

```
1: import numpy as np
2: import sympy as sp
3:
4: current_minibatch = None
5:
6: def generate_trainingdata(m=25):
7:     return np.array([0,0]) + 0.25 * np.random.randn(m,2)
8:
9:
10: def f(x, minibatch):
11:     # loss function sum_{w in training data} f(x,w)
12:     y = 0
13:     count = 0
14:     for w in minibatch:
15:         z = x - w - 1
16:         left = 10 * (z[0]**2+z[1]**2)
17:         right = (z[0]+2)**2+(z[1]+4)**2
18:         y = y + min(left, right)
19:         count = count + 1
20:     return y/count
21:
22:
23: def gradient_function_fd(minibatch, epsilon=10**(-15)):
24:     def gradient_fd(x):
25:         dydx1 = (f(x + np.array([epsilon, 0]), minibatch) - f(x, minibatch)) / epsilon
26:         dydx2 = (f(x + np.array([0, epsilon]), minibatch) - f(x, minibatch)) / epsilon
27:         return np.array([dydx1, dydx2])
28:     return gradient_fd
29:
30: def sympy_loss(minibatch):
31:     x1, x2 = sp.symbols('x1 x2', real=True)
32:     function = 0
33:     for w in minibatch:
34:         z1 = x1 - w[0] - 1
35:         z2 = x2 - w[1] - 1
36:         left = 10 * (z1**2 + z2**2)
37:         right = (z1 + 2)**2 + (z2 + 4)**2
38:         function = sp.Min(left, right) + function
39:     function = function / len(minibatch)
40:     return function
41:
42: def gradient_function(minibatch):
43:     function = sympy_loss(minibatch)
44:     def gradient(x):
45:         dydx1 = function.diff(x1)
46:         dydx2 = function.diff(x2)
47:         return np.array([
48:             dydx1.subs(x1, x[0]).subs(x2, x[1]),
49:             dydx2.subs(x1, x[0]).subs(x2, x[1]),
50:         ])
51:
52:     return gradient
53:
54:
55: def loss(x, w):
56:     z = x - w - 1
57:     left = 10 * (z[0]**2+z[1]**2)
58:     right = (z[0]+2)**2+(z[1]+4)**2
59:     return min(left, right)
60:
61:
62: def f_clear(x, minibatch):
63:     return sum(loss(x, w) for w in minibatch) / len(minibatch)
64:
65:
66: def generate_minibatches(T, N=5, seed=42, shuffle=True,):
67:     global current_minibatch
68:     if shuffle:
69:         T = T.copy()
70:         if seed:
71:             np.random.seed(seed)
72:             np.random.shuffle(T)
73:     num_rows = T.shape[0]
74:     i = 0
75:
76:     minibatch = np.zeros((N, T.shape[1]), T.dtype)
77:     while True:
78:         for j in range(N):
79:             minibatch[j] = T[i % num_rows]
80:             i += 1
81:         if shuffle and i >= num_rows:
82:             # begin next epoch
83:             np.random.shuffle(T)
84:             i = 0
85:         current_minibatch = minibatch
86:         yield minibatch
87:
88:
89: def generate_optimisation_functions(batch, minibatch_size=5, finite_difference=True, **kwargs):
90:     minibatch_generator = generate_minibatches(
91:         batch, N=minibatch_size, **kwargs)
92:     for minibatch in minibatch_generator:
93:         def optim_func(x):
94:             return f_clear(x, minibatch)
95:         gradf = None
96:         if finite_difference:
97:             gradf = gradient_function_fd(minibatch)
98:         else:
99:             gradf = gradient_function(minibatch)
100:         yield (optim_func, gradf)
```

```
101:         yield "finished"
102:
103:
104: if __name__ == "__main__":
105:     import os
106:     os.makedirs("data", exist_ok=True)
107:     T = generate_trainingdata()
108:     import pandas as pd
109:     df = pd.DataFrame(T)
110:     df.to_csv("data/T.csv", index=False)
111:
112:     x = np.array([3, 3])
```