Supplementary Material for
Bayesian Triplet Loss: Uncertainty Quantification in Image Retrieval

Frederik Warburg[†], Martin Jørgensen[‡], Javier Civera[§], and Søren Hauberg[†]
[†]Technical University of Denmark, [‡]University of Oxford, [§]University of Zaragoza
[†]{frwa,sohau}@dtu.dk, [‡]martinj@robots.ox.ac.uk, [§]jcivera@unizar.es

1. Introduction

In this supplementary material we give additional details into the derivations of the mean and the variance for the Gaussian approximation of the proposed likelihood. We show through simulations that the Gaussian approximation of the likelihood is very accurate, even for a small number of dimensions. We provide the full derivation of the Expected Lower Bound (ELBO). Finally, we provide sample code for the proposed Bayesian Triplet loss, which highlights that the proposed method is simple to implement. We will make the entire code available upon acceptance.

2. Gaussian Likelihood

First, we provide the details into the derivations of the first two moments of the proposed Gaussian approximation of the likelihood.

2.1. The Mean

The expression

$$\mathbb{E}[\tau] = \mathbb{E}[p(p-2a)] - \mathbb{E}[n(n-2a)] \tag{1}$$

consists of two means of the same form. We first evaluate

$$\mathbb{E}[p(p-2a)] = \mathbb{E}[p^2] - 2\mathbb{E}[ap] = \mathbb{E}[p^2] - 2\mathbb{E}[a]\mathbb{E}[p], \tag{2}$$

where we have used that the expectation of the product of two independent variables is the product of the expectations. Using identical arguments for the second term of $\mathbb{E}[\tau]$ we get

$$\begin{aligned}\mathbb{E}[\tau] &= \mathbb{E}[p^2] - 2\mathbb{E}[a]\mathbb{E}[p] - \mathbb{E}[n^2] + 2\mathbb{E}[a]\mathbb{E}[n] &(3)\\ &= \mathbb{E}[p^2] - \mathbb{E}[n^2] - 2\mathbb{E}[a](\mathbb{E}[p] - \mathbb{E}[n]) &(4)\\ &= \mu_p^2 + \sigma_p^2 - \mu_n^2 - \sigma_n^2 - 2\mu_a(\mu_p - \mu_n). &(5)\end{aligned}$$

2.2. The variance

The expression

$$\begin{aligned}\mathrm{var}[\tau] &= \mathrm{var}[p(p-2a) - n(n-2a)] &(6)\\ &= \mathrm{var}[p(p-2a)] + \mathrm{var}[n(n-2a)] - 2\mathrm{cov}[p(p-2a), n(n-2a)] &(7)\end{aligned}$$

consists of three terms. We treat the terms one by one

$$\begin{aligned}\mathrm{var}[p(p-2a)] &= \mathrm{var}[p^2 - 2ap] &(8)\\ &= \mathrm{var}[p^2] + 4\mathrm{var}[ap] - 4\mathrm{cov}[p^2, ap] &(9)\\ &= 2\sigma_p^4 + 4\mu_p^2\sigma_p^2 + 4\left(\mathrm{var}[a] + \mathbb{E}[a]^2\right)\left(\mathrm{var}[p] + \mathbb{E}[p]^2\right) \\ &\quad - 4\mathbb{E}[a]^2\mathbb{E}[p]^2 - 4\mathrm{cov}[p^2, ap] &(10)\\ &= 2\sigma_p^4 + 4\mu_p^2\sigma_p^2 + 4\left(\sigma_a^2 + \mu_a^2\right)\left(\sigma_p^2 + \mu_p^2\right) - 4\mu_a^2\mu_p^2 \\ &\quad - 4\mathrm{cov}[p^2, ap] &(11)\end{aligned}$$

The last covariance term is

$$\begin{aligned}
\operatorname{cov}[p^2, ap] &= \mathbb{E}[ap^3] - \mathbb{E}[ap]E[p^2] && (12)\\
&= \mathbb{E}[a]\mathbb{E}[p^3] - \mathbb{E}[a]\mathbb{E}[p]\mathbb{E}[p^2] && (13)\\
&= \mu_a(\mu_p^3 + 3\mu_p\sigma_p^2) - \mu_a\mu_p(\mu_p^2 + \sigma_p^2) && (14)\\
&= \mu_a\left[\mu_p^3 + 3\mu_p\sigma_p^2 - \mu_p^3 - \mu_p\sigma_p^2)\right] && (15)\\
&= 2\mu_a\mu_p\sigma_p^2, && (16)
\end{aligned}$$

such that

$$\operatorname{var}[p(p-2a)] = 2\left[\sigma_p^4 + 2\mu_p^2\sigma_p^2 + 2\left(\sigma_a^2 + \mu_a^2\right)\left(\sigma_p^2 + \mu_p^2\right) - 2\mu_a^2\mu_p^2 - 4\mu_a\mu_p\sigma_p^2\right]. \quad (17)$$

The second term of the variance follows the same structure,

$$\operatorname{var}[n(n-2a)] = 2\left[\sigma_n^4 + 2\mu_n^2\sigma_n^2 + 2\left(\sigma_a^2 + \mu_a^2\right)\left(\sigma_n^2 + \mu_n^2\right) - 2\mu_a^2\mu_n^2 - 4\mu_a\mu_n\sigma_n^2\right]. \quad (18)$$

The last term of the variance is

$$\begin{aligned}
\operatorname{cov}[p(p-2a), n(n-2a)] &= \operatorname{cov}[p^2 - 2ap, n^2 - 2an] && (19)\\
&= \operatorname{cov}[p^2, n^2] - 2\operatorname{cov}[p^2, an] - 2\operatorname{cov}[ap, n^2] + 4\operatorname{cov}[ap, an] && (20)\\
&= 4\operatorname{cov}[ap, an] && (21)\\
&= 4\mathbb{E}[a^2pn] - 4\mathbb{E}[ap]\mathbb{E}[an] && (22)\\
&= 4\mathbb{E}[a^2]\mathbb{E}[p]\mathbb{E}[n] - 4\mathbb{E}[a]^2\mathbb{E}[p]\mathbb{E}[n] && (23)\\
&= 4\mathbb{E}[p]\mathbb{E}[n]\left(\mathbb{E}[a^2] - \mathbb{E}[a]^2\right) && (24)\\
&= 4\mathbb{E}[p]\mathbb{E}[n]\operatorname{var}[a] && (25)\\
&= 4\mu_p\mu_n\sigma_a^2. && (26)
\end{aligned}$$

Thus, given a mean and variance estimate for each embedding, we can analytically find the mean and variance of the Gaussian that approximate our proposed likelihood. In the next section, we show experimentally that this is good approximation.

2.3. Simulation Approximation of Gaussian Likelihood

In this section, we show that the Gaussian approximation of our proposed likelihood is accurate even for low dimensions. We sample μ_a, μ_p and μ_n from a D-dimensional standard Gaussian, and σ_a^2, σ_p^2 and σ_n^2 from a 1-dimensional standard Gaussian, where we ensure that σ_a^2, σ_p^2 and σ_n^2 are positive by taking the absolute value of the sample. We then generate N samples from $a \sim N(\mu_a, \sigma_a^2)$, $p \sim N(\mu_p, \sigma_p^2)$ and $n \sim N(\mu_n, \sigma_n^2)$, and calculate $\tau = \|a-p\|^2 - \|a-n\|^2$. We plot a histogram over the samples of τ as a Monte Carlo approximation of the true CDF (histograms in the figures below). We use the means and variances of a, p and n to analytically calculate the CDF of our Gaussian approximation (black line in the figures below). The figures show that the approximation is very accurate even for a very low number of dimensions (the deviation between the analytical CDF and the Monte Carlo simulation is only significant for $D < 4$). This means that the approximation is accurate for any practical retrieval system.

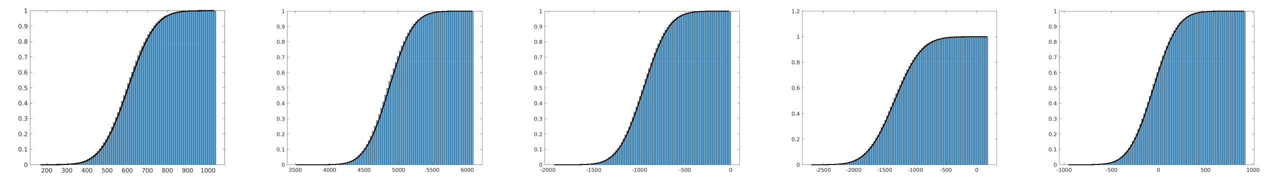

Figure 1: $D = 2048$

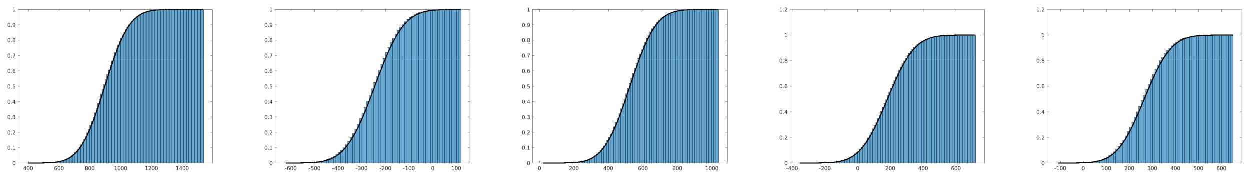

Figure 2: $D = 512$

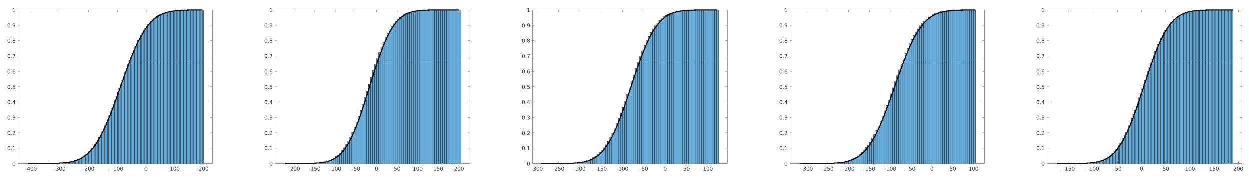

Figure 3: $D = 128$

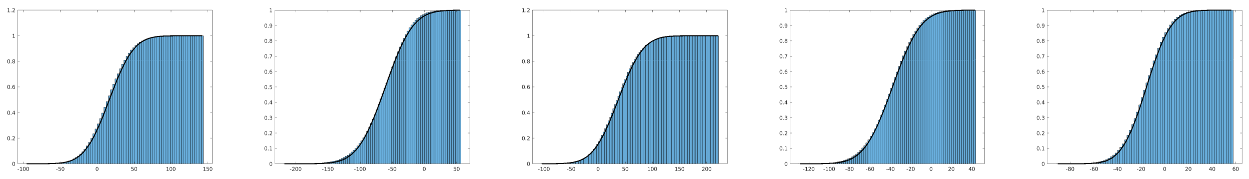

Figure 4: $D = 32$

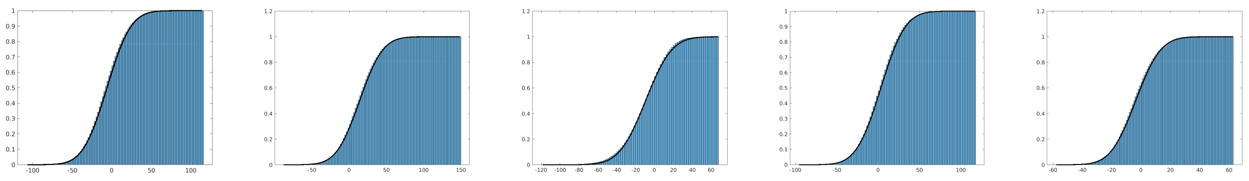

Figure 5: $D = 16$

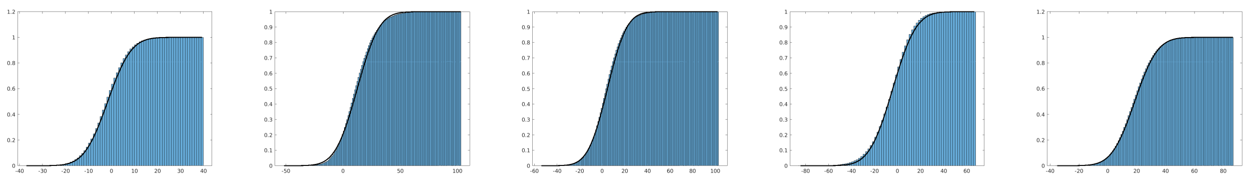

Figure 6: $D = 8$

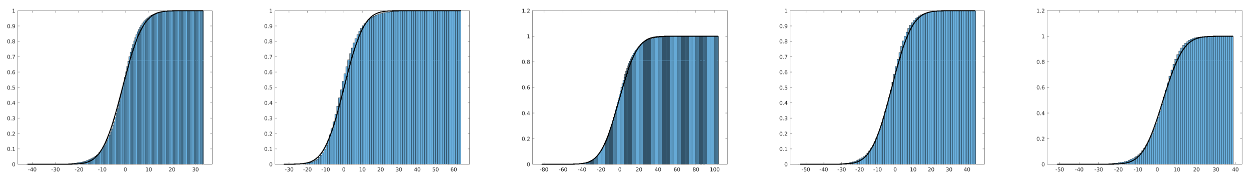

Figure 7: $D = 4$

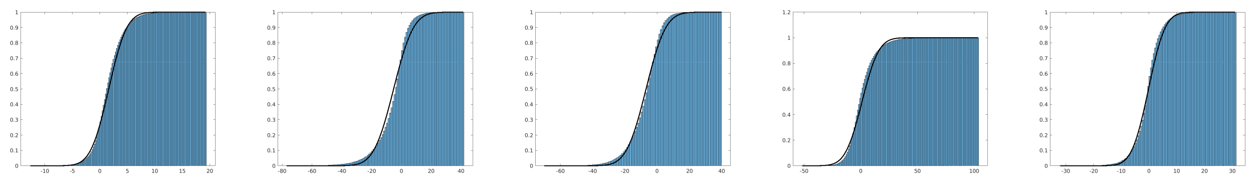

Figure 8: $D = 2$

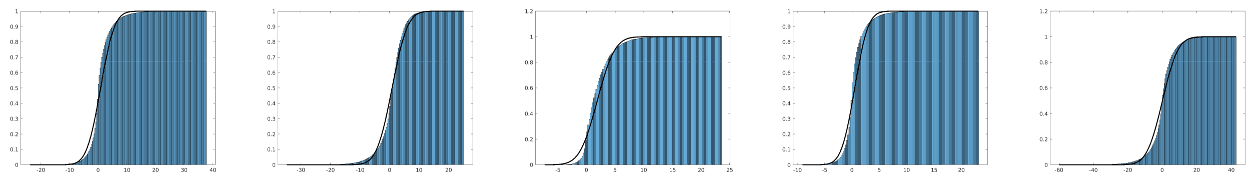

Figure 9: $D = 1$

3. Variational Approximation

In this section we derive the Expected Lower Bound. We start with the marginal probability of a triplet label $I(X,Y,Z)$.

$$logP(I(X,Y,Z)) = log\int P(I(X,Y,Z),(X,Y,Z))d(X,Y,Z) \tag{27}$$

$$= log\int\int\int P(I(X,Y,Z),X,Y,Z)dXdYdZ \tag{28}$$

$$= log\int\int\int P(I(X,Y,Z)|X,Y,Z)P(X,Y,Z)dXdYdZ \tag{29}$$

$$= log\int\int\int P(I(X,Y,Z)|X,Y,Z)P(X)P(Y)P(Z)dXdYdZ \tag{30}$$

$$= log\int\int\int \frac{P(I(X,Y,Z)|X,Y,Z)P(X)P(Y)P(Z)}{q(X)q(Y)q(Z)}q(X)q(Y)q(Z)dXdYdZ \tag{31}$$

$$\geq \int\int\int log\frac{P(I(X,Y,Z)|X,Y,Z)P(X)P(Y)P(Z)}{q(X)q(Y)q(Z)}q(X)q(Y)q(Z)dXdYdZ \tag{32}$$

$$= \int\int\int logP(I(X,Y,Z)|X,Y,Z)q(X)q(Y)q(Z) \tag{33}$$

$$+ log\frac{P(X)P(Y)P(Z)}{q(X)q(Y)q(Z)}q(X)q(Y)q(Z)dXdYdZ \tag{34}$$

$$= \int\int\int logP(I(X,Y,Z)|X,Y,Z)q(X)q(Y)q(Z)dXdYdZ \tag{35}$$

$$+ \int\int\int log\frac{P(X)P(Y)P(Z)}{q(X)q(Y)q(Z)}q(X)q(Y)q(Z)dXdYdZ \tag{36}$$

$$\tag{37}$$

We have two terms. The first part becomes

$$\int\int\int logP(I(X,Y,Z)|X,Y,Z)q(X)q(Y)q(Z)dXdYdZ \tag{38}$$

$$= \mathbb{E}_{q(X)q(Y)q(Z)}[logP(I(X,Y,Z)|X,Y,Z)] \tag{39}$$

The second part becomes

$$\int\int\int log\frac{P(X)P(Y)P(Z)}{q(X)q(Y)q(Z)}q(X)q(Y)q(Z)dXdYdZ \tag{40}$$

$$= \int\int\int [logP(X)-logq(X)+logP(Y)-logq(Y)+logP(Z)-logq(Z)]q(X)q(Y)q(Z)dXdYdZ \tag{41}$$

$$= \int\int\int [logP(X)-logq(X)]q(X)q(Y)q(Z)dXdYdZ \tag{42}$$

$$+ \int\int\int [logP(Y)-logq(Y)]q(X)q(Y)q(Z)dXdYdZ \tag{43}$$

$$+ \int\int\int [logP(Z)-logq(Z)]q(X)q(Y)q(Z)dXdYdZ \tag{44}$$

$$= \int [logP(X)-logq(X)]q(X)dX + \int [logP(Y)-logq(Y)]q(Y)dY + \int [logP(Z)-logq(Z)]q(Z)dZ \tag{45}$$

$$= -KL(q(X)||P(X)) - KL(q(Y)||P(Y)) - KL(q(Z)||P(Z)) \tag{46}$$

This means that if we choose $P(X)=P(Y)=P(Z)=N(0,1)$, we can remove the normalization layer and optimize the ELBO directly.

4. Implementation

We present our Pytorch implementation of the Bayesian triplet loss with both the von Mises Fischer and Gaussian embeddings. We will make the source code available upon acceptance.

```python
import torch
import torch.nn as nn
import functional as LF

class BayesianTripletLoss(nn.Module):

    def __init__(self, margin, varPrior, kl_scale_factor = 1e-6, distribution = 'gauss'):
        super(BayesianTripletLoss, self).__init__()

        self.margin = margin
        self.varPrior = varPrior
        self.kl_scale_factor = kl_scale_factor
        self.distribution = distribution

    def forward(self, x, label):

        # divide x into anchor, positive, negative based on labels
        D, N = x.shape
        nq = torch.sum(label.data == -1).item()  # number of tuples
        S = x.size(1) // nq  # number of images per tuple including query: 1+1+n
        A = x[:, label.data == -1].permute(1, 0).repeat(1, S - 2).view((S - 2) * nq, D).permute(1, 0)
        P = x[:, label.data == 1].permute(1, 0).repeat(1, S - 2).view((S - 2) * nq, D).permute(1, 0)
        N = x[:, label.data == 0]

        varA = A[-1:, :]
        varP = P[-1:, :]
        varN = N[-1:, :]

        muA = A[:-1, :]
        muP = P[:-1, :]
        muN = N[:-1, :]

        # calculate nll
        nll = LF.negative_loglikelihood(muA, muP, muN, varA, varP, varN, margin=self.margin)

        # KL(anchor|| prior) + KL(positive|| prior) + KL(negative|| prior)
        if self.distribution == 'gauss':

            muPrior = torch.zeros_like(muA, requires_grad = False)
            varPrior = torch.ones_like(varA, requires_grad = False) * self.varPrior

            kl = (LF.kl_div_gauss(muA, varA, muPrior, varPrior) + \
                    LF.kl_div_gauss(muP, varP, muPrior, varPrior) + \
                    LF.kl_div_gauss(muN, varN, muPrior, varPrior))

        elif self.distribution == 'vMF':
            kl = (LF.kl_div_vMF(muA, varA) + \
                    LF.kl_div_vMF(muP, varP) + \
                    LF.kl_div_vMF(muN, varN))

        return nll + self.kl_scale_factor * kl

    def __repr__(self):
        return self.__class__._Name__ + '(' + 'margin=' + '{:.4f}'.format(self.margin) + ')'
```

Listing 1: loss.py

```
import torch
from torch.distributions.normal import Normal

def negative_loglikelihood(muA, muP, muN, varA, varP, varN, margin = 0.0):

    muA2 = muA**2
    muP2 = muP**2
    muN2 = muN**2
    varP2 = varP**2
    varN2 = varN**2

    mu = torch.sum(muP2 + varP - muN2 - varN - 2*muA*(muP - muN), dim=0)
    T1 = varP2 + 2*muP2 * varP + 2*(varA + muA2)*(varP + muP2) - 2*muA2 * muP2 - 4*muA*muP*varP
    T2 = varN2 + 2*muN2 * varN + 2*(varA + muA2)*(varN + muN2) - 2*muA2 * muN2 - 4*muA*muN*varN
    T3 = 4*muP*muN*varA
    sigma2 = torch.sum(2*T1 + 2*T2 - 2*T3, dim=0)
    sigma = sigma2**0.5

    probs = Normal(loc = mu, scale = sigma + 1e-8).cdf(margin)
    nll = -torch.log(probs + 1e-8)

    return nll.mean()

def kl_div_gauss(mu_q, var_q, mu_p, var_p):

    N, D = mu_q.shape

    # kl diverence for isotropic gaussian
    kl = 0.5 * ((var_q / var_p) * D + \
            1.0 / (var_p) * torch.sum(mu_p**2 + mu_q**2 - 2*mu_p*mu_q, axis=1) - D + \
            D*(torch.log(var_p) - torch.log(var_q)))

    return kl.mean()

def kl_div_vMF(mu_q, var_q):

    N, D = mu_q.shape

    # we are estimating the variance and not kappa in the network.
    # They are propertional
    kappa_q = 1.0 / var_q
    kl = kappa_q - D * torch.log(2.0)

    return kl.mean()
```

Listing 2: functional.py