Sequential Class User Manual

06 November 2024 21:40

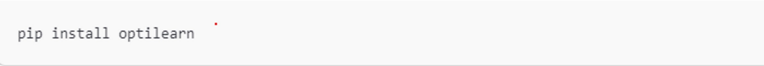
Sequential Class User Manual

Overview

The Sequential class in **Optilearn** (formerly optimal-data-selector) is designed to build and train neural networks layer-by-layer. This class is flexible and enables users to stack various neural network layers in a specified order, making it ideal for quickly building deep learning models.

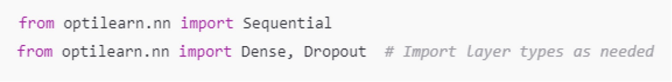
Importing Sequential

First, ensure you have **Optilearn** installed:



```
pip install optilearn
```

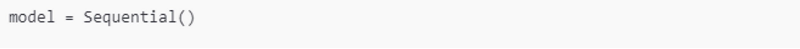
Then, import the Sequential class from optilearn:



```
from optilearn.nn import Sequential
from optilearn.nn import Dense, Dropout  # Import layer types as needed
```

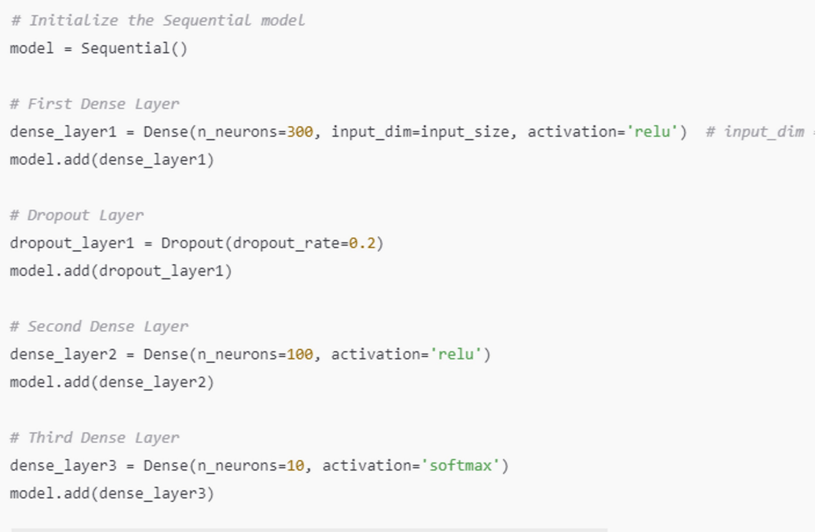
Creating a Sequential Model

Initialize the Model:



```
model = Sequential()
```

Adding Layers: Add layers in the order you want them to appear in the model. The Sequential

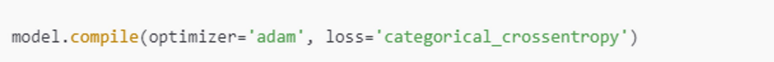


```
# Initialize the Sequential model
model = Sequential()

# First Dense Layer
dense_layer1 = Dense(n_neurons=300, input_dim=input_size, activation='relu')  # input_dim
model.add(dense_layer1)

# Dropout Layer
dropout_layer1 = Dropout(dropout_rate=0.2)
model.add(dropout_layer1)

# Second Dense Layer
dense_layer2 = Dense(n_neurons=100, activation='relu')
model.add(dense_layer2)

# Third Dense Layer
dense_layer3 = Dense(n_neurons=10, activation='softmax')
model.add(dense_layer3)
```

l model processes each layer in the order it's added.

- **Dense**: A fully connected layer.
- **Dropout**: A dropout layer for regularization (specify the dropout rate).

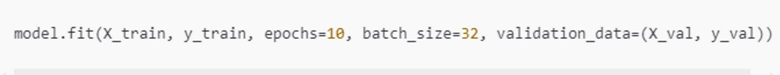
Compile the Model: Set the optimizer, loss function.



```
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

- **optimizer**: Choose the optimization algorithm (adam, sgd, etc.).
- **loss**: Define the loss function for training.

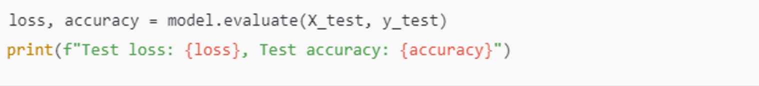
Training the Model: Use the fit method to train the model.



```
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

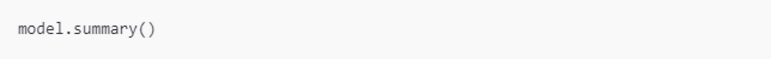
- **X_train, y_train**: Training data and labels.
- **epochs**: Number of times to iterate over the entire dataset.
- **batch_size**: Number of samples per gradient update.
- **validation_data**: Tuple (X_val, y_val) to evaluate the model on validation data during training.

Evaluating the Model: To assess model performance, use the evaluate method:



```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test loss: {loss}, Test accuracy: {accuracy}")
```

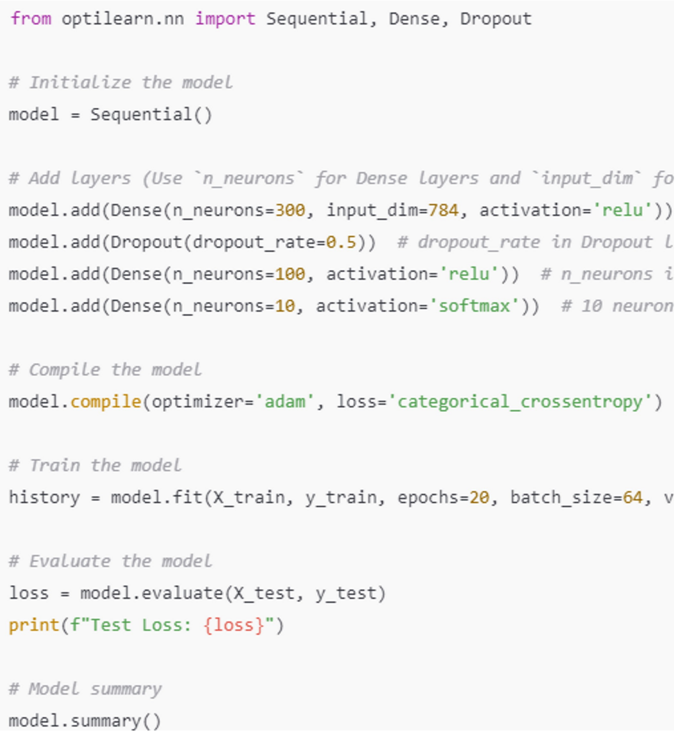
Generating Model Summary: Get a summary of the model architecture, including layer names



```
model.summary()
```

Example

Here's a full example that demonstrates a basic workflow:



```
from optilearn.nn import Sequential, Dense, Dropout

# Initialize the model
model = Sequential()

# Add layers (Use `n_neurons` for Dense layers and `input_dim` fo
model.add(Dense(n_neurons=300, input_dim=784, activation='relu'))
model.add(Dropout(dropout_rate=0.5))  # dropout_rate in Dropout l
model.add(Dense(n_neurons=100, activation='relu'))  # n_neurons i
model.add(Dense(n_neurons=10, activation='softmax'))  # 10 neuron

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=64, v

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")

# Model summary
model.summary()
```

- **Dense Layer Initialization**:
  - For the Dense layers, I replaced units with n_neurons as per your Dense class implementation.
  - The first Dense layer uses input_dim=784, which represents the number of features (e.g., 784 for MNIST images with 28x28 pixels flattened).
  - Subsequent layers only need to specify n_neurons and the activation function.
- **Dropout Layer**:
  - Changed rate to dropout_rate to match your Dropout class implementation.
- **Compile Method**:
  - Since your compile method doesn't include metrics, I removed it from the compile line.
  - If you want to track accuracy, you would need to implement it in your custom evaluate method or elsewhere in your model.
- **Model Summary**:
  - The model summary will now work based on your custom Sequential class.