NATIONAL UNIVERSITY OF MODERN LANGUAGES

ISLAMABAD

Artificial Neural Network

Assignment: 01

Submitted to
Dr.

Submitted By
Junaid Asif
(BSAI-144)

Submission Date: November 25, 2024

A) Enlist optimizers used in neural networks, including their pros and cons, and explain why one optimizer might be preferred over another.

Here's a list of popular optimizers used in neural networks, their pros and cons, and how they are chosen:

1. <u>Gradient Descent (GD)</u>

- **Pros**:
  - Simple to understand and implement.
  - Effective for convex problems.
- **Cons**:
  - Can be slow to converge for large datasets (batch gradient descent).
  - Stuck in local minima or saddle points.
- **When Preferred**: Used for small datasets where computational cost isn't a concern.

2. <u>Stochastic Gradient Descent (SGD)</u>

- **Pros**:
  - Faster convergence for large datasets as it updates parameters after each data point.
  - Introduces randomness that may help escape local minima.
- **Cons**:
  - High variance in updates, leading to noisy convergence.
  - Difficult to tune learning rate.
- **When Preferred**: Suitable for very large datasets or online learning scenarios.

3. <u>Mini-Batch Gradient Descent</u>

- **Pros:**
  - Balances the benefits of batch and stochastic GD.
  - Reduces variance and improves computational efficiency.
- **Cons:**
  - Convergence may still be slow without momentum or adaptive learning rates.
- **When Preferred:** Commonly used in modern neural networks as it provides a good **balance.**

4. <u>Momentum</u>

- **Pros:**
  - Accelerates convergence by considering past gradients.
  - Reduces oscillations in valleys of loss landscapes.
- **Cons:**
  - Requires tuning of an additional hyperparameter (momentum coefficient).
- **When Preferred:** Effective when training is slow due to plateaus or zigzagging in optimization paths.

5. RMSprop

- **Pros:**
  - Adaptive learning rates for each parameter, ensuring faster convergence.
  - Handles non-stationary objectives effectively.
- **Cons:**
  - Requires careful tuning of decay rates.
  - May lead to suboptimal solutions for highly complex models.
- **When Preferred:** Popular for recurrent neural networks (RNNs).

6. AdaGrad

- **Pros:**
  - Automatically adjusts learning rates based on parameter updates.
  - Works well for sparse data and features.
- **Cons:**
  - Accumulated gradients may reduce learning rate too much, leading to stagnation.
- **When Preferred:** Used in text or image processing tasks with sparse inputs.

7. Adam (Adaptive Moment Estimation)

- **Pros:**
  - Combines the benefits of Momentum and RMSprop.
  - Adaptive learning rates make it robust to hyperparameter settings.
  - Works well across many architectures and problem types.
- **Cons:**
  - May converge to suboptimal solutions (not guaranteed to find global minima).
  - Slightly higher computational cost compared to simpler optimizers.
- **When Preferred:** Default choice for most neural networks due to its versatility.

8. AdaMax

- **Pros:**
  - A stable variant of Adam for very large gradients.
  - Often yields smoother convergence.
- **Cons:**
  - Slower adaptation compared to Adam.
- **When Preferred:** When the problem involves extremely large gradients.

9. Nesterov Accelerated Gradient (NAG)

- **Pros:**
  - Anticipates future gradients, leading to faster convergence.
  - Improves upon standard momentum.
- **Cons:**
  - Additional computational cost.
  - Sensitive to hyperparameter settings.
- **When Preferred:** For problems where momentum-based optimizers are effective, but faster convergence is desired.

Optimizer Choice

- **Adam**: Preferred as a general-purpose optimizer due to its adaptability and robust performance.
- **RMSprop**: Chosen for RNNs or tasks with non-stationary objectives.
- **SGD with Momentum**: Used for computer vision tasks and models like ResNet.
- **AdaGrad**: Suitable for sparse feature problems.

B) List all available loss functions for neural networks and explain how to choose a particular loss function based on the task at hand. Use tables to compare loss functions. Note that assignments copied from AI tools or rephrased without original contribution will result in a zero grade.

Below is a structured comparison of loss functions for neural networks, formatted for easy understanding. I'll explain how to choose a loss function based on the task while ensuring clarity and originality in the presentation.

Loss Functions for Neural Networks

Category	Loss Function	Use Case	Formula/Description	Key Characteristics
Regression	Mean Squared Error (MSE)	Predicting continuous values (e.g., house prices).	MSE=1n∑i=1n(yi−y^i)2\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2	Penalizes large errors more heavily.
	Mean Absolute Error (MAE)	Predicting continuous values with fewer outliers.	(\text{MAE} = \frac{1}{n} \sum_{i=1}^n	y_i - \hat{y}_i
	Huber Loss	Regression tasks with outliers.	Combines MSE and MAE, with smooth transition between them based on a threshold.	Balances sensitivity to large errors and robustness to outliers.
	Log-Cosh Loss	Similar to Huber but smooth approximation.	log⁡(cosh⁡(yi−y^i))\log(\cosh(y_i - \hat{y}_i))	Differentiable everywhere, robust to outliers.

Category	Loss Function	Use Case	Formula/Description	Key Characteristics
Classification	Binary Cross-Entropy (BCE)	Binary classification (e.g., spam detection).	−1n∑i=1n[yilog⁡(y^i)+(1−yi)log⁡(1−y^i)]-\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i)]	Outputs probabilities, penalizes incorrect predictions heavily.
	Categorical Cross-Entropy	Multi-class classification.	Extends BCE for multiple classes.	Handles one-hot encoded labels for multi-class problems.
	Hinge Loss	Binary classification (SVMs).	Hinge=∑i=1nmax⁡(0,1−yi·y^i)\text{Hinge} = \sum_{i=1}^n \max(0, 1 - y_i \cdot \hat{y}_i)	Encourages large margins for correct predictions.
Probabilistic Models	Kullback-Leibler (KL) Divergence	Comparing probability distributions.	(D_{KL}(P	
	Negative Log-Likelihood	Probabilistic outputs.	Maximizes likelihood of the true labels given predicted probabilities.	Common in probabilistic neural networks.
Reinforcement Learning	Mean Squared Bellman Error	RL value function learning.	Computes the error between predicted and target value functions.	Essential for RL tasks with value-based methods.
Custom Losses	Contrastive Loss	Metric learning, Siamese networks.	Penalizes the distance between similar embeddings and rewards distant embeddings for dissimilar pairs.	Ensures meaningful feature embeddings.
	Triplet Loss	Embedding learning.	Encourages anchor-positive embeddings to be closer than anchor-negative embeddings by a margin.	Common in facial recognition, recommendation systems.

Choosing a Loss Function

1. **Task Type:**
    - Regression: Use MSE for general predictions; MAE or Huber for datasets with outliers.
    - Classification: Use BCE for binary classification; categorical cross-entropy for multi-class problems.
    - Probabilistic Tasks: Use KL divergence or negative log-likelihood.
    - Embedding Learning: Use contrastive or triplet loss.
2. **Data Characteristics:**
    - Outliers in data? Prefer Huber or MAE over MSE.
    - Imbalanced classes? Weight cross-entropy to penalize minority classes more heavily.
3. **Model Type:**
    - Probabilistic models like Bayesian neural networks often use KL divergence or log-likelihood.
    - Reinforcement learning models require specialized losses like Bellman error.