

Deep-ML problems

1.

The screenshot shows a Deep-ML interface with a code editor. The left panel displays a problem description for gradient descent variants, including sample data and parameters. The right panel shows a code editor with three variants: Batch, Stochastic, and Mini-Batch. The code uses NumPy operations like matrix multiplication (`@`) and dot products (`@`). The code editor includes standard controls like Run Code, Reset, Save Code, and Submissions.

```
3 def gradient_descent(x, y, weights, learning_rate, n_iterations,
4     for epoch in range(n_iterations):
5         if method == 'stochastic':
6             for i in range(n_samples):
7                 pred = x[i] @ w
8                 err = pred - y[i]
9                 grad = 2.0 * x[i] * err
10                w -= learning_rate * grad
11
12 elif method == 'mini_batch':
13     for start in range(0, n_samples, batch_size):
14         end = min(start + batch_size, n_samples)
15         xb = x[start:end]
16         yb = y[start:end]
17         pred = xb @ w
18         error = pred - yb
19         batch_len = end - start
20         grad = (2.0 / batch_len) * (xb.T @ error)
21         w -= learning_rate * grad
22
23 return w
```

2.

The screenshot shows a Deep-ML interface with a problem description for implementing the Adam optimization algorithm. It provides instructions and parameters for the function. The right panel shows a code editor with the implementation of the Adam optimizer. The code uses NumPy and includes variable names like `f`, `grad`, `x0`, `learning_rate`, `beta1`, `beta2`, `epsilon`, and `num_iterations`. The code editor includes standard controls like Run Code, Reset, Save Code, and Submissions.

```
1 import numpy as np
2
3 def adam_optimizer(f, grad, x0, learning_rate=0.001, beta1=0.9, beta2=0.999,
4     epsilon=1e-8, num_iterations=10):
5     # Your code here
6     x = x0.astype(float)
7     m = np.zeros_like(x)
8     v = np.zeros_like(x)
9
10    for t in range(1, num_iterations+1):
11        g = grad(x)
12        m = beta1 * m + (1 - beta1) * g
13        v = beta2 * v + (1 - beta2) * (g ** 2)
14        m_hat = m / (1 - beta1 ** t)
15        v_hat = v / (1 - beta2 ** t)
16        x = x - learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)
17
18    return x
```

3.

The screenshot shows a programming assignment interface. On the left, the problem description for "Implement Batch Normalization for BCHW" is displayed, along with input and example sections. The input section contains sample code for generating random data and parameters. The example section shows the expected output of the normalization process. On the right, the code editor shows a template for the `batch_normalization` function, which includes importing numpy, defining the function, calculating mean and variance, normalizing the data, and applying scale and shift parameters. Below the code editor are buttons for running the code, saving it, and viewing submissions. The test results section indicates 3/3 tests passed, with links to Test 1, Test 2, and Test 3.

```
1 import numpy as np
2
3 def batch_normalization(X: np.ndarray, gamma: np.ndarray, beta: np.ndarray,
4     epsilon: float = 1e-5) -> np.ndarray:
5     # Your code here
6     mean = X.mean(axis=(0, 2, 3), keepdims=True)
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
```

4.

The screenshot shows a programming assignment interface. On the left, the problem description for "Implement Layer Normalization for Sequence Data" is displayed, along with input and example sections. The input section contains sample code for generating random data and parameters. The example section shows the expected output of the normalization process. On the right, the code editor shows a template for the `layer_normalization` function, which includes importing numpy, defining the function, calculating mean and variance across the feature dimension, normalizing the data, and applying scaling and shifting parameters. Below the code editor are buttons for running the code, saving it, and viewing submissions. The test results section indicates 3/3 tests passed, with links to Test 1, Test 2, and Test 3.

```
1 import numpy as np
2
3 def layer_normalization(X: np.ndarray, gamma: np.ndarray, beta: np.ndarray,
4     epsilon: float = 1e-5) -> np.ndarray:
5     """
6         Perform Layer Normalization.
7     """
8     # Your code here
9     mean = X.mean(axis=2, keepdims=True)
10    var = X.var(axis=2, keepdims=True)
11    X_hat = (X - mean) / np.sqrt(var + epsilon)
12    Y = gamma * X_hat + beta
13
14 return Y
```

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.]])
```

Descrinnn:

Notebook Mode

```
3 class DropoutLayer:
4     def __init__(self, p: float):
5         self.mask = None
6         self.scale = 1.0 / (1.0 - self.p)
7
8     def forward(self, x: np.ndarray, training: bool = True) -> np.ndarray:
9         """Forward pass of the dropout layer."""
10        # Your code here
11        if training:
12            self.mask = np.random.binomial(1, 1.0 - self.p, size=x.shape).astype(x.dtype)
13            return x * self.mask * self.scale
14        else:
15            return x
16
17    def backward(self, grad: np.ndarray) -> np.ndarray:
18        """Backward pass of the dropout layer."""
19        # Your code here
20        return grad * self.mask * self.scale
21
22
23
24
25
```

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

Test Results 3/3

Test 1 Test 2 Test 3

Test Case Ask Tutor