

# Week Thirteen PHY-480

Lewis

November 2025

## 1 Pre Quiz Questions

### **Q24.1**

Quantum annealing solves optimization problems by starting in the ground state of a transverse field Hamiltonian and slowly turning on the problem Hamiltonian. If this is done slowly enough, the system stays in the ground state and ends in the solution. Simulated annealing instead uses temperature where it starts hot and cools down, escaping minima by thermal hopping rather than quantum tunneling.

### **Q24.2**

The adiabatic theorem says that if a Hamiltonian changes slowly enough and the energy gap does not close, a system remains in its ground state. For quantum annealing this means the annealing must be slow compared to the minimum gap, otherwise the system may leave the ground state.

### **Q24.3**

If you were writing some python code on this, define Pauli matrices, build operators for  $X_i$  and  $Z_i Z_j$ , construct the SK Hamiltonian and the transverse field Hamiltonian, set a time grid, form  $H(t) = at H_{\text{NPC}} + (1 - at)H_1$ , evolve the wavefunction forward in time, and compare the final annealed state to the exact ground state.

## Pre-Quiz 25

### **Q25.1**

A perceptron is a single artificial neuron. It multiplies each input by a weight, then adds a bias, and applies an activation function:

$$y = f \left( \sum_i w_i x_i + b \right).$$

It's "learning" the weights so the output matches the desired result.

### **Q25.2**

Underfitting happens when a model is too simple and does not learn the patterns in the data. In practice, this appears when the loss is still decreasing even after many epochs (like after 50 epochs), meaning the model has not captured the structure of the problem.

Overfitting happens when the model memorizes the training data instead of learning general patterns. This can be seen when the loss appears to converge, stops improving, and then begins to behave irregularly as training continues. In this case, the model performs well on training data but poorly on new data.

### **Q25.3**

A graph neural network (GNN) is a neural network that uses graph structure, updating each node using information from its neighbors. GNNs are used in tasks such as image processing and transportation networks. Training adjusts the model parameters using backpropagation, and testing checks performance on new data.

### **Q25.4**

A python outline for a GNN for NP-complete problems includes:

- Create training data by generating random couplings  $J_{ij}$  and computing exact ground states.
- Build a multilayer GNN with fixed intralayer couplings  $J_{ij}$  and learnable interlayer weights  $w_{ij}^{(l,l+1)}$ .
- Update nodes using

$$x_i^{(l+1)} = \alpha \tanh \left[ \alpha \frac{1}{\sqrt{N}} \sum_{j \neq i} J_{ij} x_j^{(l)} + (1 - \alpha) \frac{1}{\sqrt{N}} \sum_j w_{ij}^{(l,l+1)} x_j^{(l)} \right].$$

- Use the loss

$$f_L = \frac{1}{N n_t} \sum_{t,i} (x_i^{out,t} - x_i^{exact,t})^2,$$

and optimize the learnable weights with gradient descent.

- Test the trained GNN on new samples.