Quantum-Inspired Linear Regression

by Yassine Mouadi

1 Introduction to the Quantum System

When the rules of classical physics, such as Newton's laws, face limitations, we enter the quantum system: a collection of particles governed by fundamentally different principles.

A quantum system includes the following characteristics:

- **Quantization of energy**: We are accustomed to energy, momentum, and other quantities being continuous. However, in the quantum realm, they exist in discrete "packets," or quanta. This explains the stability of atoms and why elements have distinct chemical properties.

- **Wave-particle duality**: Instead of everything existing as a simple particle, quantum entities exhibit wave-particle duality. They are both a particle and a wave, depending on how they are observed. For their particle-like behavior, they appear at a single point upon measurement. For their wave-like behavior, when unobserved, they spread out and interfere with themselves, much like waves in water. This duality complicates our ability to visualize quantum entities and makes their nature fundamentally probabilistic until a measurement is made.

- **Superposition**: Before measurement, a quantum system can exist in all of its possible states simultaneously. For example, if a system has two states, it is not "either A or B," but rather "A and B." Upon observation, it is forced to "choose" a definite state: either A or B.

- **Entanglement**: When a connection is made between two or more quantum entities, their properties become perfectly intertwined, no matter how far apart they are. By measuring one, we can instantly determine the state of the other—at a speed surpassing light.

- **Heisenberg's uncertainty principle**: The more precisely you know the position of a quantum entity, the less precisely you can know its momentum. This is a fundamental property of nature.

- **Probabilistic Nature**: As previously discussed, before a measurement, we can only predict the probability of a quantum entity's state.

- **Tunneling**: In classical physics, a particle must have enough energy to overcome a barrier. For example, a ball at the bottom of a hill needs energy to roll to the other side. In the quantum realm, a quantum entity can pass through that hill, described as "tunneling" through the barrier.

How is a "particle" described in the quantum realm? Mathematically, a function completely describes the quantum state of a particle or a system of particles. It is probabilistic in nature and is called a probability amplitude, or "the probability cloud."

Such a function is called the **Wavefunction**, often denoted as Ψ.

It adheres perfectly to the characteristics of the quantum realm:

- It embodies wave-particle duality: its form can spread out over space, reflecting how a quantum entity can exist in different states before measurement.

- The probability density $|\Psi|^2$: This is the probability density of finding the quantum entity at a particular place at a particular time.

- Probabilistic Nature: Because the wavefunction is probability-based, we cannot predict with certainty where a quantum entity will be, but we can calculate the probability of finding it in a given region.

This wavefunction evolves over time, governed by the Schrödinger equation (which is not relevant to the subject at hand).

Any physical quantity that can be measured is called an **observable**. Whenever you measure an observable, you always get a real number as a result, not an imaginary number. Such observables have corresponding mathematical operators.

These operators are the "recipe" by which the wavefunction provides information about the observable. For example, the position operator X tells us the particle's position, the momentum operator P tells us the particle's momentum, and the Hamiltonian operator H tells us its total energy.

2 Applying Quantum Concepts to Machine Learning

This approach incorporates core quantum mechanics concepts as the algorithm's foundation.

For the traditional linear regression setup, we have:

The cost function:

$$J(\theta) = \frac{1}{2m}\sum(h_\theta(x) - y)^2$$

And the gradient descent algorithm:

$$\theta := \theta - \alpha\nabla J(\theta)$$

The key problem is that this method might get trapped in local minima, especially with non-convex regularization or complex feature interactions.

This approach extends traditional optimization by leveraging quantum principles: rather than maintaining a single parameter vector θ in the cost function, we preserve a quantum superposition of multiple candidate parameter states. This superposition-based method enables the simultaneous evaluation of potential solutions.

Thus, we maintain N parallel parameter vectors $\{\theta_1, \theta_2, \ldots, \theta_N\}$, where each θ_i evolves independently via gradient descent. This allows simultaneous exploration of multiple regions of the loss landscape.

However, these states could get lost in local minima due to stagnation. This is where the quantum tunneling analogy comes into play. As previously noted, in quantum systems, particles can tunnel through barriers they classically couldn't overcome.

To implement this, we give solutions a chance to jump to new areas of the parameter space by adding random noise. This noise is carefully controlled, with the following characteristics:

```java
double noise = (Math.random() * 2 - 1) * scale;
```

- `Math.random()` generates a random decimal between 0.0 and 1.0.
- `* 2 - 1` transforms this to the interval [-1.0, 1.0].
- `scale` controls the maximum perturbation size of the noise.

This `scale` parameter determines how aggressively we explore new areas.

3 Project Architecture Overview

The entire project is structured into modular components:

3.1 `gradientdescent.java`

This class implements **gradient descent optimization** with two key approaches: **Batch Gradient Descent**, the traditional method where all training examples are used in each iteration to compute the gradient and update parameters. Parameters may be updated asynchronously or in a state-based manner.

3.2 `QuantumEnhancedTrainer.java`

This class manages a collection of independent parameter vectors, each updating its state based on its own gradient. When a vector gets stuck (meaning its cost no longer improves), it undergoes a "quantum tunneling" process: controlled noise is injected to help it jump out of local minima.

3.3 `MemoryBank.java` and `MemoryQuantumState.java`

This class stores a limited number of the **most successful parameter vectors** discovered so far. It assigns each a probabilistic weight, or **amplitude**, calculated as:

$$\frac{1}{\text{cost}}$$

This ensures that vectors with a lower cost have a higher amplitude and are more likely to be selected for future operations.

3.4 `prediction.java`

Implements the prediction function:

$$f_{w,b}(x) = w \cdot x + b$$

3.5 `costcalculator.java`

Computes the cost function $J(\theta)$, using the Mean Squared Error (MSE).