A brief summary of Bayesian Deep Learning

Yutian Pang

Arizona State University

September 17, 2019

Overview

Linear Regression

- The simple LR, suppose data X is $1xn$, w is $nx1$, b is of size 1
- We have
  $Y = f(X, w, b)$
  $f(x, w, b) = x * w + b$
- w is the paratemeter we want
- Goal of linear regression is to find w such that,
  - $\|y - f(X, w, b)\|$ is minimized
  - Solution:

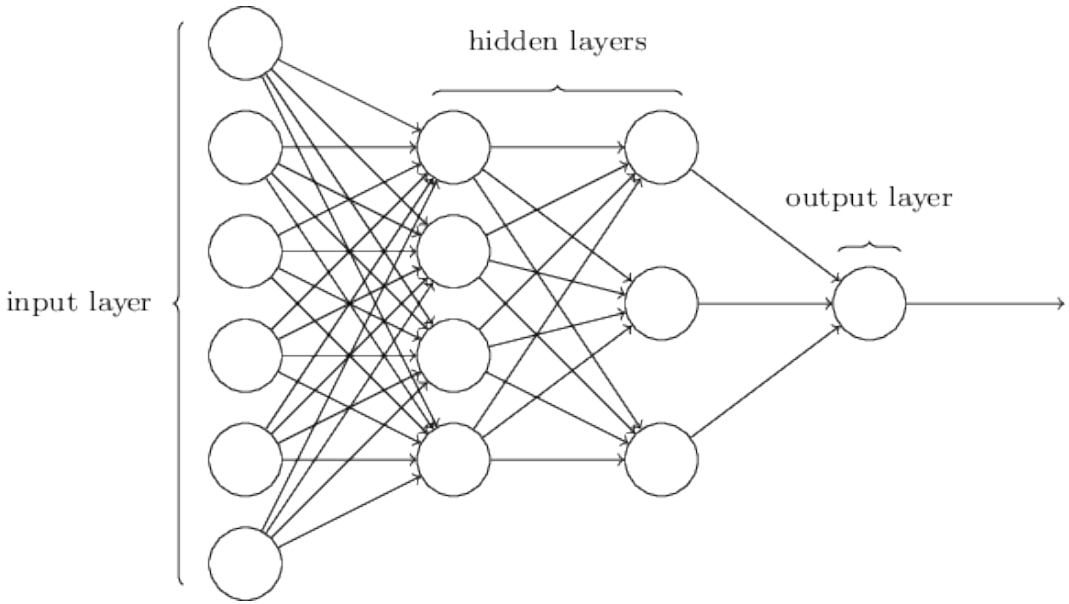
$$\begin{bmatrix} b \\ w \end{bmatrix} = (XX^T)^{-1}Xy$$

Logistic Regression

- Data X is $1xn$, w is $nx1$, b is of size 1
- We have
  $Z = f(X, w, b)$
  $Y = g(Z)$
  $g(x) = \frac{1}{1+e^{-x}}$
- Clip the output between $(0, 1)$

Neural Network

Data: $\mathcal{D} = \{(x_0, y_0), ..., (x_i, y_i), ..., (x_n, y_n)\} = \{X, Y\}$



- Neural Network is a parameterized fitting function.
- w and b are the weights of neural nets
- Fully Connected Nets

Neural Nets

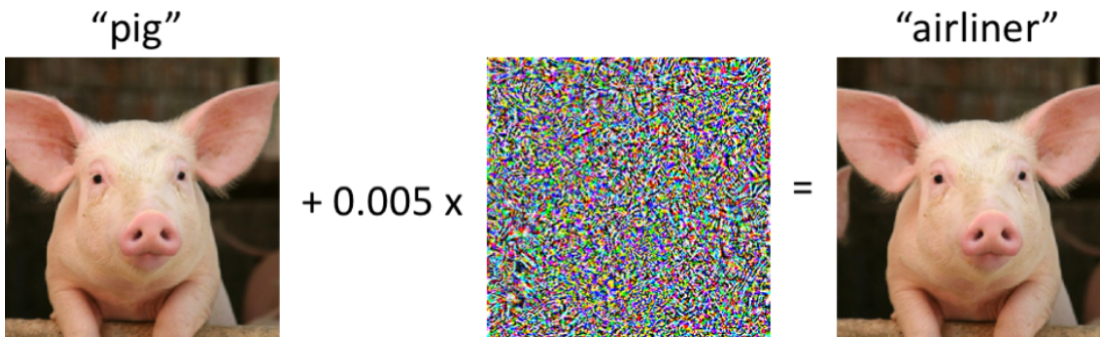
- Feedforward neural nets model $p(Y|X, w, b)$ as a composition of nonlinear functions(layers).
- $Z = \sigma(f(X, w_1, b_1))$
  $Y = ReLU(f(Z, w_2, b_2))$
- Relu, σ, Tanh are activation functions to introduce non-linearities into the network in case there are some non-linear complex functional mappings between the inputs and outputs.
- $ReLU(x) = max(x, 0)$
  $Tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
  $\sigma(x) = \frac{1}{1+e^{-x}}$
- Usually trained to maximize likelihood using variants of stochastic gradient descent (SGD) optimization algorithms.

Deep Learning

- The backprop paper first came out in 1986 when NNs are called *Connectionism*.
- The innovations of architectures and algorithms(e.g. ReLU, many layers, better initialization, learning rate decay, dropout, LSTMs,...)
- Availability of large dataset.
- Improvement of computational power (GPUs).
- Better autodiff software packages (Theano, Tensorflow, Torch).

Limitations

- Data hungry
- Computational-intensive
- Point estimate, poor at representing uncertainty
- Easily fooled by adversarial attacks
- Nontransparent black-boxes, difficult to trust



(Figures from https://medium.com/@smkirthishankar/the-unusual-effectiveness-of-adversarial-attacks-e1314d0fa4d3)

Being Bayesian

The key idea is to learn probability density over parameter space.

Bayesian Linear Regression

- Put a prior over the parameters, i.e. $w \sim N(0, \sigma)$
- Finding the posterior distribution is the goal of Bayesian method,

$$
\begin{aligned}
p(w|Y,X) &= \frac{p(Y|X,w)p(w)}{p(Y|X)} \\
&\propto \exp(\frac{1}{2\sigma_n^2}(y - X^T w)^T (y - X^T w)) \exp(-\frac{1}{2} w^T \sigma^{-1} w) \\
&\propto \exp(\frac{1}{2}(w - \bar{w})^T A (w - \bar{w}))
\end{aligned}
$$

  where $\bar{w} = \frac{1}{\sigma_n^2} A^{-1} X y$ and $A = (\frac{1}{\sigma_n^2} X X^T + \sigma^{-1})$

- Hence, $P(w|Y,X) = N(\bar{w}, A^{-1})$
- However the analytical form of posterior distribution is not always possible. The priors that can enable analytical posterior forms are called conjugate priors.

Bayesian Linear Regression

- The parameter that maximize the posterior distribution is called the maximum a posteriori (MAP) solution.

$$\bar{w}_{MAP} = \frac{1}{\sigma_n^2}(\frac{1}{\sigma_n^2}XX^T + \sigma^{-1})^{-1}Xy$$

- The solution that maximizes the likelihood distribution is called MLE (maximum likelihood estimation) solution,

$$\bar{w}_{MLE} = (XX^T)^{-1}Xy$$

- How about prediction?

Bayesian Linear Regression

Suppose we want to predict at the new input x_*, then the predictive distribution is,

$$p(y_*|x_*, X, Y) = \int p(y_*|x_*, w)p(w|X, Y)dw$$
$$= N(\frac{1}{\sigma_n^2}x_*^T A^{-1}Xy, x_*^T A^{-1}x_*)$$

where $A = (\frac{1}{\sigma_n^2}XX^T + \sigma^{-1})$
Here we apply the kernel trick, the distribution of y_* becomes,
$p(y_*|x_*, X, Y) \sim N(\mu_*, \sigma_*)$
where $\mu_* = k(x_*, X)(k(X, X) + \sigma_n^2 I)^{-1}y$ and
$\sigma_* = k(x_*, x_*) - k(x_*, X)(k(X, X) + \sigma_*^2 I)^{-1}k(X, x_*)$
This is also the weight space view of Gaussian process regression.

A picture: GPs, Regressions, SVMs

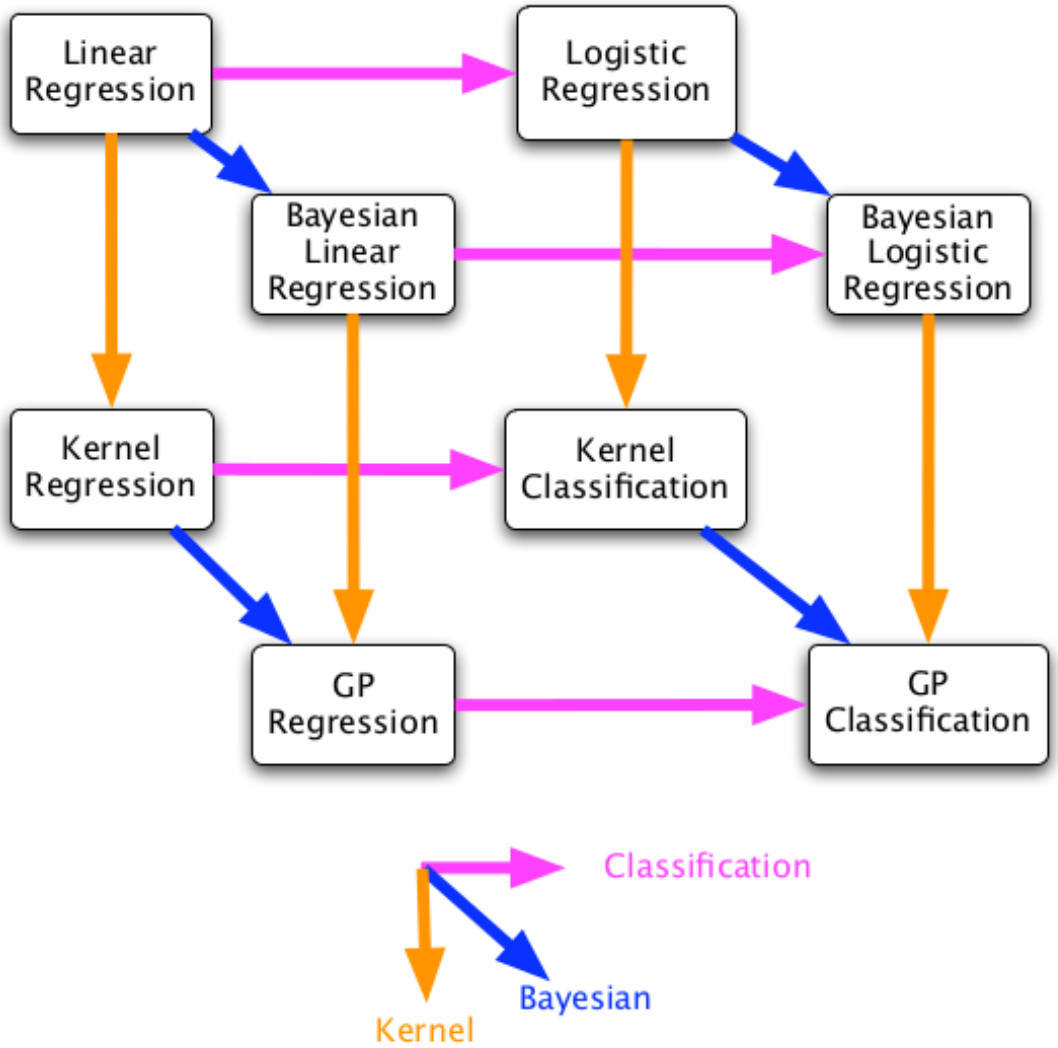
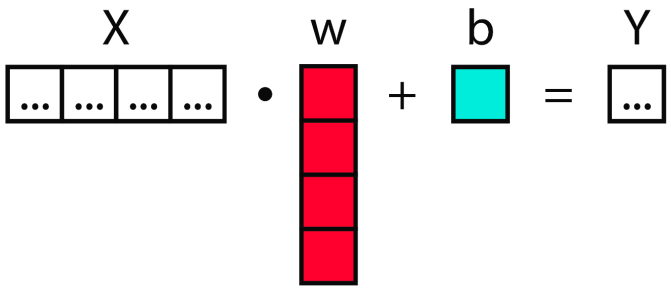


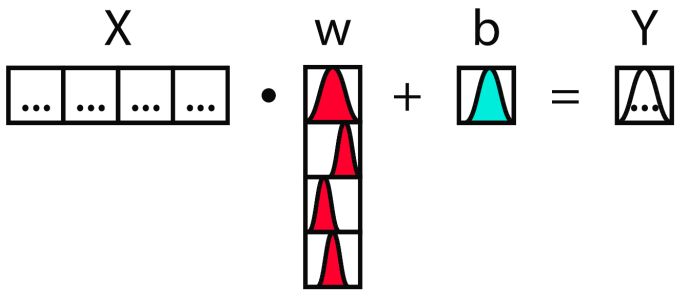
Figure from Zoubin Ghahramani's talk *"A history of Bayesian neural networks"* on NIPS 2016

Bayesian Linear Regression

BLR is as simple as changing from,



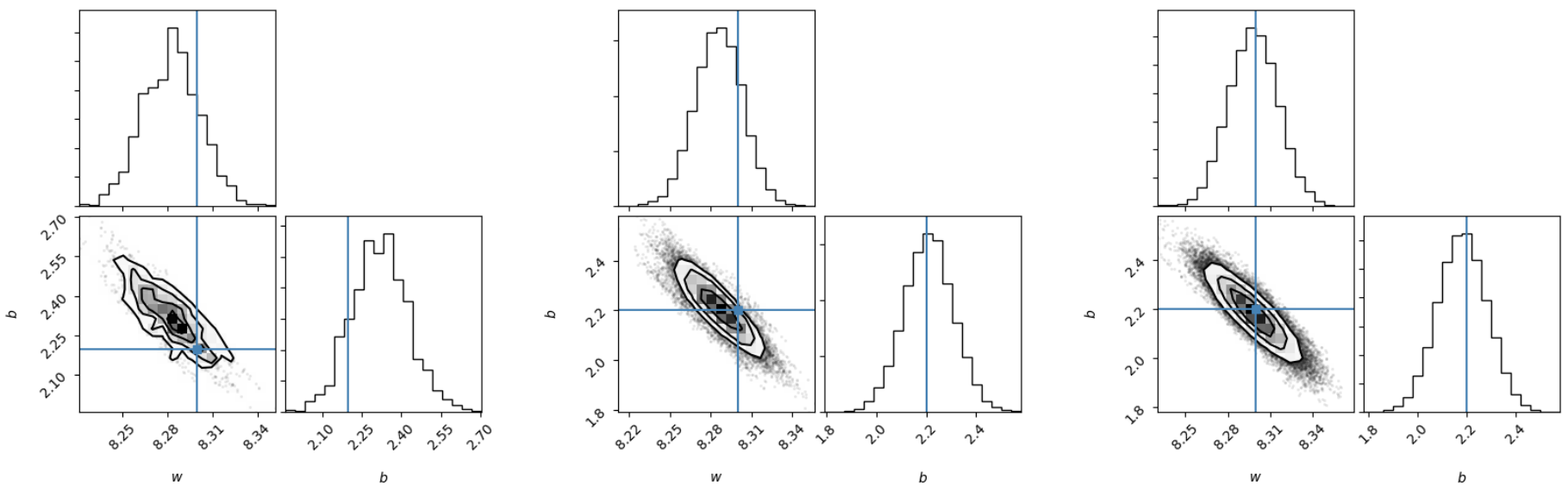
to



(Figures from Eric J. Ma's talk "*An Attempt At Demystifying Bayesian Deep Learning*" at PyData NYC, 2017)

BLR Demo with TensorFlow-probability

- The model, $y(x, w, b) = x * w + b$, assume $w \sim N(0, 10)$, $b \sim U(-10, 10)$
- Noise is added in the training data by $\epsilon \sim N(0, 1)$
- The likelihood function is equal to the joint probability distribution of the random sample evaluated at the given observations
- Sampled with Hamiltonian Monte Carlo



Num of Samples: 2000, 20000, 40000(left to right)

Models vs Algorithms

- Models:
  - HMMs, Boltzmann machines
  - CNNS, RNNs
  - GPs
- Algorithms:
  - Stochastic Gradient Descent
  - MCMC
  - Variational Bayes
  - Conjugate gradients
- "Bayesian" belongs in the Algorithms category, not the Models category. Any well defined model can be treated in a Bayesian manner. (Zoubin, 2016)

What about Bayesian Deep Learning?

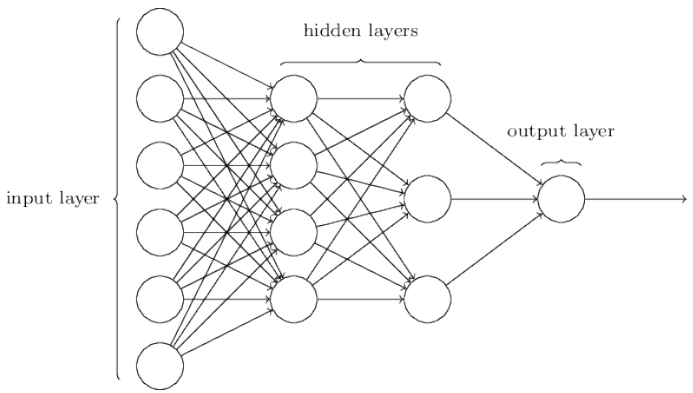
Bayesian Deep Learning

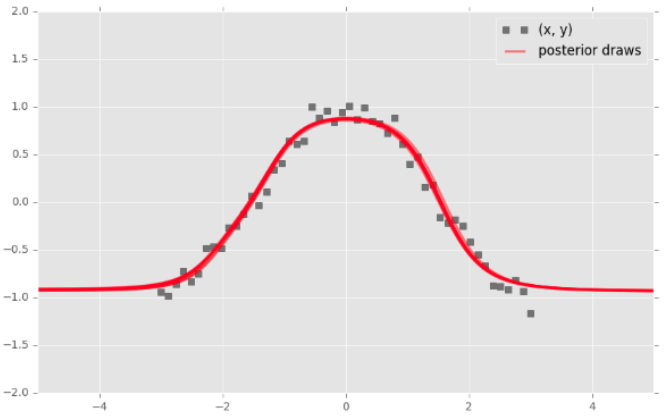
- Posterior

$$p(W|X,Y) = \frac{p(Y|X,W)p(W)}{p(Y|X)}$$

- Inference

$$p(Y|X) = \int p(Y|X,W)p(W|X,Y)dW$$





- A neural network with one hidden layer, infinitely many hidden units and Gaussian priors on the weights $\rightarrow$ *aGP*(*Neal*, 1994).

Reviews: *Minimizing the Description Length*

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In COLT, pages 5-13. ACM, 1993.

- A more complex model is always better to fit the training data.
- The best model is defined that minimizes the combined cost of describing the model and describing the misfit between the model and the data.
- In simple words,
  - There is a sender who can see both the input and the output.
  - There is a receiver who can only see the input.
- The sender first fits a NN to the entire training dataset and then send the weights to the receiver.
- By adding the discrepancy to the output of the network, the receiver can generate exactly the correct output.
- Total cost = coding the data misfits + coding the weights

Reviews: *Ensemble Learning in Neural Network*

David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. In Generalization in Neural Networks and Machine Learning Springer Verlag, 215-238, 1998.

- This method is called the variational inference nowadays.
- This paper introduce a variational distribution q(w) that approximates the posterior.
- Main contribution of this paper is to find the efficient update rule for optimizing q(w) by using full covariance Gaussian variational approximation to weights.

Reviews: *Practical Variational Inference*

Taken from Introduction:

- "In the eighteen years since variational inference was first proposed for neural networks (Hinton and Camp, 1993) it has not seen widespread use."
- "We believe this is largely due to the difficulty of deriving analytical solutions to the required integrals over the variational posteriors."
- "The approach taken here is to forget about analytical solutions and search instead for variational distributions whose expectation values (and derivatives thereof) can be efficiently approximated with numerical integration."

Reviews: *Practical Variational Inference*

- This paper is the first paper to formally use a variational inference on Bayesian neural networks.
- This paper assumes that the posterior distribution is a Gaussian distribution where the covariance matrix is diagonal and presented efficient gradients for optimizing the mean and variance.

Reviews: *Bayes by backprop*

Weight Uncertainty in Neural Networks

- All weights in the neural networks are represented by probability distributions over possible values, rather than having a single fixed value.
- Instead of training a single network, the proposed method trains an ensemble of networks, where each network has its weights sampled from a learnt probability distribution.
- This paper also changes the optimization objective

Summary of BNN

- The goal of a Bayesian Neural Network(BNN) is to find a posterior distribution:

  $p(W|X, Y) \propto p(Y|X, W)p(W)$

  where X and Y are the input and output training data and W is a set of network weights.

- Once we have $p(W|X, Y)$, the output y_* at unseen x_* is predicted through Bayesian inference:

  $p(y_*|x_*) = \int p(y|x_*, W)p(W|X, Y)dW$

- In practice, none if them is tractable.

Summary of BNN

- Variational Inference is used to handle this issue.
- Instead of finding $p(W|X, Y)$, optimize a variational distribution $q(\theta)$ by minimizing

$$KL(q_\theta(W)\|p(W|X, Y)) = \int q_\theta(W) log \frac{q_\theta(W)}{p(W|X, Y)} dW$$

- Variational inference is used to approximate the (intractable) posterior distribution with (tractable) variational distribution with respect to the KL divergence.
- But $KL(q_\theta(W)\|p(W|X, Y))$ is not tractable as well.
- We choose to compute the lower-bound called evidance lower-bound(ELBO) instead.

$$ELBO = \int q_\theta(W) log p(Y|X, W) dW - KL(q_\theta(W)\|p(W))$$

Summary of BNN

- In computing ELBO, we do not need to know $p(W|X, Y)$.

$$ELBO = \int q_\theta(W) log p(Y|X, W) dW - KL(q_\theta(W)||p(W))$$

- All we need is
  - prior: $p(W)$
  - variational distribution: $q_\theta(W)$
  - likelihood: $p(Y|X, W)$
- Minimizing the KL divergence is equivalent to maximizing the evidence lower bound (ELBO) which also contains the integral with respect to the distribution over latent variables.

Summary of BNN

- In computing ELBO, we do not need to know $p(W|X, Y)$.

$$ELBO = \int q_\theta(W) log p(Y|X, W) dW - KL(q_\theta(W) \| p(W))$$

- All we need is
  - prior: $p(W)$
  - variational distribution: $q_\theta(W)$
  - likelihood: $p(Y|X, W)$
- How to define a likelihood function?

Summary of BNN

- In computing ELBO, we do not need to know $p(W|X, Y)$.

$$ELBO = \int q_\theta(W) log p(Y|X, W) dW - KL(q_\theta(W) \| p(W))$$

- All we need is
  - prior: $p(W)$
  - variational distribution: $q_\theta(W)$
  - likelihood: $p(Y|X, W)$
- How to define a likelihood function?
- If we could recall BLR, the solution is GP approximation again!

Summary of BNN

Gaussian process approximation

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = \int p(\mathbf{w})p(b)\sigma(\mathbf{w}^T\mathbf{x}+b)\sigma(\mathbf{w}^T\mathbf{y}+b)\mathrm{d}\mathbf{w}\mathrm{d}b$$

↓ MC approximation

$$\widehat{\mathbf{K}}(\mathbf{x},\mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K}\sigma(\mathbf{w}_k^T\mathbf{x}+b_k)\sigma(\mathbf{w}_k^T\mathbf{y}+b_k)$$

↓ Apply to Gaussian processes

$$\mathbf{w}_k \sim p(\mathbf{w}),\ b_k \sim p(b),$$
$$\mathbf{W}_1 = [\mathbf{w}_k]_{k=1}^K, \mathbf{b} = [b_k]_{k=1}^K$$
$$\widehat{\mathbf{K}}(\mathbf{x},\mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K}\sigma(\mathbf{w}_k^T\mathbf{x}+b_k)\sigma(\mathbf{w}_k^T\mathbf{y}+b_k)$$
$$\mathbf{F} \mid \mathbf{X}, \mathbf{W}_1, \mathbf{b} \sim \mathcal{N}(\mathbf{0}, \widehat{\mathbf{K}}(\mathbf{X},\mathbf{X}))$$
$$\mathbf{Y} \mid \mathbf{F} \sim \mathcal{N}(\mathbf{F}, \tau^{-1}\mathbf{I}_N),$$

↓ GP Marginal likelihood

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X},\mathbf{W}_1,\mathbf{W}_2,\mathbf{b})p(\mathbf{W}_1)p(\mathbf{W}_2)p(\mathbf{b})$$

Figure from Sungjoon Choi's presentation slides *Uncertainties in Deep Learning in a nutshell*

Summary of BNN

- Then apply the re-parametrization trick ($\tilde{W} = g(W + \epsilon_W)$ where $\epsilon_W(\epsilon)$) is a random variable) to the GP approximation result and get the re-parametrized ELBO

$$\int p(\epsilon) logp(Y|X, \tilde{W}) d\epsilon - KL(q_\theta(W) \| p(W))$$

- Then approximate the integral with MC integration, we have

$$\mathcal{L}_{GP\_MC} = \sum_{n=1}^{N} logp(y_n|x_n, \tilde{W}) + KL(q_\theta(W) \| p(W))$$

- With Gaussian prior on the parameters W, we have

$$\mathcal{L}_{GP\_MC} = \sum_{n=1}^{N} logp(y_n|x_n, \tilde{W}) + \sum_{m=1}^{M} \lambda_m \| W_m \|^2$$

- Where n is the number of data and m is the number of parameters.

Summary of BNN

- Once we have the paramaters W, predictive mean and variance can be approximated as:

$$E_q(y_*) = \int y_* q(y_*|x_*) dy_*$$
$$\approx \frac{1}{T} \sum_{t=1}^{T} y_*(x_*, \tilde{W}^t)$$
$$Var_q(y_*) \approx \tau^{-1} I + \frac{1}{T} \sum_{t=1}^{T} y_*(x_*, \tilde{W}^t)^T y_*(x_*, W^t) - E_q(y_*)^T E_q(y_*)$$

where τ is called model precision hyper-parameter.

Y.Gal *Uncertainty in Deep Learning*, 2016

Uncertainty in Deep Learning

Yarin Gal

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Gonville and Caius College September 2016

Review: Dropout as Bayesian Approximation

Algorithm 2 Optimisation of a neural network with dropout

1: Given dataset $\mathbf{X}, \mathbf{Y}$,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly.
4: **repeat**
5: Sample M random variables $\widehat{\boldsymbol{\epsilon}}_i \sim p(\boldsymbol{\epsilon})$, S a random subset of $\{1, .., N\}$ of size M.
6: Calculate derivative w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \frac{\partial}{\partial\theta}\left(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\right).$$

7: Update θ:
$\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$.
8: **until** θ has converged.

Review: Dropout as Bayesian Approximation

- Gal Ghahramani: Approximate ELBO and Dropout NN objectives are identical.
- Optimising any neural network with dropout is equivalent to a form a apprimate Bayesian inference
- A network trained with dropout is already a BNN.
- However, MC Dropout requires applying dropout at every weight layer and averaging over T forward passes during testing.

Other approximation methods

$$p(W|X,Y) = \frac{p(Y|X,W)p(W)}{p(Y|X)}$$

$$p(Y|X) = \int p(Y|X,W)p(W|X,Y)dW$$

Other approximation methods for posterior and marginal likelihood estimation,

- Variational approximations
- Markov chain Monte Carlo methods (MCMC)
- Sequential Monte Carlo (SMC)
- Exact Sampling
- Bayesian Information Criterion (BIC)

Tools

The Python probabilistic programming packages provides:

- Statistical distributions
- Sampling algorithms
- Inference methods

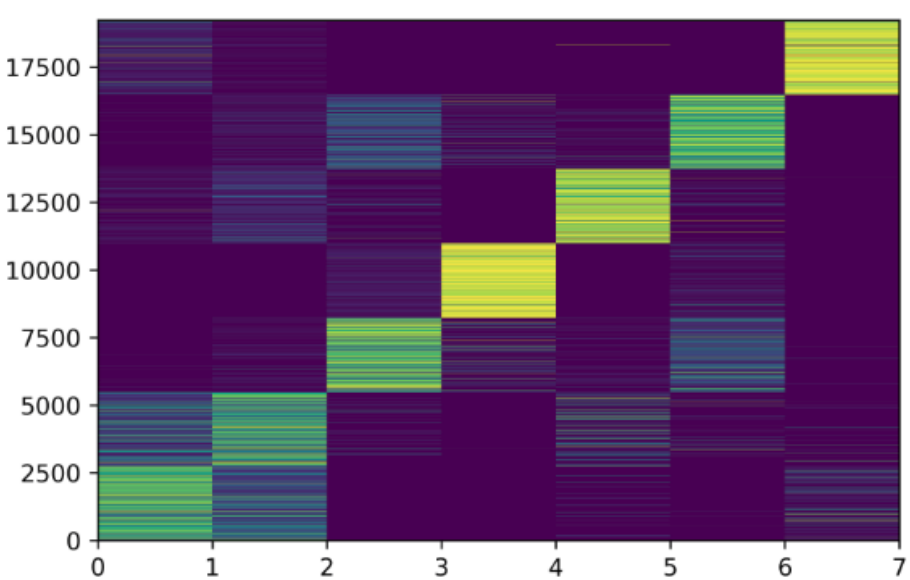
List of packages,

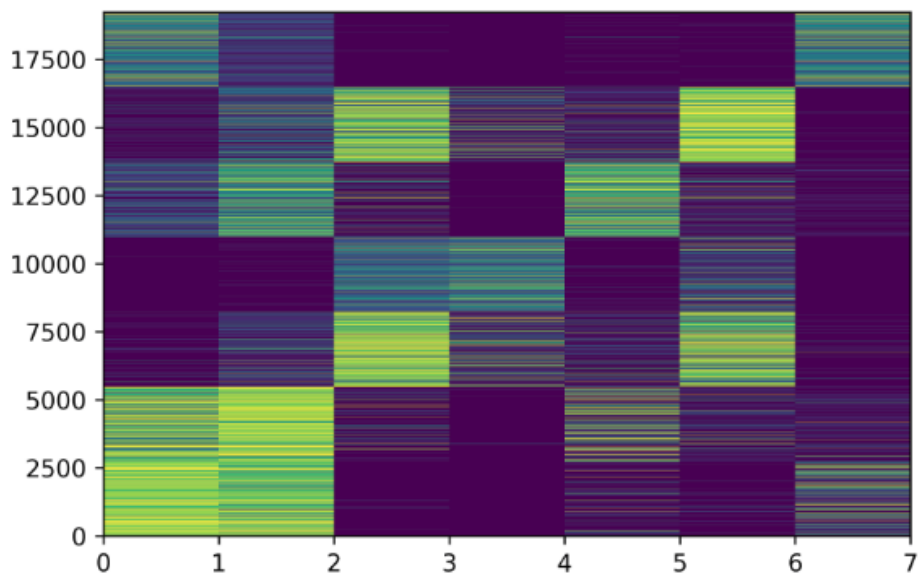
- Tensorflow-probability(tfp)
  - Edward: Dustin Tran, merging into tfp as Edward2
  - Zhusuan: TsingHua University, not officially released in pip
  - Sonnet: Deepmind
- PyMC3/4: Based on Theano but trying to move into tfp in PyMC4
- Pyro: Uber Lab, based on Torch

Demo: BNN Classification

Predict Forest Cover Type with PyMC3

- Toy Dataset: Covertype Dataset from UCI ML Repository
- Input: 66 categorical features
- Output: 7 forest cover types
- $W \sim N(0, 1)$
- Two hidden layers with 20 units each



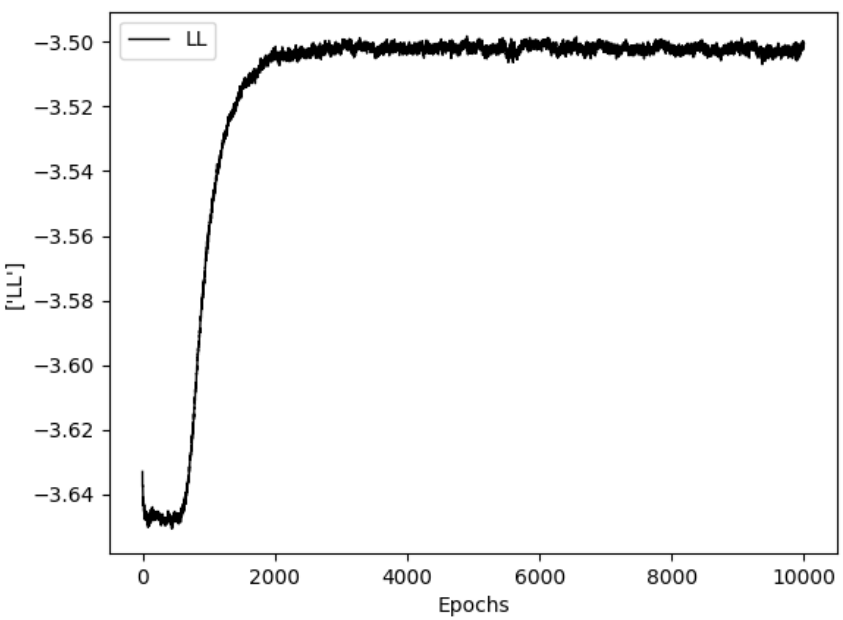


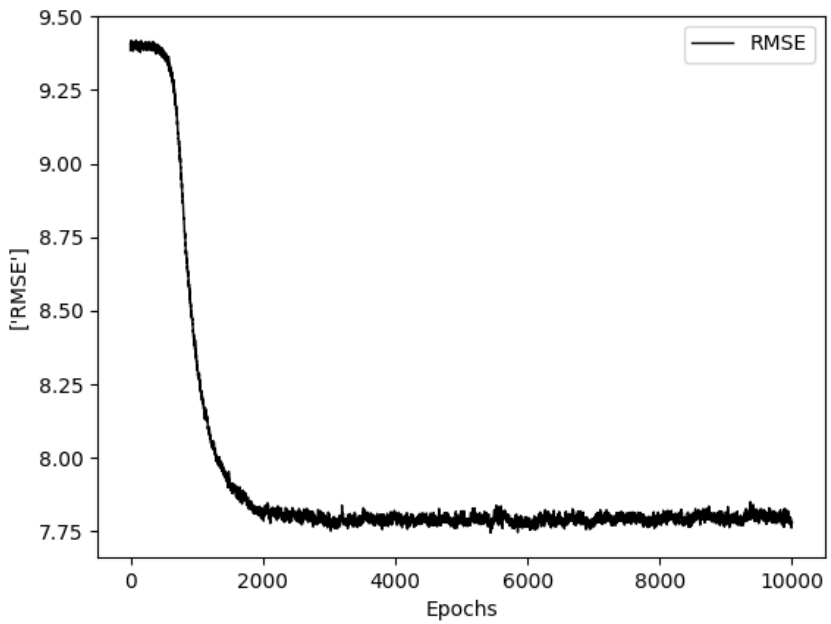
Left: μ *Right* : σ

Demo: BNN Regression

Predict Boston Housing data with Zhusuan

- Toy Dataset: Boston Housing Dataset from UCI ML Repository originally
- Input: 506 data points with 13 features
- Output: House price
- $W \sim N(0, 1)$
- Hidden layers: [32, 128, 32]

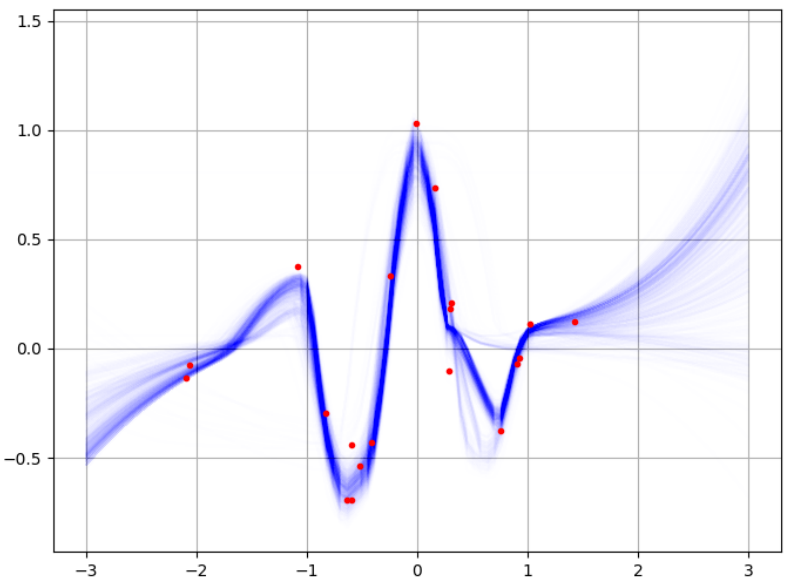


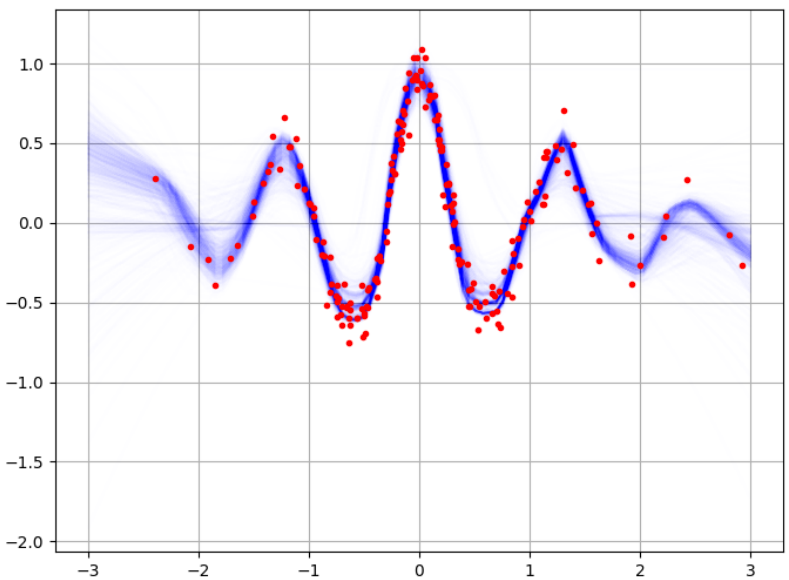


Demo:Dropout as Bayesian

Regression simple funtions with Tensorflow.distributions

- Implementation of Dropout as Bayesian
- Two hidden layers [100, 100]
- Data sampled from $N(\frac{cos(5x)}{|x|} + 1, 0.1)$
- Updated 1000 iterations





Left: 20 sample points Right: 200 sample points

Future Work

- The demo code is found online
- Computational Statistics knowledge
- Exploration

The End