Barren plateaus in quantum neural networks
&
An initialization strategy for addressing them

Informazione Quantistica
Federico Magnolfi
Prof. Paola Verrucchi & Filippo Caruso

September 17, 2020

Overview

Outline

We want to

- define a simple **quantum neural network**
- understand the **behavior of gradients**

without solving a specific learning problem.

We'll analyze the gradients:

1. analitically
2. experimentally

Outline

We want to

- define a simple **quantum neural network** $\rightarrow$ using `cirq`
- understand the **behavior of gradients**

without solving a specific learning problem.

We'll analyze the gradients:

1. analitically
2. experimentally

Outline

We want to

- define a simple **quantum neural network** → using `cirq`
- understand the **behavior of gradients** → using `tfq`

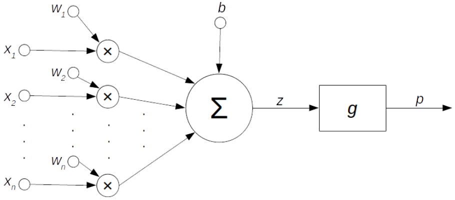
without solving a specific learning problem.

We'll analyze the gradients:

1. analitically
2. experimentally

Classical Neural Networks

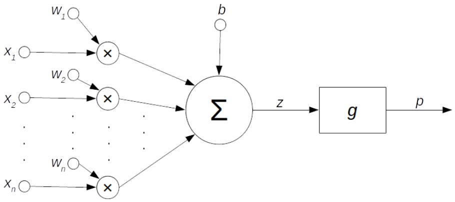
Neuron



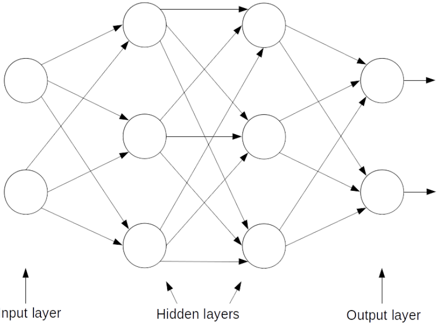
$$f_w(x) = g(w^T x + b)$$

Classical Neural Networks

Neuron



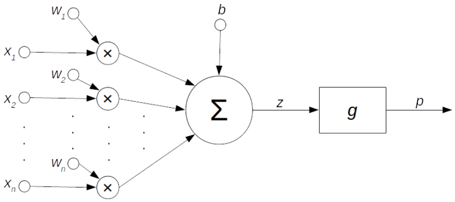
Neural Network



$$f_w(x) = g(w^T x + b)$$

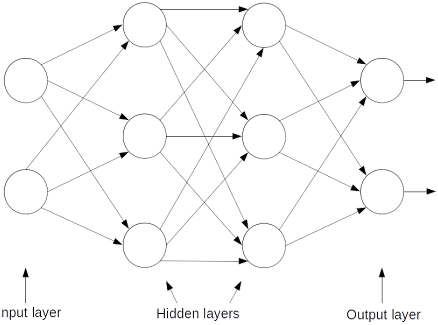
Classical Neural Networks

Neuron



$$f_w(x) = g(w^T x + b)$$

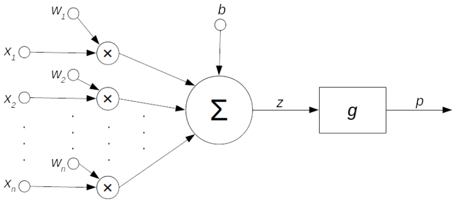
Neural Network



- objective function $L(f_W(x), y)$

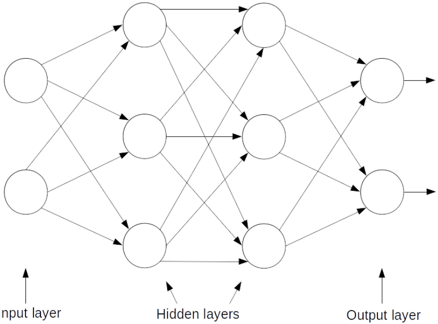
Classical Neural Networks

Neuron



$$f_w(x) = g(w^T x + b)$$

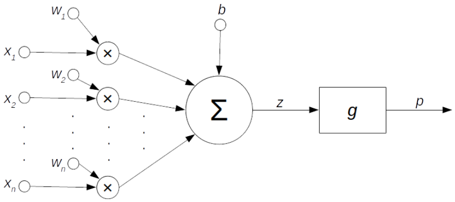
Neural Network



- objective function $L(f_W(x), y)$
- backprogation to calculate $\frac{\partial L}{\partial W}$

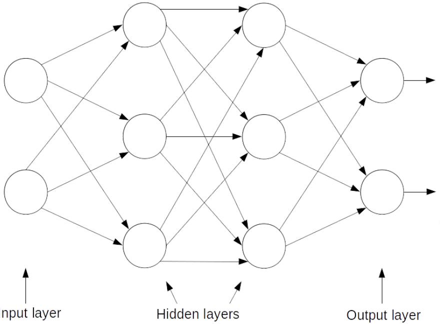
Classical Neural Networks

Neuron



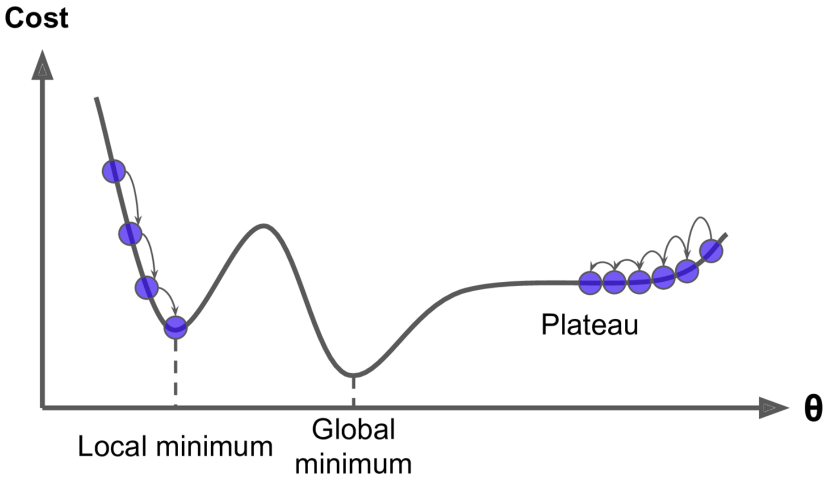
$$f_w(x) = g(w^T x + b)$$

Neural Network

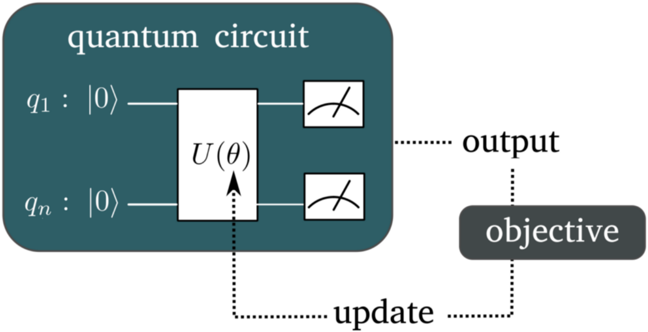


- objective function $L(f_W(x), y)$
- backprogation to calculate $\frac{\partial L}{\partial W}$
- gradient descent to update weigths $W \leftarrow W - l_r \frac{\partial L}{\partial W}$

Gradient descent

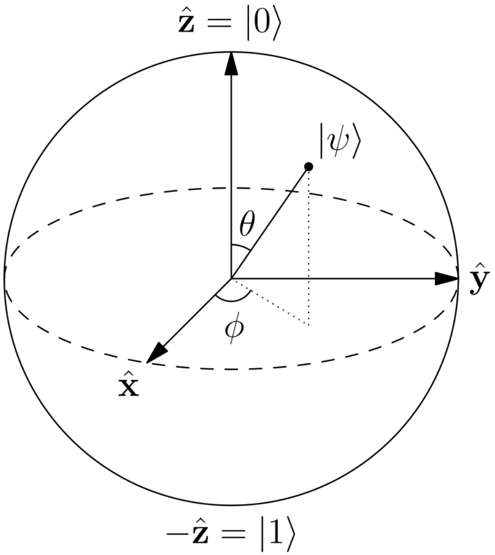


Variational quantum algorithm



We analyze the gradients w/o considering the optimization phase

Qubit: Bloch sphere



Pauli matrices and rotations

Matrices

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Rotations

$$R_X(\theta) = e^{-i\frac{\theta}{2}X} = \begin{bmatrix} cos(\frac{\theta}{2}) & -i\ sin(\frac{\theta}{2}) \\ -i\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}$$

Rotations in `cirq`

`cirq.rx`(θ)

Pauli matrices and rotations

Matrices	Rotations	Rotations in `cirq`
$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$R_X(\theta) = e^{-i\frac{\theta}{2}X} = \begin{bmatrix} cos(\frac{\theta}{2}) & -i\ sin(\frac{\theta}{2}) \\ -i\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}$	`cirq.rx(`θ`)`
$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$R_Y(\theta) = e^{-i\frac{\theta}{2}Y} = \begin{bmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}$	`cirq.ry(`θ`)`
$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$R_Z(\theta) = e^{-i\frac{\theta}{2}Z} = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$	`cirq.rz(`θ`)`

Barren Plateaus

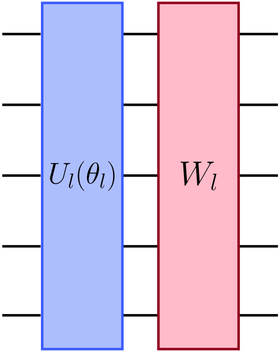
Parametrized quantum circuit

A parametrized quantum circuit can be a sequence of unitaries

$$U(\boldsymbol{\theta}) = U_L(\theta_L)W_L \cdots U_1(\theta_1)W_1 = \prod_{l=L}^{1} U_l(\theta_l)W_l$$

where:

- $U_l(\theta_l) = \exp(-i\theta_l V_l)$
- θ_l: real-valued parameter
- V_l: Hermitian operator
- W_l: fixed unitary



Objective funtion

Objective function as expectation over some Hermitian operator H

$$E(\boldsymbol{\theta}) = \langle 0 | U(\boldsymbol{\theta})^\dagger H U(\boldsymbol{\theta}) | 0 \rangle$$

H represents the observable of interest.

remind: $U_k(\theta_k) = \exp(-i\theta_k V_k)$

Objective funtion

Objective function as expectation over some Hermitian operator H

$$E(\boldsymbol{\theta}) = \langle 0|U(\boldsymbol{\theta})^\dagger H U(\boldsymbol{\theta})|0\rangle$$

H represents the observable of interest.
Assume that $\exists M$ Hermitian s.t.: $\partial_{\theta_k} U(\theta) = iUM$

$$\begin{aligned}\partial_{\theta_k} E(\boldsymbol{\theta}) &= \left\langle 0 \middle| U^\dagger H(\partial_{\theta_k} U) + (\partial_{\theta_k} U)^\dagger H U \middle| 0 \right\rangle \\ &= i \left\langle 0 \middle| U^\dagger H U M - M^\dagger U^\dagger H U \middle| 0 \right\rangle \\ &= i \left\langle 0 \middle| \left[U^\dagger H U, M \right] \middle| 0 \right\rangle\end{aligned}$$

remind: $U_k(\theta_k) = \exp(-i\theta_k V_k)$

Expectation & Variance

The expected value of the gradient is therefore:

$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i\left\langle 0 \middle| \left[\mathbb{E}[U^\dagger H U], M\right] \middle| 0 \right\rangle$$

Expectation & Variance

The expected value of the gradient is therefore:

$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i \left\langle 0 \middle| \left[\mathbb{E}[U^\dagger H U], M\right] \middle| 0 \right\rangle$$

If U is sufficiently random, then:

$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i \frac{\mathrm{Tr}(H)}{2^n} \left\langle 0 \left| [I, M] \right| 0 \right\rangle = 0$$

where:

- n: number of qubits

Expectation & Variance

The expected value of the gradient is therefore:

$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i\left\langle 0 \middle| \left[\mathbb{E}[U^\dagger H U], M\right] \middle| 0\right\rangle$$

If U is sufficiently random, then:

$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i\frac{\mathrm{Tr}(H)}{2^n}\left\langle 0 \left| [I, M] \right| 0\right\rangle = 0$$

$$\mathrm{Var}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = 2\frac{(M^2)_{00} - (M_{00})^2}{2^{2n}-1}\left(\mathrm{Tr}\left(H^2\right) - \frac{\mathrm{Tr}\left(H\right)^2}{2^n}\right)$$

where:

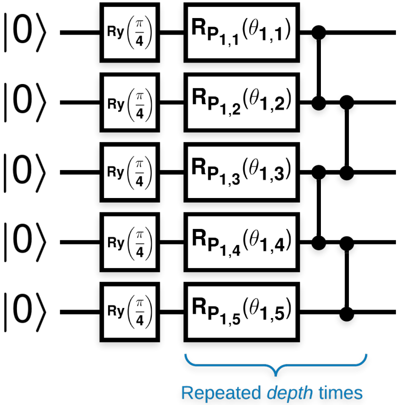
- n: number of qubits
- $M_{00} \triangleq \langle 0| M |0\rangle$
- $(M^2)_{00} \triangleq \langle 0| M^2 |0\rangle$

Parametrized quantum circuit: simplification

General formulation

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l$$

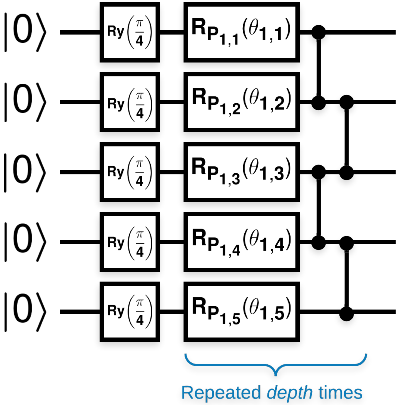
Parametrized quantum circuit: simplification



General formulation

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l$$

Parametrized quantum circuit: simplification

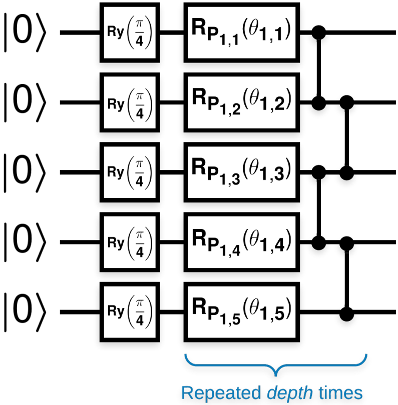


General formulation

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l$$

- $R_Y(\frac{\pi}{4})$ to avoid trivial rotations

Parametrized quantum circuit: simplification



General formulation

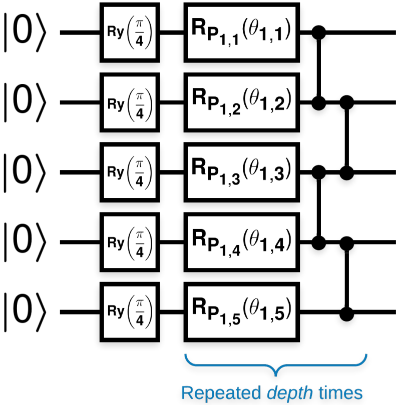
$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l$$

- $R_Y(\frac{\pi}{4})$ to avoid trivial rotations
- Pauli rotations

$$P_{j,k} \in \{X, Y, Z\}$$

$$\theta_{j,k} \in \mathbb{R}$$

Parametrized quantum circuit: simplification



General formulation

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l$$

- $R_Y(\frac{\pi}{4})$ to avoid trivial rotations
- Pauli rotations

$$P_{j,k} \in \{X, Y, Z\}$$

$$\theta_{j,k} \in \mathbb{R}$$

- controlled-Z

Parametrized quantum circuit in cirq

```
def generate_random_qnn(qubits, symbol, depth):
    circuit = cirq.Circuit()
    for qubit in qubits:
        circuit += cirq.ry(np.pi / 4.0)(qubit)

```

Parametrized quantum circuit in cirq

```
def generate_random_qnn(qubits, symbol, depth):
    circuit = cirq.Circuit()
    for qubit in qubits:
        circuit += cirq.ry(np.pi / 4.0)(qubit)
    for d in range(depth):
        for i, qubit in enumerate(qubits):
            n = np.random.uniform()
            θ=unif()*2*pi if i!=0 or d!=0 else symbol
            if n > 2/3: # add Rz
                circuit += cirq.rz(θ)(qubit)
            elif n > 1/3: # add Ry
                circuit += cirq.ry(θ)(qubit)
            else: # add Rx
                circuit += cirq.rx(θ)(qubit)

```

Parametrized quantum circuit in `cirq`

```
def generate_random_qnn(qubits, symbol, depth):
    circuit = cirq.Circuit()
    for qubit in qubits:
        circuit += cirq.ry(np.pi / 4.0)(qubit)
    for d in range(depth):
        for i, qubit in enumerate(qubits):
            n = np.random.uniform()
            θ=unif()*2*pi if i!=0 or d!=0 else symbol
            if n > 2/3: # add Rz
                circuit += cirq.rz(θ)(qubit)
            elif n > 1/3: # add Ry
                circuit += cirq.ry(θ)(qubit)
            else: # add Rx
                circuit += cirq.rx(θ)(qubit)
        # add CZ ladder
        for ctrl, target in zip(qubits, qubits[1:]):
            circuit += cirq.CZ(ctrl, target)
    return circuit
```

Experiment many random circuits

```
# create qubits
qubits = cirq.GridQubit.rect(1, num_qubits)

```

Experiment many random circuits

```
# create qubits
qubits = cirq.GridQubit.rect(1, num_qubits)

# create symbolic variable of which calculate gradient
symbol = sympy.Symbol('theta')

```

Experiment many random circuits

```
# create qubits
qubits = cirq.GridQubit.rect(1, num_qubits)

# create symbolic variable of which calculate gradient
symbol = sympy.Symbol('theta')

# create batch of random circuits
circuits = [generate_random_qnn(qubits, symbol, depth)
    for _ in range(n_circuits)]

```

Experiment many random circuits

```python
# create qubits
qubits = cirq.GridQubit.rect(1, num_qubits)

# create symbolic variable of which calculate gradient
symbol = sympy.Symbol('theta')

# create batch of random circuits
circuits = [generate_random_qnn(qubits, symbol, depth)
    for _ in range(n_circuits)]

# define operator H to use in the objective function
H = cirq.Z(qubits[0]) * cirq.Z(qubits[1])

```

Experiment many random circuits

```
# create qubits
qubits = cirq.GridQubit.rect(1, num_qubits)

# create symbolic variable of which calculate gradient
symbol = sympy.Symbol('theta')

# create batch of random circuits
circuits = [generate_random_qnn(qubits, symbol, depth)
     for _ in range(n_circuits)]

# define operator H to use in the objective function
H = cirq.Z(qubits[0]) * cirq.Z(qubits[1])

# get mean and variance of gradients across batch
mean, var = process_batch(circuits, symbol, H)
```

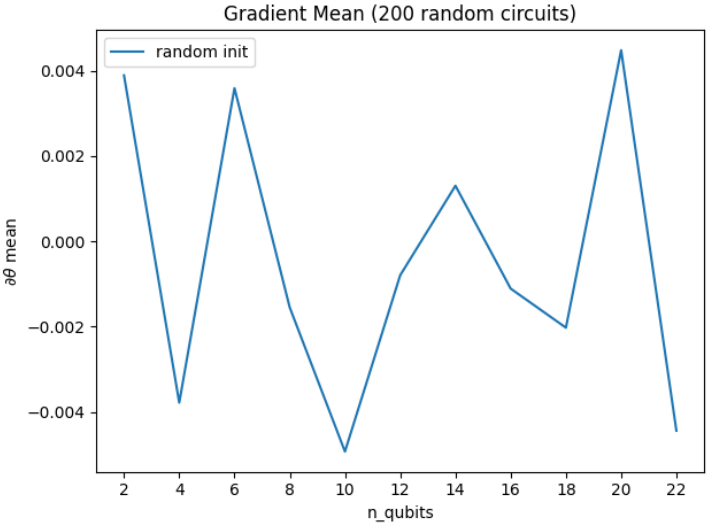
Calculate gradients with tfq

```
def process_batch(circuits, symbol, H):
    expectation = tfq.layers.Expectation()
    circuits = tfq.convert_to_tensor(circuits)
    values = tf.random.uniform(
        [len(circuits),1], 0, 2*np.pi) # theta values

```

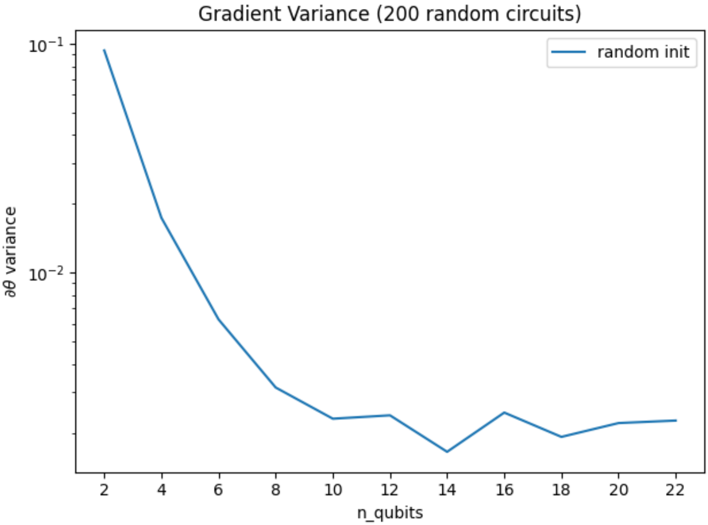
Calculate gradients with tfq

```
def process_batch(circuits, symbol, H):
    expectation = tfq.layers.Expectation()
    circuits = tfq.convert_to_tensor(circuits)
    values = tf.random.uniform(
      [len(circuits),1], 0, 2*np.pi) # theta values

    # automatic differentiation
    with tf.GradientTape() as tape:
        tape.watch(values)
        output = expectation(circuits, operators=H,
                    symbol_names=[symbol],
                    symbol_values=values)

    gradients = tape.gradient(output, values)
    return np.mean(gradients), np.var(gradients)
```

Mean


Gradient Mean (200 random circuits)
random init
0.004
0.002
0.000
−0.002
−0.004
∂θ mean
2
4
6
8
10
12
14
16
18
20
22
n_qubits

Variance

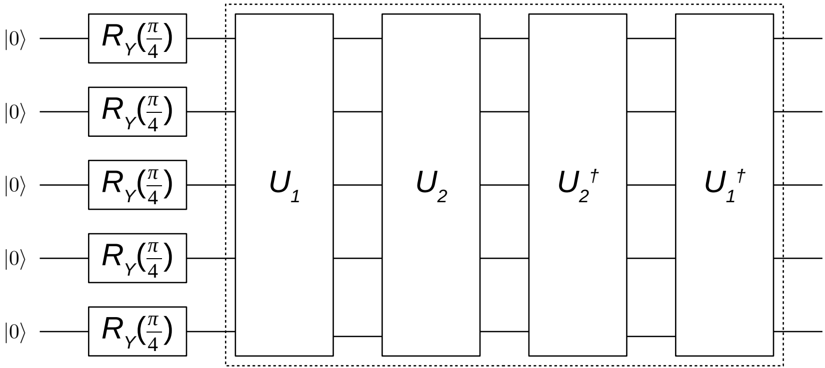

Gradient Variance (200 random circuits)
random init
10^{-1}
10^{-2}
∂θ variance
2
4
6
8
10
12
14
16
18
20
22
n_qubits

Identity Heuristic

Sequence of identity blocks

Idea

Initialize the circuit as a sequence of identity blocks



Identity initialization: single block

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_{l,1}) \prod_{l=1}^{L} U_l(\theta_{l,2})$$

Initialization

- $\theta_{l,1}$: random
- $\theta_{l,2}$: s.t. $U_l(\theta_{l,2}) = U_l(\theta_{l,1})^\dagger$

$$U(\boldsymbol{\theta}^{init}) = \prod_{l=L}^{1} U_l(\theta_{l,1}) \prod_{l=1}^{L} U_l(\theta_{l,1})^\dagger = I$$

Identity initialization: complete circuit

$$U(\boldsymbol{\theta}^{init}) = \prod_{b=B}^{1} U_b(\boldsymbol{\theta}_b) = \prod_{b=B}^{1} I = I$$

Remind

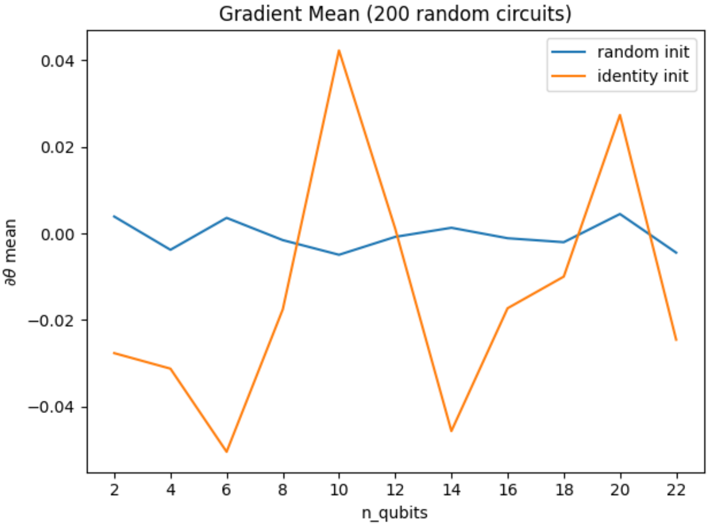
$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta})\right] = i \left\langle 0 \left| \left[\mathbb{E}[U^\dagger H U], M \right] \right| 0 \right\rangle$$

$U(\boldsymbol{\theta}^{init}) = I$

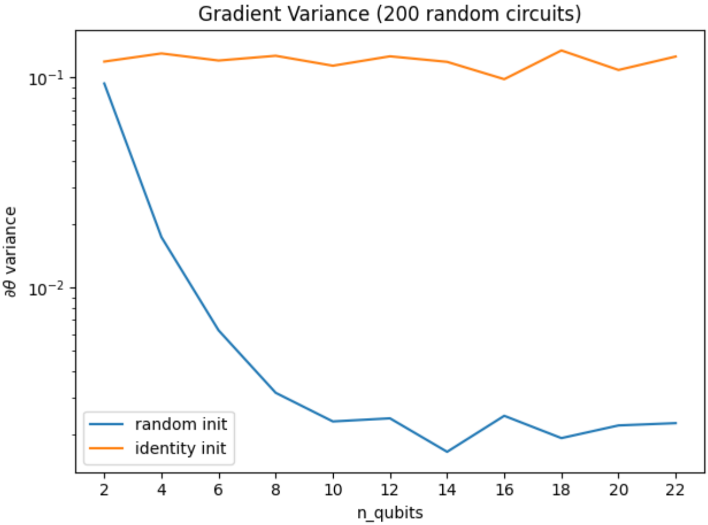
$$\mathbb{E}\left[\partial_{\theta_k} E(\boldsymbol{\theta}^{\boldsymbol{init}})\right] = i \left\langle 0 \left| [H, M] \right| 0 \right\rangle$$

Expectation is 0 only if H and M commute.

Mean


Gradient Mean (200 random circuits)
random init
identity init
0.04
0.02
0.00
−0.02
−0.04
∂θ mean
2
4
6
8
10
12
14
16
18
20
22
n_qubits

Variance


Gradient Variance (200 random circuits)
∂θ variance
10^{-1}
10^{-2}
2 4 6 8 10 12 14 16 18 20 22
n_qubits
random init
identity init

Conclusions

Conclusions

Caution

Quantum Neural Networks initialization can't be fully random

Possible solution

Identity Heuristic is a good starting point

Thanks for the attention

Exponentiation

If operator A satisfies $A^2 = I$:

$$e^{i\theta A} = cos(\theta)I + i\ sin(\theta)A$$

Usefulness of M

$\exists M$ Hermitian s.t.: $\partial_{\theta_k} U(\theta) = iUM$

$$U(\boldsymbol{\theta}) = \prod_{l=L}^{1} U_l(\theta_l) W_l = U_+ U_k(\theta_k) W_k U_-$$

where

$$U_+ \equiv \prod_{l=L}^{k+1} U_l(\theta_l) W_l \qquad U_- \equiv \prod_{l=k-1}^{1} U_l(\theta_l) W_l$$

$$\partial_{\theta_k} U(\boldsymbol{\theta}) = U_+ (\partial_{\theta_k} U_k(\theta_k)) W_k U_- = -i \; U_+ U_k(\theta_k) V_k W_k U_-$$

---

remark: $U_k(\theta_k) = \exp(-i\theta_k V_k)$

cirq overview

```
import cirq

circuit = cirq.Circuit()
q0 = cirq.NamedQubit("qubit0")
circuit += cirq.H(q0) # Bell State sqrt(2)*(|00>+|11>)
```

cirq overview

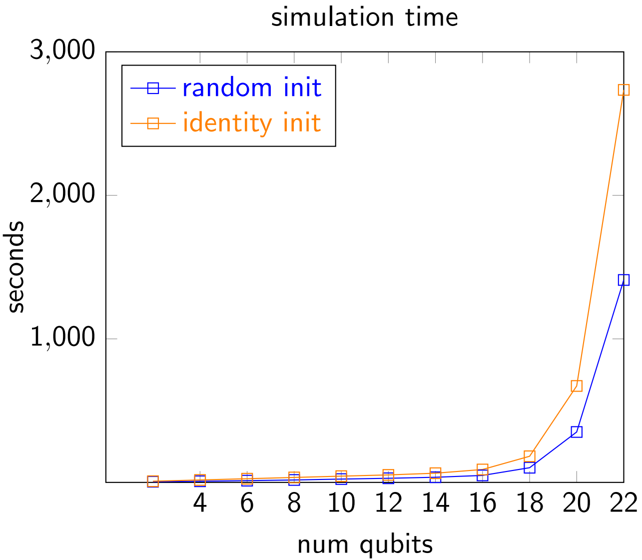
```
import cirq

circuit = cirq.Circuit()
q0 = cirq.NamedQubit("qubit0")
circuit += cirq.H(q0) # Bell State sqrt(2)*(|00>+|11>)

s = cirq.Simulator() # initialize Simulator

print('Simulate the circuit:\n' + str(circuit))
results = s.simulate(circuit)
print(results, "\n")

```

cirq overview

```python
import cirq

circuit = cirq.Circuit()
q0 = cirq.NamedQubit("qubit0")
circuit += cirq.H(q0) # Bell State sqrt(2)*(|00>+|11>)

s = cirq.Simulator() # initialize Simulator

print('Simulate the circuit:\n' + str(circuit))
results = s.simulate(circuit)
print(results, "\n")

print('Sample the circuit:') # measurement needed
circuit.append(cirq.measure(q0, key='result'))
print(circuit)
samples = s.run(circuit, repetitions=10)
print(samples)
# print results histogram
print(samples.histogram(key='result'))
```

cirq overview: output

```
Simulate the circuit:
qubit0: ───H───
measurements:  (no measurements)
output vector:  0.707|0⟩ + 0.707|1⟩

Sample the circuit:
qubit0: ───H───M('result')───
result=1101101000
Counter({1: 5, 0: 5})
```


simulation time
3,000
2,000
1,000
seconds
random init
identity init
4
6
8
10
12
14
16
18
20
22
num qubits