

**Deep-ML** Problems Labs **NEW** Collections Leaderboard Deep-0 Jobs 🔥 85 Premium 🐕 🔔

## Implement Gradient Descent Variants with MSE Loss

Medium Machine 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
```

# Initialize weights

1 import numpy as np  
2  
3 def gradient\_descent(X, y, weights, learning\_rate, n\_iterations, batch\_size=None, method='batch'):  
4 n = len(y)  
5 for \_ in range(n\_iterations):  
6 if method == 'batch':  
7 y\_ = X.dot(weights)  
8 gradient = (2/n)\*X.T.dot(y\_- y)  
9 weights = weights - learning\_rate \* gradient  
10 elif method == 'stochastic':  
11 for i in range(n):  
12 xi = X[i].reshape(1, -1)  
13 yi = y[i]  
14 y\_ = xi.dot(weights)  
15 gradient = 2 \* xi.T.dot(y\_- yi)  
16 weights = weights - learning\_rate \* gradient  
17 elif method == 'mini\_batch':  
18 for i in range(0, n, batch\_size):  
19 X\_batch = X[i:i+batch\_size]  
20 y\_batch = y[i:i+batch\_size]

**Run Code** **Reset** **Save Code** **Submissions 1**

**Test Results** 3/3

3. Optimization Techniques (7) 2/4 Completed Test Case Ask Tutor

**Deep-ML** Problems Labs **NEW** Collections Leaderboard Deep-0 Jobs 🔥 85 Premium 🐕 🔔

## Implement Adam Optimization Algorithm

Medium Deep 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.

1 import numpy as np  
2  
3 def adam\_optimizer(f, grad, x0, learning\_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8, num\_iterations=10):  
4 s = np.zeros\_like(x0)  
5 r = np.zeros\_like(x0)  
6 for i in range(num\_iterations):  
7 s = beta1\*s + (1-beta1)\*grad(x0)  
8 r = beta2\*r + (1-beta2)\*(grad(x0)\*\*2)  
9 s\_ = s/(1-beta1\*\*(i+1))  
10 r\_ = r/(1-beta2\*\*(i+1))  
11 x0 = x0 - learning\_rate\*s\_/(np.sqrt(r\_)+epsilon)  
12 return x0

**Run Code** **Reset** **Save Code** **Submissions 3**

**Test Results** 2/2

Test 1 Test 2 Ask Tutor

**Problem Description**

### 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**

### 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**

Deep-MI Problems Tags Collections Leaderboard Deep-0 Subs

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.]])
```

**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
9     def forward(self, x: np.ndarray, training: bool = True) -> np.ndarray:
10        """Forward pass of the dropout layer."""
11        if not training:
12            return x
13        self.mask = np.random.binomial(1, 1 - self.p, x.shape)
14        return x * self.mask / (1 - self.p)
15
16    def backward(self, grad: np.ndarray) -> np.ndarray:
17        """Backward pass of the dropout layer."""
18        return grad * self.mask / (1 - self.p)
```

Run Code Reset Save Code Submissions 3

Test Results 3/3

Test 1 Test 2 Test 3

Ask Tutor