

Homework 2 – Deep Learning (CS/DS 541, Murai, Spring 2022)

(Adapted from Prof. Whitehill)

You may complete this homework assignment either individually or in teams up to 2 people.

1. **XOR problem** [10 points, on paper]: Show (by deriving the gradient, setting to 0, and solving mathematically, not in Python) that the values for $\mathbf{w} = (w_1, w_2)$ and b that minimize the function $J(\mathbf{w}, b)$ in Equation 6.1 (in the *Deep Learning* textbook) are: $w_1 = 0$, $w_2 = 0$, and $b = 0.5$.
2. **L_2 -regularized Linear Regression via Stochastic Gradient Descent** [20 points, in Python]: Train a 2-layer neural network (i.e., linear regression) for age regression using the same data as in homework 1. Your prediction model should be $\hat{y} = \mathbf{x}^\top \mathbf{w} + b$. You should regularize \mathbf{w} but not b . Note that, in contrast to Homework 1, this model includes a bias term.

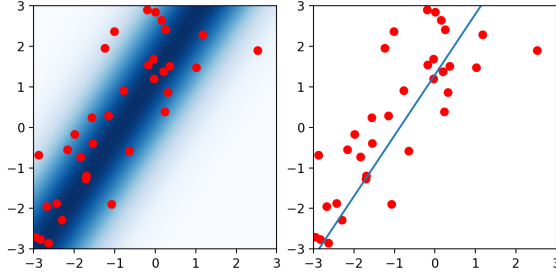
Instead of optimizing the weights of the network with the closed formula, use stochastic gradient descent (SGD). There are several different hyperparameters that you will need to choose:

- Mini-batch size \tilde{n} .
- Learning rate ϵ .
- Number of epochs.
- L_2 Regularization strength α .

In order not to cheat (in the machine learning sense) – and thus overestimate the performance of the network – it is crucial to optimize the hyperparameters **only** on a *validation set*. (The training set would also be acceptable but typically leads to worse performance.) To create a validation set, simply set aside a fraction (e.g., 20%) of the `age_regression_Xtr.npy` and `age_regression_ytr.npy` to be the validation set; the remainder (80%) of these data files will constitute the “actual” training data. While there are fancier strategies (e.g., Bayesian optimization – another probabilistic method, by the way!) that can be used for hyperparameter optimization, it’s common to just use a grid search over a few values for each hyperparameter. In this problem, you are required to explore systematically (e.g., using nested `for` loops) at least 4 different parameters for each hyperparameter.

Performance evaluation: Once you have tuned the hyperparameters and optimized the weights so as to minimize the cost on the validation set, then: (1) **stop** training the network and (2) evaluate the network on the **test** set. Report the performance in terms of *unregularized* MSE.

3. **Logistic Sigmoid Identities** [8 points, on paper]: The logistic sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.
 - (a) Prove that $\sigma(-x) = 1 - \sigma(x) \quad \forall x$.
 - (b) Prove that $\sigma'(x) = \frac{\partial \sigma}{\partial x}(x) = \sigma(x)(1 - \sigma(x)) \quad \forall x$.
4. **Regularization to encourage symmetry** [8 points, on paper]: Faces (and some other kinds of data) tend to be left-right symmetric. How can you use L_2 regularization to discourage the weights from becoming too *asymmetric*? For simplicity, consider the case of a tiny 1×2 “image”. Hint: instead of using $\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} = \frac{\alpha}{2} \mathbf{w}^\top \mathbf{I} \mathbf{w}$ as the L_2 penalty term (where α is the regularization strength), consider a different matrix in the middle. Your answer should consist of a 2×2 matrix \mathbf{S} as well as an explanation of why it works.
5. **Linear-Gaussian prediction model** [14 points, on paper]:



Probabilistic prediction models enable us to estimate not just the “most likely” or “expected” value of the target y (see figure above, right), but rather an entire *probability distribution* about which target values are more likely than others, given input \mathbf{x} (see figure above, left). In particular, a linear-Gaussian model is a Gaussian distribution whose expected value (mean μ) is a linear function of the input features \mathbf{x} , and whose variance is σ^2 :

$$P(y \mid \mathbf{x}) = \mathcal{N}(\mu = \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2}\right)$$

This prediction model is equivalent to a 2-layer linear neural network that outputs $\mu(\mathbf{x})$ as well as σ^2 . Note that, in general, σ^2 can also be a function of \mathbf{x} (heteroscedastic case). Moreover, *non-linear* Gaussian models are also completely possible, e.g., the mean (and possibly the variance) of the Gaussian distribution is output by a deep neural network. However, in this problem, we will assume that μ is linear in \mathbf{x} , and that σ^2 is the same for all \mathbf{x} (homoscedastic case).

MLE: The parameters of probabilistic models are commonly optimized by *maximum likelihood estimation* (MLE). (Another common approach is *maximum a posteriori* estimation, which allows the practitioner to incorporate a “prior belief” about the parameters’ values.) Suppose the training dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$. Let the parameters/weights of the linear-Gaussian model be \mathbf{w} , such that the mean $\mu = \mathbf{x}^\top \mathbf{w}$. Prove that the MLE of \mathbf{w} and σ^2 given \mathcal{D} is:

$$\mathbf{w} = \left(\sum_{i=1}^n \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \right)^{-1} \left(\sum_{i=1}^n \mathbf{x}^{(i)} y^{(i)} \right)$$

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)})^2$$

Note that this solution – derived based on *maximizing* probability – is exactly the same as the optimal weights of a 2-layer neural network optimized to *minimize* MSE.

Hint: Follow the same strategy as the MLE derivation for a biased coin in Lecture 3. For a linear-Gaussian model, the argmax of the likelihood equals the argmax of the log-likelihood. The log of the Gaussian likelihood simplifies beautifully.

Put your code in a Python file called `homework2_WPIUSERNAMES.py`. For the proofs, please create a PDF called `homework2_WPIUSERNAMES.pdf`. Create a Zip file containing both your Python and PDF files, and then submit on Canvas.