Quantum Formulations for Neural Network Training

Michael Ogezi
ogezi@ualberta.ca

Department of Computer Science,
University of Alberta, Edmonton, AB.

CMPUT 604: Quantum Computing for Computer Scientists

***Abstract*—With the rapid increase in the size and complexity of machine learning architectures over the past few years, we have gained improved capabilities on many tasks across Natural Language Processing, Computer Vision, and other fields. However, with Moore's Law plateauing [1], and researchers realising that they cannot continue to train larger and larger models ad infinitum, it is increasingly important that we find alternative ways to train our ever more complex deep neural networks within a reasonable time frame. In this work, I formulate the training optimisation of a deep neural network as a Binary Quadratic Model (BQM), and more specifically, a Quantum Unconstrained Binary Optimisation (QUBO) in order to facilitate faster model training and convergence on quantum annealing devices.**

I. Introduction

At its core, the process of training a learning system involves searching for the combination of weights (and biases) that best approximates some target phenomenon. While closed-form solutions exist for some linear regression problems (such as $\boldsymbol{w} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T t}$), in general, there isn't an analytical way to find the ideal weights. We, therefore, end up using a numerical method: gradient descent, or one of its variants to find the best configuration of weights. We know that training a three-node neural network is NP-Complete [2]. Furthermore, the most common data type for neural network neurons is the 32-bit floating point. It can hold 4,278,190,079 distinct values. This means that there are $\sim 1.3230542 \times 10^{81}$ (4,278,190,079 *combination* 9) possible weight configurations for a neural network with just *nine* parameters. Based on the Planck Collaboration's estimates [3], there are 5.3×10^{79} atoms in the observable universe. So, there are more possible configurations for a 9-parameter neural network (the minimum required to approximate the XOR function) than there are atoms in the universe. Going further, GPT-3 [4], OpenAI's latest large language model, has *175 billion* parameters.

Quantum computers possess special characteristics which classical computers do not. Some of these are superposition, tunnelling, interference, and entanglement. Superposition, which allows the same qubit to manifest in a combination of many possible states at once, gives quantum computers a considerable advantage when performing parallel computations. This means that we can leverage superposition to consider many more weight configurations. Tunnelling is one of the phenomena being leveraged by quantum annealing machines such as the D-Wave 2000Q. Quantum Annealing leverages quantum fluctuations to find the global optimum of a given cost function.

Although work is being done to solve linear machine learning problems using QUBO methods, there is less progress on deep neural networks. Borle et al. [5] have proposed a method of solving the linear least-squares problem using quantum annealing. Chang et al. [6], [7], in two papers, have also proposed ways of solving the least-squares problem with QUBOs and applying that to solve linear regression problems. Date et al. [8] have also improved on previous work by computing both positive and negative real-number parameters for linear regression models. All this work helps solve linear problems – which must be linearly separable. Although this work is important, many phenomena that we would like to model, especially the ones we see in the real world are not linearly separable. *OR* – as seen in Fig. 2 below – is a linearly separable function. On the other hand, *XOR* – as seen in Fig. 1 below – is not.

The reason that that deep neural networks can solve XOR is that they contain non-linearities in the form of activation functions such as:

1) Sigmoid: $\sigma(z) = \dfrac{1}{1+e^{-z}}$
2) Softmax: $S(\boldsymbol{z}) = \dfrac{e^{z_i}}{\sum_{i'} e^{z_{i'}}}$
3) Relu: $R(z) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$
4) Tanh: $T(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$

These activation functions come between layers and allow neural nets to learn and model much more complicated phenomena than linear models would allow.

$$\boldsymbol{a_1} = \boldsymbol{W_1} \cdot \boldsymbol{x} + \boldsymbol{b_1}$$
$$\boldsymbol{a_2} = \boldsymbol{W_2} \cdot \boldsymbol{a_1} + \boldsymbol{b_2} = \boldsymbol{W_2} \cdot [\boldsymbol{W_1} \cdot \boldsymbol{x} + \boldsymbol{b_1}] + \boldsymbol{b_2}$$

- $\boldsymbol{a_i}$ – activations from the i^{th} layer
- $\boldsymbol{W_i}$, $\boldsymbol{b_i}$ – weights, and biases from the i^{th} layer

As we can see in the equations above, feed-forward neural network layers without an activation function are still linear functions.

TABLE I: Data points of the XOR problem

x_1	x_2	t
0	0	**0** (●)
0	1	**1** (●)
1	0	**1** (●)
1	1	**0** (●)

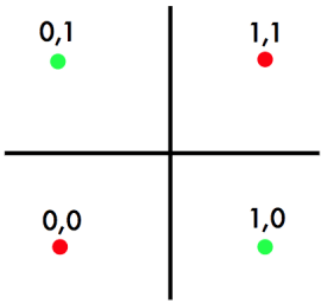

Fig. 1: XOR is not linearly separable

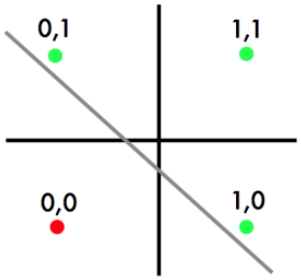

Fig. 2: OR is linearly separable

In this work, I will attempt to train a neural network to estimate the XOR function. I aim to formulate an XOR neural network optimisation task as a QUBO and compare that to a traditional classical approach designed with Keras [9], a well known deep learning library.

II. Methods

A. Mathematical Basis

I will be running experiments on D-Wave adiabatic quantum computers. These devices work to minimise a function that is modelled as a QUBO. This process is expressed in the matrix-vector form below:

$$min_{\boldsymbol{z}\in B^M}\ \boldsymbol{z^T A z} + \boldsymbol{z^T b} \tag{1}$$

B. Notations

The notations that I will be using are as follows:

- $\mathbb{R}$ is the set of real numbers.
- $\mathbb{N}$ is the set of natural numbers.
- $B = \{0, 1\}$ is the set of binary numbers.
- $\boldsymbol{z} \in B^M$ is our binary decision vector.
- $\boldsymbol{A} \in \mathbb{R}^{M\times M}$ is the QUBO matrix.
- $\boldsymbol{b} \in \mathbb{R}^M$ is the QUBO vector.
- N is the number of training samples (rows).
- d is the number of features (columns).
- $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ is the training dataset.
- $\boldsymbol{t} \in \mathbb{R}^N$ is the target class vector.

C. Objective Function

The objective of our XOR task is to minimise the loss function below. $\boldsymbol{Xw}$ represents our vector of predictions while $\boldsymbol{t}$ represents our vector of targets.

$$min_{\boldsymbol{w}\in\mathbb{R}^d}\ J(\boldsymbol{w}) = \|\boldsymbol{t} - \boldsymbol{Xw}\|_2^2 \tag{2}$$

D. Classical Formulation

The classical approach will be traditional gradient descent.

- α is the learning rate.
- $\nabla J(\boldsymbol{w})$ is the derivative (w.r.t $\boldsymbol{w}$) of the loss function in Equation 2.
- E is the number of epochs (iterations through the entire training set) I will train for.

Algorithm 1 Classical gradient descent algorithm

```
for i in 1...E do
    w ← w − α · ∇J(w)
    if should update learning rate then
        α ← α^(new)
    end if
end for
```

E. Quantum (QUBO) Formulation

Proceeding to expand our loss expression:

$$min_{\boldsymbol{w}\in\mathbb{R}^d}\ J(\boldsymbol{w}) = \|\boldsymbol{t} - \boldsymbol{Xw}\|_2^2$$
$$min_{\boldsymbol{w}\in\mathbb{R}^d}\ J(\boldsymbol{w}) = (\boldsymbol{t} - \boldsymbol{Xw})^T(\boldsymbol{t} - \boldsymbol{Xw})$$
$$min_{\boldsymbol{w}\in\mathbb{R}^d}\ J(\boldsymbol{w}) = \boldsymbol{t^T t} - \boldsymbol{t^T Xw} - \boldsymbol{w^T X^T t} + \boldsymbol{w^T X^T Xw}$$

And finally:

$$min_{\boldsymbol{w}\in\mathbb{R}^d}\ J(\boldsymbol{w}) = \boldsymbol{w^T X^T Xw} - \boldsymbol{2w^T X^T t} + \boldsymbol{t^T t} \tag{3}$$

Our mathematical definitions will be similar to those laid out by Date et al. [8]. I introduce a K-dimensional precision vector $P = \begin{bmatrix} p_1, p_2, \ldots, p_K \end{bmatrix}^T$ sorted in increasing order. Each entry in P will be an integral power of 2 which could be positive or negative i.e ($2^{-1} = \frac{1}{2}$, $2^0 = 1$, and $-(2^1) = -2$). I also introduce a K-dimensional vector $\hat{w}_i$ with binary coefficients, such that $\hat{w}_i^T P$ yields a scalar $w_i \in \mathbb{R}$. This scalar w_i represents the i^{th} entry in our weight vector, where $i \in \{1, \ldots, d+1\}$. $\hat{w}_{ik}$ (the k^{th} element of the $\hat{w}_i$ vector) can be thought of as a binary decision variable that selects (with $1 \in B$) or ignores (with $0 \in B$) entries in P. In this way, I represent w_i similarly to how binary floating-point values are converted to decimal floating-point values (recall that QUBOs can only handle binary variables). With this formulation, I can

have up to 2^K unique values for each $\hat{w}_i$. Now, I define the K(d + 1)-dimensional binary vector $w \in B^{K(d+1)}$, such that:

$$\hat{\boldsymbol{w}} = \left[\hat{w}_{11}, \ldots, \hat{w}_{1K}, \hat{w}_{21}, \ldots, \hat{w}_{2K}, \ldots, \hat{w}_{(d+1)1}, \ldots, \hat{w}_{(d+1)K}\right]^T \tag{4}$$

This is simply concatenating all the w_i vectors in a single vector.

Similarly, I can define a precision matrix as follows:

$$\rho = I \otimes P^T \tag{5}$$

Where:

- $I \in \mathbb{R}^{(d+1)\times(d+1)}$ is an identity matrix.
- $\otimes$ is the Kronecker product.
- $\rho \in \mathbb{R}^{(d+1)\times K(d+1)}$

Now our original weight vector is obtained by:

$$w = \rho\hat{w} \tag{6}$$

I can now pose the minimization problem of Equation 3 as an equivalent QUBO problem. Let us substitute the expression obtained for the weight vector w in terms of P and $\hat{w}$ into Equation 3, which yields:

$$min_{\hat{\boldsymbol{w}}\in\mathbb{B}^{K(d+1)}}\ J(\hat{\boldsymbol{w}}) = \hat{\boldsymbol{w}}^{\boldsymbol{T}}\boldsymbol{\rho}^{\boldsymbol{T}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{\rho}\hat{\boldsymbol{w}} - \boldsymbol{2}\hat{\boldsymbol{w}}^{\boldsymbol{T}}\boldsymbol{\rho}^{\boldsymbol{T}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{t} \tag{7}$$

I have neglected the term $\boldsymbol{t}^{\boldsymbol{T}}\boldsymbol{t}$ because it is a constant scalar and does not affect the solution to this unconstrained optimization problem. See that Equation 7 now has the form of a QUBO problem, as desired. Hence, we are able to solve this optimization problem for one neural network layer using a D-Wave adiabatic computer. The first phase of embedding is complete.

The second phase of embedding entails generalising the first to all the layers of the deep neural network. There is a fundamental problem with modelling an activation function within a QUBO. As we established earlier that an activation function was necessary to model the XOR function, I have to find a way around this. The first thing I do is to use the sigmoid function as the de-facto and hard-coded activation function for all layers. We know that what the sigmoid function $\sigma(z)$ does is to force a number z into $\sigma(z) \in [0, 1]$.

$$\text{let } Q = \boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{z} + \boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{b}$$

$$min_{\boldsymbol{z}\in B^M}\ Q \tag{8}$$

$$max_{\boldsymbol{z}\in B^M}\ Q \equiv min_{\boldsymbol{z}\in B^M}\ -Q \tag{9}$$

For example, $\sigma(+\inf) = 1$; and $\sigma(-\inf) = 0$. So, when I want my eventual output (after the activation function) to be 0, I minimise the QUBO expression so that it returns the smallest value it can. This results in a high-magnitude negative number z_{small} and then $\sigma(z_{small}) \approx 0$. One the other hand, when I want to output a 1, I minimise the negative of (maximise) the QUBO expression. This results in a high-magnitude positive number z_{big} and then $\sigma(z_{big}) \approx 1$.

We bring together the first and second phases of our approach for expressing a QUBO formulation for our 2-layer XOR neural network – the input layer, one hidden layer (1), and the output layer (2). We go through the following steps:

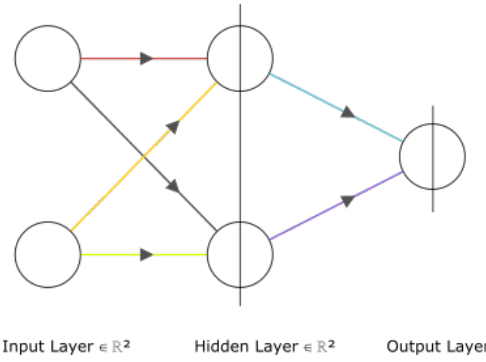

Fig. 3: Proposed XOR neural network diagram with 2 neurons in the hidden layer (vertical lines represent sigmoid function applications)

- Fix the outputs from the hidden layer and what they would lead to in the final layer. For example, the output layer weights $w_2 = \left(w_{21}, w_{22}\right)$ could be selected as $\left(0, 0\right)$ for a 0 (●) output and $\left(1, 0\right)$ for a 1 (●) output.
- Express the task to find the hidden layer weights to give this required output as a QUBO Q_1 of the form in Equation 7.
- Minimise Q_1.
- Optionally express the task to find the final layer weights to give this required output as a QUBO. The alternative is to do a simple mapping. Using the example above, I would map $w_2 = \left(0, 0\right)$ to 0 and $w_2 = \left(1, 0\right)$ to 1.
- Minimise Q_2 if I took the first choice in the previous step.

In this way, I separate the deep neural network into multiple augmented linear regression problems.

III. Results and Discussion

I will be running my primary experiments on both a *simulator*, and a *real* D-Wave quantum machine. On the other hand, I will be running my baseline on a *classical* machine, then comparing our results across all three modalities. For our QUBO formulation, I set $K = 32$. This means that we're using 32 bits to represent each weight that would eventually be used in the neural network. The 32 bits are not structured in the same way as conventional 32-bit floating points. They are broken down thus:

1) 8 bits for values using positive exponents with a negative sign $(w_{ij} <= -1)$ e.g. $-1, -2, -4$.
2) 8 bits for values using negative exponents with a negative sign $(-1 < w_{ij} < 0)$ e.g. $-\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}$.
3) 8 bits for values using negative exponents with a positive sign $(0 < w_{ij} < 1)$ e.g. $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$.
4) 8 bits for values using positive exponents with a positive sign $(w_{ij} >= 1)$ e.g. $1, 2, 4$.

With this 32-bit setup, we can represent final weight values as $\hat{w}_i^T P \in [\sim -256, \sim 256]$. We also posses fraction precision of up to 0.00390625 (2^{-8}). Now for the 2-, and 4-neuron

hidden layers, I have 288, and 544 state variables to compute respectively.

TABLE II: D-Wave Real vs Simulator vs Classical

	Hidden Layers	Mean Accuracy (%)	Time (ms)
Real	2	**81.36**	134,363
Real	4	**84**	254,937
Simulator	2	81.4	1,565.38
Simulator	4	82.2	2,564.72
Classical	2	72.5	**684.17**
Classical	4	78.5	**667.38**
Classical	8	90.5	**637.57**
Classical	16	98	**671.78**

Time is the arithmetic mean of the actual time taken to run the training phase. For classical methods, this would be the mean time taken to train the model over a specified number of epochs. For quantum annealing methods, this would be the mean of the combined annealing times taken for each annealing sample.

Mean accuracy is the arithmetic mean of the accuracy across multiple training sessions up to a number of epochs (10, 25, 50, 100, or 200), and for a specified number of hidden neurons (2, 4, 8, or 16).

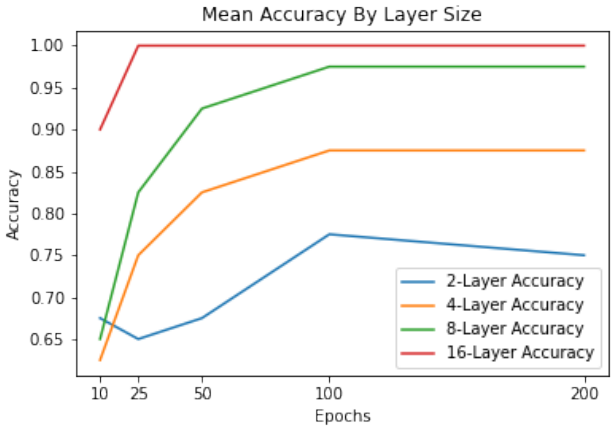

Fig. 4: Classical method: Mean accuracy by hidden neurons

As we can see from Table II as well as Figures 4, and 5, for 2-, and 4-neuron hidden-layer configurations, I achieve significantly better mean accuracy than the classical methods – for the real vs classical experiments respectively, 84.0% vs 78.5% for 4-neuron and 81.36% vs 72.5% for the 2-neuron.

However, for the time taken, the QUBO methods are vastly outperformed by the classical methods (two orders of magnitude). This holds even for the 8-, 16-layer networks which we couldn't test on the annealing device because there were too many state variables (1,056, and 2,080 respectively) to embed (with my machine access quota). My results thus show decent progress towards training deep neural networks with QUBO formulations on quantum annealing devices. Looking forward, I hope to explore training networks with many more neurons in the hidden layers.

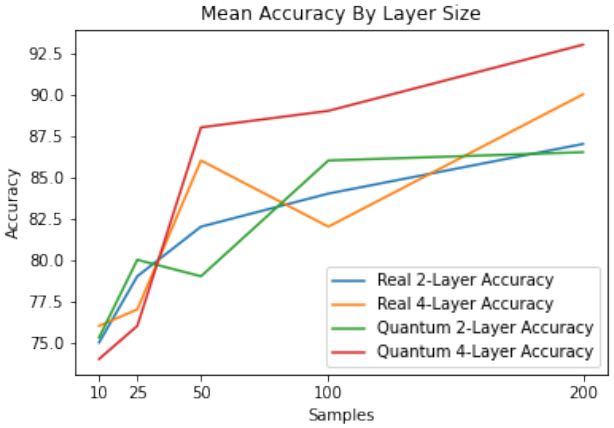

Fig. 5: Real and simulated quantum methods: Mean accuracy by hidden neurons

IV. Conclusion

The emergence of reliable, large-scale quantum annealing devices, and quantum computers more broadly could mark a fundamental paradigm shift in the field of deep learning. We have the opportunity to transition to machines that have the potential to give us several orders-of-magnitude speedups on our computations. However, as I have shown in our experiments, deep neural network training via QUBOs can only be done on a small scale at the moment. But, as quantum device technology improves, we should be able to solve increasingly complicated problems by training deeper and wider neural networks more efficiently. This will lead to faster, and cheaper systems as well as reduced carbon emissions.

Acknowledgements

This work was prepared through discussions with Professor Boulanger.

References

[1] K. Henke, G. T. Kenyon, and B. Migliori, "Machine learning in a post moore's law world: Quantum vs. neuromorphic substrates," in *2020 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, 2020, pp. 74–77.

[2] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is np-complete," in *Machine learning: From theory to applications*. Springer, 1993, pp. 9–28.

[3] and P. A. R. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. J. Banday, R. B. Barreiro, J. G. Bartlett, N. Bartolo, E. Battaner, R. Battye, K. Benabed, A. Benoît, A. Benoit-Lévy, J.-P. Bernard, M. Bersanelli, P. Bielewicz, J. J. Bock, A. Bonaldi, L. Bonavera, J. R. Bond, J. Borrill, F. R. Bouchet, F. Boulanger, M. Bucher, C. Burigana, R. C. Butler, E. Calabrese, J.-F. Cardoso, A. Catalano, A. Challinor, A. Chamballu, R.-R. Chary, H. C. Chiang, J. Chluba, P. R. Christensen, S. Church, D. L. Clements, S. Colombi, L. P. L. Colombo, C. Combet, A. Coulais, B. P. Crill, A. Curto, F. Cuttaia, L. Danese, R. D. Davies, R. J. Davis, P. de Bernardis, A. de Rosa, G. de Zotti, J. Delabrouille, F.-X. Désert, E. D. Valentino, C. Dickinson, J. M. Diego, K. Dolag, H. Dole, S. Donzelli, O. Doré, M. Douspis, A. Ducout, J. Dunkley, X. Dupac, G. Efstathiou, F. Elsner, T. A. Enßlin, H. K. Eriksen, M. Farhang, J. Fergusson, F. Finelli, O. Forni, M. Frailis, A. A. Fraisse, E. Franceschi, A. Frejsel, S. Galeotta, S. Galli, K. Ganga, C. Gauthier, M. Gerbino, T. Ghosh, M. Giard, Y. Giraud-Héraud, E. Giusarma, E. Gjerløw, J. González-Nuevo, K. M. Górski, S. Gratton,

A. Gregorio, A. Gruppuso, J. E. Gudmundsson, J. Hamann, F. K. Hansen, D. Hanson, D. L. Harrison, G. Helou, S. Henrot-Versillé, C. Hernández-Monteagudo, D. Herranz, S. R. Hildebrandt, E. Hivon, M. Hobson, W. A. Holmes, A. Hornstrup, W. Hovest, Z. Huang, K. M. Huffenberger, G. Hurier, A. H. Jaffe, T. R. Jaffe, W. C. Jones, M. Juvela, E. Keihänen, R. Keskitalo, T. S. Kisner, R. Kneissl, J. Knoche, L. Knox, M. Kunz, H. Kurki-Suonio, G. Lagache, A. Lähteenmäki, J.-M. Lamarre, A. Lasenby, M. Lattanzi, C. R. Lawrence, J. P. Leahy, R. Leonardi, J. Lesgourgues, F. Levrier, A. Lewis, M. Liguori, P. B. Lilje, M. Linden-Vørnle, M. López-Caniego, P. M. Lubin, J. F. Macías-Pérez, G. Maggio, D. Maino, N. Mandolesi, A. Mangilli, A. Marchini, M. Maris, P. G. Martin, M. Martinelli, E. Martínez-González, S. Masi, S. Matarrese, P. McGehee, P. R. Meinhold, A. Melchiorri, J.-B. Melin, L. Mendes, A. Mennella, M. Migliaccio, M. Millea, S. Mitra, M.-A. Miville-Deschênes, A. Moneti, L. Montier, G. Morgante, D. Mortlock, A. Moss, D. Munshi, J. A. Murphy, P. Naselsky, F. Nati, P. Natoli, C. B. Netterfield, H. U. Nørgaard-Nielsen, F. Noviello, D. Novikov, I. Novikov, C. A. Oxborrow, F. Paci, L. Pagano, F. Pajot, R. Paladini, D. Paoletti, B. Partridge, F. Pasian, G. Patanchon, T. J. Pearson, O. Perdereau, L. Perotto, F. Perrotta, V. Pettorino, F. Piacentini, M. Piat, E. Pierpaoli, D. Pietrobon, S. Plaszczynski, E. Pointecouteau, G. Polenta, L. Popa, G. W. Pratt, G. Prézeau, S. Prunet, J.-L. Puget, J. P. Rachen, W. T. Reach, R. Rebolo, M. Reinecke, M. Remazeilles, C. Renault, A. Renzi, I. Ristorcelli, G. Rocha, C. Rosset, M. Rossetti, G. Roudier, B. R. d'Orfeuil, M. Rowan-Robinson, J. A. Rubiño-Martín, B. Rusholme, N. Said, V. Salvatelli, L. Salvati, M. Sandri, D. Santos, M. Savelainen, G. Savini, D. Scott, M. D. Seiffert, P. Serra, E. P. S. Shellard, L. D. Spencer, M. Spinelli, V. Stolyarov, R. Stompor, R. Sudiwala, R. Sunyaev, D. Sutton, A.-S. Suur-Uski, J.-F. Sygnet, J. A. Tauber, L. Terenzi, L. Toffolatti, M. Tomasi, M. Tristram, T. Trombetti, M. Tucci, J. Tuovinen, M. Türler, G. Umana, L. Valenziano, J. Valiviita, F. V. Tent, P. Vielva, F. Villa, L. A. Wade, B. D. Wandelt, I. K. Wehus, M. White, S. D. M. White, A. Wilkinson, D. Yvon, A. Zacchei, and A. Zonca, "Planck2015 results," *Astronomy & Astrophysics*, vol. 594, p. A13, Sep. 2016. [Online]. Available: https://doi.org/10.1051/0004-6361/201525830

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] A. Borle and S. J. Lomonaco, "Analyzing the quantum annealing approach for solving linear least squares problems," 2018.

[6] T. H. Chang, T. C. Lux, and S. S. Tipirneni, "Least-squares solutions to polynomial systems of equations with quantum annealing," *Quantum Information Processing*, vol. 18, no. 12, pp. 1–17, 2019.

[7] C. C. Chang, A. Gambhir, T. S. Humble, and S. Sota, "Quantum annealing for systems of polynomial equations," *Scientific reports*, vol. 9, no. 1, pp. 1–9, 2019.

[8] P. Date, D. Arthur, and L. Pusey-Nazzaro, "Qubo formulations for training machine learning models," *Scientific Reports*, vol. 11, no. 1, May 2021. [Online]. Available: http://dx.doi.org/10.1038/s41598-021-89461-4

[9] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras