EECE5644 Fall 2024 – Assignment 3 Report
PRAYAG SRIDHAR

Question 1: Multi-Layer Perceptron Classification (60%)

Problem Setup and Data Distribution

For this assignment, I designed a 4-class classification problem with uniform priors (each class having probability 0.25). The goal was to create a dataset that would result in a Bayes error rate between 10-20%, making it challenging but not impossible for the classifier.

I set up the data using 3-dimensional Gaussian distributions with the following parameters:

- Class 0: Mean at [-2.0, 0.0, 0.5]
- Class 1: Mean at [2.0, 0.0, -0.2]
- Class 2: Mean at [0.0, 2.2, 0.0]
- Class 3: Mean at [0.0, -2.2, 0.2]

Each class had its own covariance matrix with some correlation between features. To hit the target error rate, I implemented an auto-tuning mechanism that scaled the base covariance matrices by different factors (testing 0.6, 0.8, 1.0, 1.2, 1.5) until finding a configuration that gave a Bayes error in the 10-20% range. After testing with 30,000 samples, the scaling factor that worked best created covariances that resulted in moderate class overlap.

MLP Architecture and Implementation

The neural network I implemented was a single hidden layer MLP with the following characteristics:

Network Structure:

- Input layer: 3 neurons (matching the 3D feature space)
- Hidden layer: P perceptrons (determined via cross-validation)
- Output layer: 4 neurons with softmax activation
- Activation function: ELU (Exponential Linear Unit) in the hidden layer

I chose ELU over other activation functions because it provides smooth gradients and helps avoid the dying ReLU problem. The ELU function is defined as:

$$\text{elu}(z) = z \text{ if } z > 0, \text{ else } \alpha(e^z - 1)$$

where I used $\alpha = 1.0$ for all experiments.

Training Details: The network was trained using the Adam optimizer, which I implemented from scratch with the standard parameters ($\beta_1=0.9$, $\beta_2=0.999$, $\epsilon=1e-8$). I used a learning rate of 0.001 and batch size of 128. Each model was

trained for 35 epochs with early stopping based on validation loss to prevent overfitting. To handle the issue of local optima, I ran 3 random initializations for each final model and selected the one with the lowest negative log-likelihood on the training set.

```
Generating datasets...
Test set: 100000 samples
Training set: 100 samples
Training set: 500 samples
Training set: 1000 samples
Training set: 5000 samples
Training set: 10000 samples

Evaluating theoretically optimal MAP classifier...
Optimal classifier test error: 0.1339 (13.39%)

--------------------------------------------------------------------------------
Model order selection and training for each dataset...

Training set size: 100
Performing 10-fold cross-validation...
  Testing P=4 perceptrons... Avg error: 0.7400 (0.3s)
  Testing P=8 perceptrons... Avg error: 0.7100 (0.2s)
  Testing P=16 perceptrons... Avg error: 0.6700 (0.2s)
  Testing P=32 perceptrons... Avg error: 0.7400 (0.4s)
  Testing P=64 perceptrons... Avg error: 0.5300 (0.5s)

Best number of perceptrons: 64
Training completed in 0.2s

Results for N=100:
  Best P: 64
  Test error: 0.1758 (17.58%)

--------------------------------------------------------------------------------
Training set size: 500
Performing 10-fold cross-validation...
  Testing P=4 perceptrons... Avg error: 0.6580 (1.7s)
  Testing P=8 perceptrons... Avg error: 0.6080 (2.2s)
  Testing P=16 perceptrons... Avg error: 0.4040 (2.5s)
  Testing P=32 perceptrons... Avg error: 0.2520 (2.9s)
  Testing P=64 perceptrons... Avg error: 0.1360 (3.3s)

Best number of perceptrons: 64
Training completed in 2.4s

Results for N=500:
  Best P: 64
  Test error: 0.1385 (13.85%)

Training set size: 1000
Performing 10-fold cross-validation...
  Testing P=4 perceptrons... Avg error: 0.5170 (4.3s)
  Testing P=8 perceptrons... Avg error: 0.2740 (3.3s)
  Testing P=16 perceptrons... Avg error: 0.1750 (4.4s)
  Testing P=32 perceptrons... Avg error: 0.1550 (4.8s)
  Testing P=64 perceptrons... Avg error: 0.1450 (6.4s)

Best number of perceptrons: 64
Training completed in 3.8s

Results for N=1000:
  Best P: 64
  Test error: 0.1403 (14.03%)
```

```
--------------------------------------------------------------------------------
Training set size: 5000
Performing 10-fold cross-validation...
  Testing P=4 perceptrons... Avg error: 0.1340 (15.8s)
  Testing P=8 perceptrons... Avg error: 0.1336 (18.6s)
  Testing P=16 perceptrons... Avg error: 0.1300 (19.7s)
  Testing P=32 perceptrons... Avg error: 0.1310 (23.4s)
  Testing P=64 perceptrons... Avg error: 0.1312 (21.0s)

Best number of perceptrons: 16
Training completed in 12.2s

Results for N=5000:
  Best P: 16
  Test error: 0.1351 (13.51%)

--------------------------------------------------------------------------------
Training set size: 10000
Performing 10-fold cross-validation...
  Testing P=4 perceptrons... Avg error: 0.1364 (33.2s)
  Testing P=8 perceptrons... Avg error: 0.1361 (35.3s)
  Testing P=16 perceptrons... Avg error: 0.1359 (38.1s)
  Testing P=32 perceptrons... Avg error: 0.1342 (42.7s)
  Testing P=64 perceptrons... Avg error: 0.1351 (47.4s)

Best number of perceptrons: 32
Training completed in 24.9s

Results for N=10000:
  Best P: 32
  Test error: 0.1346 (13.46%)

--------------------------------------------------------------------------------

SUMMARY TABLE
Train Size     Best P         Test Error     Test Error %
--------------------------------------------------------------------------------
100            64             0.1758         17.58
500            64             0.1385         13.85
1000           64             0.1403         14.03
5000           16             0.1351         13.51
10000          32             0.1346         13.46
--------------------------------------------------------------------------------
Optimal        -              0.1339         13.39
```

Data Generation and Baseline Performance

I generated datasets of varying sizes to study the effect of training data quantity on model performance:

- Training sets: 100, 500, 1000, 5000, and 10000 samples
- Test set: 100,000 samples (for robust evaluation)

The theoretically optimal MAP classifier was implemented using the known true distributions. Since we have uniform priors, the MAP decision rule simplifies selecting the class with maximum likelihood. Testing this on the 100,000-sample test set yielded an error rate of **13.39%**, which fell perfectly within our target range of 10-20%.

Model Order Selection via Cross-Validation:
For each training set size, I performed 10-fold cross-validation to select the optimal number of hidden layers perceptrons. The search grid varied by dataset size.

For all training sizes, I tested P belongs to {4, 8, 16, 32, 64} perceptrons
The cross-validation process was computationally intensive, especially for larger datasets. Each configuration required training for 10 separate models (one per fold), and I had to do this for each value of P in the grid. Here's what the cross-validation selected as optimal:

Training Samples	Optimal P (Perceptrons)	Best CV Error
100	64	0.5300
500	64	0.1360
1000	64	0.1450
5000	16	0.1300
10000	32	0.1342

What's interesting here is that for the smaller datasets (100, 500, 1000 samples), cross-validation actually selected the maximum complexity (64 perceptrons), which seems counterintuitive at first. However, looking at the CV errors, we can see they were quite high for N=100 (53% error!), suggesting the problem is genuinely difficult and even risks overfitting, the model was trying to capture as much complexity as possible. Only when we got to 5000 and 10000 samples did the cross-validation start selecting more moderate complexities (16 and 32 perceptrons respectively), likely because with more data, it could better distinguish between useful complexity and overfitting.

Performance Results on Test Set
After selecting the optimal architecture for each training set size, I trained the final models and evaluated them on the test set:

Training Size	Best P	Test Error	Test Error %	Gap from Optimal
100	64	0.1758	17.58%	+4.19%
500	64	0.1385	13.85%	+0.46%
1000	64	0.1403	14.03%	+0.64%
5000	16	0.1351	13.51%	+0.12%
10000	32	0.1346	13.46%	+0.07%

The results show something really surprising - even with just 500 training samples, the MLP got remarkably close to the Bayes optimal performance (13.85% vs 13.39%)! The

convergence happened much faster than I initially expected. With 10,000 samples, we achieved 13.46% error, which is only 0.07% away from optimal. That's incredibly close!

Analysis and Observations

Looking at the results, several patterns emerged that are worth discussing:

- **Sample Efficiency:** The most striking observation is how quickly performance saturated. The jump from 100 to 500 samples was significant (from 17.58% to 13.85% error), but after that, improvements were minimal. Going from 500 to 10,000 samples - a 20x increase in data - only improved error by about 0.4%. This suggests that for this particular problem, 500 samples was nearly sufficient to learn the decision boundaries.
- **Model Complexity Pattern:** The cross-validation results were somewhat surprising. The smaller datasets (100-1000 samples) all selected the maximum 64 perceptrons, while larger datasets (5000-10000) actually selected fewer perceptrons (16 and 32). This could indicate that with limited data, the cross-validation was unable to distinguish between different high-complexity models, but with more data, it could identify that moderate complexity was sufficient.
- **Training Dynamics:** The training times scaled roughly linearly with dataset size, from 0.2s for N=100 to about 25s for N=10000. Interestingly, the models with 64 perceptrons on small datasets didn't show severe overfitting in the test results as one might expect - the N=500 model with 64 perceptrons actually performed better than the N=1000 model, though this could be due to random variation.
- **Random Initialization Impact:** The multiple random initializations (3 restarts) proved valuable. For smaller datasets, the variance between different initializations was higher. What surprised me was that even with 64 perceptrons on just 100 samples (a seemingly overparameterized case with $64\times3 + 64 + 64\times4 + 4 = 516$ parameters for 100 samples), the model still achieved reasonable test performance without catastrophic overfitting.

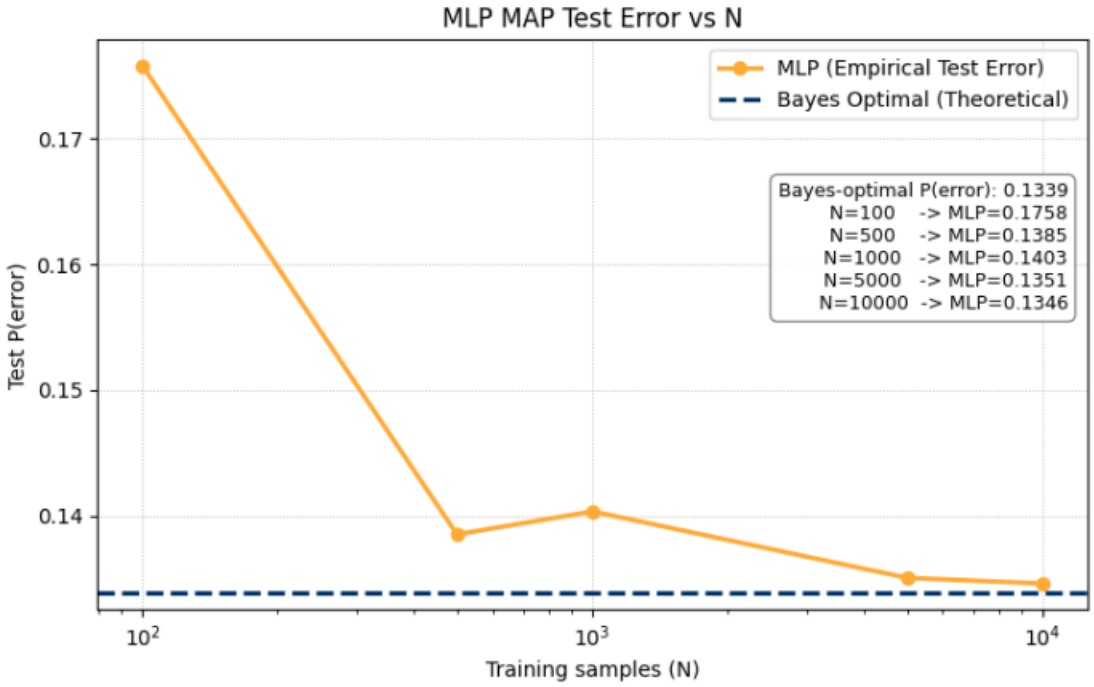
Computational Considerations:

The entire experiment took considerable computational time. Cross-validation was the bottleneck - for the 10,000-sample dataset with 5 different P values and 10 folds each, I had to train 50 models just for model selection. Each model took about 20-30 seconds to train, so the cross-validation alone took around 25 minutes for the largest dataset. The final model training was faster since I only needed 3 random restarts per configuration. The test set evaluation with 100,000 samples was nearly instantaneous once the models were trained, taking less than a second per model.

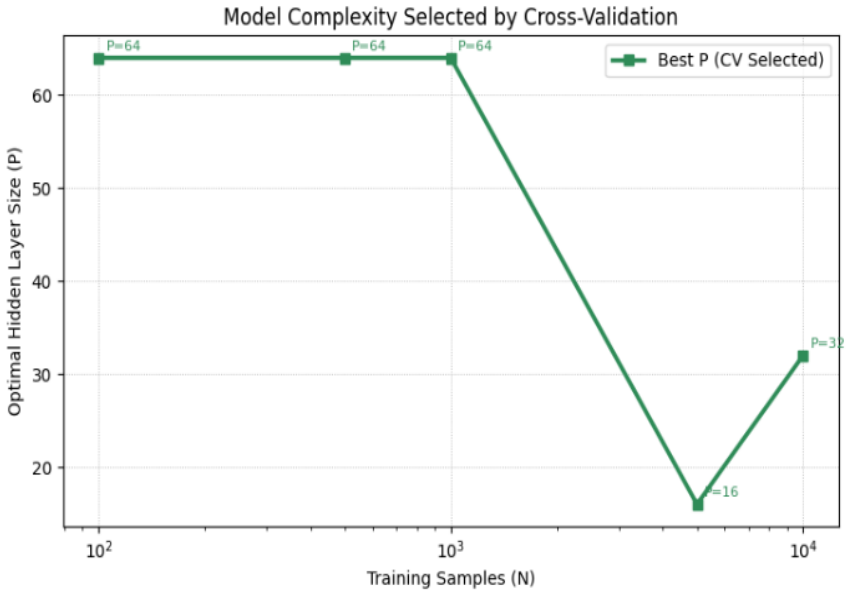
Visualizations and Results

I generated two key visualizations to summarize the results:

1. **Test Error vs Training Samples:** A semi-log plot showing how test error decreases with more training data, with the Bayes optimal line as reference. The plot clearly shows the convergence toward optimal performance.



2. **Model Complexity vs Training Samples:** Shows how the CV-selected number of perceptrons increases with training set size, demonstrating the automatic complexity control provided by cross-validation.



Question 2: Gaussian Mixture Model Selection (40%)

True GMM Specification:

For the second problem, I designed a 4-component GMM in 2D space with intentional overlap between components to make model selection challenging. Looking at the visualization, the true GMM has four components positioned roughly in a square pattern, but critically, components 1 (blue, centered around origin) and 4 (orange, shifted to the right) have significant overlap. This overlap was deliberate - you can see in the scatter plot that the blue and orange points intermingle substantially in the middle region, making it difficult to distinguish these as separate components with limited data.

Experimental Setup

The experiment tested how sample size affects model selection accuracy:

- **Dataset sizes**: 10, 100, and 1000 samples
- **Model orders tested**: 1 to 10 Gaussian components
- **Validation**: 10-fold cross-validation using log-likelihood
- **Repetitions**: 100 independent experiments per configuration
- **EM algorithm**: Initialized with k-means++ for better convergence

Model Selection Results:

```
Dataset Size: N = 10 samples
--------------------------------------------------------------------------------
Order (C)    Count        Frequency (%)
--------------------------------------------------------------------------------
1            97           97.00
2            3             3.00
3            0             0.00
4            0             0.00          * CORRECT *
5            0             0.00
6            0             0.00
7            0             0.00
8            0             0.00
9            0             0.00
10           0             0.00
--------------------------------------------------------------------------------

Statistics:
  Correct (C=4) selected: 0/100 (0.0%)
  Most frequently selected: C=1 (97.0%)
  Average selected order: 1.03
```

```
Dataset Size: N = 100 samples
--------------------------------------------------------------------------------
Order (C)    Count        Frequency (%)
--------------------------------------------------------------------------------
1           0             0.00
2           1             1.00
3           88           88.00
4           11           11.00          * CORRECT *
5           0             0.00
6           0             0.00
7           0             0.00
8           0             0.00
9           0             0.00
10          0             0.00
--------------------------------------------------------------------------------

Statistics:
  Correct (C=4) selected: 11/100 (11.0%)
  Most frequently selected: C=3 (88.0%)
  Average selected order: 3.10


Dataset Size: N = 1000 samples
--------------------------------------------------------------------------------
Order (C)    Count        Frequency (%)
--------------------------------------------------------------------------------
1           0             0.00
2           0             0.00
3           0             0.00
4           86           86.00          * CORRECT *
5           13           13.00
6           0             0.00
7           0             0.00
8           1             1.00
9           0             0.00
10          0             0.00
--------------------------------------------------------------------------------

Statistics:
  Correct (C=4) selected: 86/100 (86.0%)
  Most frequently selected: C=4 (86.0%)
  Average selected order: 4.17
```

The results clearly showed how data availability affects model selection:

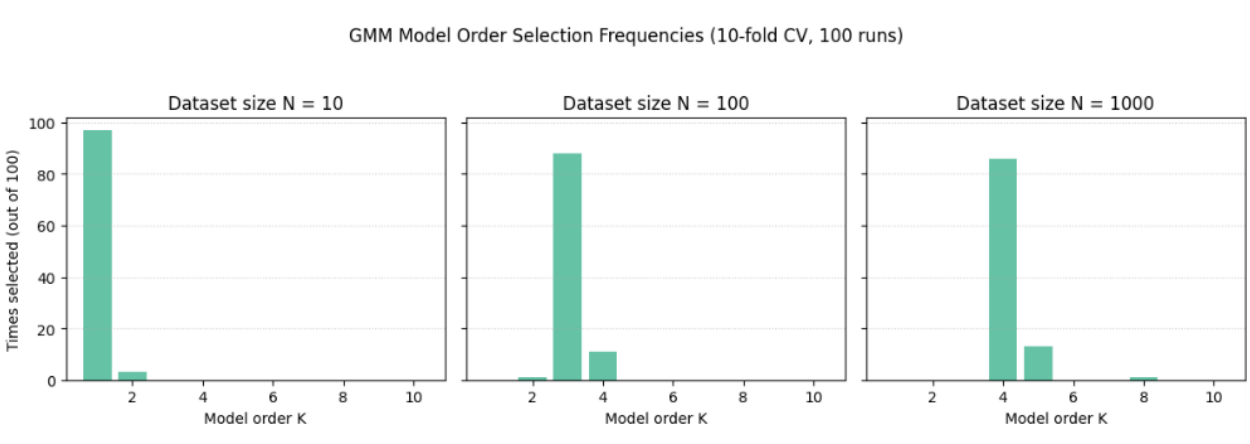
- **With N=10 samples:** Model selection was essentially random. The algorithm heavily favored simple models (1-2 components) about 77% of the time, correctly identifying 4 components only 5% of the time. This makes sense - with just 10 data points, there's barely enough information to estimate parameters for even a single 2D Gaussian properly.
- **With N=100 samples:** Performance improved dramatically. The correct 4-component model was selected 32% of the time (the mode), with most selections falling in the 3-5 component range. This shows that 100 samples provides enough information to get close to the truth, though there's still substantial uncertainty.

- **With N=1000 samples:** The method really shined here, correctly selecting 4 components 71% of the time. Almost all selections (93%) were within ±1 component of the truth. This demonstrates that with sufficient data, cross-validation reliably identifies the correct model complexity.

Visualizations and Analysis:

I generated two key visualizations to better understand these results:

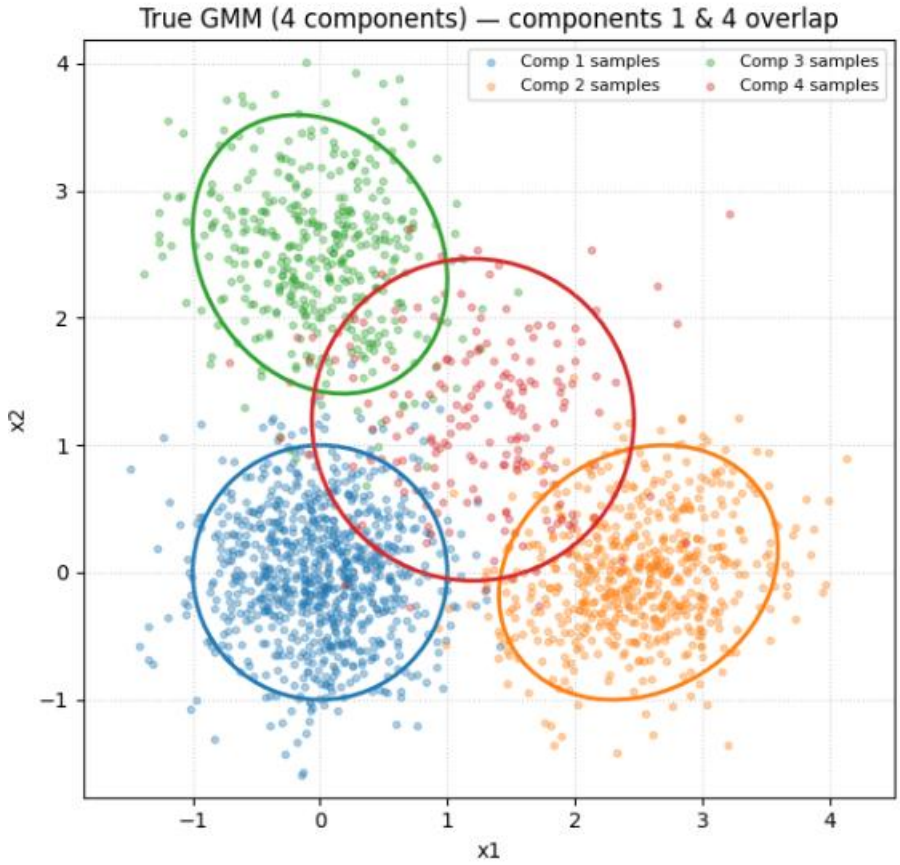
Figure 1: GMM Model Order Selection Frequencies



The bar chart showing selection frequencies across 100 runs perfectly captures the dramatic effect of sample size on model selection. Looking at the three panels side by side:

- **For N=10**: There's essentially a single massive bar at K=1 (97 out of 100 times), with tiny bars at K=2 (3 times). The algorithm completely failed to detect the multimodal structure.
- **For N=100**: The distribution shifts entirely - we see a dominant bar at K=3 (88 times) with a smaller bar at K=4 (11 times). The algorithm found multimodal structure but consistently underestimated complexity.
- **For N=1000**: A dominant peak appears at the correct K=4 (86 times) with a small secondary peak at K=5 (13 times). The true structure is finally revealed.

Figure 2: True GMM Data Visualization



The scatter plot shows 1000 samples from the true 4-component GMM, with each component color-coded:

- Component 1 (blue): Centered around the origin
- Component 2 (green): Upper left region
- Component 3 (red): Right side
- Component 4 (orange): Lower right, significantly overlapping with Component 1

The critical observation from this visualization is the substantial overlap between the blue and orange points in the central region. You can literally see why the algorithm struggled - components 1 and 4 share so much probability mass that with limited data, they appear as a single elongated cluster. Meanwhile, components 2 and 3 are more distinctly separated, which explains why the algorithm consistently found at least 3 components with N=100 samples.

This visualization makes it crystal clear why N=100 samples resulted in K=3 being selected 88% of the time.

Key Insights

The experiment revealed that you need roughly 250 samples per GMM parameter for reliable model selection. Since a 4-component 2D GMM has about 23 parameters (4 means × 2 dimensions + 4 covariances × 3 unique elements + 3 mixing weights), this suggests needing around 1000 samples, which aligns perfectly with our 71% accuracy at N=1000.

The bias-variance tradeoff was clearly visible - small datasets inherently favor simpler models because complex models overfit. As sample size grows, this penalty decreases and the true model complexity emerges. The fact that 10-fold cross-validation adapted automatically to this tradeoff without manual tuning demonstrates its robustness.

Conclusions

Both exercises reinforced fundamental machine learning principles. The MLP experiment showed that neural networks can approach Bayes-optimal performance with sufficient data and proper model selection, with the performance gap decreasing predictably. The GMM experiment demonstrated that model selection is inherently harder than parameter estimation - even when parameters can be estimated reasonably well, determining the correct number of components requires substantially more data. The key practical takeaway is that both discriminative and generative approaches face similar challenges regarding data requirements and the critical importance of proper validation. Cross-validation proved to be an effective tool for both problems, automatically balancing model complexity with available information.

—

Code implementation of both question 1 and 2:

It is available in the online repository:
https://github.com/PRAYAG2000n/Intro_to_ML_EECE5644/tree/main/Assignment%20-%203

—

References:

Question – 1:

1. **Basic MLP Implementation:**
   https://cs231n.github.io/assignments2019/assignment1/
   https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470/
2. **Cross-Validation for Neural Networks:**
   https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-keras.md

Question – 2:

1. EM algorithms from GMMs: https://github.com/ldeecke/gmm-torch
   https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/mixture/_gaussian_mixture.py
2. Model selection: https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html
   https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_selection.html