Ciência de Dados Quântica 2021/22

Variational Quantum Classification: a practical algorithm

LUÍS PAULO SANTOS

ANDRÉ SEQUEIRA

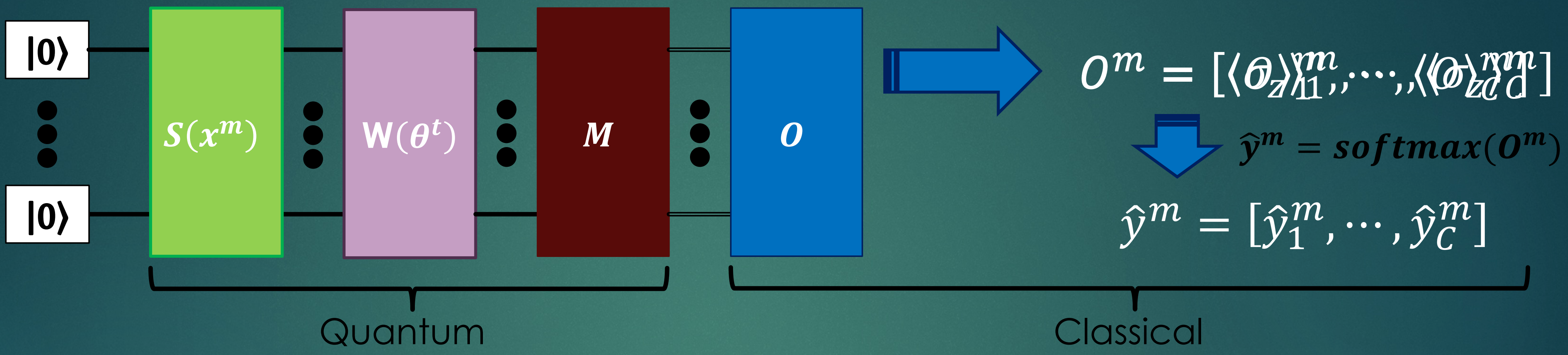
Material de Consulta

- [Schuld2021] – Chap. 5
- [Abbas2021] – 2021 Qiskit Global Summer School on Quantum Machine Learning:
  Building a Quantum Classifier
  https://learn.qiskit.org/summer-school/2021/lec5-1-building-quantum-classifier
- [Pennylane] – Parameter shift rules
  https://pennylane.ai/qml/glossary/parameter_shift.html

Notation and Setup

- Training set with M data points: $\mathcal{D} = [(x^1, y^1), \cdots, (x^M, y^M)]$
- Each point x^m is an N features column vector: $x^m = (x_1^m, \cdots, x_N^m)^T$
- Each y^m identifies which class (out of C classes) x^m belongs to. One-hot encoding is used, so y^m is a column vector: $y^m = (0, \cdots, 0,1,0, \cdots 0)^T$ with C elements and a single element equal to 1 (all others are 0)
- $\hat{y}^m$ is the estimate for x^m. It is a C elements column vector, indicating the probability of x^m belonging to each class c, with $c = 1 \ldots C$

Notation and Setup



- θ^t is the t^{th} iteration column vector of K parameters: $\theta^t = (\theta_1^t, \cdots, \theta_K^t)^T$
- O^m is the list of the expectation values for C observables, with input x^m
- For this example the observable for each class is a measurement of a single qubit in the computational basis: $O^m = [\langle\sigma_z\rangle_1^m, \cdots, \langle\sigma_z\rangle_C^m]$

Main Loop

$t = 1$

θ^t = random (K parameters)

while not termination criterion

    $\hat{Y}_\theta$ = y_hat $(\mathcal{D}, \theta^t)$

    $C(\theta^t)$ = cost $(\mathcal{D}, \hat{Y}_\theta)$

    $\vec{\nabla}_\theta C(\theta^t)$ = gradient $(\mathcal{D}, \hat{Y}_\theta, \theta^t)$

    $\theta^{t+1} = \theta^t - \eta \vec{\nabla}_\theta C(\theta^t)$

    $t = t+1$

- $\hat{Y}_\theta$ is a list of estimates for the current θ^t and each $x^m, m = 1 \cdots M$
- $C(\theta^t)$ is a scalar representing the cost of the current parameterization
- $\vec{\nabla}_\theta\, C(\theta^t)$ is the gradients column vector, with K elements one per parameter
- η is the learning rate

Computing the estimates: $\hat{Y}_\theta$

- $\hat{Y}_\theta$ is a list of M (one per x^m) $\hat{y}_\theta^m$ probability distributions over the C classes
- $\hat{y}_\theta^m = [\hat{y}_1^m, \cdots, \hat{y}_C^m]^T$

```
y_hat (𝒟,θ):
    Y_HAT = []
    for p_m in 𝒟 :
        x_m = p_m[0]
        qc = build_qc (x_m, θ)
        # compute ⟨σ_z⟩^m for all C classes
        expect_z_m = []
        for c in range (C):
            expect_z_m.append (sigma_z_expectation (qc, c, S))
        Y_HAT.append (softmax(expect_z_m))
    return Y_HAT
```

Computing the estimates: $\hat{Y}_\theta$

- We are required to estimate the expectation values of σ_z measurements: one per data point x^m and class c
- Given an observable O and state $|\psi\rangle$ we have:
  - $\langle O\rangle_\psi = \langle\psi\,|A|\psi\rangle = \sum_j \mu_j \left|\langle\psi|\mu_j\rangle\right|^2$, where μ_j are the eigenvalues and $\left|\mu_j\right\rangle$ the eigenvectors
- σ_z has $\mu_0 = 1$, $\left|\mu_0\right\rangle = |0\rangle$ and $\mu_1 = -1$, $\left|\mu_1\right\rangle = |1\rangle$: $\langle\sigma_z\rangle_\psi = p_\psi(|0\rangle) - p_\psi(|1\rangle)$

```
sigma_z_expectation (qc, c, S):
    probs = [0] * 2
    for s in range(S):
        measure = measure_and_execute (qc, c)
        probs[measure] += 1
    probs = [p/S for p in probs]
    return probs[0] - probs[1]
```

Computing the estimates: $\hat{Y}_\theta$

- $\hat{y}^m$ is computed using softmax().
- Let $\langle\sigma_z\rangle^m$ be a vector with the expectations for all C classes for x^m:

$$\langle\sigma_z\rangle^m = [\langle\sigma_z\rangle_1^m, \cdots, \langle\sigma_z\rangle_C^m]^T$$

- Then $\hat{y}_c^m = \frac{e^{\langle\sigma_z\rangle_c^m}}{\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc}^m}}$ and $\hat{y}^m = [\hat{y}_1^m, \cdots, \hat{y}_C^m]$

```
softmax (expect_val_z_m):
    y_hat_m = []
    sum = 0
    for e_val_z_m_c in expect_val_z_m:
        sum += exp(e_val_z_m_c)
    for e_val_z_m_c in expect_val_z_m:
        y_hat_m.append ( exp(e_val_z_m_c) / sum)
    return y_hat_m
```

Loss function: cross entropy

- The loss function : measurement of the dissimilarity between the true class y^m and the estimated distribution over the C classes $\hat{y}^m$
- Cross entropy is based on the notion of entropy:
  - entropy: number of bits (if log_2 is used) required to transmit an event e from a probability distribution $p(E)$ – where E is a discrete random variable
  - cross entropy compares 2 distributions (remember that y^m is a one-hot vector):

$$ce(y^m, \hat{y}^m) = -\sum_{c=1}^{C} (y_c^m * \ln(\hat{y}_c^m))$$

```
cross_entropy (y^m , ŷ^m):
    ce = 0
    for y_m_c, y_hat_m_c in zip (y^m , ŷ^m) :
        ce -= y_m_c * log(y_hat_m_c + 1e-25)
    if (abs(ce) < 1e-10): ce = 0
    return ce
```

Cost function

- The cost is the average of cross entropy across all M data points:

$$C(\theta^t) = \frac{1}{M}\sum_{m=1}^{M} \texttt{cross\_entropy}\ (y^m, \hat{y}^m) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \ln(\hat{y}_c^m)$$

```
cost (𝒟, Ŷ_θ) :
    M = len (𝒟)
    sum = 0
    for p_m, ŷ^m in zip (𝒟, Ŷ_θ):
        y^m = p_m[1]
        sum += cross_entropy (y^m , ŷ^m)
return sum / M
```

Gradients

- $\vec{\nabla}_{\theta}$ C(θ) is a vector with the gradients of the cost with respect to each of the K parameters
- $C(\theta) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \ln \hat{y}_{c,\theta}^m = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \boxed{\ln\left(\frac{e^{\langle\sigma z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma z\rangle_{cc,\theta}^m}}\right)}$

depends on θ^t

- $\nabla_{\theta} C(\theta) = \nabla_{\theta}\frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \ln\left(\frac{e^{\langle\sigma z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma z\rangle_{cc,\theta}^m}}\right)$
- $\nabla_{\theta} C(\theta) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \nabla_{\theta}\ln\left(\frac{e^{\langle\sigma z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma z\rangle_{cc,\theta}^m}}\right)$

Gradients

$$\nabla_\theta \mathrm{C}(\theta) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \nabla_\theta \ln\left(\frac{e^{\langle\sigma_z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m}}\right)$$

- For a single data point (x^m, y^m) and category c

$$\nabla_\theta \ln\left(\frac{e^{\langle\sigma_z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m}}\right) = \nabla_\theta \ln\left(e^{\langle\sigma_z\rangle_{c,\theta}^m}\right) - \nabla_\theta \ln\left(\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m}\right) =$$

$$= \nabla_\theta \langle\sigma_z\rangle_{c,\theta}^m - \frac{1}{\sum_{cc\prime=1}^{C} e^{\langle\sigma_z\rangle_{cc\prime,\theta}^m}} \sum_{cc=1}^{C} \nabla_\theta\left(e^{\langle\sigma_z\rangle_{cc,\theta}^m}\right) =$$

$$= \nabla_\theta \langle\sigma_z\rangle_{c,\theta}^m - \frac{1}{\sum_{cc\prime=1}^{C} e^{\langle\sigma_z\rangle_{cc\prime,\theta}^m}} \sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m} \nabla_\theta \langle\sigma_z\rangle_{cc,\theta}^m =$$

$$= \nabla_\theta \langle\sigma_z\rangle_{c,\theta}^m - \sum_{cc=1}^{C} \frac{e^{\langle\sigma_z\rangle_{cc,\theta}^m}}{\sum_{cc\prime=1}^{C} e^{\langle\sigma_z\rangle_{cc\prime,\theta}^m}} \nabla_\theta \langle\sigma_z\rangle_{cc,\theta}^m =$$

$$= \nabla_\theta \langle\sigma_z\rangle_{c,\theta}^m - \sum_{cc=1}^{C} \hat{y}_{cc,\theta}^m * \nabla_\theta \langle\sigma_z\rangle_{cc,\theta}^m$$

Gradients

$$\nabla_\theta \mathrm{C}(\theta) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \nabla_\theta \ln\left(\frac{e^{\langle\sigma_z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m}}\right)$$

- For a single data point (x^m, y^m) and category c

$$\nabla_\theta \ln\left(\frac{e^{\langle\sigma_z\rangle_{c,\theta}^m}}{\sum_{cc=1}^{C} e^{\langle\sigma_z\rangle_{cc,\theta}^m}}\right) = \nabla_\theta\langle\sigma_z\rangle_{c,\theta}^m - \sum_{cc=1}^{C} \hat{y}_{cc,\theta}^m * \nabla_\theta\langle\sigma_z\rangle_{cc,\theta}^m$$

- For M data points

$$\nabla_\theta \mathrm{C}(\theta) = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{C} -y_c^m * \left(\boxed{\nabla_\theta\langle\sigma_z\rangle_{c,\theta}^m} - \sum_{cc=1}^{C} \hat{y}_{cc,\theta}^m * \boxed{\nabla_\theta\langle\sigma_z\rangle_{cc,\theta}^m}\right)$$

Gradients

- Parameter shift rule:
  - for each parameter $\theta_k \in \theta$, data point (x^m, y^m) and category c:

$$\nabla_{\theta_k} \langle \sigma_z \rangle^m_{c,\theta_{\partial k}} = \frac{1}{2}\left[\langle \sigma_z \rangle^m_{c,\theta_{k+\frac{\pi}{2}}} - \langle \sigma_z \rangle^m_{c,\theta_{k-\frac{\pi}{2}}} \right]$$

Gradients

```
gradient (𝒟, Ŷ_θ,θ):
   grad = []
   M = len (𝒟)
   for k in range(K):   # for every parameter
     sum_m = 0
     for p_m, y_hat_m in zip(𝒟, Ŷ_θ ):
        x_m = p_m[0]
        y_m = p_m[1]
        grad_k = grad_k_all_classes (x_m, θ, k) # grad for all classes
        sum_cc = numpy.dot(y_hat_m, grad_k)
        sum_c = numpy.dot(y_m, (grad_k-sum_cc))
        sum_m += sum_c
    grad.append (-sum_m / M)
    return grad
```

Gradients


```
grad_k_all_classes (x_m, θ, k) # gradient θ_k for all C classes
    θ_plus = θ_minus = θ
    θ_plus[k] += π/2
    θ_minus[k] -= π/2
    qc_plus = build_qc (x_m, θ_plus)
    qc_minus = build_qc (x_m, θ_minus)
    grad_k = []
    for c in range (C):     # for all classes
        sigma_z_plus = sigma_z_expectation (qc_plus, c, S)
        sigma_z_minus = sigma_z_expectation (qc_minus, c, S)
        grad_k.append (1/2 * (sigma_z_plus - sigma_z_minus ))
    return grad_k
```