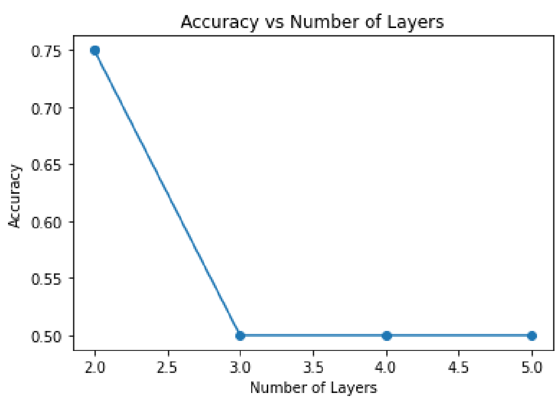
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [6]:
```
# Part 1.1
# Define the XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Build a function to create a neural network with a specified number of hidden layers and neurons per layer
def build_model(num_layers, num_neurons):
    model = keras.Sequential()
    model.add(layers.Input(shape=(2,)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Train the neural network for each configuration
results = []
for num_layers in range(2, 6):
    model = build_model(num_layers, 2)
    history = model.fit(X, y, epochs=400, verbose=0)
    accuracy = history.history['accuracy'][-1]
    results.append((num_layers, accuracy))

# Plot the results
num_layers, accuracies = zip(*results)
plt.plot(num_layers, accuracies, marker='o')
plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Number of Layers')
plt.show()
# Here, the accuracy peaks at 2 layers and then falls off precipitously elsewhere.
```

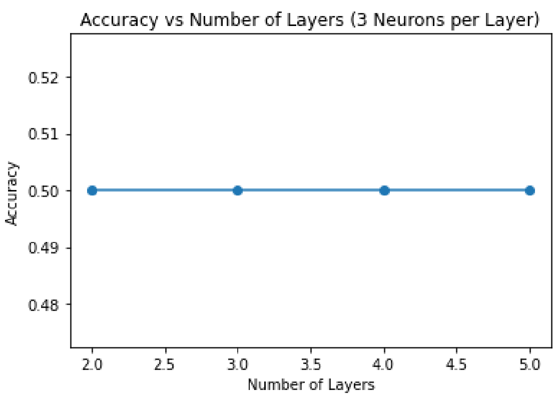


In [7]:
```
# Part 1.2
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Build a function to create a neural network with a specified number of hidden layers and neurons per layer
def build_model(num_layers, num_neurons):
    model = keras.Sequential()
    model.add(layers.Input(shape=(2,)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Train the neural network for each configuration
results = []
for num_layers in range(2, 6):
    model = build_model(num_layers, 3)
    history = model.fit(X, y, epochs=400, verbose=0)
    accuracy = history.history['accuracy'][-1]
    results.append((num_layers, accuracy))

# Plot the results
num_layers, accuracies = zip(*results)
plt.plot(num_layers, accuracies, marker='o')
plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Number of Layers (3 Neurons per Layer)')
plt.show()
# Here, the accuracy is constant across all numbers of layers.
```

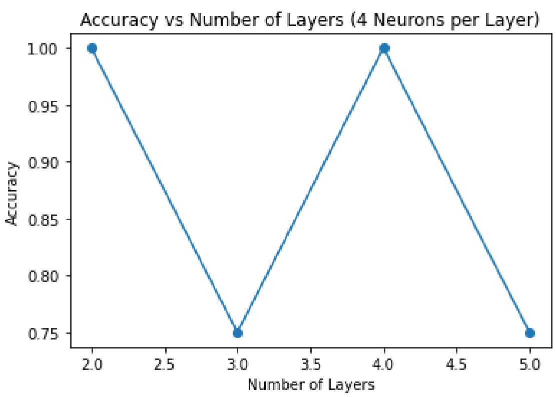


In [8]:
```
# Part 1.3
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Build a function to create a neural network with a specified number of hidden layers and neurons per layer
def build_model(num_layers, num_neurons):
    model = keras.Sequential()
    model.add(layers.Input(shape=(2,)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Train the neural network for each configuration
results = []
for num_layers in range(2, 6):
    model = build_model(num_layers, 4)
    history = model.fit(X, y, epochs=400, verbose=0)
    accuracy = history.history['accuracy'][-1]
    results.append((num_layers, accuracy))

# Plot the results
num_layers, accuracies = zip(*results)
plt.plot(num_layers, accuracies, marker='o')
plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Number of Layers (4 Neurons per Layer)')
plt.show()
# This is the best model so far, with peak accuracy at 2 and 4 layers and accuracy at 0.75 (minimum) at 3 and 5 layers
```



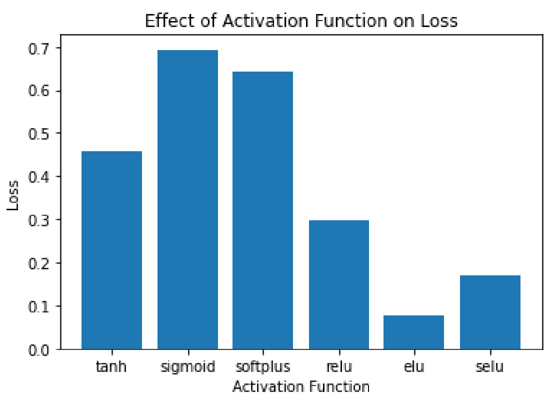
In [9]:
```
# Part 1.4
# Define the XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Build a function to create a neural network with a specified number of hidden layers, neurons per layer, and activat
def build_model(num_layers, num_neurons, activation):
    model = keras.Sequential()
    model.add(layers.Input(shape=(2,)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation=activation))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Activation functions to compare
activation_functions = ['tanh', 'sigmoid', 'softplus', 'relu', 'elu', 'selu']

# Train the neural network for each activation function
results = {}
for activation in activation_functions:
    model = build_model(3, 4, activation)  # Using the most optimal configuration (3 layers, 4 neurons per layer)
    history = model.fit(X, y, epochs=400, verbose=0)
    loss = history.history['loss'][-1]
    results[activation] = loss

# Plot the results
plt.bar(results.keys(), results.values())
plt.xlabel('Activation Function')
plt.ylabel('Loss')
plt.title('Effect of Activation Function on Loss')
plt.show()
```



In [10]:
```
#Part 1.5
# Define the XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Build a function to create a neural network with a specified number of hidden layers, neurons per layer, and optimize
def build_model(num_layers, num_neurons, optimizer):
    model = keras.Sequential()
    model.add(layers.Input(shape=(2,)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Optimizers to compare
optimizers = ['adam', 'rmsprop', 'adagrad', 'adadelta', 'adamax', 'nadam']

# Train the neural network for each optimizer
results = {}
for optimizer in optimizers:
    model = build_model(3, 4, optimizer)  # Using the most optimal configuration (3 layers, 4 neurons per layer)
    history = model.fit(X, y, epochs=400, verbose=0)
    loss = history.history['loss'][-1]
    results[optimizer] = loss

# Print the results
for optimizer, loss in results.items():
    print(f'Optimizer: {optimizer}, Loss: {loss}')
```

```
Optimizer: adam, Loss: 0.512321412563324
Optimizer: rmsprop, Loss: 0.4918256998062134
Optimizer: adagrad, Loss: 0.6180860996246338
Optimizer: adadelta, Loss: 0.6931471824645996
Optimizer: adamax, Loss: 0.4701884388923645
Optimizer: nadam, Loss: 0.6931471824645996
```

In [ ]:
```
# Part 2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
# Load the recent-grads.csv dataset
df = pd.read_csv('../../datasets/college-majors/data/recent-grads.csv')

# Preprocessing the data
# For the purpose of this example, let's assume the dataset contains features and a target variable
# Perform necessary preprocessing steps such as handling missing values, encoding categorical variables, and scaling nu

# Split the dataset into features and target variable
X = df.drop('target_variable', axis=1)  # Replace 'target_variable' with the actual target variable name
y = df['target_variable']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build a function to create a neural network with a specified configuration
def build_model(num_layers, num_neurons, activation, optimizer):
    model = keras.Sequential()
    model.add(layers.Input(shape=(X_train.shape[1],)))
    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, activation=activation))
    model.add(layers.Dense(1, activation='sigmoid'))  # Assuming binary classification
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Experiment with different configurations
best_accuracy = 0
best_config = None
for num_layers in range(2, 5):
    for num_neurons in [32, 64, 128]:
        for activation in ['relu', 'tanh', 'sigmoid']:
            for optimizer in ['adam', 'rmsprop', 'adagrad']:
                model = build_model(num_layers, num_neurons, activation, optimizer)
                model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
                y_pred = (model.predict(X_test) > 0.5).astype("int32")
                accuracy = accuracy_score(y_test, y_pred)
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_config = (num_layers, num_neurons, activation, optimizer)

# Print the best configuration and accuracy
print(f'Best Configuration: {best_config}, Accuracy: {best_accuracy}')
```