Descent Minimization on a Quantum Computer

Willard Miranker

July 2008
TR-1410

Descent Minimization on a Quantum Computer

Willard Miranker
Department of Computer Science
Yale University
7/31/08

Abstract: The parallelism of quantum computation is applied to the problem of descent minimization. That parallelism provides the setting for a diffusion technique that we introduce for improving the minimum attained, achieving the global minimum in some cases. The algorithm's quantum computational circuitry is given, and the associated critical probability concentration property is discussed. The methodologies' scope is illustrated by application to Hopfield net dynamics, the latter known to generate a variety of candidate descent minimization problems. As a byproduct, this shows how the recursive neural network (a brain model) can be implemented on a quantum computer.

Key words: descent minimization, global minimization, Hopfield nets, parallelization, probability concentration, quantum computing

1. INTRODUCTION

Quantum computation with its capacity to explore the entire computational basis in parallel and its associated probability concentration requirement is applied to descent minimization. We expect these features to furnish results superior to those of a standard computation. They also provide the setting for a diffusion technique that we introduce for improving the value of the minimum found, in some cases achieving the global minimum (improvements central for the problem of protein folding, for example).

In Sect. 2 we formulate the descent computation algorithm and develop the quantum circuits that implement it. We then show how that implementation produces the key quality of probability concentration required of a quantum computation.

In Sect. 3 we introduce a diffusion process, which when coupled with the quantum parallelism improves the value of the minimum found, in some cases reaching the global minimum. To demonstrate the scope of the methodology, we show how it applies to the relaxation dynamics of a Hopfield net; those dynamics known to be a generator of a wide variety of candidate descent minimization problems. As a byproduct this also shows how a recursive neural network (a brain model) can be implemented on a quantum computer.

The quantum algorithm generates a collection of information sharing paths on the edges of a unit cube, those paths overlap onto one another as they approach cube vertices that encode possible solutions. Compare this to the digital computer parallelization of a descent optimization algorithm of Chazan and Miranker (1970), that algorithm generating a collection of information sharing paths that converge and coalesce at the solution.

2. THE QUANTUM ALGORITHM

In Sect. 2.1 we give the circuitry for the parallel quantum implementation of a minimization descent algorithm. The probability concentration supplied by that quantum circuitry is discussed in Sect. 2.2.

2.1 The quantum circuits

We apply the parallelism of quantum computing to a descent minimization (local or global) recurrence for a function denoted by $H(v)$. As with any computer implementation, the algorithm must first be projected[1] into the quantum computer's computational basis, in order that it may be conducted as a quantum computation. So representing the descent minimization algorithm as

$$v_{j+1} = G_j(v_j),\ j = 0,2\ldots, \tag{2.1}$$

replace each function G_j that produces the local descent displacement at the j-th step of the minimization algorithm by an approximation that is itself a mapping of the set of Boolean strings $\{0,1\}^N$ into itself. We shall use the same name for a function and its approximation, since confusion will not occur. So the quantum computation may be viewed as being conducted on vertices of the unit N-cube, that is, on the binary integers $\mathbf{N} = \{0,1,\cdots,2^N - 1\}$. As modified, the functions referred to are endomorphisms of $\mathbf{N}$, the latter taken as a group with addition mod 2^N. Correspondingly, the minima of the (projected) function $H(v)$ are located at the N-cube vertices. We emphasize the descent requirement that $H(v_{j+1}) \le H(v_j)\ \forall j \ge 1$.

A vertex v of the N-cube (a binary integer) may be encoded as a quantum state $|b\rangle$, that state decoded into the corresponding binary integer by N spin measurements, as is well known (Nielsen and Chuang (2000)). This is concisely written as

$$b \underset{\substack{\textit{decoded via}\\ \textit{measurement}}}{\overset{\textit{encoded}}{\rightleftarrows}} |b\rangle,\quad \forall b \in \mathbf{N}. \tag{2.2}$$

By convention all quantum states are taken as normalized, but we shall not necessarily indicate the relevant normalizing constants.

Since $2^N = 1\underbrace{0\cdots0}_{N}$ (in binary), the kets that occur in (2.2) are specified as tensor products of qubits, namely

[1] The term rounding is used for such a projection into a digital computer's data space, the latter usually referred to as the screen of floating point numbers (Kulisch and Miranker (1981)).

(2.3) $$|0\rangle = \underbrace{|0\rangle\cdots|0\rangle}_{N} = \underbrace{|0\cdots0\rangle}_{N},\quad |1\rangle = \underbrace{|0\cdots01\rangle}_{N},\quad \cdots\quad,\quad |2^N-1\rangle = \underbrace{|1\cdots1\rangle}_{N}.$$

Now let

(2.4) $$|\phi_0\rangle = \sum\nolimits_{b=0}^{2^N-1} |b\rangle / 2^{N/2}.$$

This quantum state corresponds to the superposition of all of the vertices of the N-cube (i.e., $|\phi_0\rangle$ is a superposition of all of the kets in (2.3)). Let $\boldsymbol{H}^{\otimes N}$ denote the N-fold tensor product of the unitary Hadamard matrix $\boldsymbol{H} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$ with itself. Then

(2.5) $$|\phi_0\rangle = \boldsymbol{H}^{\otimes N} \underbrace{|0\cdots0\rangle}_{N},$$

(2.5) shows that this superposition of 2^N states (an exponentially increasing number) can be generated by N applications of $\boldsymbol{H}$, as is well known in quantum computing. Note that the ket label used for the summation variable b in (2.4) is also a unit cube vertex label. Moreover it also stands for an appropriate binary integer. The intended meaning will be clear from the context.

We shall now introduce a sequence of quantum circuits that taken together will implement the algorithm in question. Standard conventions in drawing quantum circuits are used (see Nielsen and Chuang (2000)). The first shown in Fig. 2.1 illustrates the operation in (2.5).

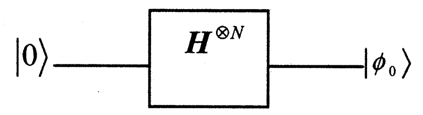
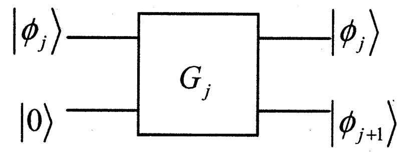


Fig. 2.1: Quantum circuit for the production of the ket $|\phi_0\rangle$

The circuit in Fig. 2.2a performs the parallel quantum evaluation of the function G_j on the state $|\phi_j\rangle = \sum\nolimits_{b=0}^{2^N-1} \alpha_{j,b}|b\rangle$ say (the $\alpha_{j,b}$ being appropriate amplitudes), producing the state $|\phi_{j+1}\rangle$, where

(2.6) $$|\phi_{j+1}\rangle = |G_j\phi_j\rangle = \sum\nolimits_{b=0}^{2^N-1} \alpha_{j,b} G_j |b\rangle.$$



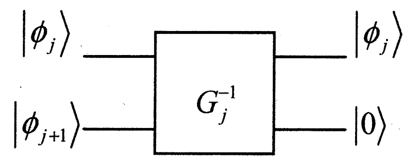


Fig. 2.2a: Parallel evaluation of G_j **Fig. 2.2b:** Defining G_j^{-1} by running G_j backwards

So the circuit in Fig. 2.2a shows the implementation of (2.6), while the circuit in Fig. 2.2b that implicitly defines the symbol G_j^{-1} consists of running the circuit in Fig. 2.2a backwards[2].

In Fig. 2.3 and Fig. 2.4, we show the circuits that compute (2.1) for $j = 0$ and $j = 1$, respectively.

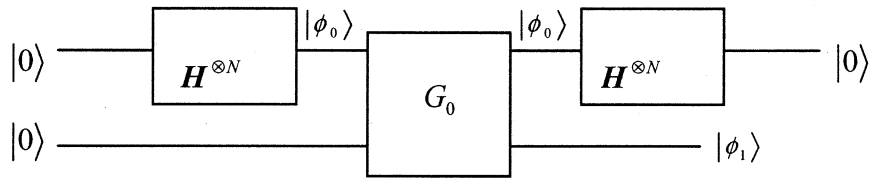


Fig. 2.3: The quantum circuit for implementing the computation of (2.1) for $j = 0$, that is, for computing the state $|0\rangle\sum_b|G_0 b\rangle$ starting with $|0\rangle|0\rangle$.

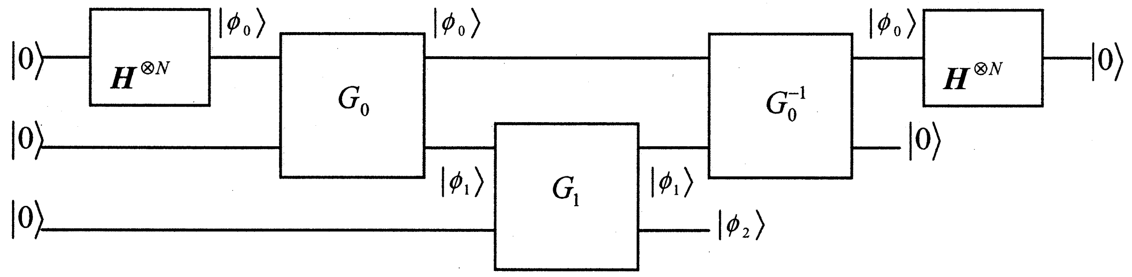


Fig. 2.4: The quantum circuit for implementing the computation of (2.1) for $j = 1$, that is, for computing $|0\rangle|0\rangle|\phi_2\rangle$ starting with $|0\rangle|0\rangle|0\rangle$.

In Fig. 2.5, we illustrate the circuit that performs the computation of (2.1) for an arbitrary value called p of $j \geq 0$. The circuit has a "V-shape", and the computational protocol is to descend the left arm of the V and then to ascend the right arm (this being suggested by the looping arrows in the figure). This protocol is more easily seen in the special cases p =1 and 2 in Figs. 2.3 and 2.4, respectively. For convenience and clarity we do not discuss nor do we display the circuitry for making the spin measurements that are required to read out the result of a computation. Assuming each operation $\boldsymbol{H}^{\otimes N}$ or of a G_j (forwards or backwards) uses one time unit, we see from Fig. 2.5 that starting with $|0\rangle = \underbrace{|0\rangle \cdots |0\rangle}_{N}$ the computation of the final state $\underbrace{|0 \cdots 0\rangle}_{p-1}|\phi_p\rangle$ requires $2(p+1)$ time units.

[2] The author is grateful to Steve Girvin for discussions, and, in particular, for the suggestion concerning running a quantum circuit backwards as in Fig. 2.4. Of course, this is also done in Fig. 2.3, since $\boldsymbol{H}$, being unitary, is its own inverse.

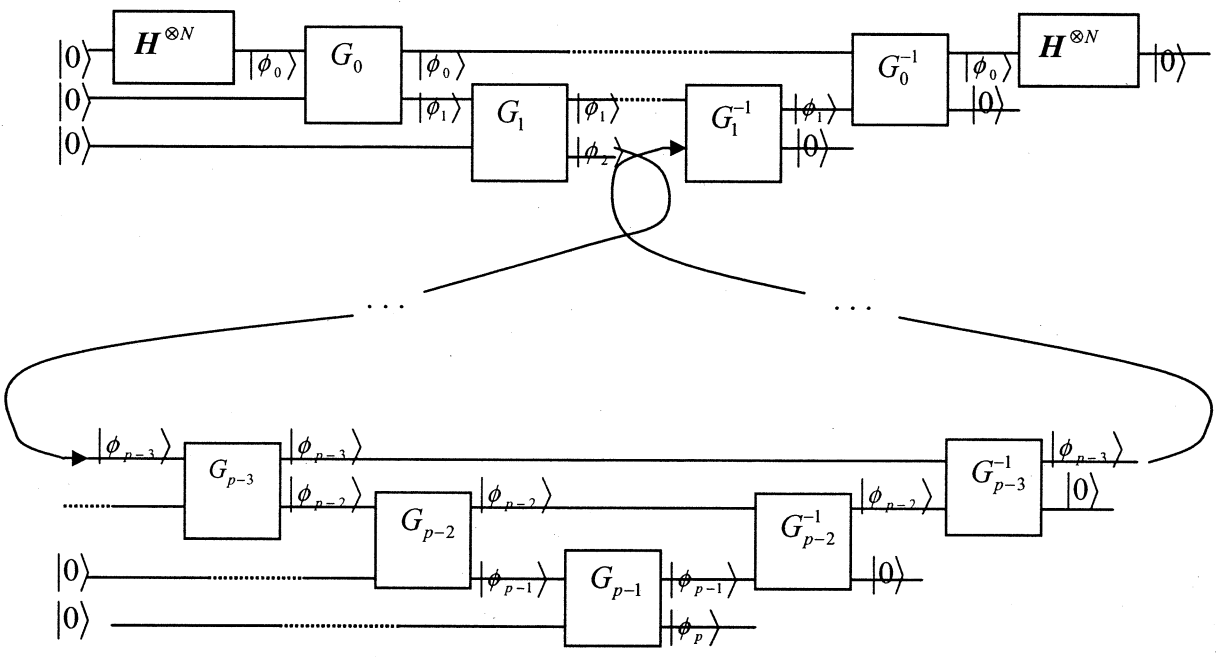


Fig. 2.5: The quantum circuit for implementing the computation of (2.1) for $j = p-1$. The looping arrows display the flow of computation that yields $\underbrace{|0\cdots0\rangle}_{p-1}|\phi_p\rangle$ starting with $\underbrace{|0\rangle\cdots|0\rangle}_{p}$.

2.3 Concentration of probability

We now argue that the quantum computation concentrates probabilities. There are 2^N possible states in the computational basis the quantum computer. These correspond to the 2^N vertices of the N-cube and are encoded superposed into the state $|\phi_0\rangle$ (consisting of N successive qubits per vertex) by the first application of $\boldsymbol{H}^{\otimes N}$ in Fig. 2.1. As the recursion commences, a path along N-cube edges emanates from each such vertex. Each path grows, one edge per recursion of the computation shown in Fig. 2.5. The output of the j-th recursion of the circuit in Fig. 2.5 delivers the state $|\phi_j\rangle$, which represents the superposition of all the leading vertices of these paths. Since the function H being minimized decreases as each path is augmented by an edge, N-cube vertices are being discarded from $|\phi_j\rangle$ for each increase $j \to j+1$. This means that the coefficient of a corresponding discarded state (an appropriate vertex of the N-cube), that is, its probability amplitude, must become zero. Indeed that particular state will never reappear with non-zero amplitude, during the course of the computation. Of course, $|\phi_j\rangle$ always contains 2^N vertices, so there are an increasing number of vertex repetitions. That is, *paths begin to overlap*; equivalently, *probabilities begin to concentrate.* Each path will approach a minimum of the function H. A terminating quantum measurement will produce an output of the descent algorithm (2.1) being executed.

3. APPROACHING THE GLOBAL MINIMUM

Suppose a diffused version (in discretized form) of $H(v)$ is woven into a recursion for determining a local minimum of $H(v)$. Diffusion will cause minima to flatten. In optimal circumstances the shallower minima will evanesce, and only the more robust will remain to serve as attractors of the augmented recursion. If we explore the entire domain of $H(v)$, we should expect diffusion to result in improvement of the result found. This method should produce the global minimum in some cases, for instance when the global minimum is the most robust. In such cases, by employing the parallel quantum methodology of Sect. 2 that explores the entire (computational) domain of H, that methodology, as it concentrates probability will yield a global minimum with probability that increases with the recursion index. Note that the combined process specifies a recursion of the form (2.1), although the feature $H(v_{j+1}) \le H(v_j)$ may be vacated following a diffusion step. This mandates that diffusion should be exploited relatively infrequently during the recursion, and certainly not at all towards the finish. With these features and restrictions, minimization with diffusion could be compared with annealing, another candidate for quantum parallelization by methods developed here (Kirkpatrick, Gelatt and Vecchi (1983)). Augmenting the recursion will cause the location of the minimum found to be a drifted away approximation to the true location. So a final post processing step that uses the location found as a starting point is suggested.

The diffusion process is specified in Sect. 3.1. These methods apply to steepest descent, but to demonstrate their scope and versatility, we show how they apply to the relaxation dynamics of a Hopfield net; those dynamics used as a generator of a variety of candidate descent minimization problems. As a byproduct, this also demonstrates how recursive neural networks can be implemented on a quantum computer. The Hopfield net example is treated in Sect. 3.2. A formalism for averaging and of diffusion in higher dimensions is given in Sect. 3.3.

3.1 Diffusion

In this section, we specify what is meant by diffusion (in the discrete case). We shall use v to denote the independent variable. Let Δv be a vector of increments in v. Then a step of diffusion replaces $H(v)$ by $H_{diff}(v)$, where

(3.1) $$H_{diff}(v) = \lambda(H(v+\Delta v) + H(v-\Delta v)) + (1-2\lambda)H(v).$$

Here λ is an arbitrary parameter. (λ may be thought of as equal to $D\Delta t/\|\Delta v\|^2$, the familiar ratio of increments arising in the finite difference approximation to the diffusion equation and where D is the diffusion constant.) For convenience we shall write Δv as Δ.

Next define the shift operators $\boldsymbol{S}^{\pm}$ (with $\boldsymbol{S}^{+} = \boldsymbol{S}$) as

(3.2) $$\boldsymbol{S}^{\pm} H(v) = H(v \pm \Delta).$$

Let $\boldsymbol{M} = \frac{1}{2}[\boldsymbol{S} + \boldsymbol{S}^{-}]$ denote the averaging operator, and let $\boldsymbol{D}_{iff} = \boldsymbol{D}_{iff}(\lambda)$ denote the following discrete diffusion operator.

(3.3) $$\boldsymbol{D}_{iff} = 2\lambda \boldsymbol{M} + (1-2\lambda)I.$$

Then

(3.4) $$H_{diff}(v) = \boldsymbol{D}_{iff} H(v).$$

Using Taylor's theorem, (3.3) and (3.4) give

(3.5) $$H_{diff}(v) = \nabla H(v) + O(\Delta^2),$$

and so, with $G = \nabla H$ in (2.1), we see that the method of steepest descent fits our framework. Note that diffusion reduces to averaging for $\lambda = 1/2$ in which case we write H_{diff} as H_{av}. Extension to averaging and diffusion in *d* dimensions is given in Sect. 3.3.

3.2 Hopfield nets

The Hopfield net supplies a formalism for generating descent minimization algorithms for a variety of combinatorial problems, and so, illustrates the range of our methodology. We start with averaging, returning to the more general diffusion in Sect. 3.2.4. For explicit Hopfield net applications, one of which is the Traveling Salesman Problem, see Hertz, Krogh and Palmer (1991). For the problem of sorting, see Atkins (1989).

3.2.1 Hopfield dynamics

The dynamics of a Hopfield net with *m* neurons are given by (Haykin, 1999)

(3.6) $$\tau_i \frac{du_i}{dt} = -u_i + \sum_{j=1}^{m} w_{ij} g(u_j), \quad i = 1, \cdots, m,$$

where *g* is the neuronal gain function, and the *m*-vectors of neuronal inputs $u = (u_i)$ to neuron *i* and outputs $v = (v_i)$ from neuron *i* are related by

(3.7) $$v = g(u) \Leftrightarrow v_i = g(u_i), \quad \forall i.$$

w_{ij} is the synaptic weight connecting neuron *j* to neuron *i*. The total weighted input to neuron *i* is

(3.8) $$u_i = \sum_{j=1}^{m} w_{ij} v_j.$$

Now discretize the time in (3.6), setting $\Delta t = 1$ and $\tau_i = 1$ for convenience. Combining the result with (3.7), we find

(3.9) $$v(n+1)=gWv(n),$$

where $W=(w_{ij})$ is the weight matrix, and n indexes the (discrete) time. Under appropriate conditions on W (namely, $w_{ij}=w_{ji},\ \forall i,j$) and on g (namely, $g'>0$), the net's dynamics, written in the discrete form (3.9) drives the associated energy function

(3.10) $$H(v)=-\frac{1}{2}\sum_{i,j} w_{ij}v_iv_j+\sum_i\int_0^{v_i} g^{-1}(\xi)d\xi$$

to a relative minimum as is well known in the theory of Hopfield nets. So gW plays the role of G in (2.1).

3.2.2 Replacement energy $H_{av}(v)$ and gain function $h(u)$ (the case of averaging)

Let the displacement vector be $\Delta=(\Delta_1,\dots,\Delta_m)$. Then using (3.10) and specializing diffusion to be averaging, we have

(3.11)
$$\begin{aligned}2H_{av}(v)&=2\boldsymbol{M}H(v)\\&=-\frac{1}{2}\sum_{ij} w_{ij}(v_i+\Delta_i)(v_j+\Delta_j)-\frac{1}{2}\sum_{ij} w_{ij}(v_i-\Delta_i)(v_j-\Delta_j)\\&\qquad+\sum_j\left[\int_0^{v_j+\Delta_j}+\int_0^{v_j-\Delta_j}\right]g^{-1}(\xi)d\xi\\&=-\sum_{ij} w_{ij}v_iv_j-\sum_{ij} w_{ij}\Delta_i\Delta_j+\sum_j\left[\int_{-\Delta_j}^{0} g^{-1}(\xi+\Delta_j)d\xi+\int_{-\Delta_j}^{0} g^{-1}(\xi-\Delta_j)d\xi\right]\\&\qquad+\sum_j\left[\int_0^{v_j} g^{-1}(\xi+\Delta_j)d\xi+\int_0^{v_j} g^{-1}(\xi-\Delta_j)d\xi\right]\\&=2\overline{H}(v)+2J(\Delta).\end{aligned}$$

Here

(3.12) $$\overline{H}(v)=-\frac{1}{2}\sum_{ij} w_{ij}v_iv_j+\sum_j\int_0^{v_j} h_j^{-1}(\xi)d\xi,$$

where

(3.13) $$h_j^{-1}(\xi)=\frac{1}{2}\left[g^{-1}(\xi+\Delta_j)+g^{-1}(\xi-\Delta_j)\right]$$

and

(3.14) $$J(\Delta)=\frac{1}{2}\sum_{ij} w_{ij}\Delta_i\Delta_j+\frac{1}{2}\sum_j\left[\int_{-\Delta_j}^{0} g^{-1}(\xi+\Delta_j)d\xi+\int_{-\Delta_j}^{0} g^{-1}(\xi-\Delta_j)d\xi\right].$$

Note that $J(\Delta)$, being a constant independent of v may be discarded in the context of minimization. Comparing (3.10) and (3.11), we see that the Hopfield net corresponding to $\overline{H}$ is the same as the original net that corresponds to H except that the gain function g is replaced by the displacement dependent gain function h_j given in (3.13).

For clarity, we shall take all of the displacements Δ_j to be equal. Then we may drop the subscripts on the h_j and on the Δ_j. Note that if g is linear in its argument, averaging changes nothing, since then h and g are identical.

3.2.3 Solving for the replacement gain function

Since $u = g^{-1}(v)$, we have $\frac{d}{dv} g^{-1}(v) = 1/g'(u)$, and $\frac{d^2}{dv^2} g^{-1}(v) = -g''(u)/(g'(u))^3$. Then Taylor's theorem gives

(3.15) $$g^{-1}(v \pm \Delta) = g^{-1}(v) \pm \frac{\Delta}{g'(u)} - \frac{g''(u)}{(g'(u))^3}\frac{\Delta^2}{2} + \cdots.$$

Inserting this into (3.9) gives

(3.16) $$h^{-1}(v) = g^{-1}(v) - \frac{g''(u)}{(g'(u))^3}\frac{\Delta^2}{2} + \cdots.$$

This specifies an equation for determining the input-output relation

(3.17) $$v = h(u)$$

for the averaged Hopfield net. In particular, combining (3.16) and (3.17) gives

(3.18) $$u = g^{-1}(v) - \frac{g''(u)}{(g'(u))^3}\frac{\Delta^2}{2} + \cdots.$$

From this, we find the gain function h of the averaged net in terms of the gain function g of the original net. Namely,

(3.19) $$h(u) = g\left(u + \frac{g''(u)}{(g'(u))^3}\frac{\Delta^2}{2} + \cdots\right).$$

Since we are to repeat this net averaging process (equivalently, the net's gain function replacement process) recursively, we index the successive stages with n. Then denote the gain functions for the succession of averaged neural nets as $g_n(u), n = 0,1\ldots$ Then $g_0(u) = g(u)$, the original gain function, and $g_1(u) = h(u)$, the gain function of the first averaged net. Also let $\Delta(n)$ denote the displacement Δ used at the n-th stage ($\Delta(0) = 0$), and let $v(n)$ denote the output of the (averaged) neural net relevant to the n-th stage. Then from (3.19) (using Taylor's theorem), we have the following approximation to the successive input-output relations $v_n = g_n(u), n \geq 0$, in terms of $g(u)$.

(3.20) $$v_n = g(u) + 2^{n-2}\Delta^2(n) g''(u)/(g'(u))^2 + \cdots, \quad n \geq 0.$$

Coupling (3.20) and (3.18) gives the relation (2.1) to be implemented in the quantum computation.

To avoid divergence of the expression in (3.20) as $n \to \infty$, the displacement $\Delta(n)$ must be made to tend to zero as n increases. A convenient choice for achieving this is $\Delta(n) = 2^{1-\frac{n}{2}}\delta$, where δ is a constant chosen suitably small, namely so that expressions like $\delta^2 g''(u)/(g'(u))^2$ are small for all values of u that may arise in the course of running the computation. Notice that these choices of the $\Delta(n)$ may be interpreted as requiring that the average of the original energy function $H(v)$, specified in (3.8), be taken over ever smaller displacements in the independent variable v. This is a common feature of recursive minimization methods wherein the displacements toward the minimum sought (are made to) tend to zero as the location of that minimum is approached.

3.2.4 Diffusion in the Hopfield net

Now we translate the development for employing averaging with a Hopfield net to the general case of diffusion.

Referring to Sects. 3.1 and 3.2 (and writing $h(u)$ as is $h_{diff}(u)$), we see that

(3.21) $$H_{diff}(v) = 4\lambda(1-\lambda)\overline{H}_{diff}(v) + 2\lambda J(\Delta).$$

Here $J(\Delta)$ is given in (3.14), and

(3.22) $$\overline{H}_{diff}(v) = -\frac{1}{2}\sum_{ij} w_{ij}v_i v_j + \sum_j \int_0^{v_j} h_{diff}^{-1}(\xi)d\xi,$$

where

(3.23) $$h_{diff}^{-1}(\xi) = \frac{1}{4\lambda(1-\lambda)}\left[\lambda g^{-1}(\xi+\Delta) + (1-2\lambda)g^{-1}(\xi) + \lambda g^{-1}(\xi-\Delta)\right].$$

The analog of (3.16) is

(3.24) $$h_{diff}^{-1}(v) = \frac{1}{2(1-\lambda)}\left[g^{-1}(v) - \frac{g''(u)}{(g'(u))^3}\frac{\Delta^2}{2} + \cdots\right].$$

The input-output relation (3.15) becomes $v = h_{diff}(u)$, where

(3.25) $$h_{diff}(u) = g\left(4\lambda(1-\lambda)u + \frac{g''(u)}{2(1-\lambda)(g'(u))^3}\frac{\Delta^2}{2} + \cdots\right).$$

(3.25) is the analog of (3.19).

Finally the analog of the input-output relation (3.20) is

$$v(n) = 4\lambda(1-\lambda)g(u) + \frac{2^{n-2}g''(u)}{2(1-\lambda)(g'(u))^2}\Delta^2(n) + \cdots, \quad n > 0. \tag{3.26}$$

3.3 Averaging and diffusion in *d* dimensions

In this section we develop some formalism for specifying and computing averages and diffusion in *d* dimensions ($v = (v_1, \cdots, v_d)$). To proceed, define the shift operators $\boldsymbol{S}_j$, $j = 1, \cdots d$ appropriate to each coordinate ($\boldsymbol{S}_j H(v) = H(v_1, \cdots, v_{j-1}, v_j + \Delta_j, v_{j+1}, \cdots, v_d)$). Next define a grid of lattice points in *d* dimensions, G^d, where

$$G = \{0, \pm 1, \cdots, \pm d\}, \tag{3.27}$$

and a corresponding grid $v(G)$ in *d*-space. Namely,

$$v(G) = \prod_{j=1}^{d} v_j, \; v_j = \{0, \pm\Delta_j, \cdots, \pm d\Delta_j\} \quad j = 1, \cdots, d. \tag{3.28}$$

We say that the grid $v(G)$ is centered at the origin, $v = 0$. Then the shift operator,

$$\boldsymbol{S}^{\mathrm{d}}(\mathrm{G}) = \prod_{\mathrm{j=1}}^{\mathrm{d}} \sum_{\mathrm{i} \in \mathrm{G}} \boldsymbol{S}_j^i, \tag{3.29}$$

when applied to $H(v)$ delivers the sum of its values over the grid $v(G)$.

Let *A* be an array of real numbers that sum to unity. *A* has the configuration of the grid G^d. That is, *A* is a $(2d+1)^d$ array in *d* dimensions.

Now let

$$\boldsymbol{M}_A = A \otimes \boldsymbol{S}^{\mathrm{d}}(\mathrm{G}), \tag{3.30}$$

where $\otimes$ denotes the direct (element-wise) product. Then $\boldsymbol{M}_A$ is a generalized averaging operator in *d* dimensions. Note that by inserting zeros appropriately into the array *A* (maintaining the sum to unity property), the grid may be shaped into a desired stencil of points. Applying $\boldsymbol{M}_A$ to *H*(*v*) delivers a generalized average of *H*(v) over a grid of the form $v(M)$ that is centered at *v* (more precisely, over a stencil specified by *A*). Discretized diffusion operators in *d* dimensions correspond to particular choices of *A*, that is, they are special cases of $\boldsymbol{M}_A$.

REFERENCES

Atkins, M., (1989), Sorting by Hopfield Net, *International Joint Conference on Neural Networks.*

Chazan, D. and Miranker, W., (1970), A nongradiednt and parallel algorithm for unconstrained minimization. *SIAM J. Control, 2. 207-217.*

Haykin, S., (1999), *Neural Networks, A Comprehensive Foundation*, Prentice Hall, Upper Saddle River.

Hertz, J., Krogh, A., and Palmer, R., (1991), *Introduction to the Theory of Neural Computation,* Addison -Wesley, Redwood City.

Kirkpatrick, S., Gelatt, C. and Vecchi, M., (1983), Optimization by simulated annealing, Science, **220**, 671.

Kulisch, U., Miranker, W., (1981), *Computer Arithmetic in Theory and Practice,* Academic, New York.

Nielsen, M. and Chuang, I., (2000), *Quantum Computation and Quantum Information,* Cambridge.