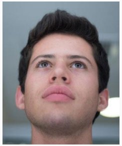
Dropout Inference in Bayesian Neural Networks

with Alpha-divergences

Yingzhen Li (presenter) and Yarin Gal

University of Cambridge



Background I: Deep learning

Conceptually simple models

Data: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\}$
Model: given matrices $\mathbf{W}$ and non-linear func. $\sigma(\cdot)$, define "network"

$$\tilde{\mathbf{y}}_i(\mathbf{x}_i) = \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \mathbf{x}_i\big)$$

Objective: find $\mathbf{W}$ for which $\tilde{\mathbf{y}}_i(\mathbf{x}_i)$ is close to $\mathbf{y}_i$ for all $i \leq N$.

Deep learning is awesome ✓

- Simple and modular
- Huge attention from practitioners and engineers
- Great software tools
- Scales with data and compute
- Real-world impact

... but has many issues ✗

- What does a model not know?
- Uninterpretable black-boxes
- Easily fooled (AI safety)
- Lacks solid mathematical foundations (mostly ad hoc)
- Crucially relies on big data

Conceptually simple models

Data: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\}$
Model: given matrices $\mathbf{W}$ and non-linear func. $\sigma(\cdot)$, define "network"

$$\tilde{\mathbf{y}}_i(\mathbf{x}_i) = \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \mathbf{x}_i\big)$$

Objective: find $\mathbf{W}$ for which $\tilde{\mathbf{y}}_i(\mathbf{x}_i)$ is close to $\mathbf{y}_i$ for all $i \leq N$.

Deep learning is awesome ✓

- Simple and modular
- Huge attention from practitioners and engineers
- Great software tools
- Scales with data and compute
- Real-world impact

... but has many issues ✗

- What does a model not know?
- Uninterpretable black-boxes
- Easily fooled (AI safety)
- Lacks solid mathematical foundations (mostly ad hoc)
- Crucially relies on big data

No uncertainty!

Recap: Bayesian neural networks

- Use your favourite loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ to define the likelihood:

$$\log p(\mathbf{y}|\mathbf{x}, \omega) = -\ell(\mathbf{y}, \hat{\mathbf{y}}_n = \mathrm{NN}_\omega(\mathbf{x})) + C$$

- In many cases regulariser induces prior info, e.g. Gaussian $\Leftrightarrow$ ℓ_2 regulariser
- Bayesian neural net in two steps (ideally):
  - observing data $\mathbf{X}, \mathbf{Y}$, compute exact posterior $p(\omega|\mathbf{X}, \mathbf{Y})$
  - for prediction, compute

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

  - ...and also obtain uncertainty estimates

Recap: Bayesian neural networks

- Use your favourite loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ to define the likelihood:

$$\log p(\mathbf{y}|\mathbf{x}, \omega) = -\ell(\mathbf{y}, \hat{\mathbf{y}}_n = \text{NN}_\omega(\mathbf{x})) + C$$

- In many cases regulariser induces prior info, e.g. Gaussian $\Leftrightarrow$ ℓ_2 regulariser
- Bayesian neural net in two steps (in practice):
  - observing data $\mathbf{X}, \mathbf{Y}$, find approximate posterior $q(\omega) \approx p(\omega|\mathbf{X}, \mathbf{Y})$
  - for prediction, compute approximate Bayesian prediction

$$\mathbf{y}^* = \int \text{NN}_\omega(\mathbf{x}^*) q(\omega) d\omega$$

  - ...and also obtain (approximated) uncertainty estimates

Recap: Bayesian neural networks

- Use your favourite loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ to define the likelihood:

$$\log p(\mathbf{y}|\mathbf{x}, \omega) = -\ell(\mathbf{y}, \hat{\mathbf{y}}_n = \mathrm{NN}_\omega(\mathbf{x})) + C$$

- In many cases regulariser induces prior info, e.g. Gaussian $\Leftrightarrow$ ℓ_2 regulariser
- Bayesian neural net in two steps (in practice):
  - observing data $\mathbf{X}, \mathbf{Y}$, find approximate posterior $q(\omega) \approx p(\omega|\mathbf{X}, \mathbf{Y})$
  - for prediction, compute approximate Bayesian prediction

$$\mathbf{y}^* = \int \mathrm{NN}_\omega(\mathbf{x}^*) q(\omega) d\omega$$

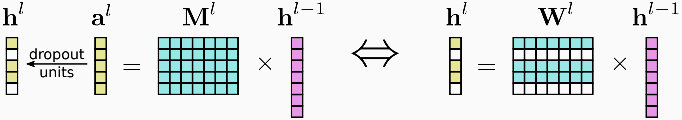
  - ...and also obtain (approximated) uncertainty estimates

Challenges: efficient approximate inference methods for deep nets

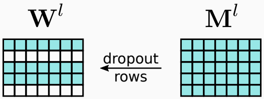
- computationally fast & memory efficient
- easy to implement

Recap: dropout as approximate Bayesian inference for BNNs [1]

Activation Dropout with rate p $\quad\Leftrightarrow\quad$ Dropout rows with rate p



Dropout rows with rate p



$\Leftrightarrow$

Sample rows $\mathbf{W}_i^l$ from

$$q(\mathbf{W}_i^l) = p\mathcal{N}(\mathbf{M}_i^l, \eta\mathbf{I}) + (1-p)\mathcal{N}(\mathbf{0}, \eta\mathbf{I})$$
$$\eta \to 0$$

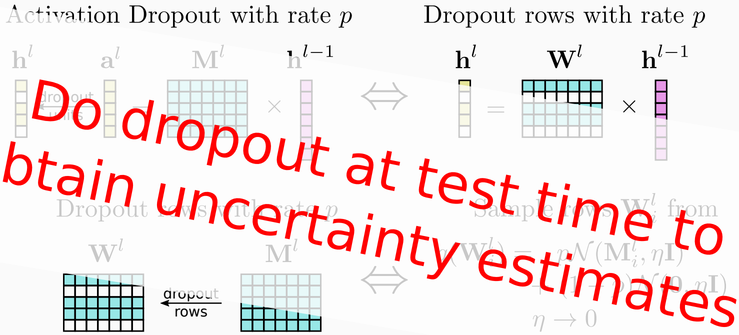
Define $q_\theta(\omega) = \prod_{l,i} q(\mathbf{W}_i^l)$, $\theta = \{\mathbf{M}_i^l\}$:

$$\sum_n \ell(\mathbf{y}_n, \hat{\mathbf{y}}_n = \text{dropout-NN}(\mathbf{x}_n)) + \ell_2(\theta) = \text{MC estimate of } -\mathbb{E}_{q_\theta(\omega)}[\log p(\mathbf{Y}|\mathbf{X}, \omega)] + \mathrm{KL}[q_\theta || p_0]$$

Training NNs with dropout $\Leftrightarrow$ Training BNNs with variational inference!

[1] Yarin Gal. Uncertainty in Deep Learning. PhD Thesis, University of Cambridge. 2016.

Recap: dropout as approximate Bayesian inference for BNNs [1]



Do dropout at test time to obtain uncertainty estimates!

Define $q_\theta(\omega) = \prod_{l,i} q(\mathbf{W}_i^l)$, $\theta = \{\mathbf{M}_i^l\}$:

$$\sum_n \ell(\mathbf{y}_n, \hat{\mathbf{y}}_n = \text{dropout-NN}(\mathbf{x}_n)) + \ell_2(\theta) = \text{MC estimate of } -\mathbb{E}_{q_\theta(\omega)}[\log p(\mathbf{Y}|\mathbf{X}, \omega)] + \mathrm{KL}[q_\theta || p_0]$$

Training NNs with dropout ⇔ Training BNNs with variational inference!

1

Recap: properties of alpha-divergences

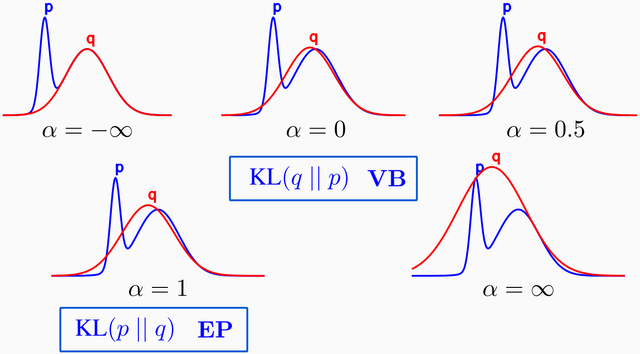


Figure source: Tom Minka

- VI/VB underestimates uncertainty (bad for the applications we considered)
- Black box alpha (BB-α): a better alternative [2]

[2] J. M. Hernandez-Lobato and **Y. Li** et al. Black box alpha-divergence minimisation. ICML 2016.

Naive combination doesn't work!

Quick explanations on why it fails:

- VI has q appeared in the KL term $\rightarrow$ efficient approximate KL
- BB-α requires explicitly evaluating $\log q(\widehat{\boldsymbol{\omega}}^k)$ for samples $\widehat{\boldsymbol{\omega}}^k \sim q(\omega)$
- Even worse: dropout implicitly samples different sets of samples $\widehat{\boldsymbol{\omega}}^{n,k}$ for different datapoints!
  - need to store NK sets of $\widehat{\boldsymbol{\omega}}$ and compute density,
  - i.e. $\mathcal{O}(NK|\omega|)$ for both time and space complexity,
  - infeasible for wide and deep NNs!

New: “power loss” for training BNNs

We propose a new objective for training BNNs:

$$\mathcal{L}_\alpha(\theta) := -\frac{1}{\alpha}\sum_n \text{log-sum-exp}_k[-\alpha\ell(\mathbf{y}_n, \text{NN}_{\widehat{\boldsymbol{\omega}}^{n,k}}(\mathbf{x}_n))] + \ell_2(\theta), \quad \widehat{\boldsymbol{\omega}}^{n,k} \sim q_\theta(\omega)$$

$\ell(\mathbf{y}, \hat{\mathbf{y}})$ is your favourite loss function, log-sum-exp performed over k axis

- Goes back to VI when $\alpha \to 0$ or $K = 1$
- If you have more computational resources, you can obtain better uncertainty estimates with our method!

Significance: compared to black-box alpha (ICML 2016)

$$\mathcal{L}_\alpha(\theta) := -\frac{1}{\alpha} \sum_n \text{log-sum-exp}_k[-\alpha \ell(\mathbf{y}_n, \text{NN}_{\widehat{\boldsymbol{\omega}}^{n,k}}(\mathbf{x}_n))] + \ell_2(\theta), \quad \widehat{\boldsymbol{\omega}}^{n,k} \sim q_\theta(\omega)$$

Our new method is easier to understand!

- To understand the original BB-alpha, need to understand power EP & go through sets of equations [3]
- Instead the new formulation has a clear link to loss function minimisation while still being an approximate Bayesian inference method (inherent from EP)
  - $\alpha = 0$: focus on decreasing prediction error
  - $\alpha = 1$: focus on increasing predictive likelihood

[3] Ask me at the poster :)

Significance: compared to black-box alpha (ICML 2016)

$$\mathcal{L}_\alpha(\theta) := -\frac{1}{\alpha}\sum_n \text{log-sum-exp}_k[-\alpha\ell(\mathbf{y}_n, \text{NN}_{\widehat{\boldsymbol{\omega}}^{n,k}}(\mathbf{x}_n))] + \ell_2(\theta), \quad \widehat{\boldsymbol{\omega}}^{n,k} \sim q_\theta(\omega)$$

Our new method is more computationally efficient!

- The original BB-alpha requires evaluating $\log q(\widehat{\boldsymbol{\omega}}^{n,k})$ at samples ($\mathcal{O}(NK|\omega|)$ time and memory)
- Instead the new formulation just need dropout or other SRTs :)
  - can still use a small number of samples to evaluate $\mathrm{KL}[q||p_0]$ if analytical solutions/approximations are not available

Significance: compared to black-box alpha (ICML 2016)

$$\mathcal{L}_\alpha(\theta) := -\frac{1}{\alpha}\sum_n \text{log-sum-exp}_k[-\alpha\ell(\mathbf{y}_n, \text{NN}_{\widehat{\omega}^{n,k}}(\mathbf{x}_n))] + \ell_2(\theta), \quad \widehat{\omega}^{n,k} \sim q_\theta(\omega)$$

Our method is much easier to implement!

(Keras example: ∼10 lines for the proposed loss function and ∼20 lines for dropout itself)

```
def bbalpha_softmax_cross_entropy_with_mc_logits(alpha):
  def loss(y_true, mc_logits):
    # mc_logits: output of GenerateMCSamples, of shape M x K x D
    mc_log_softmax = mc_logits - K.max(mc_logits, axis=2, keepdims=True)
    mc_log_softmax = mc_log_softmax - logsumexp(mc_log_softmax, 2)
    mc_ll = K.sum(y_true * mc_log_softmax, -1)  # M x K
    return - 1. / alpha * (logsumexp(alpha * mc_ll, 1) + K.log(1.0 / K_mc))
  return loss
```

(see appendix in our paper for full code details)

Adversarial attack detection: being Bayesian helps!

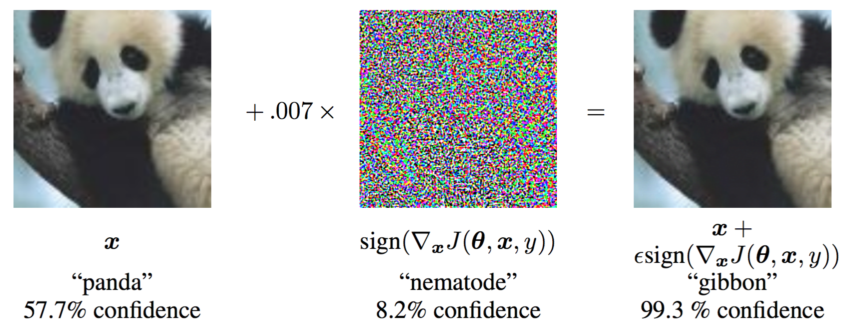


Figure source: Goodfellow et al. ICLR 2015

Why BNNs could be more robust to adversarial attacks?

A simple reasoning for improved robustness:

- Let's say you have an ensemble of neural nets
- In most cases the attacker can only access the majority vote of the ensemble
- i.e. the attacker needs to fool more than a half of them

Why BNNs could be more robust to adversarial attacks?

A simple reasoning for improved robustness:

- Let's say you have an ensemble of neural nets
- In most cases the attacker can only access the majority vote of the ensemble
- i.e. the attacker needs to fool more than a half of them

BNN is better than naive ensembling!

- Bayesian prediction $\Leftrightarrow$ constructing an infinite ensemble in a principled way
- MC sampling returns a random set of ensembles

Being robust $\neq$ being able to detect!

- Adversarial training: more robust, but still provide point estimates
- Ensembles: even when majority vote is fooled, disagreement can still exist!

(describes uncertainty in some sense)

[4] other possible idea: bootstrapping and bagging

Being robust $\neq$ being able to detect!

- Adversarial training: more robust, but still provide point estimates
- Ensembles: even when majority vote is fooled, disagreement can still exist!
  (describes uncertainty in some sense)

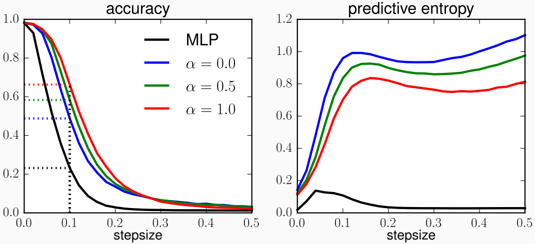
However, we need reliable "uncertainty" here:

- ideal case: uncertainty level grows as we move away from the data manifold
- meaning we need calibrated uncertainty estimates
- Bayesian method is one of the natural choice [4]

[4] other possible idea: bootstrapping and bagging

Adversarial attack detection: being Bayesian helps!

- Fast gradient sign method (FGSM):
  $\mathbf{x}_{\text{adv}} = \mathbf{x} - \eta \cdot \text{sgn}(\nabla_{\mathbf{x}} \max_y \log p(y|\mathbf{x}))$
- Detection metric: predictive entropy
  $\mathbb{H}(p(\mathbf{y}|\mathbf{x}_{\text{adv}}, \mathbf{X}, \mathbf{Y}))$
  with the predictive distribution approximated by MC-dropout
- All 3 BNNs are more robust!
- ... and indeed very uncertain at $\mathbf{x}_{\text{adv}}$



MLP	$\alpha = 0.0$	$\alpha = 0.5$	$\alpha = 1.0$
0.2318	0.4879	0.5832	0.663

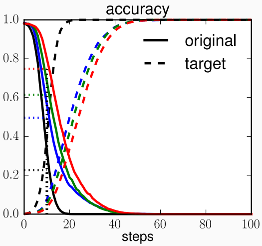
accuracy for stepsize=0.1

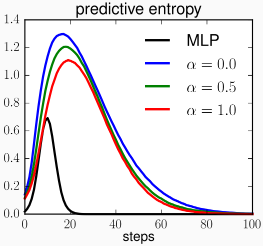
Adversarial attack detection: being Bayesian helps!

- Targeted FGSM (iterative, $\eta = 0.01$):

  $$\mathbf{x}_{\text{adv}}^{t} = \mathbf{x}_{\text{adv}}^{t-1} + \eta \cdot \text{sgn}(\nabla_{\mathbf{x}} \log p(y_{\text{target}} | \mathbf{x}_{\text{adv}}^{t-1}))$$

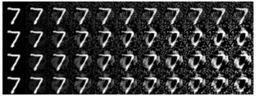
- All 3 BNNs are agian more robust!
- This attack on BNNs produces trajectories on the manifold!





MLP	$\alpha = 0.0$	$\alpha = 0.5$	$\alpha = 1.0$
0.2271	0.4960	0.6143	0.7480

original class acc. for #steps=10



Conclusions

- We proposed an approximate inference method for NNs that is
  - easy to understand and implement for deep learning people
  - while still maintain advantages of power-EP/BB-alpha
- Bayesian NNs are more useful in applications that needs calibrated uncertainty
- We think, adversarial attack detection, belongs to such set of applications
  - maybe can try BNNs + adversarial training + better metric for detection?

Thanks! # 61