

Unnormalized Vector; Kuantum mekaniksel sistemin durumunu tanımlayan ve normu (uzunluğu) 1'e eşit olmayan bir vektör.

Normalized Vector; $\text{norm} = 1$

Vektör nasıl Normalize edilir?

Bir vektörü normalize etmek için, vektörün kendisini normuna bölmemiz gereklidir. Vektörün normu, vektörün bileşenlerinin karelerinin toplamının karekökü olarak tanımlanır.

Matematiksel olarak, bir vektör (ψ) normalize edilmek isteniyorsa, normalize edilmiş vektör (ϕ) şu şekilde elde edilir:

$$\phi = \frac{\psi}{\|\psi\|}$$

Burada ($\|\psi\|$), vektör (ψ)'nin normudur ve şu şekilde hesaplanır:

$$\|\psi\| = \sqrt{\langle \psi | \psi \rangle}$$

Örneğin, ($\psi = \begin{pmatrix} a \\ b \end{pmatrix}$) gibi iki boyutlu bir vektör için norm:

$$\|\psi\| = \sqrt{a^*a + b^*b}$$

Bu normu kullanarak, normalize edilmiş vektör:

$$\phi = \begin{pmatrix} a \\ b \\ \|\psi\| \end{pmatrix}$$



Bu normu kullanarak, normalize edilmiş vektör:

$$\phi = \begin{pmatrix} \frac{a}{\|\psi\|} \\ \frac{b}{\|\psi\|} \\ 1 \end{pmatrix}$$

şeklinde hesaplanır.

Örnek

Örneğin, iki boyutlu bir kuantum durumu vektörü ($\psi = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$) olsun. Bu vektörün normu:

$$\|\psi\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Normalize edilmiş vektör ise:

$$\phi = \begin{pmatrix} \frac{3}{5} \\ \frac{4}{5} \\ 1 \end{pmatrix}$$

Numpy Array → matris çarpımı, transpozisyon, ters alma (için kullanılır).

Kuantum mekaniginde inner product alıken, vektörlerin karmaşık eslenimlerini alırız. Bu
bra ve ket kullanımına yaparız → Bra vektörü ket vektörünün Hermitian eslenigidir.
(Transpoz ve karmaşık esleniği)

Temel vektorlerin (basic states) inner product özellikleri:

$$\langle 0|0 \rangle = 1, \quad \langle 1|1 \rangle = 1, \quad \langle 0|1 \rangle = 0, \quad \langle 1|0 \rangle = 0.$$

$$2+i = (2-i)^*$$

\checkmark eslenig 1. → karmaşık sayılarında / gerçek sayıların esleniği konsidir.

Inner product = 0 → states are orthogonal

= 1 → they are the same states and state is normalized.

np.sqrt() → fonksiyon bir vektörün iç çarpımını hesaplar. Bu fonksiyon karmaşık
eslenimleri de dikkate alır.

Orthonormal két vektorlerin türümüne;

$$ket_0 = np.array([1, 0])$$

$$ket_1 = np.array([0, 1])$$

np.random.choice → numpy Library, Belirli bir aralıktan rastgele elemanlar seçmek
için kullanılır. Ör;

np.random.choice(a, size=None, replace=True, p=None)

a → Seçim yapılacak dizi veya aralıkta o tam sayı ISC. O'dan a^1 'ye kadar olan tam sayılar
arasından seçim yapılır.

size → Seçilenin eleman sayısı (belirtilmese 1 tır eleman sevdir).

replace → True: elemanlar tekrar edebilir, False: edemez.

p → her elemanın seçilme olasılığının belirten özlük dizisi. (Olasılıkların toplamı 1 olmalıdır.)

Örnek 2: Belirli Bir Diziden Birden Fazla Rastgele Eleman Seçimi

Örnek 1: Belirli Bir Diziden Rastgele Eleman Seçimi

Bir diziden rastgele bir eleman seçmek:

```
Python Kodu kopyala
import numpy as np
arr = [1, 2, 3, 4, 5]
chosen_element = np.random.choice(arr)
print(chosen_element)
```

Bu kod, arr dizisinden rastgele bir eleman seçecektir.

Bir diziden belirli sayıda rastgele eleman seçmek:

```
Python Kodu kopyala
import numpy as np
arr = [1, 2, 3, 4, 5]
chosen_elements = np.random.choice(arr, size=3)
print(chosen_elements)
```

Bu kod, arr dizisinden rastgele 3 eleman seçecektir.

Örnek 3: Elemanların Tekrar Etmediği Seçim

Bir diziden tekrar etmeyen rastgele elemanlar seçmek:

```
Python Kodu kopyala
import numpy as np
arr = [1, 2, 3, 4, 5]
chosen_elements =
np.random.choice(arr, size=3,
replace=False)
print(chosen_elements)
```

Bu kod, arr dizisinden rastgele ve tekrar etmeyen 3 eleman seçecektir.

Örnek 4: Belirli Olasılıklarla Rastgele Eleman Seçimi

Elemanların belirli olasılıklarla seçildiği bir örnek:

```
Python Kodu kopyala
import numpy as np
arr = [1, 2, 3, 4, 5]
probabilities = [0.1, 0.2, 0.3,
0.2, 0.2]
chosen_elements =
np.random.choice(arr, size=3,
p=probabilities)
print(chosen_elements)
```

Bu kod, arr dizisinden verilen olasılıklarla (probabilities) rastgele 3 eleman seçecektir. Olasılıkların toplamının 1 olması gerektiğini unutmayın.

$\text{np.abs}()$ → bir sayının veya bir dizinin
elemanlarının mutlak değerlerini
hesaplamak için kullanılır.
fakat genel bir karmaşık sayıda
sayıları için geçerlidir.

Karmaşık Sayılar İçin Kullanım

Karmaşık sayılarından oluşan bir dizide
 np.abs fonksiyonunu kullanmak:

Python

Kodu kopyala

```
import numpy as np

complex_arr = np.array([1+2j,
3-4j, -1+1j])
absolute_values =
np.abs(complex_arr)
print(absolute_values)
```

Çıktı:

Kodu kopyala

[2.23606798 5. 1.41421356]

Gerçek Sayılar İçin Kullanım

Gerçek sayılarından oluşan bir dizide
 np.abs fonksiyonunu kullanmak:

Python

Kodu kopyala

```
import numpy as np
```

```
arr = np.array([-1, -2, 3, -4])
absolute_values = np.abs(arr)
print(absolute_values)
```

Çıktı:

Kodu kopyala

Kuantum Durumu Ölçümünde np.abs Kullanımı

Kuantum hesaplamada, np.abs fonksiyonu, bir kuantum durum vektörünün genliklerinin büyüklüğünü (modülüsünü) hesaplamak için kullanılır. Ölçüm olasılıkları, bu büyüklüklerin kareleri olarak tanımlanır. Bu nedenle, karmaşık sayıların büyüklüğünü hesaplamak için np.abs kullanılır.

Örnek

Bir kuantum durum vektörünü ele alalım:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Bu durumda, (α) ve (β) genlikleri karmaşık sayılar olabilir. Ölçüm olasılıklarını hesaplamak için bu genliklerin modüllerinin karelerini kullanırız:

```
import numpy as np

def measure_state(state, num_meas):
    """Simulate a quantum measurement process.

    Args:
        state (np.array[complex]): A normalized qubit
        state vector.
        num_meas (int): The number of measurements to
        take.

    Returns:
        np.array[int]: A set of num_meas samples, 0 or
        1, chosen according to the probability
        distribution defined by the input state.
    """
    # Ölçüm sonuçlarının olasılıklarını hesapla
    prob_0 = np.abs(state[0])**2 # |\alpha|^2
    prob_1 = np.abs(state[1])**2 # |\beta|^2

    # Olasılıkları kullanarak ölçüm sonuçlarını seç
    samples = np.random.choice([0, 1], size=num_meas,
    p=[prob_0, prob_1])

    return samples

# Örnek kullanım
state = np.array([1/np.sqrt(2), 1/np.sqrt(2)]) # Eşit
olasılıklı |0⟩ ve |1⟩ süperpozisyonu
num_meas = 1000
measurement_results = measure_state(state, num_meas)
print(f"Measurements: {func çağırıldı fonksiyonun döndürdüğü:}\n{np.bincount(measurement_results)}")
```

↓ bir tam sayı dizisindeki her benzerin sayısını içeren bir dizi. (Histogram oluşturmak için kullanılır.)

```
np.bincount(x, weights=None, minlength=0)
```

Parametreler

- **x**: Sayılacak tam sayıların bulunduğu 1 boyutlu bir dizi.
- **weights**: Ağırlıklar için bir dizi. Ağırlıklı sayma işlemi yapmak için kullanılır. Varsayılan olarak None'dır.
- **minlength**: Döndürülen dizinin en küçük uzunluğu. Bu, çıktı dizisinin en azından bu uzunlukta olmasını sağlar. Varsayılan olarak 0'dır.

Ağırlıklı Sayma

Ağırlıklı sayma işlemi yapmak:

Python

Kodu kopyala

```
import numpy as np

arr = np.array([0, 1, 1, 2, 3, 3, 3,
                4])
weights = np.array([0.5, 0.2, 0.2,
                    1.0, 0.5, 0.5, 0.5, 0.5])
counts = np.bincount(arr,
                      weights=weights)
print(counts)
```

Çıktı:

Kodu kopyala

```
[0.5 0.4 1. 1.5 0.5]
```

Basit Kullanım

Bir dizideki her elemanın sayısını hesaplamak:

Python

Kodu kopyala

```
import numpy as np
```

```
arr = np.array([0, 1, 1, 2, 3, 3, 3, 4])
counts = np.bincount(arr)
print(counts)
```

Kuantum Hesaplama Kullanım

Kuantum ölçümleri simüle ederken np.bincount fonksiyonu, her ölçüm sonucunun kaç kez elde edildiğini saymak için kullanılabilir. Bu, ölçüm sonuçlarını analiz etmede oldukça yararlıdır.

Örneğin, daha önce verdığımız measure_state fonksiyonu ile birlikte np.bincount fonksiyonunu kullanarak kuantum ölçüm sonuçlarının dağılımını görebiliriz:

Manipulation of Qubit States

What sends a 2-dimensional vector to another 2-dimensional vector? Multiplication by a 2×2 Matrix, U

$$|\psi\rangle = U|\psi'\rangle$$

U is not any matrix \rightarrow The matrix must preserve the normalization of the state. Even after an operation, the measurement probabilities must sum 1 $\rightarrow |\alpha'|^2 + |\beta'|^2 = 1$.

There is a special class of matrices that preserves the length of quantum states; Unitary matrices $U^\dagger U = I$ \rightarrow indicates the taking complex conjugate of all elements in the transpose of U and I is the 2×2 matrix

np.dot(U, state) \rightarrow U matrisini input state vektörüne uyguler.

Use the functions below to simulate a quantum algorithm that does the following:

1. Initialize a qubit in state $|0\rangle$
2. Apply the provided operation U
3. Simulate measuring the output state 100 times

You'll have to complete a function for initialization, but we've provided functions for the other two.

```
import numpy as np

def initialize_state():
    """Prepare a qubit in state |0>.

    Returns:
        np.array[float]: the vector representation of state |0>.
    """
    # Durumu |0> olarak başlat
    return np.array([1, 0])

# Unitary matris U
U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)

def apply_u(state):
    """Apply a quantum operation."""
    return np.dot(U, state)

def measure_state(state, num_meas):
    """Measure a quantum state num_meas times."""
    p_alpha = np.abs(state[0]) ** 2
    p_beta = np.abs(state[1]) ** 2
    meas_outcome = np.random.choice([0, 1], p=[p_alpha, p_beta],
size=num_meas)
    return meas_outcome

def quantum_algorithm():
    """Use the functions above to implement the quantum algorithm described above.

    Try and do so using three lines of code or less!
    """

    Returns:
        np.array[int]: the measurement results after running the algorithm 100 times
    """
    # Durumu başlat, U'yu uygula ve 100 kez ölçüm al
    state = initialize_state()      # Durumu |0> olarak başlat
    new_state = apply_u(state)      # U'yu uygula
    measurements = measure_state(new_state, 100)  # 100 ölçüm al

    return measurements

# Kuantum algoritmasını çalıştır ve sonuçları yazdır
measurement_results = quantum_algorithm()
print(f"Measurement results: {np.bincount(measurement_results)}")
```

4. quantum_algorithm Fonksiyonu:

- Üç adımı bir araya getirir: durumu başlatma, unitary operasyon uygulama ve ölçüm yapma.
- Sonuçta 100 ölçümün sonuçlarını döndürür.

Beklenen Çıktı

Bu kod çalıştırıldığında, kuantum durumu ($|0\rangle$) olarak başlatılır, Hadamard kapısı uygulanır ve ardından 100 ölçüm sonucu alınır. np.bincount kullanılarak her bir ölçüm sonucunun kaç kez elde edildiği sayılır ve yazdırılır.

Plaintext

Kodu kopyala

Measurement results: [52 48]

Açıklamalar

1. initialize_state Fonksiyonu:

- ($|0\rangle$) durumu, $[1, 0]$ vektörü ile temsil edilir.

2. apply_u Fonksiyonu:

- Unitary matris (U), durum vektörüne uygulanarak yeni bir durum vektörü elde edilir.

3. measure_state Fonksiyonu:

- Durum vektöründeki genliklerin kareleri hesaplanarak ölçüm olasılıkları elde edilir ve rastgele ölçümler alınır.

Bu örnek çıktı, ölçümlerin yaklaşık olarak eşit dağıldığını gösterir. Hadamard kapısı, ($|0\rangle$) durumunu eşit olasılıklı bir süperpozisyon durumuna dönüştürdüğünden, bu beklenen bir sonuçtur. Bu sayılar, her çalışmada küçük farklılıklar gösterebilir çünkü ölçüm sonuçları rastgeledir.

You can think of a quantum circuit like a recipe, or set of instructions that tell you what to do each qubit, and when to do it. By placing and performing operations in a certain way, we can realize diff. quantum algorithms.

Operations on qubits are called Gates

Circuit depth → Number of time steps it takes for a circuit to run, if we do things as parallel as possible. Alternatively, you can think of it as the number of layers in a circuit.

Qubit State Vectors properties; Normalized \rightarrow have length 1. Thus, any matrix that operates on qubits is going to require a structure that preserves this property. \rightarrow Unitary Matrix
 $n \times n$ complex-valued matrix U is unitary if;

$$UU^T = U^T U = I_n$$

\downarrow \hookrightarrow *n-dimensional Identity matrix.*

conjugate

transpose or adjoint of U (*take the transpose of the matrix, and replace each entry with its complex conjugate*)

Unitary Parametrization:

Unitary matrisler tanımları gereği
 Özel bir yapıyla sahipler. 2×2 bir unitary matris için 8 reel
 Sayıya gerek olduğumu düşünebilirsin (her entry için bir karmaşık
 Sayı her karmaşık sayı için 2 reel sayı $(a+bi)$)
 Ama unitary matris yapısı gereğ 3 reel sayı yetiyor.

Hermitian transpose

$$U^\dagger = \frac{1}{\det(U)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \begin{pmatrix} a^* & c^* \\ b^* & d^* \end{pmatrix}$$

Unitary matrisin tersi conjugate transpose'ıdır.

$$U^{-1} = U^\dagger$$

1. Transpozunu Alma:

$$U^T = \begin{pmatrix} 1+2i & 3+4i \\ 5+6i & 7+8i \end{pmatrix}$$

2. Karmaşık Eşlenliğini Alma:

$$\overline{U^T} = \begin{pmatrix} 1-2i & 5-6i \\ 3-4i & 7-8i \end{pmatrix} = U^\dagger$$

3. Determinant:

Unitary matrisin determinantı birim çember üzerinde bir karmaşık sayıdır, yani modülü 1 olan bir karmaşık sayı. Bu nedenle, eğer U unitary ise, $\det(U)$ şu şekilde yazılabilir:

$$\det(U) = e^{i\beta} \quad \text{burada } \beta \text{ bir reel sayıdır}$$

```
import pennylane as qml
import numpy as np

dev = qml.device("default.qubit", wires=2) # Specify a device with at
least 2 wires

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)

@qml.qnode(dev)
def apply_u():
    qml.QubitUnitary(U, wires=1) # Apply the unitary operation to
wire 1
    return qml.state()
```

\rightarrow Unitaries
 in
 PennyLane .

Kuantum kapılarının matris temsillerini, bu kapıların özdeğerlerini ve özvektörlerini elde etmek için kullanır. Özdeğerler (eigenvalues) ve özvektörler (eigenvectors) bir operatörün (veya bir matrisin) belirli vektörlerin nasıl etki ettiğini ve bu etki sonucunda vektörlerin nasıl değiştiğini anlamamıza yardımcı olur.

eigenvalues and eigenvectors: Kuantum sistemlerinin enerji seviyelerini ve durumlarını anlamada kritik rol oynar. Bir kuantum sisteminin state'leri genellikle eigenvector'lardır ve bu durumların enerjileri eigenvalue'ları ile temsil edilir.

Dinamiklerin anlamlaması: Bir kuantum sisteminin zaman evrimi, Sistemin Hamiltonian Operatörünün Eigenvalue and eigenvector'ları ile açıklanır. Bu sistemi nasıl durdurduğunu ve belirli zaman dilimlerinde nasıl değiştirdiğini anlamamızı sağlar.

- 3. Kapıların Özellikleri:** Kuantum kapılarının özdeğerleri ve özvektörleri, bu kapıların performansını ve özelliklerini analiz etmede kullanılır. Örneğin, bir kapının üniter olup olmadığını veya belirli simetrlere sahip olup olmadığını anlamamıza yardımcı olur.
- 4. Hesaplama Kolaylığı:** Özdeğerler ve özvektörler, bazı hesaplamaları daha kolay hale getirir. Örneğin, bir operatörün bir özvektöre etkisi, sadece özdeğer ile çarpma işlemiyle ifade edilebilir, bu da hesaplamaları basitleştirir.
- 5. Kuantum Algoritmaları:** Özdeğerler ve özvektörler, birçok kuantum algoritmasının temelini oluşturur. Özellikle, kuantum Fourier dönüşümü ve kuantum faz tahmini gibi algoritmalar, özdeğer ve özvektör kavramlarını kullanır.