

Deep-MLProblemsLabsNEWCollectionsLeaderboardDeep-0Jobs85Premium

Problem DescriptionSolutionVideoComments0Playground

## Implement Gradient Descent Variants with MSE Loss

MediumMachine Learning

In this problem, you need to implement a single function that can perform three variants of gradient descent Stochastic Gradient Descent (SGD), Batch Gradient Descent, and Mini Batch Gradient Descent using Mean Squared Error (MSE) as the loss function. The function will take an additional parameter to specify which variant to use. Note: Do not shuffle the data

**Example:**

**Input:**

```
import numpy as np

# Sample data
X = np.array([[1, 1], [2, 1], [3, 1], [4, 1]])
y = np.array([2, 3, 4, 5])

# Parameters
learning_rate = 0.01
n_iterations = 1000
batch_size = 2
```

3. Optimization Techniques (7/7)Test 1Test 2Test 32/4 CompletedTest Case

Ask Tutor

Notebook ModeNumPy

```
1 import numpy as np
2
3 def gradient_descent(X, y, weights, learning_rate, n_iterations,
4                     batch_size=None, method='batch'):
5     n = len(y)
6     for _ in range(n_iterations):
7         if method == 'batch':
8             y_ = X.dot(weights)
9             gradient = (2/n)*X.T.dot(y_ - y)
10            weights = weights - learning_rate * gradient
11        elif method == 'stochastic':
12            for i in range(n):
13                xi = X[i].reshape(1, -1)
14                yi = y[i]
15                y_ = xi.dot(weights)
16                gradient = 2 * xi.T.dot(y_ - yi)
17                weights = weights - learning_rate * gradient
18        elif method == 'mini_batch':
19            for i in range(0, n, batch_size):
20                X_batch = X[i:i+batch_size]
21                y_batch = y[i:i+batch_size]
```

Run CodeResetSave CodeSubmissions1

Test Results3/3

Deep-MLProblemsLabsNEWCollectionsLeaderboardDeep-0Jobs85Premium

Problem DescriptionSolutionVideoComments0Playground

## Implement Adam Optimization Algorithm

MediumDeep Learning

Implement the Adam (Adaptive Moment Estimation) optimization algorithm in Python. Adam is an optimization algorithm that adapts the learning rate for each parameter. Your task is to write a function `'adam_optimizer'` that updates the parameters of a given function using the Adam algorithm.

The function should take the following parameters:

- `'f'`: The objective function to be optimized
- `'grad'`: A function that computes the gradient of `'f'`
- `'x0'`: Initial parameter values
- `'learning_rate'`: The step size (default: 0.001)
- `'beta1'`: Exponential decay rate for the first moment estimates (default: 0.9)
- `'beta2'`: Exponential decay rate for the second moment estimates (default: 0.999)
- `'epsilon'`: A small constant for numerical stability (default: 1e-8)
- `'num_iterations'`: Number of iterations to run the optimizer (default: 1000)

The function should return the optimized parameters.

3. Optimization Techniques (7/7)Test 1Test 2Test 32/4 CompletedTest Case

Ask Tutor

Notebook Mode

```
1 import numpy as np
2
3 def adam_optimizer(f, grad, x0, learning_rate=0.001, beta1=0.9,
4                   beta2=0.999, epsilon=1e-8, num_iterations=10):
5     s = np.zeros_like(x0)
6     r = np.zeros_like(x0)
7     for i in range(num_iterations):
8         s = beta1*s + (1-beta1)*grad(x0)
9         r = beta2*r + (1-beta2)*(grad(x0)**2)
10        s_ = s/(1-beta1**(i+1))
11        r_ = r/(1-beta2**(i+1))
12        x0 = x0 - learning_rate*s_/(np.sqrt(r_)+epsilon)
13    return x0
```

Run CodeResetSave CodeSubmissions3

Test Results2/2

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], [-1.00236882, -1.00581468]], [[ 0.45676349, -1.50433085], [-1.33293647, -0.27503802]]]]
```

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

6

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

7

X\_ = (X - mean) / np.sqrt(var + epsilon)

8

Y = gamma \* X\_ + beta

9

return Y

Run Code

Reset

Save Code

Submissions 1

Test Results 3/3

Test 1

Test 2

Test 3

Ask Tutor

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

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

8

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

9

X\_ = (X - mean) / np.sqrt(var + epsilon)

10

Y = gamma \* X\_ + beta

11

# Your code here

12

return Y

Run Code

Reset

Save Code

Submissions 3

Test Results 3/3

Test 1

Test 2

Test 3

Ask Tutor

Problem Description

Solution

Video

Comments 0

Dropout Layer

MediumDeep 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. ]])
```

**Reasoning:**

Notebook Mode

```
1 import numpy as np
2
3 class DropoutLayer:
4     def __init__(self, p: float):
5         """Initialize the dropout layer."""
6         self.p = p
7         self.mask = None
8     def forward(self, x: np.ndarray, training: bool = True) -> np.ndarray:
9         """Forward pass of the dropout layer."""
10        if not training:
11            return x
12        self.mask = np.random.binomial(1, 1 - self.p, x.shape)
13        return x * self.mask / (1 - self.p)
14    def backward(self, grad: np.ndarray) -> np.ndarray:
15        """Backward pass of the dropout layer."""
16        return grad * self.mask / (1 - self.p)
```

Run Code

Reset

Save Code

Submissions 3

Test Results 3/3

Test 1Test 2Test 3

Test Case

Ask Tutor