```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')

np.random.seed(42)

# Generowanie danych
nb_of_samples = 30
sequence_len = 20

# Możliwe wartości
levels = np.array([0.33, 0.66, 1.0])

# Tworzymy dane wejściowe (X)
X = np.random.choice(levels, size=(nb_of_samples, sequence_len))

# Cele: średnie odchylenie standardowe
t = np.std(X, axis=1)

# Forward propagation
def update_state(xk, sk, wx, wRec):
    return xk * wx + sk * wRec

def forward_states(X, wx, wRec):
    S = np.zeros((X.shape[0], X.shape[1]+1))
    for k in range(X.shape[1]):
        S[:,k+1] = update_state(X[:,k], S[:,k], wx, wRec)
    return S

def loss(y, t):
    return np.mean((t - y)**2)

# Backward propagation
def output_gradient(y, t):
    return 2. * (y - t)

def backward_gradient(X, S, grad_out, wRec):
    grad_over_time = np.zeros((X.shape[0], X.shape[1]+1))
    grad_over_time[:,-1] = grad_out
    wx_grad = 0
    wRec_grad = 0
    for k in range(X.shape[1], 0, -1):
        wx_grad += np.sum(np.mean(grad_over_time[:,k] * X[:,k-1],
axis=0))
        wRec_grad += np.sum(np.mean(grad_over_time[:,k] * S[:,k-1],
axis=0))
        grad_over_time[:,k-1] = grad_over_time[:,k] * wRec
    return (wx_grad, wRec_grad), grad_over_time
```

```python
#  Sprawdzenie gradientu
params = [1.0, 1.0]
eps = 1e-7

S = forward_states(X, params[0], params[1])
grad_out = output_gradient(S[:,-1], t)
backprop_grads, _ = backward_gradient(X, S, grad_out, params[1])

for p_idx in range(len(params)):
    original = params[p_idx]
    params[p_idx] += eps
    plus_loss = loss(forward_states(X, params[0], params[1])[:,-1], t)
    params[p_idx] -= 2 * eps
    min_loss = loss(forward_states(X, params[0], params[1])[:,-1], t)
    params[p_idx] = original
    grad_num = (plus_loss - min_loss) / (2 * eps)
    if not np.isclose(grad_num, backprop_grads[p_idx]):
        raise ValueError(f'Gradient check failed at param {p_idx}')
print("Gradient check passed!")
```

```
Gradient check passed!
```

```python
# Optymalizacja RProp
def update_rprop(X, t, W, W_prev_sign, W_delta, eta_p, eta_n):
    S = forward_states(X, W[0], W[1])
    grad_out = output_gradient(S[:,-1], t)
    W_grads, _ = backward_gradient(X, S, grad_out, W[1])
    W_sign = np.sign(W_grads)
    for i in range(len(W)):
        if W_sign[i] == W_prev_sign[i]:
            W_delta[i] *= eta_p
        else:
            W_delta[i] *= eta_n
    return W_delta, W_sign

# Parametry RProp
eta_p = 1.2
eta_n = 0.5
W = [0.5, 0.5]
W_delta = [0.001, 0.001]
W_sign = [0, 0]
ws_history = [tuple(W)]

for _ in range(300):
    W_delta, W_sign = update_rprop(X, t, W, W_sign, W_delta, eta_p,
eta_n)
    for i in range(len(W)):
        W[i] -= W_sign[i] * W_delta[i]
    ws_history.append(tuple(W))
```

```python
print(f"� Final weights: wx = {W[0]:.4f}, wRec = {W[1]:.4f}")
```

```
� Final weights: wx = 0.0473, wRec = 0.8967
```

```python
# Testowanie na nowej sekwencji
test_input = np.random.choice(levels, size=(1, 20))
expected_output = np.std(test_input)
model_output = forward_states(test_input, W[0], W[1])[:,-1][0]

print(f"Target (std): {expected_output:.4f}")
print(f"Model output: {model_output:.4f}")
```

```
Target (std): 0.2331
Model output: 0.2433
```