HOMEWORK 3

CSC2515 Fall 2024

- **Deadline:** Friday, November 8, 2024 at 11:59PM.
- **Submission:** You need to submit the following files through MarkUs.
  - A PDF file including all your answers and plots.
  - A Python file, `4.1.py` containing your solution to Question 4.1. (You should write your own file for this, though you are free to make use of the `data.py` helper library. There is no direct starter code for this problem).
  - A Python file, `4.2.py` containing your solution to Question 4.2. (This should be extended from the starter code for this problem).
  - A Python file, `4.3.py` containing your solution to Question 4.3. (This should be extended from the starter code for this problem).

  You can produce your written PDF file however you like (e.g. LaTeX, Microsoft Word, etc) as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. If the code does not run, you may lose most/all of your points for that question. Your report must include all the relevant figures and graphs. We may or may not look at your code or Jupyter Notebook.
- **Late Submission:** 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.
- **Collaboration:** You can discuss the assignment with up to two other students (group of three). You can work on the code together. But each of you need to **write your homework report individually**. You must mention the name of your collaborators clearly in the report and the source code.

1. Backpropagation – 20pts.

The goal of this exercise is to help you practice how backpropagation works. We consider a simple variation of the feedforward fully-connected network. In the usual feedforward fully-connected network, each layer is connected to its previous layer. The main difference here is that the second hidden layers is connected to the input too. The computation graph and how each computation is performed is as follows:

$$
\begin{aligned}
z_1 &= W^{(1)}x \quad \text{with } x \in \mathbb{R}^d \\
h &= \sigma(z_1) \quad \text{with } h \in \mathbb{R}^d \\
z_2 &= h + x \\
y &= W^{(2)}z_2 \\
\mathcal{L} &= \frac{1}{2}(y-t)^2 \text{ with } t \in \mathbb{R}.
\end{aligned}
$$

Here σ is the activation function, and you can assume that it is differentiable. Answer the following questions:

(a) **[4pt]** Determine the dimensions of $W^{(1)}$, $W^{(2)}$, z_1, and z_2.

(b) **[2pt]** Calculate the number of parameters in this network, as a function of d. You need to show how you get to the solution.

(c) **[14pt]** Compute the gradient of loss $\mathcal{L}$ with respect to all variables. That is, compute

- $\bar{y} = \frac{\partial \mathcal{L}}{\partial y} = \dots$
- $\bar{W}^{(2)} = \frac{\partial \mathcal{L}}{\partial W^{(2)}} = \dots$
- $\bar{z}_2 = \dots$
- $\bar{h} = \dots$
- $\bar{z}_1 = \dots$
- $\bar{W}^{(1)} = \dots$
- $\bar{x} = \dots$

You need to show all the steps and simplify the solution.

2. Multi-Class Logistic Regression – 10pts.

The goal of this exercise is to verify the formula on Slide 96 of Lecture 3. Consider

$$\begin{aligned}\mathbf{z} &= W\mathbf{x}\\ \mathbf{y} &= \mathrm{softmax}(\mathbf{z})\\ \mathcal{L}_{\mathrm{CE}}(\mathbf{t},\mathbf{y}) &= -\mathbf{t}^\top \log \mathbf{y} = -\sum_{k=1}^{K} t_k \log y_k\end{aligned}$$

Note that if $x \in \mathbb{R}^d$, the dimension of W is $K \times d$. We denote its k-th row by $\mathbf{w}_k$. The vector $\mathbf{y}$ is a function of W and x. And the output $\mathbf{t}$ is a one-hot encoding of the output.

Recall that the k-th component of $\mathbf{y}$ is

$$y_k = \mathrm{softmax}(z_1, \ldots, z_K)_k = \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})}.$$

(a) **[5pt]** Compute

$$\frac{\partial y_k}{\partial z_{k'}},$$

for any $k, k' = 1, \ldots, K$ (note that k and k' may or may not be the same). Try to write it in a compact form (no $\exp(\cdots)$ would be needed).

(b) **[5pt]** Compute

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}(\mathbf{t}, \mathbf{y}(\mathbf{x}; W))}{\partial \mathbf{w}_k}.$$

You need to show all the derivations in order to get the full mark. The final solution alone will not give you any mark, as it is already shown on the slide.

3. Class-Conditional Gaussians – 30 pts. In this question, you will derive the maximum likelihood estimates for class-conditional Gaussians with independent features (diagonal covariance matrices), i.e., Gaussian Naïve Bayes, with shared variances.

Start with the following generative model for a discrete class label $y \in \{1, 2, ..., k\}$ and a real-valued vector of d features $\mathbf{x} = (x_1, x_2, ..., x_d)$:

$$p(y = k) = \alpha_k, \tag{3.1}$$

$$p(\mathbf{x}|y = k, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \left(\prod_{i=1}^{D} 2\pi\sigma_i^2\right)^{-1/2} \exp\left(-\sum_{i=1}^{D} \frac{(x_i - \mu_{ki})^2}{2\sigma_i^2}\right), \tag{3.2}$$

where α_k is the prior on class k, σ_i^2 are the shared variances for each feature (common for all classes), and μ_{ki} is the mean of the feature i conditioned on class k. We use $\boldsymbol{\alpha}$ to denote the vector with elements α_k. Similarly, $\boldsymbol{\sigma}$ denotes the vector of variances. The matrix of class means is written $\boldsymbol{\mu}$ where the kth row of $\boldsymbol{\mu}$ is the mean for class k.

1. **[4pt]** Use Bayes' rule to derive an expression for $p(y = k|x, \boldsymbol{\mu}, \boldsymbol{\sigma})$. [**Hint**: Use the law of total probability to derive an expression for $p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\sigma})$.]

2. **[8pt]** Write down the expression for the negative likelihood function (NLL)

$$\ell(\boldsymbol{\theta}; D) = -\log p\Big(y^{(1)}, \mathbf{x}^{(1)}, y^{(2)}, \mathbf{x}^{(2)}, \cdots, y^{(N)}, \mathbf{x}^{(N)} \mid \boldsymbol{\theta}\Big) \tag{3.3}$$

   of a particular dataset $D = \{(y^{(1)}, \mathbf{x}^{(1)}), (y^{(2)}, \mathbf{x}^{(2)}), \cdots, (y^{(N)}, \mathbf{x}^{(N)})\}$ with parameters $\boldsymbol{\theta} = \{\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\sigma}\}$ (assume that the data points are i.i.d.)
3. **[10pt]** Take partial derivatives of the likelihood with respect to each of the parameters μ_{ki} and with respect to the shared variances σ_i^2.
4. **[8pt]** Find the maximum likelihood estimates for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

4. Handwritten Digit Classification – 65 points.
For this question you will build classifiers to label images of handwritten digits. Each image is 8 by 8 pixels and is represented as a vector of dimension 64 by listing all the pixel values in raster scan order. The images are grayscale and the pixel values are between 0 and 1. The labels y are $0, 1, 2, \cdots, 9$ corresponding to which character was written in the image. There are 700 training cases and 400 test cases for each digit; they can be found in a3digits.zip.

Starter code written in Python is provided to help you load the data. A skeleton is also provided for each question that you should use to format your solution. You are welcome to use any other programming language, as long as your code is functional and modular.

Please note that if you are asked to report/compute quantities these should be clearly displayed in your written report. It is not sufficient to simply print these as an output of your code. The same applies to plots and figures.

4.1. *Warmup: Multi-Layer Perceptron Classifier – 10pt.*

(a) **[2pt]** Load the data and plot the means for each of the digit classes in the training data (include these in your report). Given that each image is a vector of size 64, the mean will be a vector of size 64 which needs to be reshaped as an 8×8 2D array to be rendered as an image. Plot all 10 means side by side using the same scale.

(b) **[5pt]** Build a simple multi-layer perceptron neural network using PyTorch. Your input layer needs one unit for each pixel; and given that you are distinguishing among 10 classes, the output layer will need 10 units. Other than that, you are free to experiment with the network architecture (e.g., number of hidden layers).

i. Keep the size (width) of each layer fixed at some value K_w (which you choose and specify in your report). Keeping all other hyperparameters constant (e.g., learning rate, optimizer, etc.), vary the depth of the network from 0 to 10, and report the train and test classification accuracy associated with each network depth.

ii. Keep the depth (width) of each layer fixed at some value K_d (which you choose and specify in your report). Keeping all other hyperparameters constant (e.g., learning rate, optimizer, etc.), vary the width of each network layer for ten different values of your choosing. Report the train and test classification accuracy associated with each width.

(c) **[4pt]** Keeping the architecture of the network fixed at five layers of width 64 each, use 10 fold cross validation to find the optimal *dropout* for the network (apply this dropout uniformly to each network layer). You may use the KFold implementation in sklearn. Report the value of this dropout parameter along with the train, validation, and test set classification accuracies, averaged across folds where applicable.

(d) Using the best performing model from the previous section, briefly summarize its performance using the following metrics. In many practical settings, measuring MSE or error rate is insufficient; instead, the following metrics (which were not explicitly covered in lecture, but will be *very* useful for your final projects) are more commonly used in practical settings. Your response for each metric should contain **two things**: (a) a 1-2 sentence description of what each metric is measuring, and (b) the reported value

for that metric using your neural network model. Please evaluate the following five metrics:

i. ROC (Receiver Operating Characteristics) curve
ii. Confusion Matrix
iii. Accuracy
iv. Precision
v. Recall

(The following may be a good starting point to build your intuition about each of the above metrics: Receiver operating characteristic)

4.2. *Conditional Gaussian Classifier Training – 25 pts.*

Using maximum likelihood, fit a set of 10 class-conditional Gaussians with a separate, full covariance matrix for each class. Remember that the conditional multivariate Gaussian probability density is given by

$$p(\mathbf{x}|y=k,\boldsymbol{\mu},\Sigma_k) = (2\pi)^{-d/2}|\Sigma_k|^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right). \tag{4.1}$$

You should take $p(y=k)=\frac{1}{10}$. You will compute parameters μ_{kj} and Σ_k for $k \in \{0,\ldots,9\}$ and $j \in \{1,\ldots,64\}$. You should implement the covariance computation yourself, i.e., without the aid of 'np.cov'. (*Hint:* To ensure numerical stability you may have to add a small positive value to the diagonal of each covariance matrix. For this assignment you can add $0.01\mathbf{I}$ to each matrix.)

(a) **[19pt]** Implement the conditional Gaussian classifier as specified above. Plot an 8 by 8 image of the log of the diagonal elements of each covariance matrix Σ_k. Plot all ten classes side by side using the same grayscale.

(b) **[3pt]** Using the parameters you fit on the training set and the Bayes' rule, compute the average conditional log-likelihood, i.e. $\frac{1}{N}\sum_{i=1}^{N}\log p(y^{(i)}|\mathbf{x}^{(i)},\theta)$ on both the train and test set and report it. (Here θ denotes all the estimated parameters.)

(c) **[3pt]** Select the most likely posterior class for each training and test data point as your prediction, and report your accuracy on the train and test set.

4.3. *Naive Bayes Classifier Training – 30 pts.*

(a) **[1pt]** Convert the real-valued features $\mathbf{x}$ into binary features $\mathbf{b}$ using 0.5 as a threshold: $b_j = 1$ if $x_j > 0.5$ otherwise $b_j = 0$.

(b) **[15pt]** Using these new binary features $\mathbf{b}$ and the class labels, train a Bernoulli Naïve Bayes classifier using MAP estimation with prior $\mathsf{Beta}(\alpha,\beta)$ with $\alpha=\beta=2$. In particular, fit the model below on the training set.

$$p(y = k) = \frac{1}{10} \tag{4.2}$$

$$p(b_j = 1|y = k) = \eta_{kj} \tag{4.3}$$

$$p(\mathbf{b}|y = k, \eta) = \Pi_{j=1}^{d}(\eta_{kj})^{b_j}(1 - \eta_{kj})^{(1-b_j)} \tag{4.4}$$

$$p(\eta_{kj}) = \mathsf{Beta}(2, 2) \tag{4.5}$$

You should compute parameters η_{kj} for $k \in \{0, \ldots, 9\}$ and $j \in \{1, \ldots, 64\}$.

Prior as Pseudo-Counts:. Instead of explicitly considering the Beta distribution prior in the Bernoulli likelihood model, you can add two training cases to your data set for each class, one of which has every pixels OFF and the other has every pixels ON. Make sure you understand why this is equivalent to using a prior. You may use either schemes in your own code.

(c) **[2pt]** Plot each of your $\boldsymbol{\eta}_k$ vectors as an 8 by 8 grayscale image. These should be presented side by side and with the same scale.

(d) **[4pt]** Given your parameters, sample one new data point using your generative model for each of the 10 digit classes. Plot these new data points as 8 by 8 grayscale images side by side.

(e) **[4pt]** Using the parameters you fit on the training set and Bayes' rule, compute the average conditional log-likelihood, i.e. $\frac{1}{N}\sum_{i=1}^{N} \log p(y^{(i)}|\mathbf{x}^{(i)}, \theta)$ on both the train and test set and report it.

(f) **[4pt]** Select the most likely posterior class for each training and test data point, and report your accuracy on the train and test set.