}}{\arg\max} \; a\left(\{\boldsymbol{x}_1, ..., \boldsymbol{x}_b\}, \text{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})\right)$$

BALD is defined as

$$\mathbb{I}(y\,;\,\boldsymbol{\omega}\mid \boldsymbol{x},\mathcal{D}_{\text{train}}) = \mathbb{H}(y\mid \boldsymbol{x},\mathcal{D}_{\text{train}}) - \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega}\mid\mathcal{D}_{\text{train}})}\left[\mathbb{H}(y\mid \boldsymbol{x},\boldsymbol{\omega},\mathcal{D}_{\text{train}})\right]$$

BALD (Bayesian Active Learning by Disagreement) uses an acquisition function that estimates **the mutual information between the model predictions and the model parameters**.

Intuitively, it captures **how strongly the model predictions for a given data point and the model parameters are coupled**, implying that finding out about the **true label of data points with high mutual information would also inform us about the true model parameters.**

BALD was originally intended for acquiring individual data points and immediately retraining the model

$$a_{\text{BALD}}\left(\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_b\},\mathrm{p}(\boldsymbol{\omega}\mid\mathcal{D}_{\text{train}})\right) = \sum_{i=1}^{b}\mathbb{I}(y_i\,;\,\boldsymbol{\omega}\mid \boldsymbol{x}_i,\mathcal{D}_{\text{train}})$$

which reduces to picking the top b highest-scoring data points.

We propose BatchBALD whereby we jointly score points by estimating the mutual information between a **joint** of multiple data points and the model parameters.

$$a_{\texttt{BatchBALD}}\left(\{\boldsymbol{x}_1, ..., \boldsymbol{x}_b\}, \mathrm{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})\right) = \mathbb{I}(y_1, ..., y_b ; \boldsymbol{\omega} \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_b, \mathcal{D}_{\textbf{train}}).$$

Intuitively, the mutual information between two random variables can be seen as the intersection of their information content.

a signed measure μ^* can be defined for discrete random variables x, y,

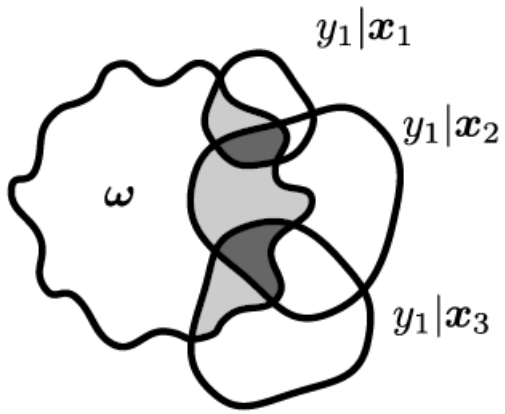
$$\mathbb{I}(x ; y) = \mu^*(x \cap y), \mathbb{H}(x, y) = \mu^*(x \cup y), \ \mathbb{E}_{\mathrm{p}(y)} \mathbb{H}(x \mid y) = \mu^*(x \setminus y)$$

Using this, BALD can be viewed as the sum of individual intersections $\sum_i \mu^*(y_i \cap \boldsymbol{\omega})$, which double counts overlaps between the y_i. Naively extending BALD to the mutual information between $y_1, ..., y_b$ and $\boldsymbol{\omega}$, which is equivalent to $\mu^*(\bigcap_i y_i \cap \boldsymbol{\omega})$, would lead to selecting *similar* data points instead of diverse ones under maximisation.

BatchBALD, on the other hand, takes overlaps into account by computing $\mu^*(\bigcup_i y_i \cap \boldsymbol{\omega})$ and is more likely to acquire a more diverse cover under maximisation:

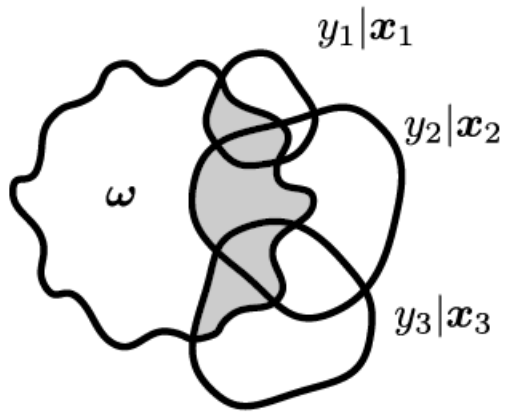
$$\mathbb{I}(y_1, ..., y_b ; \boldsymbol{\omega} \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_b, \mathcal{D}_{\text{train}}) = \mathbb{H}(y_{1:b} \mid \boldsymbol{x}_{1:b}, \mathcal{D}_{\text{train}}) - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})} \mathbb{H}(y_{1:b} \mid \boldsymbol{x}_{1:b}, \boldsymbol{\omega}, \mathcal{D}_{\text{train}})$$

$$= \mu^*\left(\bigcup_i y_i\right) - \mu^*\left(\bigcup_i y_i \setminus \boldsymbol{\omega}\right) = \mu^*\left(\bigcup_i y_i \cap \boldsymbol{\omega}\right)$$



$$\sum_i \mathbb{I}(y_i ; \boldsymbol{\omega} \mid \boldsymbol{x}_i, \mathcal{D}_{\text{train}}) = \sum_i \mu^*(y_i \cap \boldsymbol{\omega})$$

(a) BALD



$$\mathbb{I}(y_1, ..., y_b ; \boldsymbol{\omega} \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_b, \mathcal{D}_{\text{train}}) = \mu^*\left(\bigcup_i y_i \cap \boldsymbol{\omega}\right)$$

(b) BatchBALD

leave out conditioning on $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$, and $\mathcal{D}_{\textbf{train}}$, and $\mathrm{p}(\boldsymbol{\omega})$ denotes $\mathrm{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})$

$$\mathbb{I}(y_1, ..., y_b ; \boldsymbol{\omega} \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_b, \mathcal{D}_{\textbf{train}}) = \mathbb{H}(y_{1:b} \mid \boldsymbol{x}_{1:b}, \mathcal{D}_{\textbf{train}}) - \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})} \mathbb{H}(y_{1:b} \mid \boldsymbol{x}_{1:b}, \boldsymbol{\omega}, \mathcal{D}_{\textbf{train}})$$

$$a_{\texttt{BatchBALD}}\left(\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}, \mathrm{p}(\boldsymbol{\omega})\right) = \mathbb{H}(y_1, ..., y_n) - \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\left[\mathbb{H}(y_1, ..., y_n \mid \boldsymbol{\omega})\right]$$

Because the y_i are independent when conditioned on $\boldsymbol{\omega}$, computing the right term of equation is simplified as the conditional joint entropy decomposes into a sum

We can approximate the expectation using a Monte-Carlo estimator with k samples from our model parameter distribution

$$\mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\left[\mathbb{H}(y_1, ..., y_n \mid \boldsymbol{\omega})\right] = \sum_{i=1}^{n} \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\left[\mathbb{H}(y_i \mid \boldsymbol{\omega})\right] \approx \frac{1}{k} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{H}(y_i \mid \hat{\boldsymbol{\omega}}_j).$$

Applying the equality $\mathrm{p}(y) = \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\left[\mathrm{p}(y \mid \boldsymbol{\omega})\right]$

$$\mathbb{H}(y_1, \ldots, y_n) = \mathbb{E}_{\mathrm{p}(y_1,\ldots,y_n)}\left[-\log \mathrm{p}(y_1, \ldots, y_n)\right] = \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\, \mathbb{E}_{\mathrm{p}(y_1,\ldots,y_n \mid \boldsymbol{\omega})}\left[-\log \mathbb{E}_{\mathrm{p}(\boldsymbol{\omega})}\left[\mathrm{p}(y_1, \ldots, y_n \mid \boldsymbol{\omega})\right]\right]$$

$$\approx -\sum_{\hat{y}_{1:n}} \left(\frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n} \mid \hat{\boldsymbol{\omega}}_j)\right) \log\left(\frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n} \mid \hat{\boldsymbol{\omega}}_j)\right)$$

Efficient implementation

In each iteration of the algorithm, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}$ stay fixed while $\boldsymbol{x}_n$ varies over $\mathcal{D}_{\mathbf{pool}} \setminus A_{n-1}$. We can reduce the required computations by factorizing $\mathrm{p}(y_{1:n} \mid \boldsymbol{\omega})$ into $\mathrm{p}(y_{1:n-1} \mid \boldsymbol{\omega})\,\mathrm{p}(y_n \mid \boldsymbol{\omega})$. We store $\mathrm{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j)$ in a matrix $\hat{P}_{1:n-1}$ of shape $c^n \times k$ and $\mathrm{p}(y_n \mid \hat{\boldsymbol{\omega}}_j)$ in a matrix $\hat{P}_n$ of shape $c \times k$. The sum $\sum_{i=1}^{k} \mathrm{p}(\hat{y}_{1:n} \mid \hat{\boldsymbol{\omega}}_j)$ in (12) can be then be turned into a matrix product:

$$\frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n} \mid \hat{\boldsymbol{\omega}}_j) = \frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j)\,\mathrm{p}(\hat{y}_n \mid \hat{\boldsymbol{\omega}}_j) = \left(\frac{1}{k}\hat{P}_{1:n-1}\hat{P}_n^T\right)_{\hat{y}_{1:n-1},\hat{y}_n}$$

$$\frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n} \mid \hat{\boldsymbol{\omega}}_j) = \frac{1}{k}\sum_{j=1}^{k} \mathrm{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j)\, \mathrm{p}(\hat{y}_n \mid \hat{\boldsymbol{\omega}}_j) = \left(\frac{1}{k}\hat{P}_{1:n-1}\hat{P}_n^T\right)_{\hat{y}_{1:n-1},\hat{y}_n}$$

This can be further sped up by using batch matrix multiplication to compute the joint entropy for different $\boldsymbol{x}_n$. $\hat{P}_{1:n-1}$ only has to be computed once, and we can recursively compute $\hat{P}_{1:n}$ using $\hat{P}_{1:n-1}$ and $\hat{P}_n$, which allows us to sample $\mathrm{p}(y \mid \hat{\boldsymbol{\omega}}_j)$ for each $\boldsymbol{x} \in \mathcal{D}_{\textbf{pool}}$ only once at the beginning of the algorithm.

Algorithm 1: Greedy BatchBALD $1 - 1/e$-approximate algorithm

Input: acquisition size b, unlabelled dataset $\mathcal{D}_{\textbf{pool}}$, model parameters $\mathrm{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})$

$A_0 \leftarrow \emptyset$

for $n \leftarrow 1$ **to** b **do**

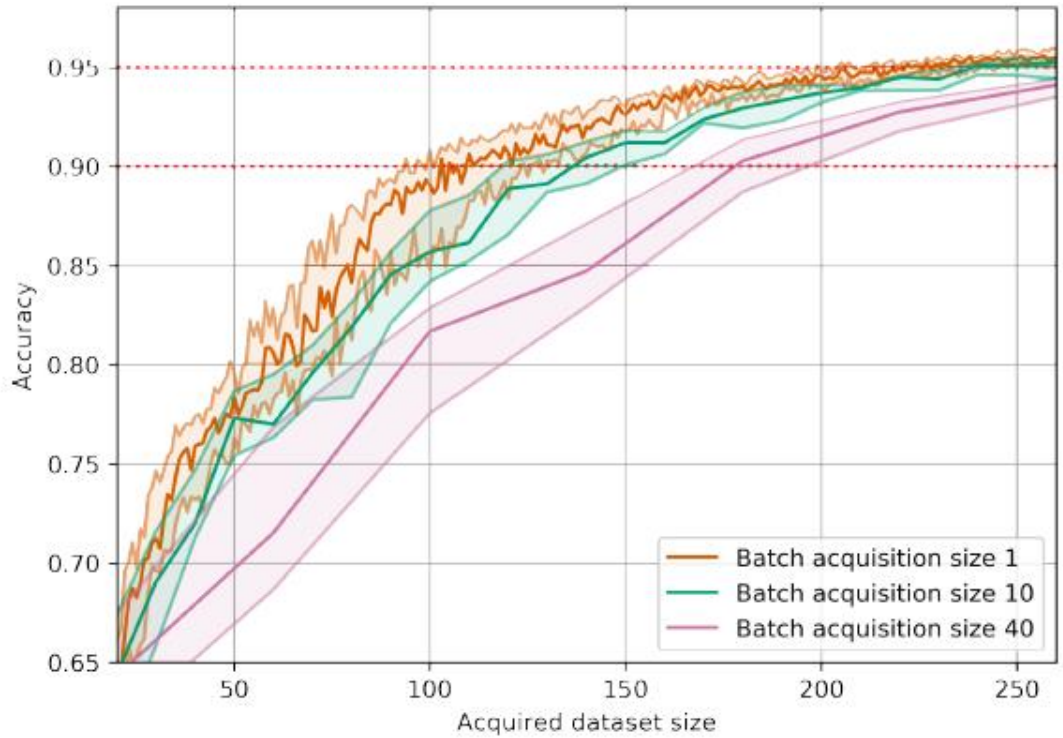
  foreach $\boldsymbol{x} \in \mathcal{D}_{pool} \setminus A_{n-1}$ **do** $s_{\boldsymbol{x}} \leftarrow a_{\texttt{BatchBALD}}\left(A_{n-1} \cup \{\boldsymbol{x}\}, \mathrm{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\textbf{train}})\right)$

  $\boldsymbol{x}_n \leftarrow \underset{\boldsymbol{x} \in \mathcal{D}_{\textbf{pool}} \setminus A_{n-1}}{\arg\max}\ s_{\boldsymbol{x}}$

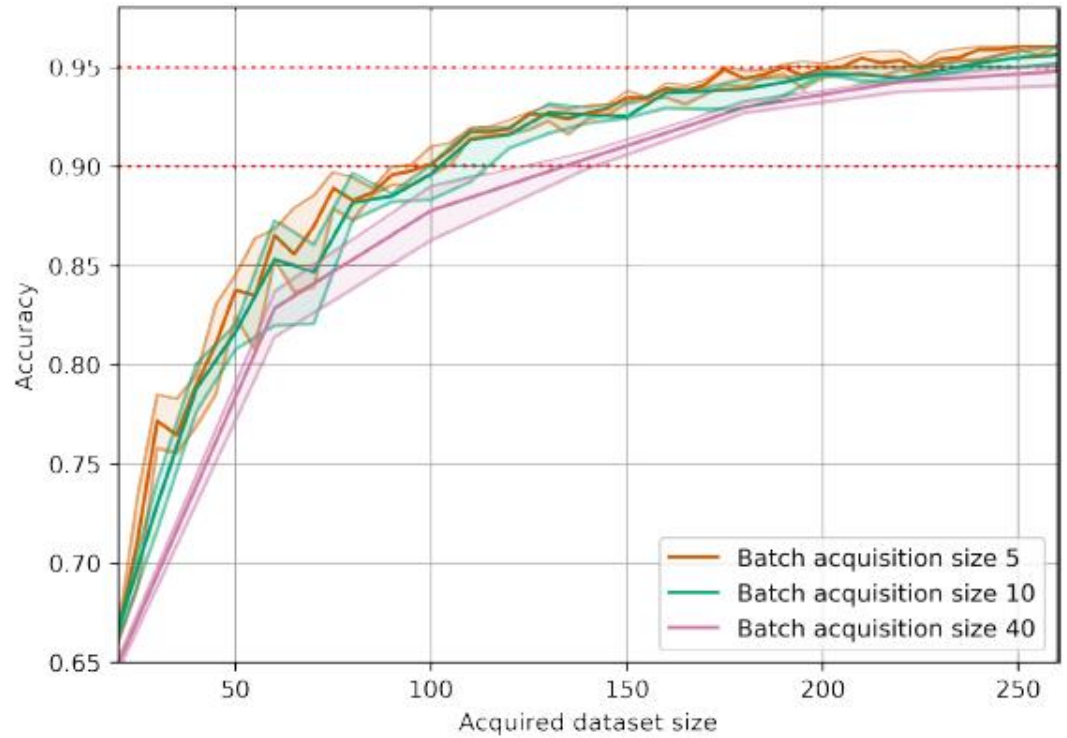
  $A_n \leftarrow A_{n-1} \cup \{\boldsymbol{x}_n\}$

end

Output: acquisition batch $A_n = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_b\}$



(a) BALD



(b) BatchBALD

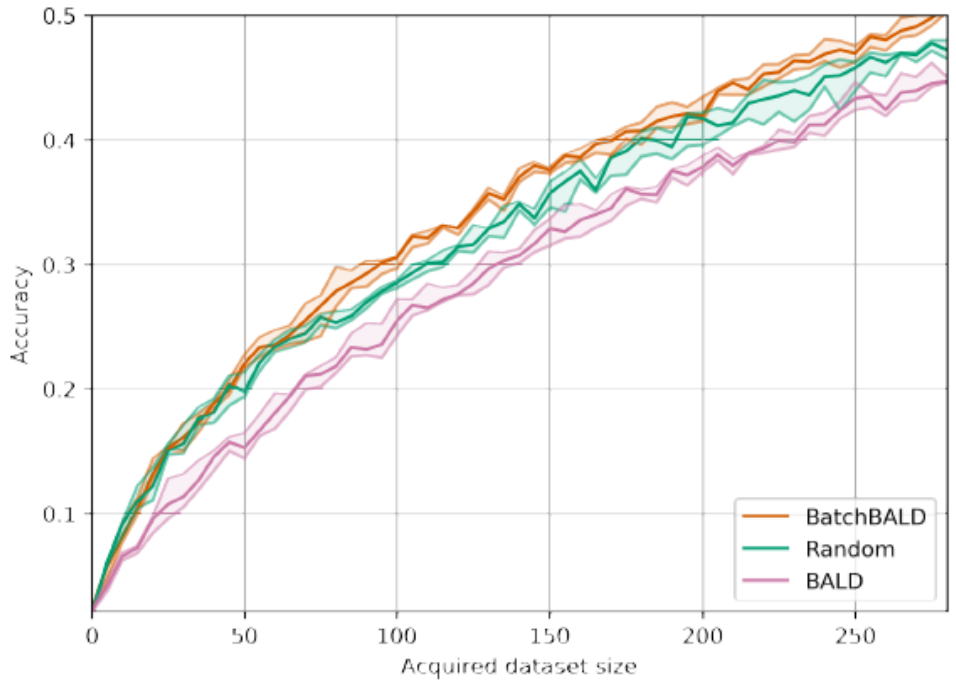


Figure 7: *Performance on* EMNIST. BatchBALD consistently outperforms both random acquisition and BALD while BALD is unable to beat random acquisition.

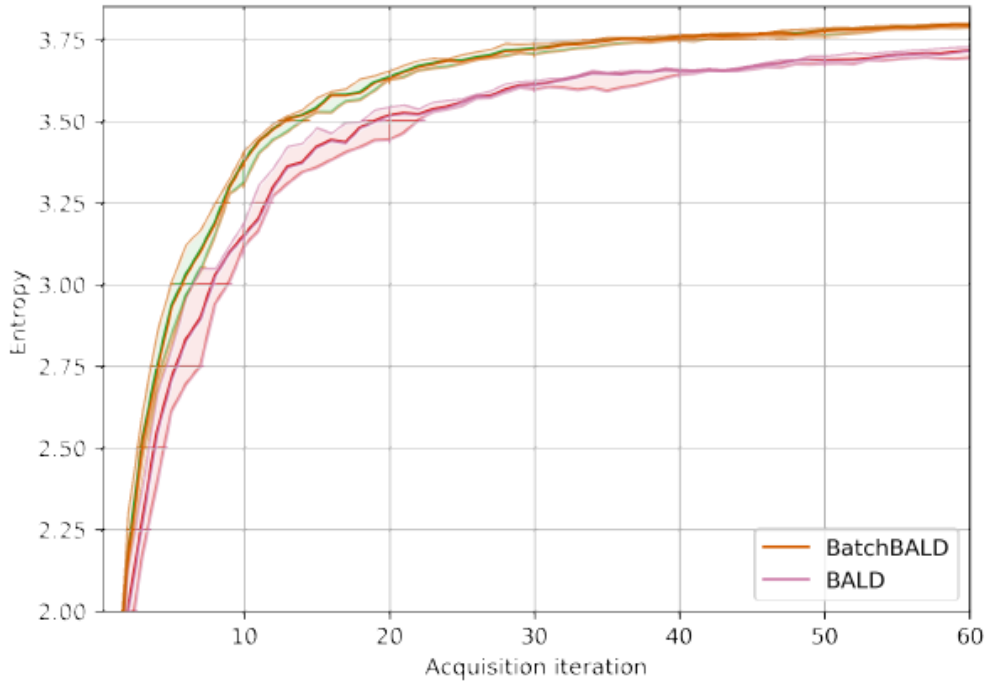


Figure 8: *Entropy of acquired class labels over acquisition steps on* EMNIST. BatchBALD steadily acquires a more diverse set of data points.

A New Outlook on Shannon's Information Measures

Raymond W. Yeung, *Member*, *IEEE*

$$\mu^*(\tilde{X} \cup \tilde{Y}) = H(X,Y)$$
$$\mu^*(\tilde{X}) = H(X)$$
$$\mu^*(\tilde{Y}) = H(Y)$$
$$\mu^*(\tilde{X} \cap \tilde{Y}) = I(X;Y)$$
$$\mu^*(\tilde{X} - \tilde{Y}) = H(X|Y) \qquad (\tilde{X} - \tilde{Y} = \tilde{X} \cap \tilde{Y}^c)$$
$$\mu^*(\tilde{Y} - \tilde{X}) = H(Y|X)$$
$$\mu^*\big((\tilde{X} \cap \tilde{Y})^c\big) = H(X|Y) + H(Y|X)$$

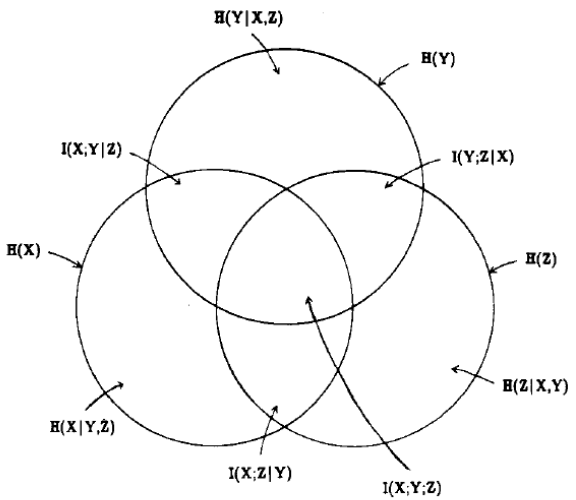


Fig. 2. *I*-Diagram for X, Y, and Z.

$$\mu^*\Big(\Big(\bigcup_{X\in G} X\Big)\cap\Big(\bigcup_{Y\in G'} Y\Big)-\Big(\bigcup_{Z\in G''} Z\Big)\Big)$$
$$\equiv \mu^*\Big(\Big(\bigcup_{X\in G} X\Big)\cup\Big(\bigcup_{Z\in G''} Z\Big)\Big)$$
$$+\mu^*\Big(\Big(\bigcup_{Y\in G'} Y\Big)\cup\Big(\bigcup_{Z\in G''} Z\Big)\Big)$$
$$-\mu^*\Big(\Big(\bigcup_{X\in G} X\Big)\cup\Big(\bigcup_{Y\in G'} Y\Big)\cup\Big(\bigcup_{Z\in G''} Z\Big)\Big)$$
$$-\mu^*\Big(\bigcup_{Z\in G''} Z\Big) \qquad \text{by Lemma 1a}$$
$$\equiv H((X, X\in G),(Z, Z\in G''))$$
$$+H((Y, Y\in G'),(Z, Z\in G''))$$
$$-H((X, X\in G),(Y, Y\in G'),(Z, Z\in G''))$$
$$-H(Z, Z\in G'')$$
$$\equiv I(X, X\in G; Y, Y\in G'|Z, Z\in G'')$$

Example 1: We first point out that $I(X;Y;Z)$ is symmetrical in X, Y, and Z. We see from Fig. 2 that

$$I(X;Y) - I(X;Y|Z)$$
$$\equiv I(Y;Z) - I(Y;Z|X)$$
$$\equiv I(X;Z) - I(X;Z|Y)$$
$$(\equiv I(X;Y;Z)).$$

This identity is not well known although it is simple.

Example 2: Let X and Z be independent. Then

$$I(X;Z) = I(X;Z|Y) + I(X;Y;Z) = 0.$$

Since $I(X;Z|Y)$ is nonnegative, $I(X;Y;Z)$ must be nonpositive. Therefore

$$I(X;Y) = I(X;Y|Z) + I(X;Y;Z) \le I(X;Y|Z).$$

This is readily obtained by inspection of Fig. 2.

Example 3: If X, Y, and Z are pairwise independent, then

$$I(X;Y) = I(Y;Z) = I(X;Z) \qquad (=0).$$

Then it can be seen by inspection of Fig. 2 that

$$I(X;Y|Z) = I(Y;Z|X) = I(X;Z|Y).$$