Homework 2 – Deep Learning (CS/DS 541, Whitehill, Spring 2021)

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 $b = 0.5$. *Answer*: Let $\mathbf{w} = [w_1, w_2]^\top$.

$$
\begin{aligned}
J(w_1, w_2, b) &= \frac{1}{4} \sum (\mathbf{x}^{(i)^\top} \mathbf{w} + b - y^{(i)})^2 \\
&= \frac{1}{4} \big((w_1 * 0 + w_2 * 0 + b - 0)^2 + (w_1 * 1 + w_2 * 0 + b - 1)^2 + \\
&\qquad (w_1 * 0 + w_2 * 1 + b - 1)^2 + (w_1 * 1 + w_2 * 1 + b - 0)^2 \big)
\end{aligned}
$$

Now we find each partial derivative, set to 0, and solve:

$$
\begin{aligned}
\frac{\partial J}{\partial w_1} &= \frac{1}{2}(w_1 + b - 1 + w_1 + w_2 + b) && (1) \\
&= \frac{1}{2}(2w_1 + w_2 + 2b - 1) && (2) \\
\frac{\partial J}{\partial w_2} &= \frac{1}{2}(w_2 + b - 1 + w_1 + w_2 + b) && (3) \\
&= \frac{1}{2}(2w_2 + w_1 + 2b - 1) && (4) \\
\frac{\partial J}{\partial b} &= \frac{1}{2}(b + w_1 + b - 1 + w_2 + b - 1 + w_1 + w_2 + b) && (5) \\
&= \frac{1}{2}(2w_1 + 2w_2 + 4b - 2) && (6)
\end{aligned}
$$

Setting to 0 and then subtracting Eq. 4 from 2, we have:

$$
\begin{aligned}
w_1 - w_2 &= 0 && (7) \\
w_1 &= w_2 && (8)
\end{aligned}
$$

Substituting $w_2 = w_1$ into Eq. 6 and setting to 0, we obtain:

$$
\begin{aligned}
\frac{1}{2}(4w_1 + 4b - 2) = 2w_1 + 2b - 1 &= 0 && (9) \\
w_1 &= \frac{1 - 2b}{2} && (10)
\end{aligned}
$$

Finally, we substitute back into Eq. 4 and set to 0:

$$
\begin{aligned}
\frac{1}{2}(3(1 - 2b)/2 + 2b - 1) &= 0 && (11) \\
3/2 - 3b + 2b - 1 &= 0 && (12) \\
b &= 1/2 && (13)
\end{aligned}
$$

and finally we deduce from Eq. 6:

$$
\begin{aligned}
2w_1 + 2(1/2) - 1 &= 0 && (14) \\
w_1 = w_2 &= 0 && (15)
\end{aligned}
$$

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.

Solution: Here are two of the key methods, where `alpha` is the regularization strength parameter:

```
def f_MSE_unreg (w, b, X, y, alpha):
    yhat = X.dot(w) + b
    return 0.5/len(y) * np.sum(np.power(yhat - y, 2))

def grad_f_MSE (w, b, X, y, alpha):
    yhat = X.dot(w) + b
    grad_w = 1./len(y) * np.dot(X, yhat - y) + alpha * w
    grad_b = 1./len(y) * np.sum(yhat - y)
    return grad_w, grad_b
```

3. **Regularization to encourage symmetry** [10 points, on paper]: Faces tend to be left-right symmetric. How can you use L_2 regularization to discourage the weights from becoming too *a*symmetric? For simplicity, consider the case of a tiny 1×2 "image". Hint: instead of using $\alpha \mathbf{w}^\top \mathbf{w} = \alpha \mathbf{w}^\top \mathbf{I} \mathbf{w}$ as the L_2 penalty term, 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.

   Solution: Let $\mathbf{w} = [w_1, w_2]$ be the weights on the left and right pixels, respectively. We want $(w_1 - w_2)^2$ to be small so that these two weights have similar values. We can express this quantity as:

   $$\frac{\alpha}{2}\left(\mathbf{w}^\top \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right)\left(\begin{bmatrix} 1 & -1 \end{bmatrix} \mathbf{w}\right) = \frac{\alpha}{2}\mathbf{w}^\top \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{w}$$

   Thus, $\mathbf{S} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$.

4. **Recursive state estimation in Hidden Markov Models** [10 points, on paper]: Teachers try to monitor their student's knowledge of the subject-matter, but teachers cannot directly peer inside students' brains. Hence, they must make *inferences* about what the student knows based on students'

observable behavior, i.e., how they perform on tests, their facial expressions during class, etc. Let random variable (RV) X_t represent the student's *state*, and let RV Y_t represent the student's observable behavior, at time t. We can model the student as a Hidden Markov Model (HMM):

(a) X_t depends *only* on the previous state X_{t-1}, *not* on any states prior to that (*Markov* property), i.e.

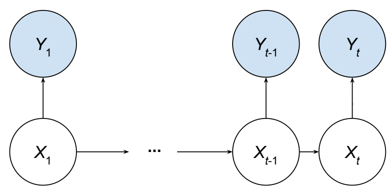
$$P(x_t \mid x_1, \ldots, x_{t-1}) = P(x_t \mid x_{t-1})$$

(b) The student's behavior Y_t depends only on his/her current state X_t, i.e.:

$$P(y_t \mid x_t, y_1, \ldots, y_{t-1}) = P(y_t \mid x_t)$$

(c) X_t cannot be observed directly (it is *hidden*).

A probabilistic graphical model for the HMM is shown below, where only the observed RVs are shaded (the latent ones are transparent):



Suppose that the teacher already knows:

- $P(y_t \mid x_t)$ (*observation likelihood*), i.e., the probability distribution of the student's behaviors given the student's state.
- $P(x_t \mid x_{t-1})$ (*transition dynamics*), i.e., the probability distribution of the student's current state given the student's previous state.

The goal of the teacher is to estimate the student's current state X_t given the *entire* history of observations $Y_1, \ldots, Y_t$ he/she has made so far. Show that the teacher can, at each time t, update his/her belief *recursively*:

$$P(x_t \mid y_1, \ldots, y_t) \propto P(y_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1} \mid y_1, \ldots, y_{t-1})$$

where $P(x_{t-1} \mid y_1, \ldots, y_{t-1})$ is the teacher's belief of the student's state from time $t-1$, and the summation is over every possible value of the previous state x_{t-1}. **Hint**: You will need to use Bayes' rule, i.e., for any RVs A, B, and C:

$$P(a \mid b, c) = \frac{P(b \mid a, c) P(a \mid c)}{P(b \mid c)}$$

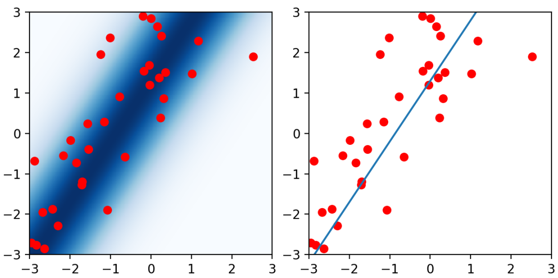
However, since the denominator in the right-hand side does not depend on a, this can also be rewritten as:

$$P(a \mid b, c) \propto P(b \mid a, c) P(a \mid c)$$

Solution:

$$\begin{aligned}
P(x_t \mid y_1, \ldots, y_t) &\propto P(y_t \mid x_t, y_1, \ldots, y_{t-1}) P(x_t \mid y_1, \ldots, y_{t-1}) \quad \text{(Bayes rule)} \\
&= P(y_t \mid x_t) \sum_{x_{t-1}} P(x_t, x_{t-1} \mid y_1, \ldots, y_{t-1}) \quad \text{(Total prob.)} \\
&= P(y_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}, y_1, \ldots, y_{t-1}) P(x_{t-1} \mid y_1, \ldots, y_{t-1}) \quad \text{(Cond. prob.)} \\
&= P(y_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1} \mid y_1, \ldots, y_{t-1}) \quad \text{(Cond. indep.)}
\end{aligned}$$

5. **Linear-Gaussian prediction model** [15 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:

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

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.

Solution:

$$
\begin{aligned}
\log P(\mathcal{D} \mid \mathbf{w}, \sigma^2) &= \log \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2) \\
&= \sum_{i=1}^{n} \log P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2) \\
&= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-(\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2/(2\sigma^2)) \\
&= \sum_{i=1}^{n} \left(-(\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2/(2\sigma^2) - \log\sigma\right) + C \\
&= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2 \ - n\log\sigma + C
\end{aligned}
$$

where C is some constant that does not depend on $\mathbf{w}$ or σ^2. To find the MLE, we differentiate the negative log-likelihood w.r.t. both $\mathbf{w}$ and σ, set to 0, and solve:

$$
\begin{aligned}
\nabla_{\mathbf{w}} \log P(\mathcal{D} \mid \mathbf{w}, \sigma^2) &= \nabla_{\mathbf{w}} \left(\frac{1}{2\sigma^2} \sum_{i=1}^{n} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2 \ + n\log\sigma + C \right) \\
0 &= \frac{1}{2\sigma^2} \sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)}) \\
\mathbf{w} &= \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)}{}^\top \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right) \\
\nabla_{\sigma} \log P(\mathcal{D} \mid \mathbf{w}, \sigma^2) &= \nabla_{\sigma} \left(\frac{1}{2\sigma^2} \sum_{i=1}^{n} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2 \ + n\log\sigma + C \right) \\
0 &= -\frac{1}{\sigma^3} \sum_{i=1}^{n} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2 + \frac{n}{\sigma} \\
\sigma^2 &= \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)}{}^\top \mathbf{w} - y^{(i)})^2
\end{aligned}
$$