Week Eight PHY-480

Lewis

October 16, 2025

1. Strategies for finding a local minimum of an unconstrained objective function

Three main methods are used to find local minima in complex landscapes:

1. **Gradient Descent:** Iteratively updates position using
$$\vec{x}_{n+1} = \vec{x}_n - \eta \nabla f(\vec{x}_n),$$
where η is the step size.
2. **Newton's Method:** Uses curvature from the Hessian,
$$\vec{x}_{n+1} = \vec{x}_n - \mathbf{H}^{-1} \nabla f(\vec{x}_n),$$
for faster local convergence.
3. **Heuristic Methods:** Techniques like simulated annealing (used in Class 13) or random restarts help escape poor local minima.

2. Strategies for including constraints

Common ways to handle constraints include:

1. **Lagrange Multipliers:** Add constraints $g_i(\vec{x}) = 0$ to the objective:
$$\mathcal{L} = f(\vec{x}) + \sum_i \lambda_i g_i(\vec{x}).$$
2. **Penalty Functions:** Add terms that penalize constraint violations, e.g.
$$g(x) = 1 - 2x^2 + x^4,$$
leading to
$$H = \vec{x}^T \mathbf{M} \vec{x} + \lambda \sum_i g(x_i).$$
3. **Projection or Relaxation:** Project solutions back into the feasible region or relax discrete constraints to continuous ones.

3. Gradient Descent vs Newton's Method

- **Gradient Descent:** Uses first derivatives only; slower but simpler.
- **Newton's Method:** Uses second derivatives (Hessian); faster near minima but costlier to compute.

4. Gradient Descent Code for Continuous Ising Model

Modified from the Class 13 Ising model by allowing continuous spins $x_i \in [-1, 1]$ and adding a penalty term.

```
import numpy as np
import matplotlib.pyplot as plt

# Gradient Descent for Continuous Ising Spin Glass
N = 50
lam = 1.0
eta = 0.01
steps = 5000
np.random.seed(42)

# Random symmetric coupling matrix
J = np.random.normal(0, 1/np.sqrt(N), (N, N))
J = (J + J.T) / 2

# Initialize continuous spins in [-1, 1]
x = np.random.uniform(-1, 1, N)

def H(x):
    return x.T @ J @ x + lam * np.sum(1 - 2*x**2 + x**4)

def grad(x):
    return 2 * J @ x + lam * (-4*x + 4*x**3)

energies = []
for k in range(steps):
    x -= eta * grad(x)
    x = np.clip(x, -1, 1)
    energies.append(H(x))

plt.plot(energies)
plt.xlabel('Iteration')
plt.ylabel('Energy')
plt.title('Gradient Descent on Continuous Ising Spin Glass')
plt.grid(True)
plt.show()

print("Final approximate minimum energy:", energies[-1])
```

This replaces discrete spin flips with gradient updates, enforcing $|x_i| \approx 1$ via the penalty term.

5. Mapping between Max-Cut, QUBO, and Ising Models

The weighted Max-Cut problem divides graph vertices into two sets (V_A, V_B) to maximize the sum of edge weights across the cut. It maps directly to the Ising form:

$$Max - Cut : \max_{S_i=\pm 1} \frac{1}{2} \sum_{i,j} w_{ij}(1 - S_i S_j),$$

which is equivalent to minimizing the Ising Hamiltonian:

$$H = \sum_{i,j} J_{ij} S_i S_j.$$

The QUBO form uses binary variables $x_i \in \{0, 1\}$ related by $S_i = 2x_i - 1$. Thus, Max-Cut, QUBO, and Ising formulations describe the same problem in different variable spaces.

6. Goemans–Williamson Bound (0.87856)

The Goemans–Williamson (GW) algorithm uses a Semi-Definite Programming (SDP) relaxation of Max-Cut. Its rounding process guarantees a solution with at least 87.856% of the optimal cut weight:

$$\frac{Cut_{GW}}{Cut_{opt}} \geq 0.87856.$$

The algorithm is not in P because Max-Cut is NP-complete, but the SDP relaxation step itself can be solved in polynomial time.

7. GW-SDP Python Program Outline

```
import cvxpy as cp
import numpy as np

# Weighted adjacency matrix
N = 10
W = np.random.rand(N, N)
W = (W + W.T) / 2

# SDP variable (positive semi-definite matrix)
X = cp.Variable((N, N), PSD=True)
```

```
# Objective: maximize (1/2) _ij w_ij (1 - X_ij)
objective = cp.Maximize(0.5 * cp.sum(cp.multiply(W, (1 - X))))

# Constraints: X_ii = 1
constraints = [cp.diag(X) == 1]

# Solve SDP
prob = cp.Problem(objective, constraints)
prob.solve()

# Random hyperplane rounding
v = np.random.randn(N)
cut = np.sign(X.value @ v)
cut_value = 0.5 * np.sum(W * (1 - np.outer(cut, cut)))

print("Approx. Max-Cut value:", cut_value)
```

8. Using Ising Ground State to Compute Max-Cut Cost

For a complete graph with weights J_{ij} drawn from a GOE:

- Compute the Ising ground state energy $E_{GS} = \sum_{i<j} J_{ij} S_i S_j$ using your Ising model code.
- Convert it to the Max-Cut cost:

$$Cut_{max} = \frac{1}{2} \sum_{i,j} J_{ij}(1 - S_i S_j) = -E_{GS} + constant.$$

- This means a low Ising energy corresponds to a large Max-Cut weight. The same random matrix J can be reused for both Ising and Max-Cut comparisons.