Quantum Process Tomography

Atithi Acharya

Guides: Dr. Giacomo Torlai

Dr.Guiseppe Carleo

July 10, 2019

QPT

Measurements

The measurements are taken from the Pauli-4 POVM $\boldsymbol{M}_{\text{Pauli}-4} = \{M^{(0)} = \frac{1}{3} \times |0\rangle\langle 0|\,, M^{(1)} = \frac{1}{3} \times |+\rangle\langle +|\,, M^{(2)} = \frac{1}{3} \times |l\rangle\langle l|\,, M^{(3)} = \frac{1}{3} \times (\,|1\rangle\langle 1| + |-\rangle\langle -| + |r\rangle\langle r|\,)\}$ where $\{|0\rangle\,, |1\rangle\}$, $\{|+\rangle\,, |-\rangle\}$, and $\{|r\rangle\,, |l\rangle\}$ stand for the eigenbases of the Pauli operators σ^z, σ^x, and σ^y, respectively. Pauli-4 can be physically implemented with the same ease as Pauli-6 and Pauli-4 is neither symmetric nor rank-1. Also, apart from the computational advantage of having only $m = 4$ outcomes, its overlap matrix defined as $T_{\boldsymbol{a},\boldsymbol{a}'} = \text{Tr}\left[M^{(\boldsymbol{a})} M^{(\boldsymbol{a}')}\right]$ is invertible.

$$\boldsymbol{T}_{\text{Pauli}-4} = \frac{1}{9}\begin{bmatrix} 1 & 1/2 & 1/2 & 1 \\ 1/2 & 1 & 1/2 & 1 \\ 1/2 & 1/2 & 1 & 1 \\ 1 & 1 & 1 & 6 \end{bmatrix}. \tag{1}$$

Toy Process

The N-qubit measurement is given by $\boldsymbol{M} = \{M^{(a_1)} \otimes M^{(a_2)} \otimes \dots M^{(a_N)}\}_{a_1,\dots a_N}$. To begin with a simpler life we can choose a 2 qubit system thus reducing our Measurement to $\boldsymbol{M_{in}} = \{M^{(a_1)} \otimes M^{(a_2)}\}$. Then we ask the question if we pass the two qubits through an unitary operation (e.g. CNOT gate) , and make another set of measurements say $\boldsymbol{M_{out}} = \{M^{(b_1)} \otimes M^{(b_2)}\}$ with what probability are we going to find $\{\mathbf{b}\}$ as an outcome of the final measurement given we started by getting $\{\mathbf{a}\}$ with the initial measurement where $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$. Since we are finding conditional probabilities by stating that we get $\mathbf{a} = (a_1, a_2)$ to begin with the state is given by , the initial density matrix can be given by $\varrho_{in} = |a_1\rangle\,|a_2\rangle\,\langle a_1|\,\langle a_2|$. After application of the unitary matrix which in this case is a CNOT, the new density matrix is given by $\varrho_u = U_{cx}\varrho_{in}U_{cx}^{\dagger}$ where U_{cx} is the representation of CNOT gate. Before the final measurement, we thus have a density matrix which looks like

$$\varrho_u = U_{cx}\varrho_{in}U_{cx}^{\dagger} = U_{cx}\,|a_1, a_2\rangle\,\langle a_1, a_2|\,U_{cx}^{\dagger} \tag{2}$$

Now after the final set of measurements the probability to find a state in $\mathbf{b} = (b_1, b_2)$ is given by

$$P(\boldsymbol{b}|\boldsymbol{a}) = \text{Tr}\left[M^{(\boldsymbol{b})}\,\varrho_u\right] = \text{Tr}[M^{(b_1)} \otimes M^{(b_2)} U_{cx}\,|a_1, a_2\rangle\,\langle a_1, a_2|\,U_{cx}^{\dagger}] \tag{3}$$

Hence our aim would be to learn this probability $P(b_1, b_2|a_1, a_2)$ using neural networks let's call it $P^{NN}(\boldsymbol{b}|\boldsymbol{a})$.

Density Matrix Reconstruction

For an IC (Informationally Complete) POVM (positive-operator valued measure) we can have have a construction where the overlap matrix is invertible as in the chosen case Pauli-4 . In such a setting , the final density matrix can be reconstructed in the following way.

$$\varrho_u^{\mathbf{a}} = \sum_{\alpha} \chi_u(\alpha/\mathbf{a}) M^{\alpha} \tag{4}$$

$$P(\boldsymbol{b}/\boldsymbol{a}) = \text{Tr}\left[M^{(\boldsymbol{b})}\,\varrho_u\right] \tag{5}$$

Now, in this step assuming T(overlap matrix) is invertible as discussed, we can write

$$\implies \chi_u(\alpha/\mathbf{a}) = \sum_{\mathbf{b}} P(\boldsymbol{b}|\boldsymbol{a}) T^{-1}_{\mathbf{b}\alpha} \tag{6}$$

Thus we can conclude that we can write the density matrix as

$$\varrho^{\mathbf{a}}_u = \sum_{\alpha,\mathbf{b}} P(\boldsymbol{b}|\boldsymbol{a}) T^{-1}_{\mathbf{b}\alpha} M^{\alpha} \tag{7}$$

After we learn the probabilities a neural network we can reconstruct the density matrix by simply writing it as $\varrho^{NN}_u = \sum_{\alpha,\mathbf{b}} P^{NN}(\boldsymbol{b}|\boldsymbol{a}) T^{-1}_{\mathbf{b}\alpha} M^{\alpha}$. The quantum fidelity defined between the two density matrix will also be a criteria to judge the training that we perform via our neural network model.

Recurrent Neural Network Model

Although there might me many advantages of using a recurrent neural network over standard network for sequential data but it can be broadly listed as

- The ability to learn and predict with sequences of varied sizes.
- While the traditional Neural Network doesn't share features across different positions of the sequence , RNN has various ways to address the issue.

Notations:
Let X and Y be the input and output sequence. The length of the input output and sequence is given by T_x and T_y respectively. Depending on T_x and T_y in the RNN architecture it is often classified into various types: One to one , one to many, many to one and many to many. In our case we will be dealing with a many to many architecture with T_x=T_y

Within the many to many architecture of RNN, there are two important models that are studied,

- Language Model: Problem statement for a language model is to find $P(y^{<1>}, y^{<2>}..., y^{<T_y>})$
- Machine Translation: The probelme statement here is $P(y^{<1>}, ..., y^{<T_y>}|x^{<1>}, ..., x^{<T_x>})$

We will build our model to close resemblance to machine translation however without having to deal with cases with $T_x \neq T_y$. A conventional RNN is incapable of having long term

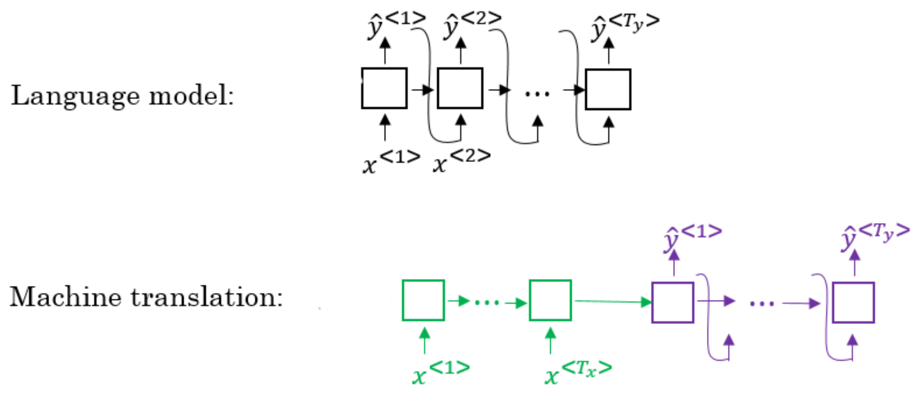


Figure 1: Language model and machine translation with the same input and output lengths in our case

dependencies that is to say if we have

Machine Translation

Although we will not be using Machine Translation in it's original form , but the idea behind using multi-layer LSTM to model the conditional probabilities can be understood by looking at a standard Ecoder-Decoder Model. Let us denote the source sequence also called source phrase by $X = \left(x^{<1>}, ..., x^{<T_x>}\right)$ and a target sequence or target phrase by $Y = \left(y^{<1>}, ..., y^{<T_y>}\right)$.

Each phrase is a sequence of K-dimensional one-hot vectors, such that only one element of the vector is 1 and all the others are 0. The index of the active (1) element indicates the word represented by the vector. For the simple case of 2 qubits, $T_x = T_y = 16$ and $K = 16$ since we are considering Pauli-4 measurements at each qubits and the dimensions in general will go as $K = 2^{2N}$ where N is the number of qubits and thus variables used in the above notation $T_x = T_y = N = 16$. The reason to make the choice is because we want to learn all the $K = 2^{2N}$ probabilities once we finish training.

Encoder

Each term of the source phrase is embedded in a 16-dimensional vector space: $e(x^{<i>}) \in \mathbf{R}^{16}$.

In one of the many architectures used to model, the hidden state of an encoder consists of 64 hidden units, and each one of them at time t is computed by

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + (1 - z_j)\tilde{h}_j^{\langle t\rangle}, \tag{8}$$

where

$$\tilde{h}_j^{\langle t\rangle} = \tanh\left(\left[\mathbf{W}_e(\mathbf{x}^{<t>})\right]_j + \left[\mathbf{U}\left(\mathbf{r} \odot \mathbf{h}_{\langle t-1\rangle}\right)\right]_j\right) \tag{9}$$

$$z_j = \sigma\left(\left[\mathbf{W}_z e(\mathbf{x}^{<t>})\right]_j + \left[\mathbf{U}_z \mathbf{h}_{\langle t-1\rangle}\right]_j\right) \tag{10}$$

$$r_j = \sigma\left(\left[\mathbf{W}_r e(\mathbf{x}^{<t>})\right]_j + \left[\mathbf{U}_r \mathbf{h}_{\langle t-1\rangle}\right]_j\right) \tag{11}$$

σ and $\odot$ are a logistic sigmoid function and an element-wise multiplication (Note: $\mathbf{h}_{<t-1>}$ is the whole set of the hidden units for the previous time step), respectively and j is the j^{th} hidden unit. Starting from the bottom 11, the *reset* gate r_j is computed by using σ which is the logistic sigmoid function, and $[.]_j$ denotes the j-th element of a vector. $\mathbf{x}$ and $\mathbf{h}_{<t-1>}$ are the input and the previous hidden state, respectively. $\mathbf{W}_r$ and $\mathbf{U}_r$ are weight matrices which are learned.

Likewise, the *update* gate z_j is computed by

$$z_j = \sigma\left(\left[\mathbf{W}_z e(\mathbf{x}^{<t>})\right]_j + \left[\mathbf{U}_z \mathbf{h}_{\langle t-1\rangle}\right]_j\right)$$

The actual activation of the proposed unit h_j is then computed by

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + (1 - z_j)\tilde{h}_j^{\langle t\rangle},$$

where

$$\tilde{h}_j^{\langle t\rangle} = \tanh\left(\left[\mathbf{W}_e(\mathbf{x}^{<t>})\right]_j + \left[\mathbf{U}\left(\mathbf{r} \odot \mathbf{h}_{\langle t-1\rangle}\right)\right]_j\right)$$

In this formulation, when the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state and reset with the current input only.

Once the hidden state at the N step (the end of the source phrase) is computed, the representation of the source phrase $\mathbf{c}$ is

$$\mathbf{c} = \tanh\left(\mathbf{V}\mathbf{h}^{\langle N\rangle}\right).$$

Decoder

The decoder starts by initializing the hidden state with

$$\mathbf{h'}^{\langle 0\rangle} = \tanh\left(\mathbf{V'}\mathbf{c}\right),$$

where we will use $\cdot'$ to distinguish parameters of the decoder from those of the encoder.

The hidden state at time t of the decoder is computed by

$$h'^{\langle t\rangle}_j = z'_j h'^{\langle t-1\rangle}_j + (1 - z'_j)\tilde{h}'^{\langle t\rangle}_j,$$

where

$$\tilde{h}'^{\langle t\rangle}_j = \tanh\left(\left[\mathbf{W'}e(\mathbf{y}_{t-1})\right]_j + r'_j\left[\mathbf{U'}\mathbf{h'}_{\langle t-1\rangle} + \mathbf{C}\mathbf{c}\right]\right),$$

$$z'_j = \sigma\left(\left[\mathbf{W'}_z e(\mathbf{y}_{t-1})\right]_j + \left[\mathbf{U'}_z\mathbf{h'}_{\langle t-1\rangle}\right]_j + \left[\mathbf{C}_z\mathbf{c}\right]_j\right),$$

$$r'_j = \sigma\left(\left[\mathbf{W'}_r e(\mathbf{y}_{t-1})\right]_j + \left[\mathbf{U'}_r\mathbf{h'}_{\langle t-1\rangle}\right]_j + \left[\mathbf{C}_r\mathbf{c}\right]_j\right)$$

and $e(\mathbf{y}_0)$ is an all-zero vector. Similarly to the case of the encoder, $e(\mathbf{y})$ is an embedding of a target word. $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{C}$ are all weight matrices. The $\mathbf{C}$ matrix used enables us to use the representation we gained by encoding the source phrase.

Unlike the encoder which simply encodes the source phrase, the decoder is learned to generate a target phrase or give us the probability to go to a specific target phrase given we started with an input phrase. At each time t, the decoder computes the probability of generating j-th word by $p(y_{t,j} = 1 \mid \mathbf{y}_{t-1}, \dots, \mathbf{y}_1, X)$ by applying a soft-max layer. To draw a parallel between the parameters used in the actual implementations and the description, it is convenient to note that we have batch sizes of 10 which in the terminology of the machine translation will be like saying we have 10 different phrases of 1 word say coming from 2 characters(number of qubits) and each of the characters(single qubit measurement) can come from a vocabulary of 4 (since we use Pauli-4 measurements). And once we flatten the two words the vocabulary is now $4 \times 4 = 16$. And each permutation of the two set of 4 letters has now been encoded into 16 one hot vectors. Hence the shape of the output from a lstm layer would be (10,16,64) where 64 is a arbitrary choice of number of hidden nodes.

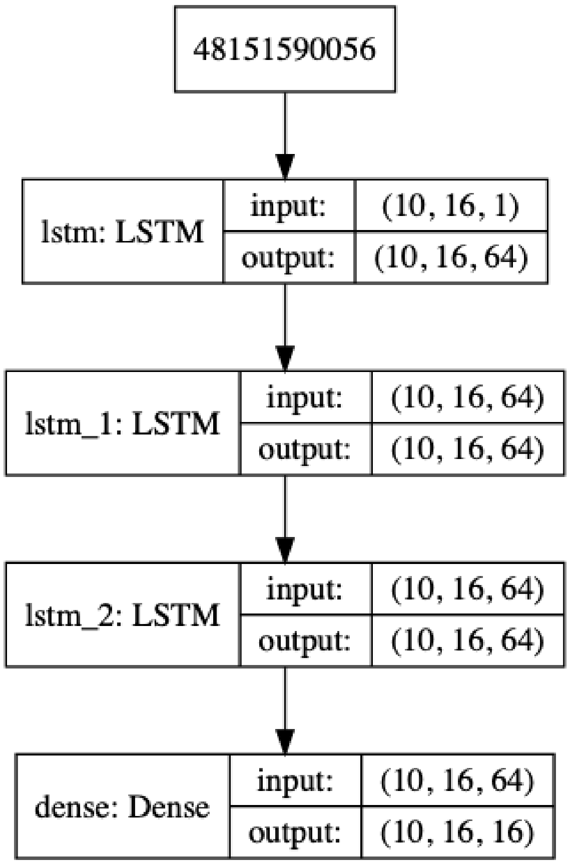


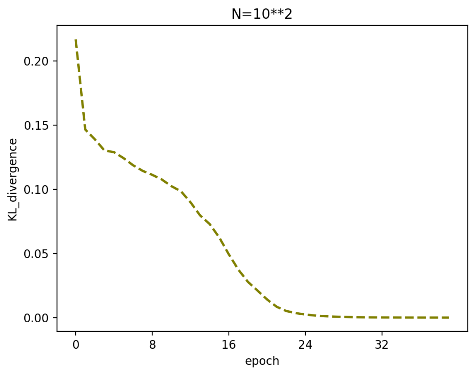
Figure 2: The architecture of a model used consisting 3 LSTM layers which predicts the conditional probability for all the input cases

The loss function used is KL-divergence which is defined by

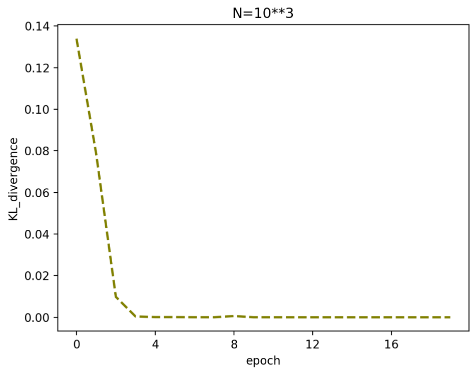
$$\sum_{b_1,b_2} P^{true}(b_1, b_2|a_1, a_2) log(\frac{P^{true}(b_1, b_2|a_1, a_2)}{P^{NN}(b_1, b_2|a_1, a_2)}) \tag{12}$$

where P^{true} are the exact probabilities and P^{NN} are the ones generated by the network. There are two kind of normalization that needs to be taken care of , one due to the fact that we are modelling the conditional probabilities here. For a given set of measurements all possible outcome probabilities will sum up-to 1. The other normalization that has been carefully chosen according to the batch size. To avoid aggressiveness of having Stochastic Gradient Descent (batch size=1), we use mini batch gradient where the descent is computed by the average of gradients over a batch of points in the neighbourhood of a training set.

Elementary Results



[**a**]



[**b**]

Figure 3: Learning the conditional probabilities with two different number of samples

As discussed earlier the quantum fidelity defined between the two density matrix will also be a criteria to judge the training that we perform via our neural network model. And for the case with 100 samples per given input states, we have a fidelity of **0.99x** after training the model with epoch greater than 30 and similarly with an increased number of samples(10^3) we get **0.999x** after training the model for more than 10 epochs. These numbers are reported for a particular architecture with a fixed number of hidden nodes in each layer and a specific batch size.

References

[1] , .