Exercise Sheet #10

due on Monday, 10 July 2023, 10am sharp

1. **Theory Question:** The last lectures discussed the Laplace approximation for deep neural networks $f(x, \boldsymbol{\theta}) : \mathbb{X} \times \mathbb{R}^D \rightarrow \mathbb{R}^K$ (for simplicity, we will here assume $K = 1$) with parameters $\boldsymbol{\theta}$, trained by regularized empirical risk minimization to find the minimum of the loss (negative log posterior)

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \sum_{i=1}^{N} \ell(y_i, f(\boldsymbol{x}_i, \boldsymbol{\theta})) + r(\boldsymbol{\theta}).$$

and approximating the associated posterior as $p(\boldsymbol{\theta}|X, \boldsymbol{y}) \approx \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_*, -\Psi^{-1})$, where $\Psi = \nabla\nabla^\intercal \mathcal{L}(\boldsymbol{\theta}_*)$ is the Hessian of the loss at the optimum. An obvious challenge for this formalism is that the Hessian Ψ is of size $D \times D$, and thus inverting it is computationally expensive. In this exercise, we will explore a popular way to address this issue, known as the *Generalized Gauss-Newton (GGN)* approximation.[1]

   (a) Using the Chain rule, show that the Hessian Ψ can be written as

$$\Psi = \sum_{i=1}^{N} \left(J(x_i, \boldsymbol{\theta}) H(y_i, f(x_i, \boldsymbol{\theta})) J(x_i, \boldsymbol{\theta})^\intercal + L(y_i, f(x_i, \boldsymbol{\theta})) F(x_i, \boldsymbol{\theta}) \right) + \nabla\nabla^\intercal r(\boldsymbol{\theta})$$

   where $H \in \mathbb{R}$, $J \in \mathbb{R}^D$, $F \in \mathbb{R}^{D \times D}$ and $L \in \mathbb{R}$ and

$$[J(x_i, \boldsymbol{\theta})]_j = \frac{\partial f(x_i, \boldsymbol{\theta})}{\partial \theta_j} \qquad \text{is the Jacobian of } f \text{ wrt. the parameters}$$

$$H(y_i, f(x_i, \boldsymbol{\theta})) = \frac{\partial^2 \ell(y_i, f(x_i, \boldsymbol{\theta}))}{\partial f^2} \qquad \text{is the Hessian of } \ell \text{ wrt. the network}$$

$$[F(x_i, \boldsymbol{\theta})]_{nm} = \frac{\partial^2 f(x_i, \boldsymbol{\theta})}{\partial \theta_n \partial \theta_m} \qquad \text{is the Hessian of } f$$

$$L(y_i, f(x_i, \boldsymbol{\theta})) = \frac{\partial \ell(y_i, f(x_i, \boldsymbol{\theta}))}{\partial f(x_i, \boldsymbol{\theta})} \qquad \text{is the Jacobian of } \ell \text{ wrt. the network}$$

   Note that J is the vector that we also used in the lecture to contruct the Laplace tangent kernel (for multi-output models, it is a rectangular matrix).

   (b) Consider the two standard choices $\ell_2(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$ (square loss) and $\ell_{\text{ce}}(y_i, f(x_i)) = -\log(1 + \exp(-y_i f(x_i)))$ (cross-entropy loss). Compute the Jacobian L and Hessian H of the loss wrt. the network output $f(x_i)$ for both choices of loss. Use the results to show that, in each case, a "well-trained network" (i.e. a choice $\boldsymbol{\theta}_*$ that minimizes the loss) satisfies $L(y_i, f(x_i, \boldsymbol{\theta}_*)) \approx 0$, and $H \geq 0$. Thus, the Hessian can be computed efficiently (in $\mathcal{O}(DN)$) using only the first term in Ψ above, and the resulting approximation is positive semi-definite.[2]

   (c) If we consider $r(\boldsymbol{\theta}) = \alpha I$, then the above results suggest that we can approximately write the Hessian as a scalar plus rank-N matrix :

$$\Psi = \alpha I + \sum_{i=1}^{N} g_i g_i^\intercal \qquad \left(\text{with } g_i = J(x_i, \boldsymbol{\theta}_*) \sqrt{H(y_i, f(x_i, \boldsymbol{\theta}_*))}\right)$$

[1] Nicol N Schraudolph, *Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent.* Neural computation, **14**(7), 2002.

[2] Specifically for these two choices, the resulting form of Ψ is also equal to the Fisher information matrix of the model. But this is not important to understand this question. Cf. J. Martens *New insights and perspectives on the natural gradient method.* Journal of Machine Learning Research, **21**(146):1–76, 2020.

Use the matrix inversion Lemma to provide an explicit expression of the inverse Ψ^{-1} used in the Laplace approximation. What is the computational complexity of computing Ψ^{-1} in this case, in terms of N and D?

2. **Practical Question:** can be found in `Ex10.ipynb`