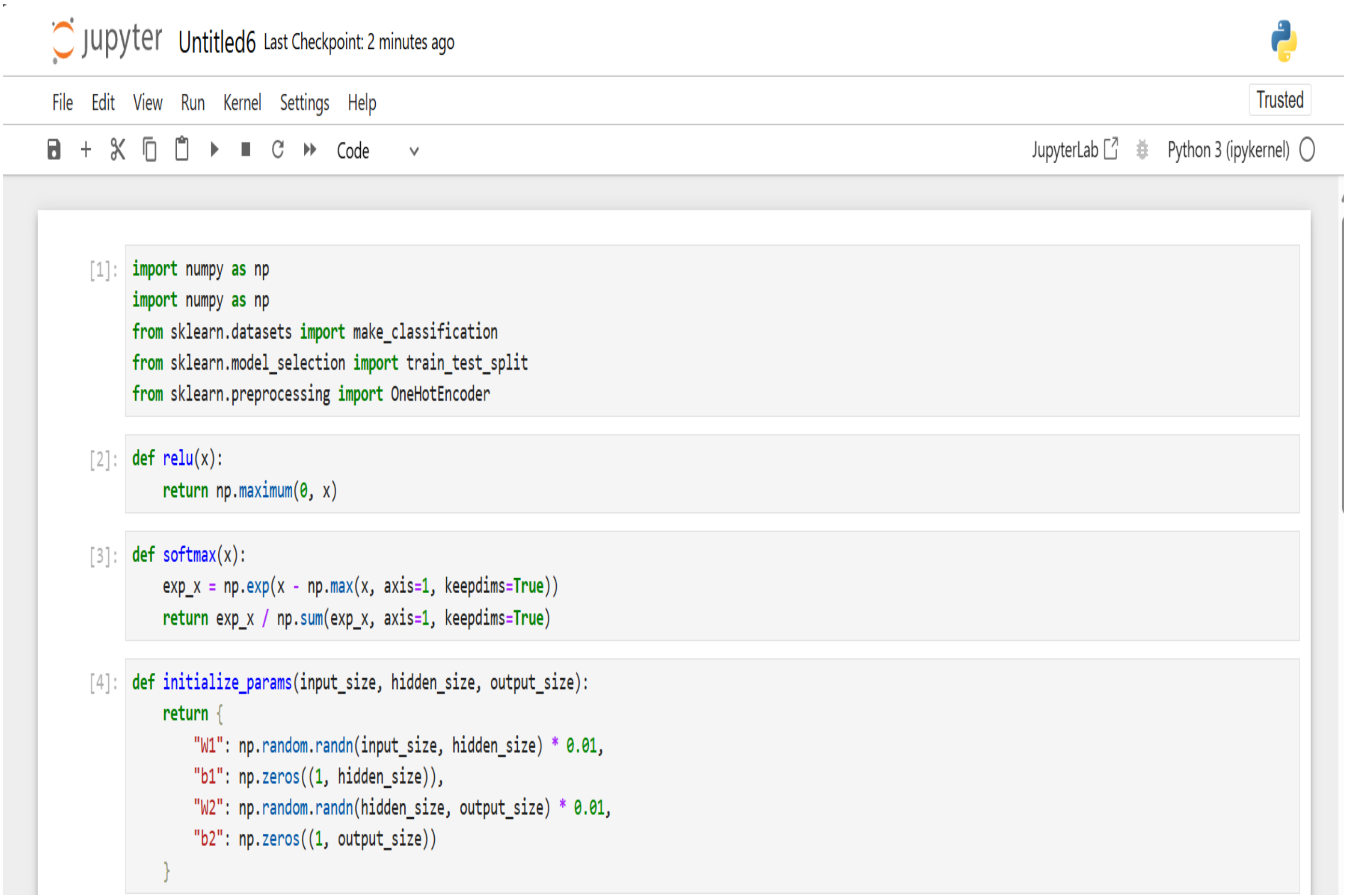


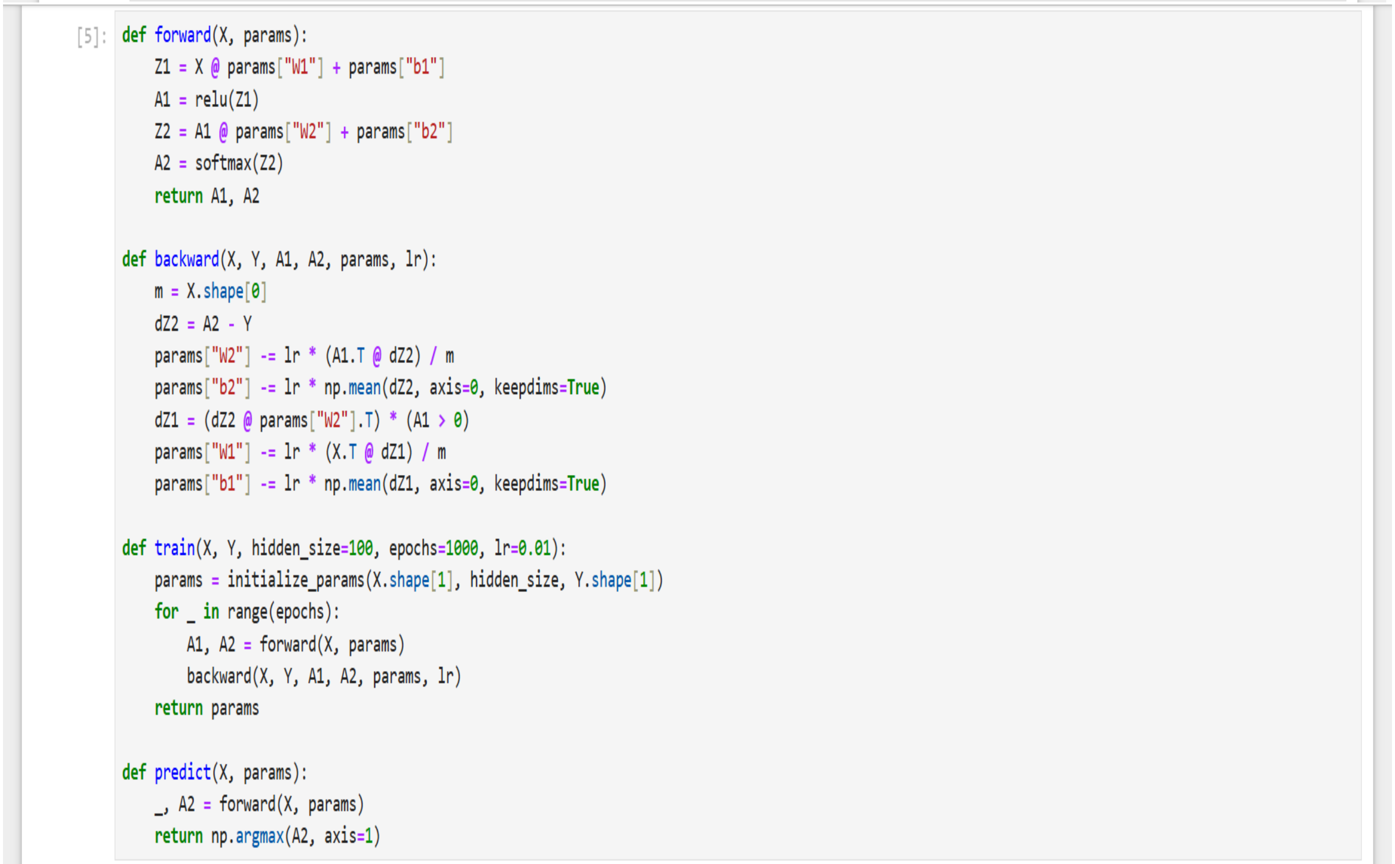
jupyter Untitled6 Last Checkpoint: 2 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[1]: import numpy as np
     import numpy as np
     from sklearn.datasets import make_classification
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import OneHotEncoder

[2]: def relu(x):
         return np.maximum(0, x)

[3]: def softmax(x):
         exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
         return exp_x / np.sum(exp_x, axis=1, keepdims=True)

[4]: def initialize_params(input_size, hidden_size, output_size):
         return {
             "W1": np.random.randn(input_size, hidden_size) * 0.01,
             "b1": np.zeros((1, hidden_size)),
             "W2": np.random.randn(hidden_size, output_size) * 0.01,
             "b2": np.zeros((1, output_size))
         }
```

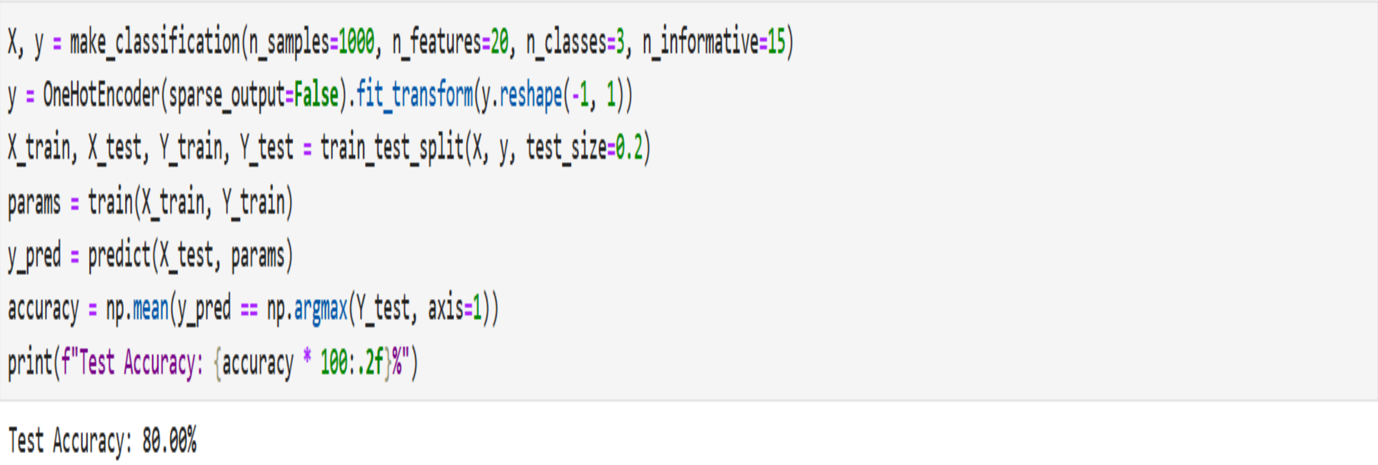


```
[5]: def forward(X, params):
         Z1 = X @ params["W1"] + params["b1"]
         A1 = relu(Z1)
         Z2 = A1 @ params["W2"] + params["b2"]
         A2 = softmax(Z2)
         return A1, A2

     def backward(X, Y, A1, A2, params, lr):
         m = X.shape[0]
         dZ2 = A2 - Y
         params["W2"] -= lr * (A1.T @ dZ2) / m
         params["b2"] -= lr * np.mean(dZ2, axis=0, keepdims=True)
         dZ1 = (dZ2 @ params["W2"].T) * (A1 > 0)
         params["W1"] -= lr * (X.T @ dZ1) / m
         params["b1"] -= lr * np.mean(dZ1, axis=0, keepdims=True)

     def train(X, Y, hidden_size=100, epochs=1000, lr=0.01):
         params = initialize_params(X.shape[1], hidden_size, Y.shape[1])
         for _ in range(epochs):
             A1, A2 = forward(X, params)
             backward(X, Y, A1, A2, params, lr)
         return params

     def predict(X, params):
         _, A2 = forward(X, params)
         return np.argmax(A2, axis=1)
```



```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=3, n_informative=15)
y = OneHotEncoder(sparse_output=False).fit_transform(y.reshape(-1, 1))
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
params = train(X_train, Y_train)
y_pred = predict(X_test, params)
accuracy = np.mean(y_pred == np.argmax(Y_test, axis=1))
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Test Accuracy: 80.00%
```