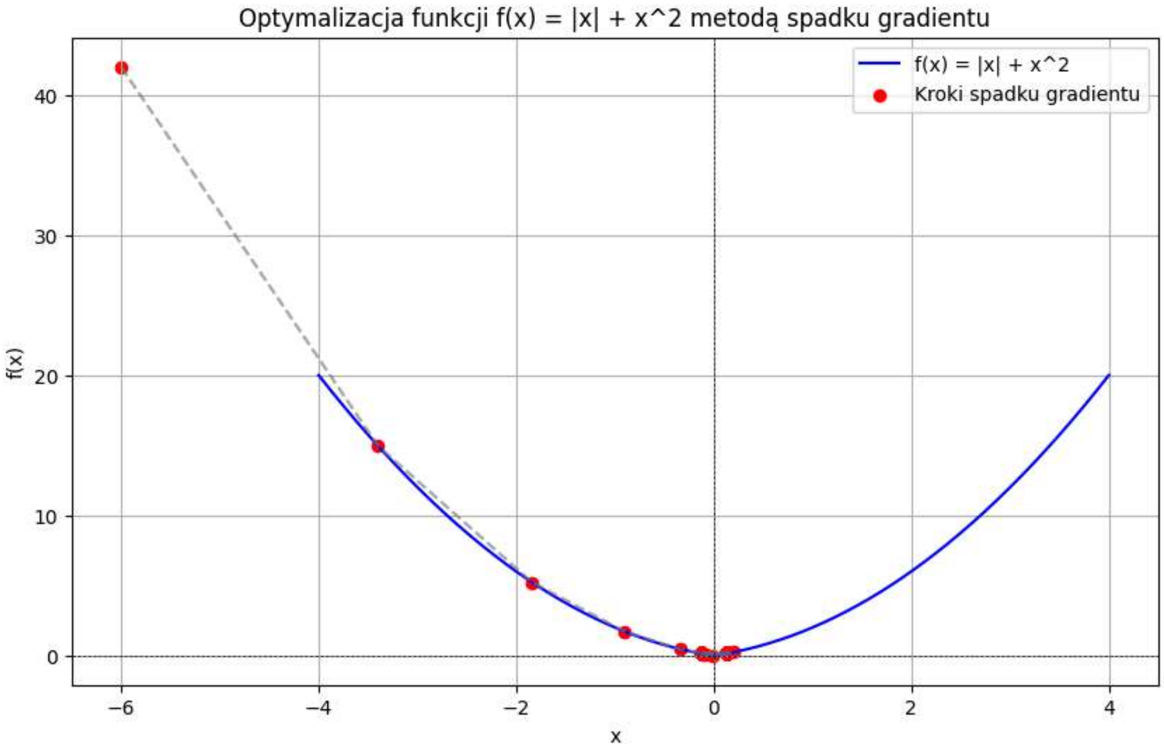
In [2]:

```
# 1. Krzysztof Świerczek Zrealizuj w Pythonie optymalizację funkcji metodą spadk
# Wariant drugi, funkcja: f(x) = |x| + x^2 metodą spadku gradientu i wizualizac

import numpy as np
import matplotlib.pyplot as plt

def f(x): # Funkcja celu
    return np.abs(x) + x**2

def gradient(x): # Gradient funkcji celu
    return 1 + 2*x if x > 0 else -1 + 2*x

def gradient_descent(start_x, learning_rate, tolerance, max_iters): # Spadek gra
    x = start_x
    history = [x]
    for _ in range(max_iters):
        grad = gradient(x)
        new_x = x - learning_rate * grad
        history.append(new_x)
        if abs(new_x - x) < tolerance: # Sprawdzanie warunku stopu
            break
        x = new_x
    return x, history

# Parametry algorytmu
start_x = -6.0          # Punkt początkowy
learning_rate = 0.2     # Krok uczenia
tolerance = 1e-6        # Tolerancja
max_iters = 250         # Maksymalna liczba iteracji

optimal_x, history = gradient_descent(start_x, learning_rate, tolerance, max_ite

# Wizualizacja
x_vals = np.linspace(-4, 4, 500)
y_vals = f(x_vals)
plt.figure(figsize=(10, 6))
plt.plot(x_vals, y_vals, label='f(x) = |x| + x^2', color='blue')
plt.scatter(history, [f(x) for x in history], color='red', label='Kroki spadku g
plt.plot(history, [f(x) for x in history], linestyle='--', color='gray', alpha=0
plt.title('Optymalizacja funkcji f(x) = |x| + x^2 metodą spadku gradientu')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.legend()
plt.grid()
plt.show()

print(f"Wartość optymalna x: {optimal_x}")
print(f"Wartość funkcji w minimum f(x): {f(optimal_x)}")
```



```
Wartość optymalna x: 0.12500000000000006
Wartość funkcji w minimum f(x): 0.14062500000000006
```

In [11]:

```
# 2. Zrealizuj w Pythonie najprostszą sieć neuronową wraz z ewaluacją i progno
# Temat: sieć neuronowa do klasyfikacji binarnej.

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Generowanie przykładowych danych binarnych (2000 próbek z dwoma cechami)
np.random.seed(42)
X = np.random.rand(2000, 2)  # wejście
y = (X[:, 0] + X[:, 1] > 1).astype(int)  # etykiety: klasa 1, jeśli suma cech >

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

model = Sequential([
    Dense(4, activation='relu', input_shape=(2,)),  # Warstwa ukryta z 4 neurona
    Dense(1, activation='sigmoid')  # Warstwa wyjściowa (klasyfikacja binarna)
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(X_train, y_train, epochs=50, batch_size=8, verbose=0) # Tren

loss, accuracy = model.evaluate(X_test, y_test, verbose=0) # Ewaluacja modelu na
print(f'Loss: {loss:.4f}, Accuracy: {accuracy:.4f}')

predictions = (model.predict(X_test) > 0.5).astype(int) # Prognozowanie na podst

print("\nClassification Report:\n") # Raport klasyfikacji
print(classification_report(y_test, predictions))


new_data = np.array([[0.1, 0.4], [0.8, 0.7]]) # Przykładowe prognozy dla nowych
```

```python
predictions_new = (model.predict(new_data) > 0.5).astype(int)
print("\nNew data predictions:")
print(predictions_new)
```

```
C:\Users\krzys\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\sr
c\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Loss: 0.1112, Accuracy: 0.9975
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

Classification Report:

              precision    recall  f1-score   support

           0       0.99      1.00      1.00       181
           1       1.00      1.00      1.00       219

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
weighted avg       1.00      1.00      1.00       400

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step

New data predictions:
[[0]
 [1]]
```

In [14]:

```python
# 3. Zrealizuj projektowanie, trenowanie i testowanie sieci konwolucyjnej na pod
# Wariant drugi: Zaprojektuj, wytrenuj i przetestuj sieć konwolucyjną na zbiorze

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F  # Dodany import

# Ustawienia urządzenia
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Wczytanie zbioru danych CIFAR-10
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False)

# Definiowanie architektury sieci konwolucyjnej
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Inicjalizacja modelu, funkcji straty i optymalizatora
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Pętla treningowa
num_epochs = 10
train_losses = []
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(trainloader)
    train_losses.append(avg_loss)
    print(f'Epoka [{epoch + 1}/{num_epochs}], Strata: {avg_loss:.4f}')

print('Zakończone trenowanie modelu')

# Testowanie modelu
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Dokładność modelu na 10000 testowych obrazach: {100 * correct / total:.2

# Wizualizacja strat
plt.plot(train_losses)
plt.title('Strata modelu w kolejnych epokach')
plt.xlabel('Epoka')
plt.ylabel('Strata')
plt.show()
```

```
---------------------------------------------------------------------------
SSLCertVerificationError                  Traceback (most recent call last)
File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:1348, in AbstractHTTPHandler.do_open(self, http_class, req, **http_conn_args)
   1347 try:
-> 1348     h.request(req.get_method(), req.selector, req.data, headers,
   1349               encode_chunked=req.has_header('Transfer-encoding'))
   1350 except OSError as err: # timeout error

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:1276, in HTTPConnection.request(self, method, url, body, headers, encode_chunked)
   1275 """Send a complete request to the server."""
-> 1276 self._send_request(method, url, body, headers, encode_chunked)

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:1322, in HTTPConnection._send_request(self, method, url, body, headers, encode_chunked)
   1321     body = _encode(body, 'body')
-> 1322 self.endheaders(body, encode_chunked=encode_chunked)

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:1271, in HTTPConnection.endheaders(self, message_body, encode_chunked)
   1270     raise CannotSendHeader()
-> 1271 self._send_output(message_body, encode_chunked=encode_chunked)

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:1031, in HTTPConnection._send_output(self, message_body, encode_chunked)
   1030 del self._buffer[:]
-> 1031 self.send(msg)
   1033 if message_body is not None:
   1034 
   1035     # create a consistent interface to message_body

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:969, in HTTPConnection.send(self, data)
    968 if self.auto_open:
--> 969     self.connect()
    970 else:

File ~\AppData\Local\Programs\Python\Python310\lib\http\client.py:1448, in HTTPSConnection.connect(self)
   1446     server_hostname = self.host
-> 1448 self.sock = self._context.wrap_socket(self.sock,
   1449                                       server_hostname=server_hostname)

File ~\AppData\Local\Programs\Python\Python310\lib\ssl.py:512, in SSLContext.wrap_socket(self, sock, server_side, do_handshake_on_connect, suppress_ragged_eofs, server_hostname, session)
    506 def wrap_socket(self, sock, server_side=False,
    507                 do_handshake_on_connect=True,
    508                 suppress_ragged_eofs=True,
    509                 server_hostname=None, session=None):
    510     # SSLSocket class handles server_hostname encoding before it calls
    511     # ctx._wrap_socket()
--> 512     return self.sslsocket_class._create(
    513         sock=sock,
    514         server_side=server_side,
    515         do_handshake_on_connect=do_handshake_on_connect,
    516         suppress_ragged_eofs=suppress_ragged_eofs,
    517         server_hostname=server_hostname,
    518         context=self,
    519         session=session
    520     )

File ~\AppData\Local\Programs\Python\Python310\lib\ssl.py:1070, in SSLSocket._create(cls, sock, server_side, do_handshake_on_connect, suppress_ragged_eofs, server_hostname, context, session)
   1069             raise ValueError("do_handshake_on_connect should not be specified for non-blocking sockets")
-> 1070         self.do_handshake()
   1071 except (OSError, ValueError):

File ~\AppData\Local\Programs\Python\Python310\lib\ssl.py:1341, in SSLSocket.do_handshake(self, block)
   1340         self.settimeout(None)
-> 1341     self._sslobj.do_handshake()
   1342 finally:

SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:997)

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
Cell In[14], line 19
     14 # Wczytanie zbioru danych CIFAR-10
     15 transform = transforms.Compose([
     16     transforms.ToTensor(),
     17     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
     18 ])
---> 19 trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
     20 trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True)
     21 testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\datasets\cifar.py:66, in CIFAR10.__init__(self, root, train, transform, target_transform, download)
     63 self.train = train  # training set or test set
     65 if download:
---> 66     self.download()
     68 if not self._check_integrity():
     69     raise RuntimeError("Dataset not found or corrupted. You can use download=True to download it")

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\datasets\cifar.py:139, in CIFAR10.download(self)
    137 if self._check_integrity():
    138     return
--> 139 download_and_extract_archive(self.url, self.root, filename=self.filename, md5=self.tgz_md5)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\datasets\utils.py:391, in download_and_extract_archive(url, download_root, extract_root, filename, md5, remove_finished)
    388 if not filename:
    389     filename = os.path.basename(url)
--> 391 download_url(url, download_root, filename, md5)
    393 archive = os.path.join(download_root, filename)
    394 extract_archive(archive, extract_root, remove_finished)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\data
sets\utils.py:121, in download_url(url, root, filename, md5, max_redirect_hops)
    118     _download_file_from_remote_location(fpath, url)
    119 else:
    120     # expand redirect chain if needed
--> 121     url = _get_redirect_url(url, max_hops=max_redirect_hops)
    123     # check if file is located on Google Drive
    124     file_id = _get_google_drive_file_id(url)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\data
sets\utils.py:66, in _get_redirect_url(url, max_hops)
     63 headers = {"Method": "HEAD", "User-Agent": USER_AGENT}
     65 for _ in range(max_hops + 1):
---> 66     with urllib.request.urlopen(urllib.request.Request(url, headers=heade
rs)) as response:
     67         if response.url == url or response.url is None:
     68             return url

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:216, in urlo
pen(url, data, timeout, cafile, capath, cadefault, context)
    214 else:
    215     opener = _opener
--> 216 return opener.open(url, data, timeout)

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:519, in Open
erDirector.open(self, fullurl, data, timeout)
    516     req = meth(req)
    518 sys.audit('urllib.Request', req.full_url, req.data, req.headers, req.get_
method())
--> 519 response = self._open(req, data)
    521 # post-process response
    522 meth_name = protocol+"_response"

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:536, in Open
erDirector._open(self, req, data)
    533     return result
    535 protocol = req.type
--> 536 result = self._call_chain(self.handle_open, protocol, protocol +
    537                           '_open', req)
    538 if result:
    539     return result

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:496, in Open
erDirector._call_chain(self, chain, kind, meth_name, *args)
    494 for handler in handlers:
    495     func = getattr(handler, meth_name)
--> 496     result = func(*args)
    497     if result is not None:
    498         return result

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:1391, in HTT
PSHandler.https_open(self, req)
   1390 def https_open(self, req):
-> 1391     return self.do_open(http.client.HTTPSConnection, req,
   1392         context=self._context, check_hostname=self._check_hostname)

File ~\AppData\Local\Programs\Python\Python310\lib\urllib\request.py:1351, in Abs
tractHTTPHandler.do_open(self, http_class, req, **http_conn_args)
```

```
   1348         h.request(req.get_method(), req.selector, req.data, headers,
   1349                   encode_chunked=req.has_header('Transfer-encoding'))
   1350     except OSError as err: # timeout error
-> 1351         raise URLError(err)
   1352     r = h.getresponse()
   1353 except:

URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify fail
ed: unable to get local issuer certificate (_ssl.c:997)>
```

```
In [ ]:
```