structured and efficient variational deep learning with matrix gaussian posteriors

christos louizos, max welling

1 kl divergence between matrix variate gaussian prior and posterior

let $m_{\gamma 0}(m_0, u_0, v_0)$ and $m_{\gamma 1}(m_1, u_1, v_1)$ be two matrix variate gaussian distributions for random matrices of size $n \times p$. we can use the fact that the matrix variate gaussian is a multivariate gaussian if we flatten the matrix, i.e. $m_{\gamma 0}(m_0, u_0, v_0) = n_0(\text{vec}(m_0), v_0 \otimes u_0)$, and as a result use the kl-divergence between two multivariate gaussians:

$$
\text{KL}(n_0 || n_1) = \frac{1}{2} \left( \text{tr}(\Sigma_1^{-1} \Sigma_0) + (\mu_1 - \mu_0)^{T} \Sigma_1^{-1} (\mu_1 - \mu_0) - k + \log |\Sigma_1| \right)
$$

$$
= \frac{1}{2} \left( \text{tr} \left( (v_1 \otimes u_1)^{-1} (v_0 \otimes u_0) \right) + \left( \text{vec}(m_1) - \text{vec}(m_0) \right)^{T} \\
(v_1 \otimes u_1)^{-1} \left( \text{vec}(m_1) - \text{vec}(m_0) \right) - np + \log |v_1 \otimes u_1| \\
|v_0 \otimes u_0| \right)
$$

now to compute each term in the kl efficiently we need to use some properties of the vectorization and kronecker product:

$$
t_a = \text{tr} \left( (v_1 \otimes u_1)^{-1} (v_0 \otimes u_0) \right) \\
= \text{tr} \left( (v_1^{-1} \otimes u_1^{-1}) (v_0 \otimes u_0) \right) \\
= \text{tr} \left( (v_1^{-1} v_0) \otimes (u_1^{-1} u_0) \right) \\
= \text{tr} (u_1^{-1} u_0) \text{tr} (v_1^{-1} v_0)
$$

$$
t_b = \left( \text{vec}(m_1) - \text{vec}(m_0) \right)^{T} (v_1 \otimes u_1)^{-1} \left( \text{vec}(m_1) - \text{vec}(m_0) \right) \\
= \text{vec}(m_1 - m_0)^{T} (v_1^{-1} \otimes u_1^{-1}) \text{vec}(m_1 - m_0) \\
= \text{vec}(m_1 - m_0)^{T} \text{vec}(u_1^{-1} (m_1 - m_0) v_1^{-1}) \\
= \text{tr} \left( (m_1 - m_0)^{T} u_1^{-1} (m_1 - m_0) v_1^{-1} \right)
$$

$$
t_c = \log \left| \frac{v_1 \otimes u_1}{v_0 \otimes u_0} \right| \\
= \log \left| \frac{u_1 |v_1|}{u_0 |v_0|} \right|^n \\
= p \log |u_1| + n \log |v_1| - \\
- p \log |u_0| - n \log |v_0|
$$

1
So putting everything together we have that:

\[
KL(\mathcal{M}_N, \mathcal{M}_1) = \frac{1}{2} \left( \text{tr}(U_1^{-1}U_0)\text{tr}(V_1^{-1}V_0) + \text{tr}((M_1 - M_0)^T U_1^{-1}(M_1 - M_0)V_1^{-1}) - np + p\log|U_1| + n\log|V_1| - p\log|U_0| - n\log|V_0| \right)
\]

(4)

2 Different toy dataset

We also performed an experiment with a different toy dataset that was employed in [2]. We generated 12 inputs from \(U[0, 0.6]\) and 8 inputs from \(U[0.8, 1]\). We then transform those inputs via:

\[
y_i = x_i + \epsilon_i + \sin(4(x_i + \epsilon_i)) + \sin(13(x_i + \epsilon_i))
\]

where \(\epsilon_i \sim \mathcal{N}(0, 0.0009)\). We continued in fitting four neural networks that had two hidden-layers with 50 units each. The first was trained with probabilistic back-propagation [1], and the remaining three with our model while varying the nonlinearities among the layers: we used ReLU, cosine and hyperbolic tangent activations. For our model we set the upper bound of the variational dropout rate to 0.2 and we used 2 pseudo data pairs for the input layer and 4 for the rest. The resulting predictive distributions can be seen at Figure 1.

![Figure 1](image)

Figure 1: Predictive distributions for the toy dataset. Grey areas correspond to \(\pm\{1, 2\}\) standard deviations around the mean function.
References
