Question 1 (60%)

In this exercise, you will train many multilayer perceptrons (MLPs) to approximate class label posteriors. The MLPs will be trained using maximum likelihood parameter estimation (or equivalently, minimum average cross-entropy loss). You will then use the trained models to approximate a maximum a posteriori (MAP) classification rule in an attempt to achieve minimum probability of error (i.e. minimize empirical risk with 0-1 loss assignments for correct-incorrect decisions).

Data Distribution: For $C = 4$ classes with uniform priors, specify Gaussian class-conditional pdfs for a 3-dimensional real-valued random vector $\mathbf{x}$. Pick your own mean vectors and covariance matrices for each class. Adjust these parameters of the multivariate Gaussian distribution so that the MAP classifier achieves between $10\%-20\%$ probability of error when using the true data pdf.

MLP Structure: Use a 2-layer MLP (one hidden layer of perceptrons) that has P perceptrons in the first (hidden) layer with smooth-ramp style activation functions (e.g., Smooth-ReLU, ELU, etc)[1]. At the second/output layer use a softmax function to ensure all outputs are positive and sum up to 1 (representing C probabilities). The best number of perceptrons, P^*, for your custom problem will be selected using K-fold cross-validation.

Generate Data: Using your specified data distribution, generate multiple datasets: training sets with 100, 200, 500, 1000, 2000, 5000 samples and a test set with 100000 samples. You will use the **test** set **only for performance evaluation**.

Theoretically Optimal Classifier: Using your knowledge of the true data pdf, construct a minimum-probability-of-error classifier. Then apply this optimal classifier to the test set and empirically estimate the probability of error for this theoretically optimal classifier. This provides the aspirational performance level for the MLP classifiers you will next develop.

Model Order Selection: For each of the training sets with a different number of samples, perform 10-fold cross-validation using minimum probability of error as the objective function to select the best number of perceptrons P (as justified by available training data). This will involve you training multiple MLPs on each of the six training sets: one per k-fold, per chosen P.

Model Training: After identifying the optimal number of perceptrons P^* per training set using cross-validation, now train an MLP with P^* perceptrons via maximum likelihood parameter estimation (minimum cross-entropy loss) on each training set. Note that this will utilize all of the data in each training set, rather than $K-1$ folds of it as in cross-validation for hyperparameter selection of P. Hence, these are your final trained MLP models for the class posteriors, each with possibly a different number of perceptrons and different weights.

Performance Assessment: Using each trained MLP as a model for the class posteriors, classify the samples in the test set using a MAP decision rule (aiming to minimize the probability of error). Empirically estimate the probability of error for each of the trained MLPs.

[1]Feel free to use ReLU as well, Smooth-ReLU or "softplus" is less efficient and effective in practice. See https://deeplearninguniversity.com/pytorch/pytorch-activation-functions/ for a list of activation functions implemented in PyTorch.

Report Process and Results: Describe your process of developing the solution; numerically and visually report the test set empirical probability of error estimates for the theoretically optimal and multiple MLP classifiers. For instance, show a plot of the empirically estimated test Pr(error) for each trained MLP versus number of training samples used in optimizing it (with a semi-log plot, logarithmic scale on the x-axis), as well as a horizontal line that runs across the plot indicating the empirically estimated test Pr(error) for the theoretically optimal classifier.

Note: You may use open-source software libraries for all aspects of your implementation. I especially recommend PyTorch for its ease of use, including things like handling weight initialization on your behalf. Recall that PyTorch offers a dedicated API for you to construct and optimize neural networks: the torch.nn package. With this package you can perform backpropagation to derive the parameters of an MLP according to maximum-likelihood (minimum cross-entropy) loss:

- CrossEntropyLoss function, where your MLPs could be optimized w.r.t to this loss criterion;
- Backprop for Gradients of Loss to compute the weights, which for cross-entropy loss is equivalent to maximum likelihood estimation of the parameters.

Question 2 (40%)

In this question, you will derive the solution for **ridge regression** in the case of a linear model with additive white Gaussian noise corrupting both the input and output data. Then you will implement it and select the hyperparameter (prior's regularization parameter) via cross-validation. Use $n = 10$, $N_{train} = 50$, and $N_{test} = 1000$ for this question.

Generate Data: Select an arbitrary non-zero n-dimensional vector $\mathbf{a}$. Pick an arbitrary Gaussian with nonzero-mean $\boldsymbol{\mu}$ and non-diagonal covariance matrix $\boldsymbol{\Sigma}$ for an n-dimensional random vector $\mathbf{x}$. Draw N_{train} iid samples of n-dimensional samples of $\mathbf{x}$ from this Gaussian pdf. Draw N_{train} iid samples of an n-dimensional random variable $\mathbf{z}$ from a zero-mean, $\varepsilon\mathbf{I}$-covariance-matrix Gaussian pdf (noise corrupting the input). Draw N_{train} iid samples of a scalar random variable v from a zero-mean, unit-variance Gaussian pdf. Calculate N_{train} scalar values of a new random variable as follows $y = \mathbf{a}^\intercal(\mathbf{x}+\mathbf{z}) + v$ using the samples of $\mathbf{x}$ and v. This is your training dataset that consists of $(\mathbf{x}, y)$ pairs. Similarly, generate a separate test dataset that consists of N_{test} $(\mathbf{x}, y)$ pairs.

Model Parameter Estimation: We think that the relationship between the $(\mathbf{x}, y)$ pairs is linear $y = \mathbf{w}^\intercal\mathbf{x} + w_0 + v$, and that v is an additive white Gaussian noise term (with zero-mean and unit-variance). We are unaware of the presence of the additional noise term $\mathbf{z}$ in the true generative process. We also believe that this process has linear model parameters close to zero, so we use a zero-mean, $\beta\mathbf{I}$-covariance-matrix Gaussian pdf as a prior $p(\boldsymbol{\theta})$ for the model parameters $\boldsymbol{\theta} = \mathbf{w}$ (which contain w_0). We will use MAP parameter estimation to determine the optimal weights for this model using the training data.

Hyperparameter Optimization: The prior $p(\boldsymbol{\theta})$ for the $n+1$-dimensional weight vector (+ bias w_0) in the model has a scalar parameter β that needs to be selected. Use 5-fold cross-validation on the training set to select this parameter. As your cross-validation objective function, use max-log-likelihood of the validation data, averaged over the 5 folds of data. Utilize the MAP parameter estimation solution in the process with candidate hyperparameter values.

Model Optimization & Evaluation: Once the best hyperparameter value is identified, use the entire training dataset to optimize the model parameters with MAP parameter estimation. Specifically use the best hyperparameter found from your model selection (cross-validation) procedure. Evaluate the "-2 $\times$ log-likelihood" of the trained model when applied to the test data.

While keeping $\mathbf{a}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ constant, vary the noise parameter ε gradually from very small (e.g., $10^{-3} \times \text{trace}(\boldsymbol{\Sigma})/n$) to very large (e.g., $10^{3} \times \text{trace}(\boldsymbol{\Sigma})/n$), introducing increasing levels of noise to the input variable. Generate new training and test sets with each ε, repeat the process, and analyze the impact of this noise parameter on the solution, selection of the hyperparameter, and performance on the test set.

Show your math for the derivation of the MAP parameter estimate, explain how your code implements the solution, and present numerical and visual results, along with their discussions.