Can you give an example of the code for a neural network?

Certainly! Here's a simplified example of **Python** code to create a feedforward neural network using the popular deep learning library TensorFlow/Keras. This example demonstrates how to build a basic neural network for a binary classification task:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Create a sequential model
model = keras.Sequential()

# Add an input layer with 10 neurons (input shape should match your data)
model.add(layers.Input(shape=(10,)))

# Add one or more hidden layers with ReLU activation
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))

# Add an output layer with one neuron (for binary classification) and sigmoid activation
model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model, specifying the optimizer, loss function, and metrics
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print a summary of the model architecture
model.summary()

# Now, you can train the model with your data using model.fit() and evaluate it.
# For training data X_train and labels y_train, you can use:
# model.fit(X_train, y_train, epochs=10, batch_size=32)

# For evaluation, you can use:
# loss, accuracy = model.evaluate(X_test, y_test)
```

In this example:

1. We import TensorFlow and Keras, which provides a high-level API for building neural networks.
2. We create a sequential model, a linear stack of layers.
3. We add an input layer with 10 neurons, assuming the input data has 10 features.
4. We add two hidden layers with ReLU activation functions.
5. We add an output layer with one neuron and sigmoid activation for binary classification.

6. We compile the model with an optimizer ('adam' in this case), a loss function ('binary_crossentropy' for binary classification), and metrics (accuracy in this case).
7. We print a summary of the model's architecture to see the number of parameters and layer shapes.
8. Finally, we would typically train the model using model.fit() with your training data and labels and evaluate its performance using model.evaluate() on test data.

Remember that in a real-world scenario, you'll need to preprocess your data, split it into training and testing sets, and adjust hyperparameters like the number of neurons and layers based on your specific problem and dataset.