presentation

November 3, 2022

[9]:
```
%%javascript
MathJax.Hub.Config({
    TeX: { equationNumbers: { autoNumber: "AMS" } }
});
```

```
<IPython.core.display.Javascript object>
```

[4]:
```
%%html
<style type='text/css'>
.CodeMirror{
    font-size: 14px;
}

div.output_area pre {
    font-size: 14px;
}
</style>
```

```
<IPython.core.display.HTML object>
```

1 Tensor Networks with Qiskit

1.0.1 Isaac Nunez

1.0.2 William Huggins et al. (2018) *Towards Quantum Machine Learning with Tensor Networks*

1.1 What exactly are Tensors?

In essence, they are generalization of vectors and matrices which helps us represent and manipulate multi-dimensional data.

`<img src="./images/tensor_vector.png" style="height: 50%; width: 5%; float: left"/>`

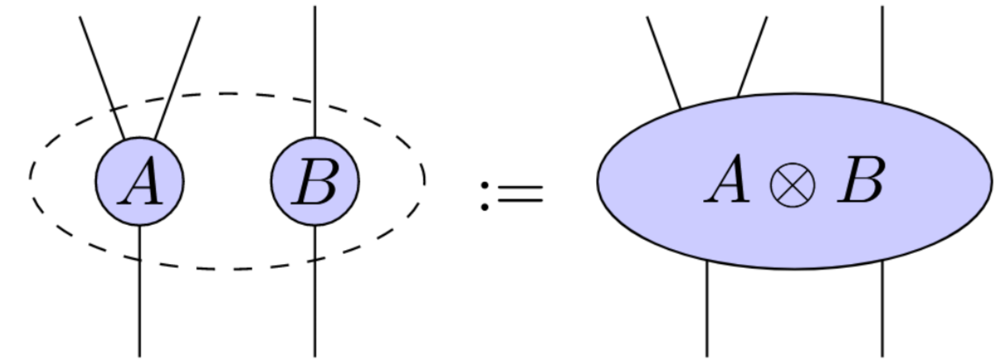
`<img src="./images/tensor_matrix.png" style="height: 40%; width: 15%; float: left"/>`

`<img src="./images/tensor_3d.png" style="height: 40%; width: 15%; float: left"/>`

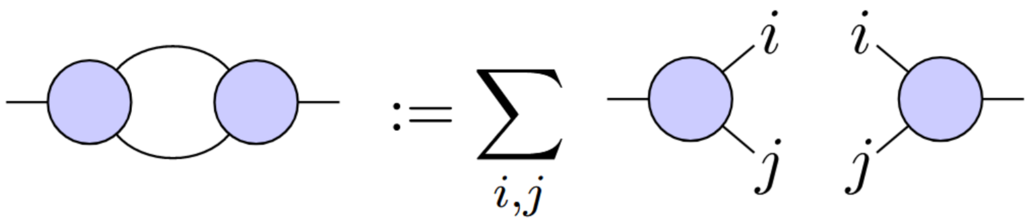
1.2 How do they differ from what we already know?

They introduce certain generalizations to vector and matrix operations.

1. **Tensor products (⊗):** they are a generalization of the outer product of vectors.



2. **Contraction**: it abstracts vector inner products (or matrix-matrix multiplication).



It can also be understood through the *Einstein Notation* where:

$$C_l^k = \sum_{i,j} A_{i,j}^k \cdot B_l^{i,j}$$

1.3 How do they relate to Quantum Computing?

Depending on the context, the shape of the tensor and position of the legs can provide a clue to the properties of the tensor or its indeces.

One example is differencing between $|\phi\rangle$ and $\langle\phi|$ based on the position of the legs.

This will allow to reject certain contractions.

1.4 The bridge between Machine Learning and Quantum Computing

Machine Learning uses Tensor Networks to represent its multi-dimensional data and speedup computation

Quantum Circuits are a special case of Tensor Networks where the arrangement and types are restricted.

Specifically, *Huggins et al.* use Tree Tensor Networks (TNN) and Matrix Product States (MPS) to implement their **discriminative** model

1.5 Towards Quantum Machine Learning with Tensor Networks

In their paper, *Huggins et al.* implement a Tensor Network for binary classification with the MNIST Dataset

They propose two approaches:

- Discriminative
- Generative

The authors select a special case of Tree Tensor Networks: *MPS* for their discriminative model.

`<img src="./images/ttn_model.png" style="height: 100%; float: left; width: 30%"/>`

`<img src="./images/mps_model.png" style="height: 100%; float: right; width: 30%"/>`

1.5.1 Learning on Quantum Circuits

First, we need a way to represent each pixel from the images as a quantum state.

They propose the mapping

$$x \mapsto |\Phi(x)\rangle = \begin{bmatrix} \cos\left(\frac{\pi}{2}x_1\right) \\ \sin\left(\frac{\pi}{2}x_1\right) \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} \cos\left(\frac{\pi}{2}x_N\right) \\ \sin\left(\frac{\pi}{2}x_N\right) \end{bmatrix}$$

They also abstract the concept of layers and implement them as unitary gates

These gates represent the parameters of the model on which the network will be trained for.

1.5.2 How is the model evaluated?

The authors define a **loss function** based on the binary results from the Quantum Circuit

$$\begin{aligned} p_{maxfalse}(\Lambda, x) &= \max_{l \neq l_x}[p_l(\Lambda, x)] \\ L(\Lambda, x) &= \max(p_{maxfalse}(\Lambda, x) - p_{l_x}(\Lambda, x) + \lambda, 0)^\eta \\ L(x) &= \frac{1}{|\text{data}|} \sum_{x \in \text{data}} L(\Lambda, x) \end{aligned} \tag{1}$$

1.5.3 How to minimize the loss function?

The authors chose the **S**imultaneous **P**erturbation **S**tochastic **A**proximation (SPSA) with a modification to include a momentum factor.

Then, the optimization process follows as:

1. Initialize Λ randomly and set v to zero
2. Choose the hyperparameters a, b, A, s, t, γ, n, and M
3. For each $k \in \{0, 1, \dots, M\}$, divide the dataset into random batches of n images and:
    1. Choose $\alpha_k = \dfrac{a}{(k+1+A)^s}$ and $\beta_k = \dfrac{b}{(k+1)^t}$
    2. Generate a perturbation δ.
    3. Evaluate $g = \dfrac{L(\Lambda_k + \alpha_k\delta) - L(\Lambda_k - \alpha_k\delta)}{2 \cdot \alpha_k}$ with $L(x)$ as defined in (1)
    4. $v_{new} = \gamma \cdot v_{old} - g \cdot \beta_k \cdot \delta$
    5. $\Lambda_{new} = \Lambda_{old} + v_{new}$

1.5.4 The networks...

The base model

The efficient model

1.5.5 The results...

The bottom line: They are not great

They claimed above 95% accuracy yet during training, accuracy was not above 55%

Despite the batch size defined by the authors, the models performed better using a smaller batch size.

1.5.6 Shortcomings

- My testing was using the efficient architecture yet the authors only tested the base model.
- The base model has 1008 parameters for an image of 8x8 while the efficient one has 15616.
- The chosen hyperparameters are only for the base model.
- The hyperparameters are outside the range of convergence required by the authors of SPSA.
- The authors did not test the efficient model nor they proposed hyperparameters for it.
- After almost two weeks running the base model, the training accuracy was not higher than 65%
- Smaller batch sizes show a small improvement in a short term.
- The authors did not use Tensorflow, Tensornetwork, and/or Qiskit for their development but rather they own C++ Tensor Library. The code for their paper is not available.

2 Future improvements

- The main author recommends to move from SPSA to the Shift Parameter Rule (SPR) as means to calculate the gradients of a Quantum Circuit.
  - The SPR for Unitary gates is called Stochastic SPR and requires **three** times more unitary gates.
- Many of the examples of SSPR rely on Pytorch for feature extraction and they introduce the Quantum Circuit as another layer on the network. Examples from Qiskit show better accuracy.