A. Code Example

The following is a code snippet showing how our inference can be implemented with a few lines of Keras code (Chollet, 2015). We define a new loss function `bbalpha_softmax_cross_entropy_with_mc_logits`, that takes MC sampled logits as an input. This is demonstrated for the case of classification. Regression can be implemented in a similar way.

```
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
```

MC samples for this loss can be generated using `GenerateMCSamples`, with `layers` being a list of Keras initialised layers:

```
def GenerateMCSamples(inp, layers, K_mc=20):
  output_list = []
  for _ in xrange(K_mc):
    output_list += [apply_layers(inp, layers)]
  def pack_out(output_list):
    output = K.pack(output_list) # K_mc x nb_batch x nb_classes
    return K.permute_dimensions(output, (1, 0, 2)) # nb_batch x K_mc x nb_classes
  def pack_shape(s):
    s = s[0]
    return (s[0], K_mc, s[1])
  out = Lambda(pack_out, output_shape=pack_shape)(output_list)
  return out
```

The above two functions rely on the following auxiliary functions:

```
def logsumexp(x, axis=None):
  x_max = K.max(x, axis=axis, keepdims=True)
  return K.log(K.sum(K.exp(x - x_max), axis=axis, keepdims=True)) + x_max

def apply_layers(inp, layers):
  output = inp
  for layer in layers:
      output = layer(output)
  return output
```

B. Alpha-divergence minimisation

There are various available definitions of α-divergences, and in this work we mainly used two of them: Amari's definition (Amari, 1985) adapted to EP context (Minka, 2005), and Rényi divergence (Rényi, 1961) which is more used in information theory research.

- Amari's α-divergence (Amari, 1985):

$$\mathrm{D}_\alpha[p||q] = \frac{1}{\alpha(1-\alpha)}\left(1 - \int p(\omega)^\alpha q(\omega)^{1-\alpha} d\omega\right).$$

- Rényi's α-divergence (Rényi, 1961):

$$\mathrm{R}_\alpha[p||q] = \frac{1}{\alpha - 1}\log \int p(\omega)^\alpha q(\omega)^{1-\alpha} d\omega.$$

These two divergence can be converted to each other, e.g. $\mathrm{D}_\alpha[p||q] = \frac{1}{\alpha(1-\alpha)}\left(1 - \exp\left[(\alpha-1)\mathrm{R}_\alpha[p||q]\right]\right)$. In power EP (Minka, 2004), this α-divergence is minimised using projection-based updates. When the approximate posterior q has an exponential family form, minimising $\mathrm{D}_\alpha[p||q]$ requires moment matching to the "tilted distribution" $\tilde{p}_\alpha(\omega) \propto p(\omega)^\alpha q(\omega)^{1-\alpha}$. This projection update might be intractable for non-exponential family q distributions, and instead BB-α deploys a gradient-based update to search a local minimum. We will present the original derivation of the BB-α energy below and discuss how it relates to power EP.

C. Original Derivation of BB-α Energy

Here we include the original formulation of the BB-α energy for completeness. Consider approximating a distribution of the following form

$$p(\omega) = \frac{1}{Z} p_0(\omega) \prod_n^N f_n(\omega),$$

in which the prior distribution $p_0(\omega)$ has an exponential family form $p_0(\omega) \propto \exp\left[\lambda_0^T \phi(\omega)\right]$. Here λ_0 is called natural parameter or canonical parameter of the exponential family distribution, and $\phi(\omega)$ is the sufficient statistic. As the factors f_n might not be conjugate to the prior, the exact posterior no longer belongs to the same exponential family as the prior, and hence need approximations. EP construct such approximation by first approximating each complicated factor f_n with a simpler one $\tilde{f}_n(\omega) \propto \exp\left[\lambda_n^T \phi(\omega)\right]$, then constructing the approximate distribution as

$$q(\omega) = \frac{1}{Z(\lambda_q)} \exp\left[\left(\sum_{n=0}^N \lambda_n\right)^T \phi(\omega)\right],$$

with $\lambda_q = \lambda_0 + \sum_{n=1}^N \lambda_n$ and $Z(\lambda_q)$ the normalising constant/partition function. These *local* parameters are updated using the following procedure (for $\alpha \neq 0$):

1 compute cavity distribution $q^{\backslash n}(\omega) \propto q(\omega)/\tilde{f}_n(\omega)$, equivalently. $\lambda^{\backslash n} \leftarrow \lambda_q - \lambda_n$;

2 compute the tilted distribution by inserting the likelihood term $\tilde{p}_n(\omega) \propto q^{\backslash n}(\omega) f_n(\omega)$;

3 compute a projection update: $\lambda_q \leftarrow \arg\min_\lambda \mathrm{D}_\alpha[\tilde{p}_n||q_\lambda]$ with q_λ an exponential family with natural parameter λ;

4 recover the site approximation by $\lambda_n \leftarrow \lambda_q - \lambda^{\backslash n}$ and form the final update $\lambda_q \leftarrow \sum_n \lambda_n + \lambda_0$.

When converged, the solutions of λ_n return a fixed point of the so called *power EP energy*:

$$\mathcal{L}_{\text{PEP}}(\lambda_0, \{\lambda_n\}) = \log Z(\lambda_0) + (\frac{N}{\alpha} - 1) \log Z(\lambda_q) - \frac{1}{\alpha} \sum_{n=1}^N \log \int f_n(\omega)^\alpha \exp\left[(\lambda_q - \alpha\lambda_n)^T \phi(\omega)\right] d\omega. \tag{8}$$

But more importantly, before convergence all these local parameters λ_n are maintained in memory. This indicates that power EP does not scale with big data: consider Gaussian approximations which has $\mathcal{O}(d^2)$ parameters with d the dimensionality of ω. Then the space complexity of power EP is $\mathcal{O}(Nd^2)$, which is clearly prohibitive for big models like neural networks that are typically applied to large datasets. BB-α provides a simple solution of this memory overhead by sharing the local parameters, i.e. defining $\lambda_n = \lambda$ for all $n = 1, ..., N$. Furthermore, under the mild condition that the exponential family is regular, there exist a one-to-one mapping between λ_q and λ (given a fixed λ_0). Hence we arrive at a "global" optimisation problem in the sense that only one parameter λ_q is optimised, where the objective function is the BB-α energy

$$\mathcal{L}_\alpha(\lambda_0, \lambda_q) = \log Z(\lambda_0) - \log Z(\lambda_q) - \frac{1}{\alpha} \sum_{n=1}^N \log \mathbb{E}_q \left[\left(\frac{f_n(\omega)}{\exp\left[\lambda^T \phi(\omega)\right]}\right)^\alpha\right]. \tag{9}$$

One could verify that this is equivalent to the BB-α energy function presented in the main text by considering exponential family q distributions.

Although empirical evaluations have demonstrated the superior performance of BB-α, the original formulation is difficult to interpret for practitioners. First the local alpha-divergence minimisation interpretation is inherited from power EP, and

the intuition of power EP itself might already pose challenges for practitioners. Second, the derivation of BB-α from power EP is ad hoc and lacks theoretical justification. It has been shown that power EP energy can be viewed as the dual objective to a continuous version of Bethe free-energy, in which λ_n represents the Lagrange multiplier of the constraints in the primal problem. Hence tying the Lagrange multipliers would effectively changes the primal problem, thus losing a number of nice guarantees. Nevertheless this approximation has been shown to work well in real-world settings, which motivated our work to extend BB-α to dropout approximation.

D. Full Regression Results

Table 1. Regression experiment: Average negative test log likelihood/nats

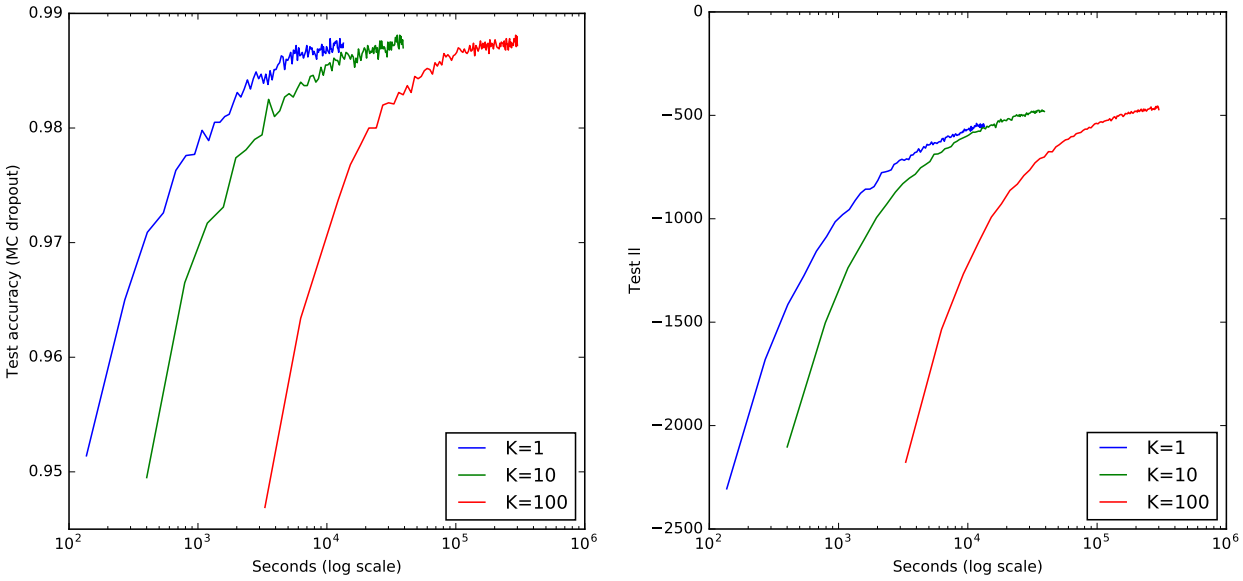
Dataset	N	D	$\alpha = 0.0$	$\alpha = 0.5$	$\alpha = 1.0$	**HMC**	**GP**	**VI-G**
boston	506	13	2.42±0.05	2.38±0.06	2.50±0.10	2.27±0.03	2.22±0.07	2.52±0.03
concrete	1030	8	2.98±0.02	2.88±0.02	2.96±0.03	2.72±0.02	2.85±0.02	3.11±0.02
energy	768	8	1.75±0.01	0.74±0.02	0.81±0.02	0.93±0.01	1.29±0.01	0.77±0.02
kin8nm	8192	8	-0.83±0.00	-1.03±0.00	-1.10±0.00	-1.35±0.00	-1.31±0.01	-1.12±0.01
power	9568	4	2.79±0.01	2.78±0.01	2.76±0.00	2.70±0.00	2.66±0.01	2.82±0.01
protein	45730	9	2.87±0.00	2.87±0.00	2.86±0.00	2.77±0.00	2.95±0.05	2.91±0.00
red wine	1588	11	0.92±0.01	0.92±0.01	0.95±0.02	0.91±0.02	0.67±0.01	0.96±0.01
yacht	308	6	1.38±0.01	1.08±0.04	1.15±0.06	1.62±0.01	1.15±0.03	1.77±0.01
naval	11934	16	-2.80±0.00	-2.80±0.00	-2.80±0.00	-7.31±0.00	-4.86±0.04	-6.49±0.29
year	515345	90	3.59±NA	3.54±NA	-3.59±NA	NA±NA	0.65±NA	3.60±NA

Table 2. Regression experiment: Average test RMSE

Dataset	N	D	$\alpha = 0.0$	$\alpha = 0.5$	$\alpha = 1.0$	**HMC**	**GP**	**VI-G**
boston	506	13	2.85±0.19	2.97±0.19	3.04±0.17	2.76±0.20	2.43±0.07	2.89±0.17
concrete	1030	8	4.92±0.13	4.62±0.12	4.76±0.15	4.12±0.14	5.55±0.02	5.42±0.11
energy	768	8	1.02±0.03	1.11±0.02	1.10±0.02	0.48±0.01	1.02±0.02	0.51±0.01
kin8nm	8192	8	0.09±0.00	0.09±0.00	0.08±0.00	0.06±0.00	0.07±0.00	0.08±0.00
power	9568	4	4.04±0.04	4.01±0.04	3.98±0.04	3.73±0.04	3.75±0.03	4.07±0.04
protein	45730	9	4.28±0.02	4.28±0.04	4.23±0.01	3.91±0.02	4.83±0.21	4.45±0.02
red wine	1588	11	0.61±0.01	0.62±0.01	0.63±0.01	0.63±0.01	0.57±0.01	0.63±0.01
yacht	308	6	0.76±0.05	0.85±0.06	0.88±0.06	0.56±0.05	1.15±0.09	0.81±0.05
naval	11934	16	0.01±0.00	0.01±0.00	0.01±0.00	0.00±0.00	0.00±0.00	0.00±0.00
year	515345	90	8.66±NA	8.80±NA	8.97±NA	NA±NA	0.79±NA	8.88±NA

E. Run time trade-off

We provide an assessment of the running time trade-offs of using an increasing number of samples at training time. Unlike VI, in our inference we rely on a large number of samples to reduce estimator bias. When a small number of samples is used ($K = 1$) our method collapses to standard VI. In Figure 8 we see both test accuracy as well as test log likelihood for a fully connected NN with four layers of 1024 units trained on the MNIST dataset, with $\alpha = 1$. The two metrics are shown as a function of wall-clock run time for different values of $K \in \{1, 10, 100\}$. As can be seen, $K = 1$ converges to test accuracy of 98.8% faster than the other values of K, which converge to the same accuracy. On the other hand, when assessing test log likelihood, both $K = 1$ and $K = 10$ attain value -600 within 1000 seconds, but $K = 10$ continues improving its test log likelihood and converges to value -500 after 3000 seconds. $K = 100$ converges to the same value but requires much longer running time, possibly because of noise from other processes.



(a) Test accuracy

(b) Test log likelihood

Figure 8. Run time experiment on the MNIST dataset for different number of samples K.