

Deep-ML problems

1.

The screenshot shows the Deep-ML interface. On the left, the 'Problem Description' tab is active, displaying an example input and a code snippet for a gradient descent implementation. The code includes sample data, parameters (learning_rate, n_iterations, batch_size), and initialization of weights. On the right, the 'Notebook Mode' tab is active, showing a code editor with a Python function for gradient descent. The function takes parameters X, y, weights, learning_rate, and n_iterations, and returns the optimized weights. The interface also includes a 'Run Code' button, a 'Reset' button, a 'Save Code' button, and a 'Submissions' button.

```
def gradient_descent(X, y, weights, learning_rate, n_iterations):
    for epoch in range(n_iterations):
        if method == 'stochastic':
            for i in range(n_samples):
                pred = xi @ w
                err = pred - yi
                grad = 2.0 * xi * err
                w -= learning_rate * grad
        elif method == 'mini_batch':
            for start in range(0, n_samples, batch_size):
                end = min(start + batch_size, n_samples)
                Xb = X[start:end]
                yb = y[start:end]
                preds = Xb @ w
                error = preds - yb
                batch_len = end - start
                grad = (2.0 / batch_len) * (Xb.T @ error)
                w -= learning_rate * grad
    return w
```

2.

The screenshot shows the Deep-ML interface. On the left, the 'Problem Description' tab is active, displaying the 'Implement Adam Optimization Algorithm' problem. The problem description includes the Adam algorithm's purpose and the parameters it takes. On the right, the 'Notebook Mode' tab is active, showing a code editor with a Python function for the Adam optimizer. The function takes parameters f, grad, x0, learning_rate, beta1, beta2, epsilon, and num_iterations, and returns the optimized parameters. The interface also includes a 'Run Code' button, a 'Reset' button, a 'Save Code' button, and a 'Submissions' button.

```
def adam_optimizer(f, grad, x0, learning_rate=0.001, beta1=0.9, beta2=0.999,
                  epsilon=1e-8, num_iterations=10):
    # Your code here
    x = x0.astype(float)
    m = np.zeros_like(x)
    v = np.zeros_like(x)
    for t in range(1, num_iterations+1):
        g = grad(x)
        m = beta1 * m + (1 - beta1) * g
        v = beta2 * v + (1 - beta2) * (g ** 2)
        m_hat = m / (1 - beta1 ** t)
        v_hat = v / (1 - beta2 ** t)
        x = x - learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)
    return x
```

3.

Problem Description

Solution

Video

Comments 0

Implement Batch Normalization for BCHW Input

Easy Deep Learning

Implement a function that performs Batch Normalization on a 4D NumPy array representing a batch of feature maps in the BCHW format (batch, channels, height, width). The function should normalize the input across the batch and spatial dimensions for each channel, then apply scale (gamma) and shift (beta) parameters. Use the provided epsilon value to ensure numerical stability.

Example:

Input:

```
B, C, H, W = 2, 2, 2, 2; np.random.seed(42); X = np.random.randn(B, C, H, W); gamma = np.ones(C).reshape(1, C, 1, 1); beta = np.zeros(C).reshape(1, C, 1, 1)
```

Output:

```
[[[[[ 0.42859934, -0.51776438], [ 0.65360963, 1.95820707]], [[ 0.02353721, 0.02355215], [ 1.67355207, 0.93490043]]], [[[-1.01139563, 0.49692747], [ 0.40566697, -0.00464601]], [[-0.46262449, -0.04330601], [ 0.33036647, 0.33036647]]]]]
```

Notebook Mode

1 import numpy as np

2

3 def batch_normalization(X: np.ndarray, gamma: np.ndarray, beta: np.ndarray, epsilon: float = 1e-5) -> np.ndarray:

4 # Your code here

5 mean = X.mean(axis=(0, 2, 3), keepdims=True)

6

7 var = X.var(axis=(0, 2, 3), keepdims=True)

8 X_hat = (X - mean) / np.sqrt(var + epsilon)

9 Y = gamma * X_hat + beta

10

11 return Y

Run Code

Reset

Save Code

Submissions 2

Test Results 3/3

Test 1 Test 2 Test 3

Test Case

Ask Tutor

4.

Problem Description

Solution

Video

Comments 0

Implement Layer Normalization for Sequence Data

Medium Machine Learning

Implement a function to perform Layer Normalization on an input tensor. Given a 3D array representing batch_size, sequence length, and feature dimensions, normalize the data across the feature dimension for each sequence, then apply scaling and shifting parameters.

Example:

Input:

```
np.random.seed(42); X = np.random.randn(2, 2, 3); gamma = np.ones(3).reshape(1, 1, -1); beta = np.zeros(3).reshape(1, 1, -1); layer_normalization(X, gamma, beta)
```

Output:

```
[[[ 0.47373971 -1.39079736  0.91705765]
 [ 1.41420326 -0.70711154 -0.70709172]]
 [[ 1.13192477  0.16823009 -1.30015486]
 [ 1.4141794  -0.70465482 -0.70952458]]]
```

Notebook Mode

1 import numpy as np

2

3 def layer_normalization(X: np.ndarray, gamma: np.ndarray, beta: np.ndarray, epsilon: float = 1e-5) -> np.ndarray:

4 """

5 Perform Layer Normalization.

6 """

7 # Your code here

8 mean = X.mean(axis=2, keepdims=True)

9 var = X.var(axis=2, keepdims=True)

10 X_hat = (X - mean) / np.sqrt(var + epsilon)

11 Y = gamma * X_hat + beta

12 return Y

Run Code

Reset

Save Code

Submissions 1

Test Results 3/3

Test 1 Test 2 Test 3

Test Case

Ask Tutor

5.

Problem Description

Solution

Video

Comments 0

Dropout Layer

Medium Deep Learning

Implement a dropout layer that applies random neuron deactivation during training to prevent overfitting in neural networks. The layer should randomly zero out a proportion of input elements based on a dropout rate p , scale the remaining values by $1/(1-p)$ to maintain expected values, and pass inputs unchanged during inference. During backpropagation, gradients must be masked with the same dropout pattern and scaled by the same factor to ensure proper gradient flow.

Example:

Input:

```
x = np.array([1.0, 2.0, 3.0, 4.0]), grad = np.array([0.1, 0.2, 0.3, 0.4]), p = 0.5
```

Output:

```
output = array([[2., 0., 6., 0. ]]), grad = array([[0.2, 0., 0.6, 0. ]])
```

Baseninn

Notebook Mode

3

class DropoutLayer:

4

def __init__(self, p: float):

8

self.mask = None

9

self.scale = 1.0 / (1.0 - self.p)

10

11

12

def forward(self, x: np.ndarray, training: bool = True) -> np.ndarray:

13

"""Forward pass of the dropout layer."""

14

Your code here

15

if training:

16

self.mask = np.random.binomial(1, 1.0 - self.p, size=x.shape).

17

astype(x.dtype)

18

return x * self.mask * self.scale

19

else:

20

return x

21

22

def backward(self, grad: np.ndarray) -> np.ndarray:

23

"""Backward pass of the dropout layer."""

24

Your code here

25

return grad * self.mask * self.scale

Run Code

Reset

Save Code

Submissions 3

Test Results 3/3

Test 1 Test 2 Test 3

Test Case

Ask Tutor