Supplementary to Laplacian Segmentation Networks Improve Epistemic Uncertainty Quantification

1 Implementation and Training Details

Training configurations are provided in Table 1. All models were trained with the Adam optimizer. The U-net backbone was constructed with feature maps of size $8, 16, 32, 64, 128$. Uncertainty measures were approximated from 50 samples from the posterior and 20 samples from the logit distribution.

Table 1: Implementation and Training Details. Dropout models for the ISIC dataset were trained with a 0.0005 learning rate to improve convergence.

	Dataset		
Configuration	ISIC	Prostate	Brats
Epochs	60	150	600
Batch Size	32	10	32
Learning Rate	0.001*	0.001	0.001

2 Fast Hessian Approximation

Consider a neural network (NN) $f_\theta : \mathcal{X} \to \mathcal{Y}$ with L layers. The parameter $\theta = (\theta_1, \dots, \theta_L) \in \Theta$ is the concatenation of the parameters for each layer $i \in \{1, ..., L\}$. The NN $f_\theta = f^{(L)}_{\theta_L} \circ f^{(L-1)}_{\theta_{L-1}} \circ \ldots \circ f^{(2)}_{\theta_2} \circ f^{(1)}_{\theta_1}$ is a composition of L functions $f^{(L)}, f^{(L-1)}, \dots, f^{(1)}$, where $f^{(i)}$ is parametrized by θ_i. Let $x_0 \in \mathcal{X}$ be the input and $x_i := f^{(i)}_{\theta_i}(x_{i-1})$ for $i = 1, \dots, L$, such that the NN output is $x_L \in \mathcal{Y}$. We define the *diagonal* operator $\mathcal{D} : \mathbb{R}^{m\times m} \to \mathbb{R}^{m\times m}$ on quadratic matrices as

$$[\mathcal{D}(M)]_{ij} := \begin{cases} M_{ij} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \qquad \forall i, j = 1, \dots, m.$$

The **Jacobian** $J_\theta f_\theta(x_0)$ of the NN has a layer block structure, block i is

$$J_{\theta_i} f_\theta(x_0) = J_{\theta_i}\left(f^{(i)}_{\theta_i} \circ \cdots \circ f^{(L)}_{\theta_L}\right)(x_{i-1}) = \left(\prod_{j=L}^{i+1} J_{x_{j-1}} f^{(j)}_{\theta_j}(x_{j-1})\right) J_{\theta_i} f^{(i)}_{\theta_i}(x_{i-1}).$$

The Laplace approximation requires the Hessian $\mathbf{H}$ of the loss w.r.t. the parameters $\nabla^2_\theta \mathcal{L}(f_\theta(x_0)) \in \mathbb{R}^{|\theta|\times|\theta|}$. Using the chain rule it holds, that

$$\underbrace{\nabla^2_\theta \mathcal{L}(f_\theta(x_0))}_{=:H_\theta} = \underbrace{J_\theta f_\theta(x_0)^\top \cdot \nabla^2_{x_L} \mathcal{L}(x_L) \cdot J_\theta f_\theta(x_0)}_{=:\mathrm{GGN}_\theta} + \sum_{o=1}^{|x_L|} [\nabla_{x_L} \mathcal{L}(x_L)]_o \cdot \nabla^2_\theta [f_\theta(x_0)]_o,$$

where $[v]_o$ refers to the o-th component of vector v and $|v|$ to its length. We can write the diagonal block $\textsc{ggnb}_\theta^{(i)} = J_{\theta_i} f_\theta(x_0)^\top \mathbf{H}^{\mathcal{L}} J_{\theta_i} f_\theta(x_0)$ of the i-th layer as

$$\textsc{ggnb}_\theta^{(i)} = J_{\theta_i} f_\theta(x_0)^\top \cdot \mathbf{H}^{\mathcal{L}} \cdot J_{\theta_i} f_\theta(x_0) \tag{1}$$

From this expression, plus the chain rule expansion of the Jacobian, we can build an efficient backpropagation-like algorithm to compute $\textsc{ggnb}_\theta$, it start from $\mathbf{H}^{\mathcal{L}}$ and then iterated backward over layers. The same holds for the diagonal approximation, which we refer to as $\textsc{ggnd}_\theta := \mathcal{D}(\textsc{ggn}_\theta) = \mathcal{D}(\textsc{ggnb}_\theta)$. This approach already scales linearly in the number of parameter $|\theta|$. On top of that, the diagonal backpropagation approximates the diagonal of the Generalized Gauss-Newton matrix. It is defined, for each layer i, by adding a diagonalization operator in between each Jacobian product, marked red in Algorithm 1. Without this extra operator the algorithm would return the exact diagonal.

Algorithm 1 Computation of $\textsc{db}_\theta$

$M = \mathbf{H}^{\mathcal{L}}$
for $j = L, L-1, \dots, 1$ **do**
  $\textsc{db}_\theta^{(j)} = \mathcal{D}\left(J_{\theta_j} f_{\theta_j}^{(j)}(x_{j-1})^\top \cdot M \cdot J_{\theta_j} f_{\theta_j}^{(j)}(x_{j-1})\right)$
  $M = \mathcal{D}\left(J_{x_{j-1}} f_{\theta_j}^{(j)}(x_{j-1})^\top \cdot M \cdot J_{x_{j-1}} f_{\theta_j}^{(j)}(x_{j-1})\right)$
end for
$\textsc{db}_\theta = (\textsc{db}_\theta^{(1)}, \dots, \textsc{db}_\theta^{(L)})$
return $\textsc{db}_\theta$

Proposition. For an autoencoder network, the memory requirement of the Algorithm scale *linearly* both in number of parameter and in number of pixels.

Proof. The bottlenecks are the storage of the matrixes $\textsc{db}_\theta^{(j)} \in \mathbb{R}^{|\theta_j|}$ and $M \in \mathbb{R}^{|x_{j-1}|}$ at each step j

Skip-connections For any given submodule g_θ, a skip-connection layer $\textsc{sc}(g)_\theta$ is defined as $x \longmapsto (g_\theta(x), x)$. The Jacobian with respect to the parameter is the same as the Jacobian of the g_θ while the Jacobian with respect to the input is $J_x \textsc{sc}(g)_\theta(x) = \begin{pmatrix} J_x g_\theta(x) \\ \hline \mathbb{I} \end{pmatrix} \in \mathbb{R}^{(O+I)\times I}$.

Proposition. If M is diagonal, then one step of Alg 1 can be computed as

$$\mathcal{D}\left(J_x \textsc{sc}(g)_\theta(x)^\top \cdot M \cdot J_x \textsc{sc}(g)_\theta(x)\right) = \mathcal{D}\left(J_x g_\theta(x)^\top M_{11} J_x g_\theta(x)\right) + \mathcal{D}(M_{22}).$$

Proof. Let $M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}$ and then

$$\begin{aligned} J_x \textsc{sc}(g)_\theta(x)^\top \cdot M \cdot J_x \textsc{sc}(g)_\theta(x) &= \left(J_x g_\theta(x)^\top | \mathbb{I}\right) \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} J_x g_\theta(x) \\ \hline \mathbb{I} \end{pmatrix} \\ &= J_x g_\theta(x)^\top M_{11} J_x g_\theta(x) + M_{12} J_x g_\theta(x) + J_x g_\theta(x)^\top M_{21} + M_{22} \end{aligned}$$