AMATH 445/645 – Practice Problems for Lectures 9–12

1. What is clustering? How is it different from classification?

   Clustering is an unsupervised learning method that groups similar data points together based on their features, without using any labels. The algorithm discovers natural groupings in the data.
   Classification is supervised learning that uses labeled data to learn a decision boundary that separates predefined classes.

2. What is the difference between hard clustering and soft clustering? Give an example of each.

   - **Hard clustering**: Each data point belongs to exactly one cluster. Example: K-means clustering assigns each point to a single cluster.
   - **Soft clustering**: Each data point can belong to multiple clusters with varying degrees of membership (probabilities). Example: Gaussian Mixture Models (GMM) assign probabilities of belonging to each cluster.

3. Write the objective function of K-means clustering. What does it minimize?

$$\min_{\{\mu_j\}_{j=1}^K} \sum_{i=1}^{N} \min_{1\le j\le K} \|x_i - \mu_j\|^2$$

   This minimizes the **within-cluster sum of squares (WCSS)**, the sum of squared distances between each point and its assigned cluster centroid. It encourages tight, compact clusters.

4. Describe the K-means algorithm step by step.

   (a) **Initialization**: Randomly select K data points as initial centroids $\mu_1, \mu_2, \ldots, \mu_K$.
   (b) Repeat until convergence:
      - **Assignment step**: For each data point x_i, assign it to the nearest centroid.
      - **Update step**: Recompute each centroid as the mean of points assigned to it.
   (c) **Convergence**: Stop when centroids change very little between iterations or when assignments stop changing.

5. Given 1D data points: $[1, 2, 3, 10, 11, 12]$ and $K = 2$ with initial centroids $\mu_1 = 2$, $\mu_2 = 11$:

   (a) Perform the assignment step.
   (b) Update the centroids.
   (c) Has the algorithm converged? Why or why not?

(a) **Assignment**:

- Distances to $\mu_1 = 2$: $|1-2| = 1$, $|2-2| = 0$, $|3-2| = 1$, $|10-2| = 8$, $|11-2| = 9$, $|12-2| = 10$
- Distances to $\mu_2 = 11$: $|1-11| = 10$, $|2-11| = 9$, $|3-11| = 8$, $|10-11| = 1$, $|11-11| = 0$, $|12-11| = 1$
- Assignments: Cluster 1: $[1, 2, 3]$, Cluster 2: $[10, 11, 12]$

(b) **Update**:

- $\mu_1 = (1+2+3)/3 = 6/3 = 2$
- $\mu_2 = (10+11+12)/3 = 33/3 = 11$

(c) **Convergence check**: The new centroids are exactly the same as the initial centroids (2 and 11). Therefore, the algorithm has converged after just one iteration.

6. Give three limitations of K-means clustering.

(a) **Assumes spherical clusters**: K-means works best for clusters that are isotropic (spherical) and roughly equal in size. It fails for elongated or oddly shaped clusters.

(b) **Sensitive to initialization**: Different random initializations can lead to different final clusterings (local minima problem).

(c) **Requires specifying K in advance**: In real problems, the true number of clusters is often unknown.

(d) **Sensitive to outliers**: Outliers can dramatically shift centroids and distort clusters.

(e) **Assumes clusters are convex**: Cannot handle non-convex shapes.

7. What is K-means++?

K-means++ is an improved initialization method for K-means that spreads out initial centroids:

(a) Randomly select the first centroid from the data points

(b) For each remaining point, compute its distance to the nearest already-chosen centroid

(c) Select the next centroid with probability proportional to the square of this distance (points farther from existing centroids are more likely to be chosen)

(d) Repeat until K centroids are selected

8. Write the perceptron prediction rule for binary classification with labels $y \in \{-1, +1\}$.

$$\hat{y} = \text{sign}(w^T x + b) = \begin{cases} +1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{if } w^T x + b < 0 \end{cases}$$

where w is the weight vector and b is the bias term.

9. A point $x = (2, 1)$ with true label $y = -1$ is misclassified by the current perceptron with $w = (1, 1)$ and $b = 0$, learning rate $\eta = 1$.

(a) Compute the current prediction.

(b) Perform one perceptron update.

(c) Verify that the updated weights correctly classify this point.

(a) Current prediction: $w^T x + b = (1,1) \cdot (2,1) + 0 = 2 + 1 = 3 > 0$, so predict $+1$. True label is -1, so misclassified.

(b) Perceptron update rule for misclassified point: $w \leftarrow w + \eta y x$, $b \leftarrow b + \eta y$
$w_{\text{new}} = (1,1) + (1)(-1)(2,1) = (1,1) + (-2,-1) = (-1,0)$
$b_{\text{new}} = 0 + (1)(-1) = -1$

(c) New prediction: $w_{\text{new}}^T x + b_{\text{new}} = (-1,0) \cdot (2,1) - 1 = -2 + 0 - 1 = -3 < 0$, so predict -1; correct!

10. Why are activation functions necessary in neural networks? What happens if you use only linear activations?

Activation functions introduce **nonlinearity** into the network. Without nonlinearity:

- Each layer performs a linear transformation: $z^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)}$
- Composition of linear functions is still linear: $W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)} = (W^{(2)}W^{(1)})x + (W^{(2)}b^{(1)} + b^{(2)})$
- A multi-layer network collapses to a single linear transformation
- Cannot learn complex patterns like image recognition, etc.

Nonlinear activations allow the network to learn hierarchical, complex representations.

11. Name four common activation functions and give one advantage of each.

(a) **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Advantage: Smooth, outputs between 0 and 1 (good for probabilities)
- Disadvantage: Vanishing gradient for large $|z|$

(b) **Tanh**: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Advantage: Zero-centered (outputs between -1 and 1)
- Disadvantage: Still suffers from vanishing gradients

(c) **ReLU**: $f(z) = \max(0, z)$
- Advantage: Computationally efficient, no vanishing gradient for positive inputs
- Disadvantage: "Dying ReLU" problem (neurons can become inactive)

(d) **Softmax**: $f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
- Advantage: Outputs a probability distribution over classes (used in output layer for multi-class)
- Disadvantage: Can saturate with extreme values

12. What does the Universal Approximation Theorem state?

A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$, provided the activation function is non-constant, bounded, and continuous (e.g., sigmoid, tanh, ReLU variants).

13. For a binary classification problem, what activation function would you use in the output layer? What loss function would you pair with it?

Output activation: Sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$
Why? Sigmoid outputs a value between 0 and 1, which can be interpreted as the probability of belonging to the positive class.
Loss function: Binary cross-entropy loss

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

This combination works well because:

- Cross-entropy penalizes confident wrong predictions more heavily than squared error
- Gradient doesn't vanish when predictions are far from targets (unlike MSE with sigmoid)

14. For a multi-class classification problem with C classes, what activation function would you use in the output layer? What loss function?

    Output activation: Softmax function

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}}$$

    Softmax ensures outputs are positive and sum to 1, forming a valid probability distribution over C classes.
    Loss function: Categorical cross-entropy loss

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{C} y_{ik} \log(\hat{y}_{ik})$$

    where y_{ik} is 1 if sample i belongs to class k, else 0 (one-hot encoding).

15. Name three loss functions used in neural networks and when you would use each.

    (a) **Mean Squared Error (MSE)**: $L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
        - Used for regression tasks (predicting continuous values)
        - Sensitive to outliers (squares the error)

    (b) **Mean Absolute Error (MAE)**: $L = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$
        - Used for regression tasks
        - More robust to outliers than MSE
        - Gradient is constant, making optimization different from MSE

    (c) **Cross-Entropy Loss**:
        - Binary cross-entropy: Used for binary classification
        - Categorical cross-entropy: Used for multi-class classification
        - Encourages calibrated probability estimates

16. What is the vanishing gradient problem?

    During backpropagation, gradients are multiplied by the derivative of the activation function at each layer. For activation functions like sigmoid and tanh, derivatives are ≤ 0.25, so gradients shrink exponentially as they propagate backward through many layers.

17. Consider a simple 2-layer network with:

- Input x, hidden layer with sigmoid activation $g(\Sigma) = \frac{1}{1+e^{-\Sigma}}$, output $\hat{y} = w_2 z + w_{20}$ (linear output)
- Loss: $L = \frac{1}{2}(y - \hat{y})^2$

Derive $\frac{\partial L}{\partial w_1}$ and $\frac{\partial L}{\partial w_2}$ using the chain rule.

Forward pass definitions:

$$\Sigma_1 = w_1 x + w_{10} \qquad \text{(weighted sum at hidden layer)}$$
$$z = g(\Sigma_1) = \frac{1}{1 + e^{-\Sigma_1}} \qquad \text{(hidden activation)}$$
$$\hat{y} = w_2 z + w_{20} \qquad \text{(output)}$$
$$L = \frac{1}{2}(y - \hat{y})^2 \qquad \text{(loss)}$$

For $\frac{\partial L}{\partial w_2}$:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = (y - \hat{y}) \cdot (-1) \cdot z = -(y - \hat{y})z$$

For $\frac{\partial L}{\partial w_1}$: (chain rule through multiple layers)

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial \Sigma_1} \cdot \frac{\partial \Sigma_1}{\partial w_1}$$

Compute each term:

$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$$
$$\frac{\partial \hat{y}}{\partial z} = w_2$$
$$\frac{\partial z}{\partial \Sigma_1} = g(\Sigma_1)(1 - g(\Sigma_1)) = z(1 - z) \qquad \text{(derivative of sigmoid)}$$
$$\frac{\partial \Sigma_1}{\partial w_1} = x$$

Multiply together:

$$\frac{\partial L}{\partial w_1} = -(y - \hat{y}) \cdot w_2 \cdot z(1 - z) \cdot x$$

The negative sign is typically absorbed into the update rule.

18. What are the update rules for batch gradient descent, stochastic gradient descent (SGD), and mini-batch gradient descent.

Update rules:

- Batch GD: $w^{(t+1)} = w^{(t)} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla L_i(w)$
- SGD: $w^{(t+1)} = w^{(t)} - \alpha \nabla L_i(w)$ (single random sample)
- Mini-batch: $w^{(t+1)} = w^{(t)} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla L_i(w)$

19. What is mini-batch gradient descent and why is it the most common choice in practice?

Mini-batch gradient descent computes gradients using a small random subset of the training data (typically 32-512 samples) and updates parameters after each batch.
Why it's preferred:

(a) **Computational efficiency**: Can be vectorized and run on GPUs (unlike pure SGD)

(b) **Memory efficiency**: Only need to store one batch in memory at a time

(c) **Stable convergence**: Less noisy than SGD but faster than batch GD

(d) **Regularization effect**: Noise from random batches helps generalization

(e) **Flexibility**: Batch size is a tunable hyperparameter

Most modern deep learning frameworks are optimized for mini-batch training.

20. Explain the intuition behind momentum in optimization. Write the update equations.

Intuition: Momentum accumulates a moving average of past gradients to accelerate convergence and smooth out oscillations. Like a ball rolling down a hill; it gains speed in consistent directions and resists changes in direction.

Update equations:

$$v^{(t+1)} = \beta v^{(t)} + (1 - \beta)\nabla L(w^{(t)})$$
$$w^{(t+1)} = w^{(t)} - \alpha v^{(t+1)}$$

where:

- v is the velocity (momentum term)
- β is the momentum coefficient (typically 0.9)
- α is the learning rate

Benefits:

- Accelerates convergence in consistent gradient directions
- Dampens oscillations in narrow valleys
- Can help escape flat regions and shallow minima

21. What is dropout? Explain how it works during training vs. testing.

Dropout is a regularization technique that randomly deactivates (drops) a fraction of neurons during training.

During training:

(a) For each training sample (or mini-batch), randomly select neurons to drop with probability p (dropout rate)

(b) Dropped neurons contribute 0 to forward and backward passes

(c) Remaining activations are scaled by $1/(1-p)$ (inverted dropout) to maintain expected magnitude

(d) Different random subset dropped for each sample/batch

During testing/inference:

- All neurons are active (no dropout)
- Weights are used as-is (no scaling needed if inverted dropout was used during training)

Why it helps:

- Prevents co-adaptation of neurons (can't rely on specific other neurons)
- Acts like training an ensemble of many subnetworks
- Reduces overfitting

22. What is data augmentation? Provide three examples for image data and three for text data.

Data augmentation artificially expands the training dataset by creating modified versions of existing data while preserving labels. It's a form of regularization that improves generalization.
Image data examples:

(a) Rotation: Rotate images by small angles
(b) Flipping: Horizontal/vertical flips
(c) Cropping: Randomly crop parts of images
(d) Color jitter: Adjust brightness, contrast, saturation
(e) Noise injection: Add Gaussian noise

Text data examples:

(a) Synonym replacement: Replace words with synonyms
(b) Back translation: Translate to another language and back
(c) Random insertion/deletion: Add or remove words
(d) Word dropout: Randomly mask words

23. What is an epoch? How does it relate to iterations and batch size?

Epoch: One complete pass through the entire training dataset.
Relationship:

$$\text{Iterations per epoch} = \frac{\text{Total training samples}}{\text{Batch size}}$$

Example: With 50,000 training samples and batch size 100:

- 500 iterations (batches) per epoch
- After 500 iterations, model has seen every sample once = 1 epoch

Typical training: 10-100+ epochs depending on dataset size and model complexity.

24. Name three ways physics can be incorporated into machine learning models. Briefly explain each.

(a) **Physics-informed loss functions (soft constraints)**: Add terms to the loss function that penalize violations of physical laws (e.g., PDE residuals). The model learns to satisfy physics approximately.
(b) **Physics-informed architectures (hard constraints)**: Design network architectures that inherently respect physical principles (e.g., conservation laws, symmetries, boundary conditions). Physics is exactly enforced by construction.
(c) **Physics-based features or representations**: Transform input data using physics-based features (e.g., invariant quantities, dimensionless numbers) or embed physical knowledge in data preprocessing.

Soft constraints are more flexible but may not be exactly satisfied; hard constraints guarantee physics but are harder to design.

25. Consider the diffusion equation:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}, \quad x \in [0,1], t \in [0,T]$$

A neural network $u_\theta(x,t)$ approximates the solution. Define the PDE residual $r_\theta(x,t)$.

The PDE residual measures how much the network output violates the governing equation:

$$r_\theta(x,t) = \frac{\partial u_\theta}{\partial t} - D\frac{\partial^2 u_\theta}{\partial x^2}$$

If u_θ exactly satisfies the PDE, then $r_\theta(x,t) = 0$ everywhere. In practice, we minimize the squared residual over a set of collocation points.

26. How are the derivatives $\frac{\partial u_\theta}{\partial t}$ and $\frac{\partial^2 u_\theta}{\partial x^2}$ computed in a PINN?

Derivatives are computed using **automatic differentiation** (autodiff), not numerical approximations like finite differences.

(a) The network $u_\theta(x,t)$ is a differentiable function of its inputs and parameters

(b) Automatic differentiation applies the chain rule to compute exact derivatives (up to machine precision)

(c) Frameworks like TensorFlow/PyTorch compute these derivatives efficiently via back-propagation

(d) This allows PINNs to handle high-order derivatives without numerical errors or mesh generation

This is a key advantage of PINNs over traditional numerical methods — they are mesh-free and derivative-free.

27. Write the physics-informed loss term L_{physics} for the diffusion equation using N_c collocation points (x_i, t_i).

$$L_{\text{physics}}(\theta) = \frac{1}{N_c}\sum_{i=1}^{N_c} |r_\theta(x_i,t_i)|^2 = \frac{1}{N_c}\sum_{i=1}^{N_c} \left|\frac{\partial u_\theta}{\partial t}(x_i,t_i) - D\frac{\partial^2 u_\theta}{\partial x^2}(x_i,t_i)\right|^2$$

The collocation points are typically randomly sampled from the domain during training.

28. Write the full PINN loss function including data, initial condition, boundary condition, and physics terms. Explain what each term enforces.

$$L(\theta) = \lambda_D L_{\text{data}} + \lambda_I L_{\text{initial}} + \lambda_B L_{\text{boundary}} + \lambda_P L_{\text{physics}}$$

where:

- $L_{\text{data}} = \frac{1}{N_d}\sum_{i=1}^{N_d} |u_\theta(x_i,t_i) - u_i^{\text{obs}}|^2$: Enforces fit to observed data
- $L_{\text{initial}} = \frac{1}{N_i}\sum_{i=1}^{N_i} |u_\theta(x_i,0) - u_0(x_i)|^2$: Enforces initial condition
- $L_{\text{boundary}} = \frac{1}{N_b}\sum_{i=1}^{N_b} |u_\theta(x_i,t_i) - g(x_i,t_i)|^2$: Enforces boundary conditions
- $L_{\text{physics}} = \frac{1}{N_c}\sum_{i=1}^{N_c} |r_\theta(x_i,t_i)|^2$: Enforces PDE satisfaction

The λ coefficients balance the relative importance of each term.

29. Distinguish between forward and inverse problems in the context of PINNs.

- **Forward problem**: Given known PDE parameters (e.g., D), initial conditions, and boundary conditions, find the solution $u(x,t)$. The network $u_\theta(x,t)$ is trained with physics loss + IC/BC loss (data may be optional).
- **Inverse problem**: Given some observations of $u(x,t)$, find unknown PDE parameters (e.g., D) or even the full governing equation. Both $u_\theta(x,t)$ and the unknown parameters are learned simultaneously from data.

In inverse problems, the physics loss depends on both θ and the unknown parameters, which are treated as trainable variables.

30. In a PINN inverse problem for the diffusion equation, what additional parameter is learned? Write the modified physics loss and the gradient update rules.

The unknown diffusion coefficient D becomes a trainable parameter alongside the network weights θ.
The physics loss becomes:

$$L_{\text{physics}}(\theta, D) = \frac{1}{N_c}\sum_{i=1}^{N_c}\left|\frac{\partial u_\theta}{\partial t}(x_i,t_i) - D\frac{\partial^2 u_\theta}{\partial x^2}(x_i,t_i)\right|^2$$

The optimization problem is:

$$\min_{\theta,D}\left[\lambda_D L_{\text{data}}(\theta) + \lambda_P L_{\text{physics}}(\theta, D) + \text{IC/BC terms}\right]$$

Both θ and D are updated via gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} - \eta_\theta \frac{\partial L}{\partial \theta}$$

$$D^{(k+1)} = D^{(k)} - \eta_D \frac{\partial L}{\partial D}$$

31. Why can PINN training become ill-conditioned? Explain the problem with loss term magnitudes.

Ill-conditioning in PINNs arises from:

(a) **Scale mismatch**: Different loss terms (L_{data}, L_{physics}, $L_{\text{IC/BC}}$) can have very different magnitudes

(b) **Stiff PDEs**: Some PDEs have solutions with rapidly changing features, making residuals large in some regions

(c) **Gradient imbalance**: Gradients from different loss terms may have different scales, causing optimization to focus on the largest terms

(d) **Competing objectives**: Physics loss and data loss may conflict, especially with noisy data

Consequences:

- Optimization may get stuck in poor local minima
- Some loss terms may be ignored if their weights are too small
- Training can become unstable or converge very slowly

32. Consider the 1D heat equation:

$$\frac{\partial u}{\partial t} = 0.1\frac{\partial^2 u}{\partial x^2}, \quad x \in [0,1], t \in [0,1]$$

with initial condition $u(x,0) = \sin(\pi x)$ and boundary conditions $u(0,t) = u(1,t) = 0$. You have sparse observations at 5 random points. Design a PINN training setup including:

(a) Network architecture

(b) Loss function terms

(c) Sampling strategy for collocation points

(d) Potential challenges

(a) **Network architecture**:

- Input layer: 2 neurons (x, t)
- 3-5 hidden layers with 20-50 neurons each
- Activation: tanh (smooth, good for derivatives)
- Output: 1 neuron ($u_\theta(x,t)$)
- Xavier initialization

(b) **Loss function terms**:

$$L_{\text{data}} = \frac{1}{5}\sum_{i=1}^{5} |u_\theta(x_i,t_i) - u_i^{\text{obs}}|^2$$

$$L_{\text{initial}} = \frac{1}{N_i}\sum_{j=1}^{N_i} |u_\theta(x_j,0) - \sin(\pi x_j)|^2$$

$$L_{\text{boundary}} = \frac{1}{N_b}\sum_{k=1}^{N_b} [u_\theta(0,t_k)^2 + u_\theta(1,t_k)^2]$$

$$L_{\text{physics}} = \frac{1}{N_c}\sum_{l=1}^{N_c} \left|\frac{\partial u_\theta}{\partial t} - 0.1\frac{\partial^2 u_\theta}{\partial x^2}\right|^2$$

$$L_{\text{total}} = \lambda_D L_{\text{data}} + \lambda_I L_{\text{initial}} + \lambda_B L_{\text{boundary}} + \lambda_P L_{\text{physics}}$$

(c) **Sampling strategy**:

- Initial condition: Uniform x samples at $t = 0$ (e.g., 100 points)
- Boundary: Uniform t samples at $x = 0$ and $x = 1$ (e.g., 50 each)
- Collocation: Random (x,t) pairs in $[0,1]^2$ (e.g., 1000 points per iteration)
- Resample collocation points each epoch for better coverage

(d) **Potential challenges**:

- Balancing loss weights (data vs physics)
- Gradient vanishing/exploding for second derivatives
- With only 5 data points, may need strong physics regularization
- Possible non-uniqueness without sufficient boundary enforcement