Faster Quantum Alternative to Softmax Selection in Deep Learning and Deep Reinforcement Learning

Oscar Galindo, Christian Ayub,
Martine Ceberio, and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso, El Paso, Texas 79968, USA,
ogalindomo@miners.utep.edu, cayub@miners.utep.edu,
mceberio@utep.edu, vladik@utep.edu

1. Need for Machine Learning in Engineering and Science

- One of the main objectives of science and engineering in general is:
  - to find the state of the world,
  - to predict the future state of the world, and
  - to find the control and/or design that leads to a better future state.
- The state of the world can be described by values of the corresponding physical quantities.
- A control can be described by the values of the corresponding control parameters.
- The details of a design can be described by the values of the corresponding design parameters.

2. Need for Machine Learning (cont-d)

- In these terms, the prediction and control problems can be described in a similar way:
  - we know – e.g., from measurements – the values of some quantities $x_1, \ldots, x_n$, and
  - we want to find the value of the desired quantity y based on the known values x_i.
- In some cases, the situation is straightforward:
  - we know the algorithm $f(x_1, \ldots, x_n)$ that enables us to estimate y based on the known values x_i, and
  - this algorithm can be feasibly implemented.
- In many cases, we do not have the equations.
- All we have is the records of observing the values x_i and the corresponding values y:

$$\left(x_1^{(k)}, \ldots, x_n^{(k)}, y^{(k)}\right), \quad 1 \le k \le K.$$

3. Need for Machine Learning (cont-d)

- Based on these records, we need to find a feasible algorithm $f(x_1, \ldots, x_n)$ for which

$$y^{(k)} \approx f\left(x_1^{(k)}, \ldots, x_n^{(k)}\right) \text{ for all } k.$$

- The procedure of producing such an algorithm based on the known records is known as *machine learning*.
- In many other situations:
  - while the corresponding equations are known,
  - the resulting system is very complex,
  - so, the algorithm would produce the result way after the event that we are trying to predict.
- This is the case, e.g., of predicting tornado trajectories, and of many other practical problems.
- In such situations, we cannot use the model directly.

4. Need for Machine Learning (cont-d)

- Instead, we can use the results $y^{(k)}$ of running this model on different inputs $\left(x_1^{(k)}, \ldots, x_n^{(k)}\right)$.
- Then, we apply machine learning to get a feasible algorithm based on these records.
- It is OK that this model takes too long to run:
  - we can spend as much time as necessary on generating the records,
  - as long as they help us to eventually come up with the feasible algorithm for prediction or control.

5. Need for Reinforcement Learning

- In many practical situations, the available records are not sufficient to reconstruct the algorithm $f(x_1, \ldots, x_n)$.
- In such situations, we need to perform additional measurements or simulations.
- In this case, it is desirable that the computer will instruct us which values $(x_1, \ldots, x_n)$ to try next:
  - so as to minimize the measuring effort and
  - get the desired algorithm as soon as possible.
- Learning that also involves additional requests for data is known as *reinforcement learning*.

6. Deep Learning, Deep Reinforcement Learning

- The most efficient machine learning technique is a neural network called *deep learning* (Go champion etc.).
- In neural networks, neurons – elementary data processing units – are places in several consequent layers:
  - starting with the input layer (that inputs the values $x_1, \ldots, x_n$),
  - going through so-called *hidden layers* where the signals are processed, and
  - ending up in the output layer that generates the desired approximation to the value $y = f(x_1, \ldots, x_n)$.
- The signals coming out of the neurons of each layer serve as inputs to neurons of the next layer.

7. Deep Learning (cont-d)

- At first, the network is *trained*:
  - we find the weights in the formulas describing how each neuron processes its inputs
  - so as the results of applying this network to $(x_1^{(k)}, \ldots, x_n^{(k)})$ become as close as possible to $y^{(k)}$.
- Once the network is trained, the weights are fixed ("frozen"), and the network is ready to use.
- Traditional neural networks mostly used one hidden layer.
- In contrast, modern neural networks use from three to sixteen and more hidden layers.
- Because of the presence of multiple layers, they are called *deep* neural networks.

8. Deep Learning and Deep Reinforcement Learning: Remaining Challenges

- One of the main challenges: they require a large amount of computation time.
- Most other machine learning techniques which can be easily performed on the user's computer.
- In contrast, deep learning and deep reinforcement learning require the use of high performance computers.
- This limitation severely limits the use of deep learning techniques, especially in engineering applications.
- It is therefore desirable to be able to speed up deep learning and deep reinforcement learning.

9. Softmax Selection: One of the Steps That Requires a Large Amount of Computation Time

- The main objective is to find a function that provides the best match for all the known records.
- We minimize the distance between the observed values $y^{(k)}$ and the values $f\left(x_1^{(k)}, \ldots, x_n^{(k)}\right)$.
- A natural idea to speed up the corresponding computations is to parallelize them.
- A natural way to parallelize the process is to run several optimization techniques in parallel for some time.
- Then, we select the one that leads to the most promising result.
- However, in optimization, it is well known that this can lead us to a local minimum.

10. Softmax Selection (cont-d)

- To avoid local minima, it is necessary to not always select the best alternative.
- It is necessary to sometimes select other alternatives as well.
- Still the best alternative is selected with the highest probability.
- Not so good alternatives are selected with much smaller probabilities.
- This need appeared way before deep learning.
- For this purpose, a heuristic technique called *simulated annealing* was invented.
- It often allows is to avoid local minima.

11. Softmax Selection (cont-d)

- In this technique:
  - when we want to minimize $J(a)$,
  - we select each alternative a with the probability proportional to $\exp(-c \cdot J(a))$ for some value c.
- The smaller $J(a)$, the higher the probability of selecting the alternative a.
- When $c \to \infty$, we select the minimum with prob. 1.
- From this viewpoint, the above formula can be viewed as a "soft" version of minimum.
- We usually get the minimum, but sometimes we also get alternatives with a larger value of $J(a)$.
- Because of this, this method is also known as *softmin*.
- Similarly, to maximize $J(a)$, we select each alternative a with probability proportional to $\exp(c \cdot J(a))$.

12. Softmax Selection (cont-d)

- This is known as *softmax*.
- Softmax is exactly what is usually used in deep learning.
- Other probabilistic ways were also tried, and they did not seem to lead to worse results.
- In reinforcement learning, there is an additional need for such softening.
- We would like to produce the values $(x_1, \ldots, x_n)$ for which:
  - estimating y and adding the resulting record
  - will add the largest amount of information to our knowledge.
- In this case too, to avoid the local maximum, it is desirable to use something like softmax.

13. Softmax Selection (cont-d)

- Softmax usually takes much more computation time than the usual optimization.
- Indeed, in optimization:
  - once we have the largest value seen so far,
  - then we can simply dismiss all these alternatives for which the value is smaller.
- In contrast, in softmax, we cannot dismiss anything:
  - we need to take all alternatives into account and estimate their values $J(a)$
  - so that we will be able to generate them with the corresponding probabilities.

14. Softmax Selection (cont-d)

- This takes a lot of time.
- So, to speed up deep learning, it is desirable to speed up the corresponding computations.
- It does not have to be necessarily exactly softmax.
- However, we do need some way to, in general, generate:
  - alternatives with larger values of objective function with higher probabilities and
  - alternatives with smaller values of the objective function with lower probabilities.

15. How Quantum Computing Can Help

- We will use Grover's quantum algorithm for finding an element in an unsorted list.
- In non-quantum computing, the only way to make sure that this element is found is to check all n elements.
- Thus, in non-quantum computing, the solution of this problem requires n computational steps.
- Grover's algorithm enables us to find this element in time $O(\sqrt{n})$.
- To be more precise, Grover's algorithm finds the element with some probability $1 - \varepsilon$.
- For very small $\varepsilon > 0$, we can safely ignore the possibility of a wrong answer.
- Let us show how Grover's algorithm can be used for optimization.

16. Quantum Computing (cont-d)

- We have n alternatives, with the values $v_1, \ldots, v_n$ of the corresponding objective function.
- In mathematical terms, we want to find the index i_0 for which the value v_{i_0} is the largest, i.e., for which

$$v_{i_0} = \max_i v_i.$$

- From the practical viewpoint, the values v_i are only known with some accuracy δ.
- Thus, it is sufficient to find the value which is δ-close to the desired maximum, i.e., for which

$$\left| v_{i_0} - \max_j v_j \right| \le \delta.$$

- Usually, we know a priori bounds m and M for all the values of the objective function.

17. Quantum Computing (cont-d)

- The interval $[m, M]$ will be the starting interval of our iterative process.
- At each stage, we will find a narrower interval $[\underline{v}, \overline{v}]$ containing v_{i_0}.
- We stop when the resulting interval gets width $\leq \delta$.
- At the beginning of each narrowing step, we compute a midpoint $\widetilde{v} \stackrel{\text{def}}{=} \dfrac{\underline{v} + \overline{v}}{2}$.
- Then, we use Grover's algorithm to check if there exists an index i for which $v_i \geq \widetilde{v}$.
- If there exists an index i for which $v_i \geq \widetilde{v}$, this means that $v_{i_0} = \max\limits_j v_j \geq v_i \geq \widetilde{v}$.
- So, v_{i_0} is in the half-size interval $[\widetilde{v}, \overline{v}]$;

18. Quantum Computing (cont-d)

- If there is no index i for which $v_i \geq \widetilde{v}$, this means that $v_i < \widetilde{v}$ for all i and thus, $v_{i_0} = \max\limits_j v_j \leq \widetilde{v}$.
- In this case, v_{i_0} is in the half-size interval $[\underline{v}, \widetilde{v}]$.
- In both cases:
  - by spending $O(\sqrt{n})$ computational steps of Grover's algorithm,
  - we divide the width of the interval $[\underline{v}, \overline{v}]$ containing v_{i_0} by half.
- In k steps, the interval's width decreases to 2^{-k} of its original width $M - m$.
- Since the values v_i are known with accuracy δ, it makes no sense to locate the value v_{i_0} with higher accuracy.
- So we should stop when the width $2^{-k} \cdot (M - m)$ of the resulting interval $[\underline{v}, \overline{v}]$ becomes smaller than δ.

19. Quantum Computing (cont-d)

- At this stage, we know that $v_{i_0} \geq \underline{v}$.
- So, we apply Grover's algorithm one more time to find the corresponding index i_0.
- This computation requires $k+1$ applications of Grover's algorithm.
- The value k can be determined as the smallest value for which $2^{-k} \cdot (M - m) \leq \delta$, i.e., $k = \left\lceil \dfrac{M - m}{\delta} \right\rceil$.
- This value does not depend on n, so the overall computational complexity of this algorithm is $O(\sqrt{n})$.
- Non-quantum computations would require that we look at every single value v_i and thus, would require $n \gg \sqrt{n}$ computational steps.

20. Towards Quantum Computing-Based Analogue of Softmax

- Grover's algorithm provides an answer with some probability $\geq 1 - \varepsilon$.
- The probability of error ε depends on the number of this algorithm's iterations.
- Usually, we select this number of iterations in such a way that the probability ε is very small.
- We do it to avoid deviations from the actual maximum.
- However, in our problem, we *are* in interested in deviating from the exact maximum.
- Thus, a natural way to get the desired quantum version of softmax is to decrease the number of iterations.
- Then, the probability ε becomes larger.

21. Quantum Softmax (cont-d)

- If we use this version of Grover's algorithm in the above optimization scheme, we get exactly what we want:
  - we get the exact maximum with high probability but also
  - we get other, smaller values with a reasonable non-zero probability.
- Here:
  - in contrast to softmax, which needs $O(n)$ computational steps,
  - the proposed quantum computing-based algorithm takes much smaller time $O(\sqrt{n}) \ll n$.

22. Acknowledgments

This work was supported in part by the National Science Foundation via grants:

- 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science),
- and HRD-1242122 (Cyber-ShARE Center of Excellence).