Problem Set: Python OOP in Machine Learning

Say OL

October 06, 2024

Problem Set

1. **Creating a Neuron Class**
   Description: Implement a class `Neuron` that simulates a single neuron with methods to set weights, perform a forward pass, and apply the sigmoid activation function.
   Sample Data: Create an instance with weights `[0.5, -0.6]` and inputs `[1.0, 2.0]`.
   Output: The output of the neuron after applying the sigmoid function.

2. **Building a Layer Class**
   Description: Implement a class `Layer` that represents a layer of neurons. It should contain methods to initialize the layer, perform forward propagation, and compute outputs for all neurons in that layer.
   Sample Data: Create a layer with 3 neurons and weights `[[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]]` and inputs `[1.0, 2.0]`.
   Output: The outputs of the layer.

3. **Creating a Neural Network Class**
   Description: Create a class `NeuralNetwork` that contains multiple layers. Implement methods to add layers and perform a forward pass through the entire network.
   Sample Data: Add two layers with sample weights and perform a forward pass with inputs `[1.0, 2.0]`.
   Output: The final output of the network.

4. **Loss Function Class**
   Description: Implement a class `MeanSquaredError` with methods to compute the loss and its gradient.
   Sample Data: Use true values `[1.0, 0.0]` and predicted values `[0.9, 0.1]`.
   Output: The computed loss and gradient.

5. **Creating a Training Loop**
   Description: Extend the `NeuralNetwork` class to include a method `train` that takes training data, true labels, and epochs. Implement a simple training loop to adjust weights using gradient descent.
   Sample Data: Train with inputs `[[1.0, 0.0], [0.0, 1.0]]` and labels `[1.0, 0.0]` for 10 epochs.
   Output: The loss after training.

6. **Model Evaluation Class**
   Description: Create a class `ModelEvaluator` that takes a neural network and test data, and computes accuracy.
   Sample Data: Use a trained network and test data `[[1.0, 0.0], [0.0, 1.0]]` with labels `[1.0, 0.0]`.
   Output: The accuracy of the model.

7. **Saving and Loading Models**
   Description: Implement methods in the `NeuralNetwork` class to save the model to a file and load it from a file.
   Sample Data: Save a trained model and load it back.
   Output: Confirm that the loaded model's weights match the saved model.

8. **Regularization Class**
   Description: Create a class `Regularizer` that implements L1 and L2 regularization methods. Integrate this class into the training loop of the neural network.

Sample Data: Use L2 regularization with a lambda value of 0.01 during training.
Output: The adjusted weights after regularization.

9. **Hyperparameter Tuning**
   Description: Create a class `HyperparameterTuner` that performs grid search for different learning rates and layer sizes.
   Sample Data: Test with learning rates `[0.01, 0.1]` and layer sizes `[1, 2]`.
   Output: The best combination of hyperparameters based on validation loss.

10. **Visualization Class**
    Description: Implement a class `Plotter` to visualize loss over epochs and the decision boundary of the neural network.
    Sample Data: Use the training history of losses over 10 epochs: : `loss_history = [0.9, 0.8, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, 0.4, 0.35]`.
    Output: A plot of loss vs. epochs and a plot showing the decision boundary.

Solutions