Statistical Deep Learning (MT7042) - Project 1

Instructions: This project is divided into two main problems worth a total of 100 points. Problem 1 covers theory, while Problem 2 involves a practical application.

- **Problem 1:** Submit a written report in .pdf format to the course webpage. Handwritten solutions are accepted, but we strongly advise using LaTeX. Illegible answers will not be graded.
- **Problem 2:** Submit your completed solution as a Jupyter Notebook (`.ipynb`) if you are using Python, or an R Markdown file (`.Rmd`) if you are using R.

Additional Guidelines:

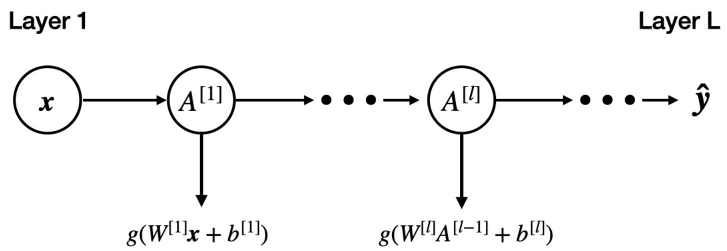
- You are encouraged to discuss the problems. However, all reports are to be written individually.
- Ensure that all code is runnable.

Problem 1

Consider a fully connected feedforward neural network, as illustrated in Figure 1, consisting of L layers, where layer 1 is the input layer and layer L is the output layer. The width of layer l is denoted by n_l. For $l = 2, \dots, L$, let

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

represent the linear output at layer l, where $W^{[l]}$ is the $n_l \times n_{l-1}$ matrix of weights, $b^{[l]}$ is the row vector of biases, and $A^{[l]} = g^{[l]}(Z^{[l]})$ is the row vector containing the nonlinear activations at layer l. Note that $A^{[1]} = Z^{[1]} = x$, representing the input layer.



Figur 1: High-level feedforward neural network diagram.

Define $\delta^{[l]} = \frac{\partial J}{\partial Z^{[l]}}$ for $l = 2, \dots, L$, where J denotes the cost function.

- **Task 1 (25 p):** Show that

$$\delta^{[l]} = g^{[l]'}(Z^{[l]}) \odot \left(W^{[l+1]^T} * \delta^{[l+1]}\right), \quad l = 2, \dots, L-1$$

and

$$\delta^{[L]} = J'(A^{[L]}) \odot g^{[L]'}(Z^{[L]})$$

where $\odot$ denotes element-wise multiplication and $*$ denotes ordinary matrix multiplication. Here $g^{[L]'}(Z^{[L]})$ and $J'(A^{[L]})$ denote the derivatives of the univariate functions applied element-wise to the vectors. If any operations with tensors are used, these need to be clearly defined.

- **Task 2 (10 p):** Derive the expressions for the gradients of the cost function J with respect to the weights and biases, i.e., $\frac{\partial J}{\partial W^{[l]}}$ and $\frac{\partial J}{\partial b^{[l]}}$ for $l = 2, 3, \dots, L$, under the following setting:
  - Cost function: Mean Squared Error (MSE)
  - Activation function for hidden layers: Rectified Linear Unit (ReLU)
  - Activation function for the output layer: Identity function

- **Task 3 (5 p):** How would you avoid an exponential blowup of computation when computing the gradients?

Problem 2

- **Task 1 (2 p)**: Load The Forest Covertype Dataset from the CSV file `covertype.csv` found in the projects directory. Make sure to read and understand the description before proceeding.

- **Task 2 (3 p)**: Check the class distribution of the data. Is class imbalance an issue? If so, explain why it could negatively affect the network performance.

- **Task 3 (5 p)**: Standardize the dataset by writing your own function without using any external libraries other than, e.g., NumPy if you are using Python or equivalents if you are using R.

- **Task 4 (5 p)**: Explain why standardization is important in the context of training a neural network. Should this step be performed before or after splitting the data? Justify your answer.

- **Task 5 (3 p)**: Split the dataset into training (80%), validation (10%), and test (10%) sets. Ensure you set a *seed* for reproducibility so that your splits yield consistent results each time the code is run.

- **Task 6 (30 p)**: Implement the training of a neural network using the package of your choice. Motivate your choice of depth and width, activation function, cost function, output function, parameter initialization, and training algorithm. A sentence or two is sufficient for each justification. During training, monitor the training error as well as the validation error.

- **Task 7 (10 p)**: Plot the validation error and training error curve, where the x-axis indicates the training epoch and the y-axis indicates the error.

  - Judging from the training error, does it appear that the training algorithm has converged?
  - Does the validation curve look as you would expect? If not, can you explain why that is?
  - Does the validation curve indicate overfitting? In the case of overfitting, how would you proceed to prevent it?

- **Task 8 (2 p)**: Use the neural network that you trained to make predictions on the test dataset. What is the test accuracy? Discuss the performance briefly.