LETTER **Communicated by Mark McDonnell**

Deep Learning with Dynamic Spiking Neurons and Fixed Feedback Weights

Arash Samadi
ars2023@med.cornell.edu
Department of Physiology, University of Toronto, Toronto, Ontario, M5S 1A8, Canada

Timothy P. Lillicrap
timothylillicrap@google.com
Google DeepMind, London, EC4A 3TW, U.K.

Douglas B. Tweed
douglas.tweed@utoronto.ca
Department of Physiology, University of Toronto, Toronto, Ontario, M5S 1A8, Canada, and Centre for Vision Research, York University, Toronto, Ontario, M3J 1PC, Canada

Recent work in computer science has shown the power of deep learning driven by the backpropagation algorithm in networks of artificial neurons. But real neurons in the brain are different from most of these artificial ones in at least three crucial ways: they emit spikes rather than graded outputs, their inputs and outputs are related dynamically rather than by piecewise-smooth functions, and they have no known way to coordinate arrays of synapses in separate forward and feedback pathways so that they change simultaneously and identically, as they do in backpropagation. Given these differences, it is unlikely that current deep learning algorithms can operate in the brain, but we that show these problems can be solved by two simple devices: learning rules can approximate dynamic input-output relations with piecewise-smooth functions, and a variation on the feedback alignment algorithm can train deep networks without having to coordinate forward and feedback synapses. Our results also show that deep spiking networks learn much better if each neuron computes an intracellular teaching signal that reflects that cell's nonlinearity. With this mechanism, networks of spiking neurons show useful learning in synapses at least nine layers upstream from the output cells and perform well compared to other spiking networks in the literature on the MNIST digit recognition task.

A.S. is now at Weill Cornell Medical College, New York, NY.

Neural Computation **29, 578–602** (2017)
doi:10.1162/NECO_a_00929

1 Introduction

Recent results in computer science have revealed the power of deep learning (Bengio, 2009; Farabet, Couprie, Najman, & LeCun, 2013; Hinton, Osindero, & Teh, 2006; Hinton & Salakhutdinov, 2006; Krizhevsky, Sutskever, & Hinton, 2012; Schmidhuber, 2015). But it is unclear which insights from this work apply to the brain because current algorithms for deep learning are designed for networks of very simple neurons. Real neurons are different in at least three crucial respects. First, real neurons communicate by streams of voltage spikes, or action potentials, whereas neurons in most artificial deep networks have continuous, graded outputs. Second, real neurons are dynamic in the sense that their activity at any moment depends not only on their inputs and synaptic weights at that moment but also on their inputs and weights over the last few milliseconds (Eliasmith & Anderson, 2002). And third, real neurons almost certainly lack weight transport, meaning they cannot send each other detailed information about the weights (i.e., strengths) of all their synapses in the way that is required in current algorithms for deep learning (Chinta & Tweed, 2012; Crick, 1989; Grossberg, 1987; Kolen & Pollack, 1994; Levine, 2000; Rolls & Deco, 2002; Stork, 1989).

Of course, these three aspects of real neurons are not necessarily flaws or shortcomings, as spiking and dynamics may bring computational advantages (Hinton, 2016; Maass & Markram, 2004; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). And of course real neurons differ from artificial ones in other ways besides these three. But these three properties do suggest that the computations underlying biological learning must differ from those of current deep learning algorithms in computer science. And the same three issues are also relevant to networks embodied in very large-scale integrated (VLSI) circuits (Azghadi, Iannella, Al-Sarawi, Indiveri, & Abbott, 2014) and field-programmable gate arrays (FPGA) (Neil & Liu, 2014). We describe the computational problems raised by these three issues and then show how those problems can be solved.

To begin with spiking and dynamics, the key issue is that in real neurons, spiking depends on current and past inputs and synaptic weights, whereas in the artificial neurons of most nonrecurrent deep networks, output depends only on existing inputs and parameters (weights and biases). In the best-performing algorithms for deep learning, each neuron receives a drive v, which depends on its inputs and parameters. The neuron emits a signal a, which is a function of v—$a = f(v)$ where f is called the activation function. This function matters because deep learning algorithms rely on the backpropagation algorithm, which works by computing the derivative of the network's current output error with respect to the weights of all the synapses in the net, and these derivatives depend on the derivative of a with respect to v, da/dv (Ciresan, Meier, Gambardella, & Schmidhuber, 2010; Hinton et al., 2006; Krizhevsky et al., 2012; Sermanet et al., 2013; Srivastava et al., 2014).

But in a dynamic neuron, there is no function relating the present a to the present v, and so there is no derivative da/dv. Of course, a real neuron's outputs are still related to its inputs, but not by a function in the mathematical sense, which implies that any one input v is always paired with the same output a. One could tackle this problem by working from the fact that the current a is a function of current and past vs. But that approach increases the dimensionality of the problem. In this letter, we apply a simpler method, which uses, in place of the activation function, the function relating the expected value of a to v (O'Connor, Neil, Liu, Delbruck, & Pfeiffer, 2013).

The third difference we are considering—the brain's lack of weight transport—sets up further barriers to the backpropagation algorithm. Backpropagation works by sending error derivatives along a feedback path that drives learning in the forward part of the network. But those derivatives depend on the weights of the synapses in the forward path, which means that the feedback circuits that drive learning must have information about those weights. In the brain, there is no known way for them to get that information.

Specifically, backpropagation continually adjusts the synaptic weights in the feedback path so that each one stays equal to its corresponding weight in the forward path, with the result that the matrix of feedback weights in each layer equals the transpose of the matrix of forward weights in that layer (in convolutional networks, there is more complicated coordination of weights). In a computer, it is easy to set each feedback weight equal to the appropriate forward-path weight at each time step. But the brain, lacking weight transport, has no mechanism to coordinate large numbers of evolving synapses on different neural pathways in this way (Chinta & Tweed, 2012; Crick, 1989; Grossberg, 1987; Kolen & Pollack, 1994; Levine, 2000; Rolls & Deco, 2002; Stork, 1989).

Surprisingly, though, it has recently been found that layered networks can learn even if synapses in the feedback path are not coordinated at all with those in the forward path but are instead frozen at random values. This algorithm is called *feedback alignment*, because in it, the forward-path synapses evolve to resemble the fixed synapses in the feedback circuits, so that in the end, it is as if those feedback synapses had been set equal to the forward ones as required by backpropagation. The reasons that feedback alignment works are not fully understood, but what is known is described in Lillicrap, Cownden, Tweed, and Akerman (2014) and Hinton (2016).

Here we show that a variant of feedback alignment can drive deep learning in dynamic, spiking networks. Connections between our results and other recent discoveries in the field of spiking networks (Beyeler, Dutt, & Krichmar, 2013; Bohte, Kok, & La Poutre, 2002; Brader, Senn, & Fusi, 2007; Diehl & Cook, 2015; Diehl et al., 2015; Eliasmith et al., 2012; Henderson, Gibson, & Wiles, 2015; Jimenez Rezende & Gerstner, 2014; Maass & Markram, 2004; Neftci, Das, Pedroni, Kreutz-Delgado, & Cauwenberghs, 2014; Neil & Liu, 2014; O'Connor et al., 2013) are laid out in section 4.

2 Methods

2.1 Neurons. We use a mathematical model called the leaky-integrate-and-fire (LIF), neuron (Eliasmith & Anderson, 2002), which is popular because it strikes a useful balance between realism and complexity.

At any moment, an LIF neuron has a drive v, which depends on its bias current, b; its inputs $a_{(in)j}$ (where the index j runs from 1 to the number of inputs); and its synaptic weights, W_j (Eliasmith & Anderson, 2002). If the neuron gets its inputs from other spiking cells, then the $a_{(in)j}$ are all 0s and 1s, where 1 means a spike and 0 means the absence of a spike. If the neuron gets inputs from sensory receptors or nonspiking neurons, then the $a_{(in)j}$ may be other real numbers besides 0s and 1s. In either case, drive is determined by the equation

$$v = \Sigma_j W_j a_{(in)j} + b. \tag{2.1}$$

An LIF neuron also has an axon hillock potential, h, which determines when the cell fires. This h depends on v: it is driven upward by positive v and downward by negative v, and it also has an intrinsic tendency to drain away toward the cell's resting potential, which for convenience we call 0. In other words h is updated by passing v through a leaky integrator,

$$\Delta h = (v - h)\Delta t/\tau, \tag{2.2}$$

where Δt is the time step used in the numerical integration and τ is the integrator's time constant; in our simulations, $\Delta t = 0.25$ ms and $\tau = 2$ ms (Eliasmith & Anderson, 2002).

The neuron fires an action potential whenever h reaches a threshold value h_{th}, equal to 0.4, on a scale where the peak potential during a spike has the value 1; in neurophysiology, membrane potentials are usually expressed in mV and average about -70 for resting, -30 for the threshold, and $+30$ for a spike, but the 0-to1 scale, in units of decivolts, is convenient and does not change anything essential (Eliasmith & Anderson, 2002). So we have

$$h > h_{th} \rightarrow a = 1. \tag{2.3}$$

That is, the cell's output snaps to 1, meaning that the cell is spiking, when its hillock potential crosses the threshold. It stays at 1 for 1 ms and then falls back to 0 when the spike is over. During that 1 ms, the hillock potential h does not obey equation 2.2 but stays pinned at 0; that is, the neuron is refractory during the 1 ms of the action potential. Figure 1 illustrates the relations between v, h, and the spike output a.

2.2 Backpropagation and Feedback Alignment. In the backpropagation learning algorithm, synaptic weights evolve according to the formula

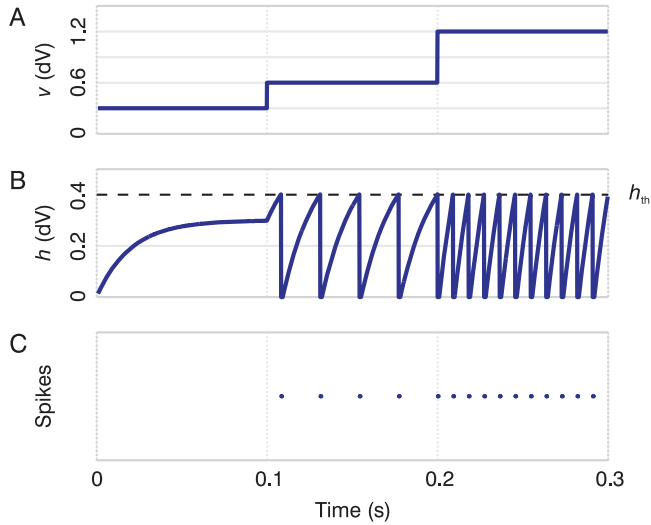

Figure 1: Relations between drive and activity in a dynamic spiking neuron. (A) Drive v holds at 0.3 decivolts (dV), steps up to 0.6, and then steps to 1.2. (B) When v is small, the hillock potential h rises to an equilibrium below the spiking threshold h_{th} (dashed black line) and the cell never spikes. When v is larger, h rises faster toward a higher equilibrium, and so hits the threshold, causing (C) spikes.

$$\Delta W_{(n)ij} = -\eta_{(n)} \delta_{(n)i} a_{(n-1)j}. \tag{2.4}$$

That is, the change in the weight $W_{(n)ij}$ connecting cell j in layer $n-1$ to cell i in layer n depends on three factors: a positive number called the learning rate constant $\eta_{(n)}$, the activity $a_{(n-1)j}$ of cell j in layer $n-1$, and a feedback signal $\delta_{(n)i}$.

The feedback signal is

$$\delta_{(n)i} = \Sigma_k W_{(n+1)ki} \delta_{(n+1)k} \; da_{(n)i}/dv_{(n)i}. \tag{2.5}$$

Here $v_{(n)i}$ is the drive to neuron i of layer n, which depends on that cell's inputs and incoming synaptic weights; $a_{(n)i}$ is the activity of that cell, which in most backpropagation nets is a function of its drive, $v_{(n)i}$; $da_{(n)i}/dv_{(n)i}$ is the derivative of that function; $\delta_{(n+1)k}$ is the feedback signal to cell k in layer $n+1$; and $W_{(n+1)ki}$ is the synaptic weight from cell i in layer n to cell k in layer $n+1$ in the forward path of the network.

The key point is that in backpropagation, all the variables of the form $W_{(n+1)ki}$ play a double role: they represent the synaptic weights in the forward path, but they also appear in equation 2.5, where they multiply signals

$\delta_{(n+1)k}$ in the feedback path. In other words, each $W_{(n+1)ki}$ acts as a synapse in two different neural pathways. In the brain the forward and feedback synapses are of course physically distinct, which means that for backpropagation to run in the brain, each synapse in the feedback path would have to always stay equal to its specific corresponding synapse in the forward path, even though the latter synapse is constantly evolving as the network learns. This is the weight transport problem, which is one of the main reasons backpropagation is not considered feasible in the brain (Chinta & Tweed, 2012; Crick, 1989; Grossberg, 1987; Kolen & Pollack, 1994; Levine, 2000; Rolls & Deco, 2002; Stork, 1989).

This problem is solved in the feedback alignment algorithm by adopting equation 2.4 from backpropagation but altering its feedback formula, equation 2.5, to give

$$\Delta W_{(n)ij} = -\eta_{(n)}\delta_{(n)i}a_{(n-1)j}, \quad \delta_{(n)i} = \Sigma_k B_{(n)ik}\delta_{(n+1)k}\ da_{(n)i}/dv_{(n)i}, \tag{2.6}$$

where $B_{(n)ik}$ is the synapse from the feedback cell carrying teaching signal $\delta_{(n+1)k}$ to the feedback cell carrying teaching signal $\delta_{(n)i}$ (Lillicrap et al., 2014). Crucially, these $B_{(n)ik}$ are all fixed, random weights. In contrast with backpropagation, there is no need to keep feedback synapses equal to forward-path synapses, and therefore no weight transport problem. So feedback alignment removes this barrier to backpropagation in the brain (Crick, 1989; Grossberg, 1987; Stork, 1989).

2.3 Broadcast Alignment. Equation 2.6, like equation 2.5, requires that feedback neurons multiply signals together: the neuron whose output is $\delta_{(n)i}$ must compute the product of two factors, $\Sigma_k B_{(n)ik}\delta_{(n+1)k}$ and $da_{(n)i}/dv_{(n)i}$ In the brain, those two factors would have to be represented by streams of action potentials on at least two separate axons, so multiplying them would mean multiplying variables coded in spike trains. There may be real neurons that do this, but LIF neurons cannot; they can only apply synaptic weights to incoming spikes and summate, as shown in equation 2.1. For that reason we will focus on a simplified version of equation 2.6, which avoids spike signal multiplication and which we call broadcast alignment. In this scheme, the feedback signals are

$$\delta_{(n)i} = \Sigma_k B_{(n)ik}e_k. \tag{2.7}$$

Here e_k is the output error for neuron k in the final layer of the network. So now every neuron in every layer receives the same feedback signals e_k, weighted by random, constant synapses $B_{(n)ik}$. Broadcast alignment needs fewer feedback neurons than backpropagation or feedback alignment: rather than propagating error signals through many layers, broadcast

alignment uses just one layer of feedback cells, which code the errors e_k and project directly to the learning cells.

Each neuron receives its feedback δ as a stream of spikes, and from it computes a single, scalar, intracellular teaching signal ι (iota) by weighting δ by a factor proportional to the derivative da/dv of its activation function; for cell i of layer n, we have

$$\iota_{(n)i} = \eta_{(n)}(da_{(n)i}/dv_{(n)i})\delta_{(n)i}. \tag{2.8}$$

This signal then influences the change in each of the cell's synaptic weights:

$$\Delta W_{(n)ij} = -\iota_{(n)i}a_{(n-1)j}. \tag{2.9}$$

We consider the biological implications of equations 2.8 and 2.9 in sections 2.6 and 4.

Feedback alignment and broadcast alignment differ in their handling of the derivatives da/dv. Feedback alignment includes multilayer information about da/dv in its feedback signals; for example, the formula for $\delta_{(n)i}$ in equation 2.6 contains the nth layer derivative term $da_{(n)i}/dv_{(n)i}$ and also includes $\delta_{(n+1)k}$, which in turn was computed using information about the $n+1$st layer derivatives $da_{(n+1)k}/dv_{(n+1)k}$, and so on through all the layers. That is, in feedback alignment as in backpropagation, the feedback signals accumulate information about the derivatives da/dv of all downstream neurons. Broadcast alignment, in contrast, omits da/dv from its feedback signals $\delta_{(n)i}$ in equation 2.7, but incorporates $da_{(n)i}/dv_{(n)i}$ into its intracellular learning mechanism in equation 2.8. Therefore, learning in any one neuron is based solely on the derivative of its own activation function and gets no information about any downstream da/dv. So broadcast alignment delivers less information to each learning neuron than backpropagation or feedback alignment does. In section 3 we show that it learns very effectively nonetheless.

We also looked at whether this simplification could be pushed one step further by omitting all information about da/dv from the learning algorithm, that is, by combining derivative-free feedback, equation 2.7, with the derivative-free intracellular process, equation 2.4. But we will show that this derivative-free algorithm does not learn nearly as well as broadcast alignment. That is, the minimal derivative information in equation 2.8 is very useful for deep learning.

2.4 Dynamics and Activation. Broadcast alignment requires that learning neurons have information about the derivative da/dv of their activation function. But LIF neurons have no such function. The leaky integrator in equation 2.2 makes LIF cells dynamic: the values of h and therefore a depend not only on the drive v at this moment but also on the values v has had over

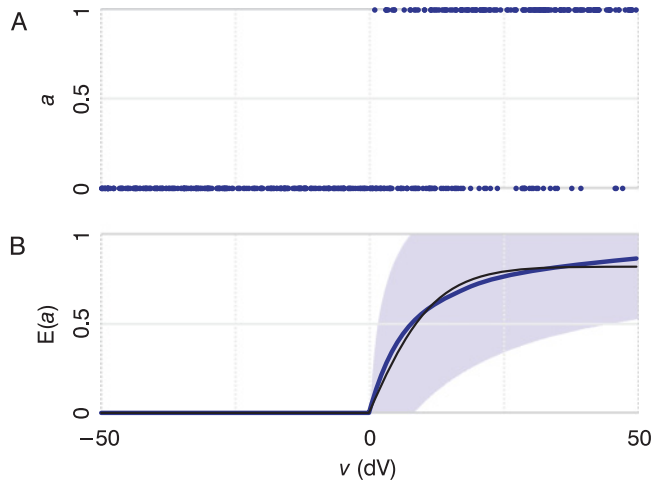

Figure 2: In a dynamic spiking neuron, activity a is not a function of drive v. (A) For any drive $v > 0$, its activity a can be either 0 or 1. (B) The expected value of a is a piecewise smooth function of v (blue curve), well approximated by equation 2.10 (black curve). For further details see the main text and appendix A.

the last few milliseconds. Plotting a versus v, as in Figure 2A, illustrates the problem. The blue dots show as for 500 random values of v: for any $v > 0$, a can be either 0 or 1 depending on recent history, and the graph does not resemble any smooth curve with a derivative da/dv.

But if we instead plot the average or expected value of a given v, which we will write as $\mathrm{E}(a)$ (blue curve in Figure 2B), then the graph is smoother (for a similar approach, see, e.g., O'Connor et al., 2013). The formula for this $\mathrm{E}(a)$ curve is $t_{\mathrm{ref}}/(t_{\mathrm{ref}} + \tau \log(v/(v - h_{\mathrm{th}})))$ for $v > h_{\mathrm{th}}$ (and $\mathrm{E}(a) = 0$ for $v \leq h_{\mathrm{th}}$), where τ is the time constant of 0.02 s from equation 2.2 and t_{ref} is the duration of the neuron's refractory period, which in our case coincides with the action potential duration of 1 ms (see Koch, 1999, and Eliasmith & Anderson, 2002, for equivalent formulas). We will, however, approximate the curve using a function of the form

$$\mathrm{E}(a) \approx \max(0, c_1 \tanh(c_2 v)). \tag{2.10}$$

The best-fitting function, plotted as a thin black curve in Figure 2B, has coefficients $c_1 = 0.82$ and $c_2 = 0.08$, and does resemble the $\mathrm{E}(a)$ graph. We chose tanh as the basis of our fitting function because it is a popular bounded activation function in the machine learning literature. Other function forms based on logarithms are also possible and actually yield slightly better fits to $\mathrm{E}(a)$ because they saturate more slowly, but in our preliminary experiments,

spiking networks based on these alternative functions learned no better or worse than tanh-based ones. The question is whether equation 2.10 can play the role of the activation function in deep learning, given that the standard deviation of a about E(a) is so wide (light blue region in Figure 2B). We address this question with simulations in section 3.

2.5 Error Feedback. All neurons in our networks, except first-layer neurons, which simply carry input signals, learn by adjusting their weights and biases based on feedback. Learning is driven by spiking error signals e_k, which are the differences between the desired outputs of the network y_k^* and its actual outputs y_k. For instance if a network has three layers, the y_k are the activities of the third-layer neurons, $y_k = a_{(3)k}$. Both y_k and y_k^* always consist of 0s and 1s (where, again, 1 means a spike and 0 means no spike), and therefore the error signals $e_k = y_k - y_k^*$ consist of 0s, 1s, and -1s. Because real neurons cannot produce negative spikes, we propose two populations of error feedback neurons, all of them carrying signals of 0 or 1, but half of them being inhibitory cells, whose spikes signal negative errors.

This scheme does not imply the existence of any unphysiological "supervisor" guiding the learning. For convenience, we speak of desired outputs y_k^*, as in the machine learning literature, but the network need not receive any y_k^* signals. All that matters is that it get signals representing the errors e_k. For instance learning circuits in the cerebellum adjust the processing in the vestibulo-ocular reflex so that when the head moves, the eyes counterrotate in the head at just the right velocity to keep the visual images stable on the retinas. This learning is driven by error signals from the visual system, which code retinal-image slip velocity; retinal-slip signals provide a useful error vector, with no need for any signals coding desired eye velocities (Lisberger, 1994). Another source of teacher signals that seems plausible physiologically is the networks' own inputs, as in the artificial networks known as autoencoders, which learn useful representations of sense data based on error signals that are differences between the networks' own inputs and outputs (Bengio, 2009; Hinton, 2016).

2.6 Learning Mechanism. To implement equation 2.8 in LIF neurons, we replace the activation function by the approximate E(a) function in equation 2.10, and da/dv by the derivative of that function, which is 0 whenever the drive $v \leq 0$ and is otherwise $c_1 c_2 \operatorname{sech}^2(c_2 v)$, which means that for $v > 0$,

$$\iota_{(n)i} = \eta_{(n)} \operatorname{sech}^2(c_2 v_{(n)i}) \delta_{(n)i} \tag{2.11}$$

(the $c_1 c_2$ term in the derivative is omitted because scaling is handled by $\eta_{(n)}$). As the notation $\eta_{(n)}$ suggests, all the neurons in any one layer n have the same learning rate constant $\eta_{(n)}$. It is possible that real neurons may have

individual, adaptive η's as do the cells in many artificial nets, but we do not explore that option here. Each layer's $\eta_{(n)}$ is inversely proportional to the number of inputs onto each of its neurons (e.g., for second-layer neurons in MNIST trials, $\eta_{(2)}$ is 1/784). There are many possible ways of setting ηs, but this method worked well in our simulations. The rationale for dividing by the number of inputs (e.g., 784) is that it helps keep all neurons in the network learning at about the same rate, whereas without the division, all weights evolve at about the same rate.

For $v > 0$, the sech^2 is a simple, indeed monotonic function that slopes down from its peak at 0 like the right half of a gaussian. Hence, the sech^2 term in equation 2.11 means that these neurons are more responsive to error signals when they are less excited. Biologically, it implies that some intracellular agent of synaptic change varies its activity as a function of the cell's overall drive. We know of no cell-biological evidence for or against such a dependence, but it is not implausible, and it does greatly improve learning, as we will show. For that reason, we use equation 2.11 in all our spiking network simulations; that is, we propose that each learning neuron computes its own ι based on its error feedback δ and its drive v.

It is convenient also to let ι drive adjustments in the neurons' bias currents, as well as in their synapses:

$$\Delta b_{(n)i} = -\iota_{(n)i}. \tag{2.12}$$

This feature makes the network a more flexible learner (though the same can be achieved by weight adjustments alone if we add just one more neuron to each nonoutput layer).

We can further improve learning by adding momentum (Sutskever, 2013), which means that the weight adjustment depends in part on how the weight changed in the previous time step:

$$\Delta W_{(n)ij} = \mu \Delta W_{(n)ij} - \iota_{(n)i} a_{(n-1)j}, \tag{2.13}$$

where μ is a number between 0 and 1. In the simulations presented in this letter, we set $\mu = 0.9$, as this is a common value in the machine learning literature, though we have observed that the exact value is not critical, and spiking networks without momentum also learn well (see section 4). Biologically, momentum means that the processes underlying synaptic plasticity persist for a few milliseconds after the error signals that trigger them. Momentum can be applied in an analogous way to Δb.

Our proposed learning mechanism, equations 2.7 to 2.9 and 2.12, is summarized in pseudocode form in Table 1.

2.7 Simulations. In all simulations, we computed dynamics by Euler integration with a time step Δt of 0.25 ms. Network Ws and bs were initialized so that the vs of all neurons in all layers had a mean of 8 and a

Table 1: Pseudocode for Broadcast Alignment in an LIF Network.

time step $\Delta t \leftarrow 0.25$ ms, time constant $\tau \leftarrow 20$ ms,
threshold $h_{\text{th}} \leftarrow 0.4$, activation-function fitting constant $c_2 \leftarrow 0.08$

initialize forward weights $W_{(n)ij}$, biases $b_{(n)i}$, feedback weights $B_{(n)ik}$,
drives $v_{(n)i}$, hillock potentials $h_{(n)i}$, activities $a_{(n)i}$,
times since refractory periods began $ref_{(n)i}$,
and learning rate constants $\eta_{(n)}$

for each example
 sample inputs $a_{(1)i}$ and desired outputs y_k^*
 for time $t = 0 \ldots 100$ ms step Δt

  for each layer n in the network
   for each cell i in the layer
    $v_{(n)i} \leftarrow \Sigma_j W_{(n)ij} a_{(n-1)j} + b_{(n)i}$ // drive
    if $ref_{(n)i} > 0$, $ref_{(n)i} \leftarrow ref_{(n)i} + \Delta t$ // how long has the cell been refractory?
    if $ref_{(n)i} > 1$ ms, $ref_{(n)i} \leftarrow 0$ // end refractory period
    if $ref_{(n)i} > 0$, $h_{(n)i} \leftarrow 0$ else $h_{(n)i} \leftarrow h_{(n)i} + (v_{(n)i} - h_{(n)i})\Delta t/\tau$ // hillock potential
    if $h_{(n)i} > h_{\text{th}}$, $ref_{(n)i} \leftarrow \varepsilon$ (any tiny positive number) // start a refractory period
    if $ref_{(n)i} > 0$, $a_{(n)i} \leftarrow 1$ else $a_{(n)i} \leftarrow 0$ // activity
   end for
  end for

  if $t > 20$ ms
   $e_k \leftarrow a_{(n\_layers)k} - y_k^*$ // error
   for each layer n from output back to layer 2
    for each cell i in the layer
     $\delta_{(n)i} \leftarrow \Sigma_k B_{(n)ik} e_k$ // feedback
     if $v_{(n)i} > 0$, $da_{(n)i}/dv_{(n)i} \leftarrow \text{sech}^2(c_2 v_{(n)i})$ else $da_{(n)i}/dv_{(n)i} \leftarrow 0$
     $\iota_{(n)i} \leftarrow \eta_{(n)} \delta_{(n)i} da_{(n)i}/dv_{(n)i}$ // intracellular teaching signal
     $W_{(n)ij} \leftarrow W_{(n)ij} - \iota_{(n)i} a_{(n-1)j}$ // weights
     $b_{(n)i} \leftarrow b_{(n)i} - \iota_{(n)i}$ // bias
    end for
   end for
  end if
 end for
end for

standard deviation of 10, because with these values, the neurons' activity is spread out over the middles of their operating ranges, as shown in section 3 in Figure 2B (see appendix B for details of this initialization).

During training, we used minibatches of 100 examples. It seems unlikely that the brain uses minibatches, but using them in our experiments reduced the computer run times and did not alter anything essential in the proposed learning model.

As in other learning studies with dynamic neurons, each input was presented for a brief interval of simulated time (100 ms in our case) rather

than for a single time step, as is done with static, graded neurons. And the network did not adjust its Ws or bs until it had been viewing an image for 20 ms. Similarly during testing, we ignored the network's outputs for the first 20 ms; we averaged its output activity vector over the remaining 80 ms and took that average as the network's answer. One motivation for these numbers is that humans need about 100 ms of viewing time to recognize objects in pictures.

In MNIST trials, performance was assessed in the usual way: the network was regarded as giving the correct answer when the appropriate output neuron was more active than all the others. For instance, when the handwritten digit is a 3, then the fourth of the 10 output neurons should be spiking and the other 9 should all be silent, so the output was considered correct when the fourth neuron produced more spikes than any of the others during the 80 ms answering period.

3 Results

3.1 Performance in Nonspiking Networks. First we tested our candidate deep learning algorithm, broadcast alignment, against three other methods: derivative-free learning, feedback alignment and backpropagation. These last two algorithms cannot run on LIF neurons, and therefore the tests of all four were run on networks of nonspiking neurons. Although the neurons were nonspiking, their activation function equaled the approximate activation function of LIF neurons, given in equation 2.10, for better comparison with the spiking neuron results in section 3.2.

In all these tests, the learning network had the same deep and narrow structure, with 2 input neurons, 2 output neurons, and 8 hidden layers of 10 neurons each. The task of the learning network was to match the outputs of a nonspiking teacher, or target, network. The target network was also deep and narrow, again with 10 layers and 2 input and 2 output neurons, to create tasks where deep learning was likely to be useful. To make the tasks more challenging, the target net had different types of neurons than the learner in all layers but the first: the 8 hidden layers each consisted of 2 nonrectified tanh cells, and the output layer had two nontanh, one-hot output cells; that is, the only possible outputs were (1, 0) and (0, 1). The probabilities of these two outputs were always close to equal, that is, always within 0.001 of 0.5.

Each algorithm was tested 500 times, each time with new, random weights in the target network and new, random initializations of the learning network, to present the learners with a large and varied set of tasks.

We ran these 500 tests on each of nine versions of each algorithm, which differed in their depth of learning. For instance, all tests of backpropagation ran on the 10-layer learning nets described above, but in the depth-1 version, learning was restricted to the synapses in the tenth (i.e., the output) layer of the net, and all upstream synapses stayed fixed at their initial values. In the depth-2 version, synapses in the last two layers were adjusted, and so

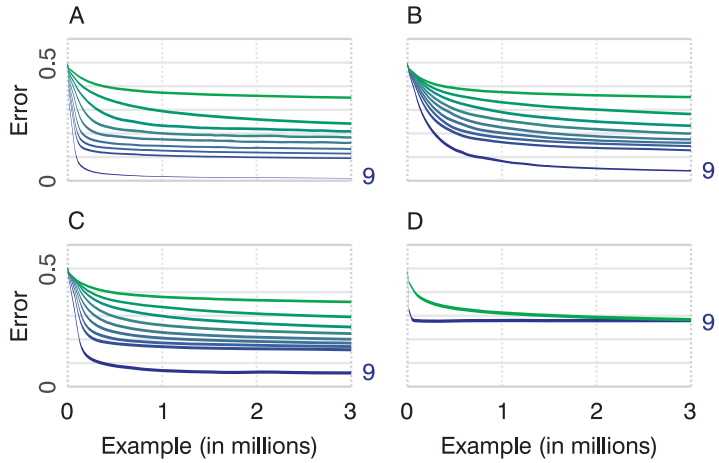

Figure 3: Comparing algorithms' depth of learning in nonspiking networks. In each panel, 10-layer nets learn to mimic other 10-layer nets. The uppermost, bright green curve shows depth-1 learning, where only output-layer synapses adapt. The lowermost, blue curve (labeled "9" on the right) shows depth-9 learning, where all 9 layers of synapses adapt. Other curves show intermediate depths. (A) With the backpropagation algorithm, the depth-9 curve lies well below all the others, showing that useful teaching signals reached even the deepest synapses. (B) The same holds for feedback alignment and (C) broadcast alignment (D) but not for derivative-free learning, where depth-9 was no better than depth-1.

on down to depth-9, the deepest possible version where all the synapses in the network were adjusted. The point of these comparisons was to see how far upstream each algorithm was able to deliver useful teaching signals.

Figure 3A shows the results for backpropagation. Each of the nine curves shows the performance error, averaged over 500 tests, for one of the depth versions of the algorithm: the top-most, bright green curve for the shallowest, or depth-1 version; the bottom blue curve for the deepest, depth-9, version; and the curves in between for the seven intermediate depths. Each curve is centered on the mean of its 500 trials, and its thickness equals 2 standard errors of the mean. Trial-to-trial variance was large because each trial used a different target function, but after 500 trials, the standard errors were small enough that the nine bands are distinctly separate. In particular, the lowest of the nine learning curves lies well below the second-lowest, showing that depth-9 learning was better than depth-8. This finding means that backpropagation delivered useful teaching signals all the way to the deepest layer of synapses in the network.

Figure 3B shows that for feedback alignment also, depth-9 learning was clearly better than depth-8. Figure 3C shows the same for broadcast

alignment. That is, these two algorithms also delivered useful teaching signals to the deepest parts of the net. Their error rates, though, were slightly higher than those of backpropagation: in this class of tasks, backpropagation was slightly better than feedback alignment, which in turn was slightly better than broadcast alignment.

Figure 3D shows that with the derivative-free algorithm, depth-9 learning was no better than depth-1 on average by the ends of the trials (curves for only those two depths are shown, to reduce clutter). The deeper version was faster and so gave better results early in the trials (near the left sides of the graphs). But neither worked as well as the deeper versions of the other three algorithms.

In summary, of the two candidate deep-learning algorithms compatible with LIF neurons, broadcast alignment and derivative-free learning, the former worked much better than the latter. Therefore, we chose broadcast alignment for implementation in spiking nets.

3.2 Broadcast Alignment in Spiking Networks. We tested an LIF version of broadcast alignment on the same task as in section 3.1 The target network was identical to that in section 3.1. The learning net had the same structure as in section 3.1 except that it contained only spiking neurons after the input layer. That is, in the learning network, the two neurons of the first layer represented sensory receptors and so had graded activity—their activities $a_{(1)k}$ were real numbers, not necessarily 0s or 1s. All other neurons in the learning network were of the LIF type—all neurons in forward layers 2 through 10 and all the feedback neurons. We ran 100 trials, with a different target function in each trial.

Figure 4 shows that this LIF version learned about as well as the nonspiking version of broadcast alignment in Figure 3C, in particular, that it also delivered useful teaching signals all the way to the deepest parts of the network.

3.3 High Dimensions. To show that the same principles still hold in higher-dimensional problems, we trained networks to recognize the handwritten digits in the MNIST database (LeCun, Bottou, Bengio, & Haffner, 1998). Again we started with nonspiking networks so we could compare all four algorithms: backpropagation, feedback alignment, broadcast alignment, and derivative-free learning. We considered two networks. One had three layers (including the input layer), with 784 input neurons representing the input image (i.e., the grayscale values of a 28-by-28 array of pixels), then 1000 neurons in the second layer, and 10 in the output layer. The other network had four layers (including input), with 784, 630, 370, and 10 neurons. We ran three trials of each algorithm in each architecture, and again we tested different depths of learning.

Table 2 summarizes the results. In the three-layer network with depth-2 learning (i.e., adjusting both layers of synapses), backpropagation correctly

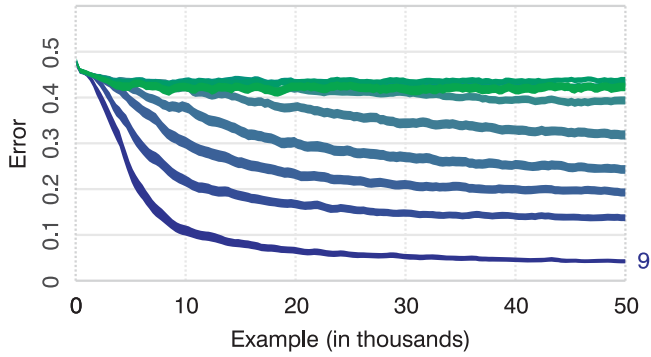

Figure 4: Spiking networks are capable of deep learning by broadcast alignment. Tasks and plotting are as in Figure 3. The bottom curve, representing depth-9 learning, lies well below the other curves, which means that useful teaching signals reached the deepest synapses in the net.

Table 2: Mean MNIST Scores of Nonspiking and Spiking (LIF) Networks.

Algorithm	Network Depth	Learning Depth	Score
BP, FA, BA	3	1	95.98
DF	3	1	95.29
BP	3	2	98.56
FA	3	2	98.42
BA	3	2	97.67
DF	3	2	96.12
BP	4	3	98.60
FA	4	3	98.22
BA	4	3	97.64
DF	4	3	95.62
LIF-BA	3	1	90.49
LIF-BA	3	2	96.02
LIF-BA	4	3	97.05

Notes: BP: backpropagation; FA: feedback alignment; BA: broadcast alignment; DF: derivative-free learning. The first 10 rows show results of nonspiking networks; the last 3, LIF networks.

classified 98.56% (mean over the three trials) of the 10,000 images in the test set; feedback alignment managed 98.42%; broadcast alignment 97.67%; and derivative-free learning 96.12%. In the same three-layer network but with depth-1 (i.e., shallow) learning, backpropagation, feedback alignment, and broadcast alignment all managed 95.98% (because these three algorithms are identical in this setting), and derivative-free learning 95.29%. So the key finding was that derivative-free learning was again scarcely better than

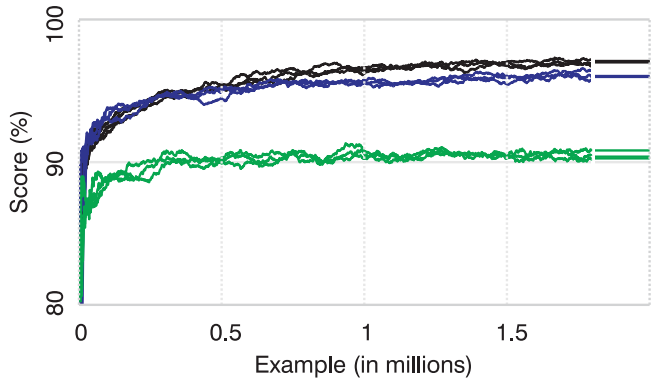

Figure 5: Deep spiking networks learn to read MNIST handwritten digits. Four-layer networks (uppermost curves, in black) outperform 3-layer ones (blue), which outperform 3-layer ones where only the output layer learns (green). Horizontal bars at the right show final scores on the full 10,000-element test set.

shallow learning, whereas broadcast alignment was again able to deliver useful teaching signals to upstream synapses.

None of the algorithms did appreciably better in the four-layer network. Backpropagation managed 98.60%, feedback alignment 98.22%, broadcast alignment 97.64%, and derivative-free 95.62. Most likely backpropagation was at the limit of what can be achieved without some form of regularization, such as convolution, dropout, or data augmentation. The others might have done better with devices such as cross-entropy loss and annealing, but we avoided those methods because they would have been complicated or controversial to include in the LIF network.

Turning now to the LIF networks, the three-layer net running broadcast alignment managed an average score of 96.02%, as shown by the blue curves in Figure 5. Specifically, these three curves depict three runs. In each run, the network learned from the 60,000 images in the MNIST training set. After every 1000 training examples, the network was tested on 100 test examples—100 images randomly drawn from a test set of 10,000 images that were never used for training, only for assessment. These test scores are plotted in the graph to show the network's improvement. After 1.8 million training examples (30 passes through the training set), we tested the network on all 10,000 images in the test set and plotted its score as a horizontal line at the right side of the plot, though the three lines, for the three runs, are too close together to distinguish in the graph: they range from 95.97% to 96.11%.

With depth-1 learning (i.e., when synaptic adjustment was restricted to the third layer of the network), performance was not as good. The three runs (green curves in Figure 5) achieved final scores ranging from 90.26%

to 90.82%, mean 90.49%. So the 96% achieved in the earlier tests (the blue curves) depended on synaptic adjustments in the upstream, second layer.

We also tested the four-layer network of 784, 630, 370, and 10 neurons. It had the same total number of neurons as the three-layer network, but fewer synaptic weights and far fewer cells and synapses in the shallower parts—the last and second-last layers. Nevertheless, it outperformed the three-layer version, achieving scores in the range 96.99% to 97.09% in its three runs (black curves in Figure 5), for a mean of 97.05%.

4 Discussion

We have shown that dynamic spiking networks can learn by applying a variant of the feedback alignment algorithm and replacing its factor da/dv with the derivative of E(a). Deeper networks learn better than shallower ones, showing that with this method, useful teaching signals reach upstream layers.

Using the algorithm described in equations 2.7 to 2.9, 2.12, and 2.13, our four-layer networks scored 97% on MNIST, which so far as we know, is the best score yet achieved by learning by any all-spiking network. In what follows, we relate our results to other recent discoveries involving spiking networks. In cases where these other studies also used the MNIST task, we will report their scores on it. But we emphasize that these different studies often had widely different aims and that most of them, like our own, were not concerned with setting records on MNIST but with demonstrating computational principles.

Several labs have looked into creating useful spiking networks not by training them directly but by training nonspiking networks and then translating the results into spiking nets. By this method, Diehl et al. (2015) created spiking networks that achieved 98.68% on MNIST and convolutional spiking nets that managed 99.12%. By similar methods, Eliasmith et al. (2012) and O'Connor et al. (2013) both constructed spiking networks that achieved 94%, and Neil and Liu (2014) managed 92%.

Other labs have devised spiking networks that do learn with one layer of plastic synapses. Beyeler et al. (2013) developed a network of 71,026 neurons that learned to score 92% on MNIST. Diehl and Cook (2015) achieved 91.9% with 2384 neurons and 95.0% with 7184. Jimenez Rezende and Gerstner (2014) trained networks to reproduce temporal patterns of spikes. Neftci et al. (2014) achieved 91.9% on MNIST with a restricted Boltzmann machine of 1324 stochastic spiking neurons. Brader et al. (2007) achieved 96.5% with just 934 neurons.

Few labs have considered deep learning in spiking networks. Bohte et al. (2002) developed the SpikeProp algorithm and used it to train three-layer networks on several tasks. But SpikeProp is not fully spiking: its forward layers spike, but its feedback signals are real valued. It also requires weight transport, as backpropagation does. In contrast, Henderson et al. (2015)

used fixed feedback weights and only spiking neurons in both the forward and feedback paths, and scored 87.4% on a subset of MNIST with a four-layer network of 4058 cells.

In many of these other cited studies, as in our own, the MNIST scores were achieved without the benefit of several devices used in the best-performing nonspiking networks: no cross-entropy, no weight decay, no adaptive gradients, no dropout (Srivastava et al., 2014), no validation set to monitor for overfitting, no data augmentation (Ciresan et al., 2010), no annealing or variation of momentum (Sutskever, 2013), and no convolution (Fukushima, 1979, 2013; Krizhevsky et al., 2012; LeCun et al., 1998; Sermanet et al., 2013). And the networks learned for only a few epochs rather than thousands. So there is scope for improvement.

Our results on deep learning are biologically interesting because it seems likely that at least some of the brain's learning circuits are multilayered. In the best-studied learning circuit in motor physiology, the cerebellum, most research has focused on a single layer of synapses—those between the parallel fibers and Purkinje cells (Sakurai, 1987)—but other synapses, from mossy fibers onto cerebellar granule cells are also plastic (D'Angelo & De Zeeuw, 2008). Therefore, this system appears to have at least two layers and may form part of a deeper circuit including deep cerebellar or brainstem nuclei (Lisberger, 1994; Medina & Mauk, 2000).

Theoretically, deep learning has advantages and disadvantages. Its main drawback is its complexity. In networks with one-layer learning, such as support vector machines, gaussian processes, and other kernel methods (Liu, Príncipe, & Haykin, 2010), there is a simple, usually linear relation between the network's output errors e_k and all its adjustable weights. As a result the risk surface (e.g., the graph of squared error as a function of the weights) is convex, sloping down smoothly in all dimensions toward a single optimum. In networks where two or more layers learn, there is nonlinear processing between e and some of the weights. This nonlinearity complicates the risk surface and also means that information about the form of the nonlinearity must be delivered to upstream synapses, as in equations 2.8 and 2.11.

On the positive side, deep learning makes networks more flexible by reducing nonoptimized parameters. That is, a network with just one layer of learning almost always needs at least one additional processing layer upstream for expansion recoding (Liu et al., 2010). If that upstream layer cannot learn, then its synapses stay frozen forever at suboptimal values (or decay or otherwise change through some process other than learning). It is true that kernel algorithms have clever ways to initialize frozen synapses, and natural selection may have done this for our brains, but even so, a network with all its synapses unfrozen will be more adaptable.

Another advantage is thought to be that deep networks contain a kind of hierarchy of layers that reflects the hierarchies in many stimuli (Saxe, McClelland, & Ganguli, 2013; for example, many images show objects made

of parts that are made of smaller parts (Yamins et al., 2014). In other words deep networks perform a useful kind of regularization (Bengio & LeCun, 2007; Ba & Caruana, 2014).

Our findings show that multilayer networks of dynamic spiking neurons can learn by mechanisms similar to the backpropagation algorithm that is used with the static, nonspiking artificial neurons of the deep-learning literature. But the feedback calculations in our method, in equations 2.7 and 2.8, are simpler than those in backpropagation.

In all the simulations in Figures 3, 4, and 5, we used a momentum value of 0.9. We chose that value because it is common in machine learning; at present, we have no biological justification for it except that it works. But it may be that the precise value of momentum is not critical. With broadcast alignment, we have observed that even momentum-free four-layer networks sized like those in Figure 5 can still achieve 97% on MNIST (results not shown).

In equation 2.8, we assumed that the learning mechanism within each neuron has information about the derivative da/dv of its own activation function. We tried removing that assumption, with our derivative-free algorithm, but learning suffered badly. Hence, it appears that deep learning in a spiking network is more effective if each neuron's learning reflects its own nonlinearity in this sense (i.e., if neurons respond more strongly to error signals when their drive is weaker). We suggest that real neurons may show a similar dependence, on the grounds that it would be very useful for deep learning

This letter has addressed three computational issues but deferred many other questions as topics for future study. For instance, we have treated synapses as simple, scalar weights that multiply their incoming signals, whereas real synapses are more complex. We have also ignored issues of timing: like most other neural network simulations, ours send their feedback signals to all learning cells simultaneously and without delay, and all their variables are updated abruptly and then stay constant for the duration of one time step. Further, we have described no biochemical implementations for the computations in our model, including those of momentum and the intracellular teaching signal ι in equation 2.8. Also in equation 2.8, it remains to be seen how precisely the variable da/dv must be represented. We have shown that if it is omitted entirely (i.e., assumed to be 1), as in our derivative-free algorithm, then learning is poor. But if a cell's estimate of da/dv were only slightly inaccurate, then the consequences might be less extreme. Even an inexact estimate might make a network learn better than it would with the derivative-free algorithm. There are also open questions about the variables that might feed into the calculation of da/dv in equation 2.8. In equation 2.11, we based the computation of da/dv on the drive variable; that is, we proposed that the cell estimates its da/dv based on some intracellular correlate of v. But da/dv might instead be estimated based on h, perhaps directly or perhaps by first filtering h to yield an

estimate of v, and there are other ways neurons might estimate da/dv (Hinton, 2016).

Appendix A: Plotting a and v

To compute the data in Figure 2A we presented a series of 1000 drives v, all between -50 and 50, to a single LIF neuron, with each v applied for 0.1 s. We ignored the neuron's activity, a, over the first 0.02 s, because the neuron was dynamic and its activity was settling over that time. Then we recorded 320 vs and as over the remaining 0.08 s. In all, then, we recorded 320,000 input-output pairs (v, a)—320 for each of the 1000 vs. The 500 blue dots in Figure 2A are a random subset of those pairs.

Appendix B: Initialization

We initialized the network weights and biases using techniques closely analogous to those used in computer science. The mechanisms used in the brain are likely quite different and outside the scope of this letter. Our methods were simply a fast way to get weights and biases that prevented the forward and feedback signals from vanishing or saturating.

Network Ws and bs were initialized so that the vs of all neurons, at the start of training, had means and ranges appropriate for LIF neurons—not too large and not too small—so they rarely fell in the ranges where the neuron's activity was 0 or near maximal. Simpler initializations than this one also worked well in preliminary tests, but this approach has a clear rationale. Specifically, we initialized all biases b to a physiological value of $\bar{b} = 0.8$ (Eliasmith & Anderson, 2002). We then defined a desired mean for v, namely, $\bar{v} = 8$, and a desired standard deviation $\sigma_v = 10$, chosen to keep the neurons' drives in the range where the E(a) curve in Figure 2A is not flat. From these values, we computed the desired second moment $\bar{\bar{v}}$. We also defined a linear approximation to the LIF activation function, equation 2.10,

$$a = \alpha v, \tag{A.1}$$

where $\alpha = c_1 c_2$, the product of the fitted constants c_1 and c_2 in equation 2.10, which gave us $\alpha = 0.066$. Then for each layer n after the first, we computed its fan-in N—the number of inputs to each neuron—and from that the desired mean of the weights

$$\bar{W}_{(n)} = (\bar{v} - \bar{b})/(\alpha N \bar{v}), \tag{A.2}$$

and the desired second moment,

$$\bar{\bar{W}}_{(n)} = (\bar{\bar{v}} + \alpha^2(N - N^2)\bar{W}_{(n)}^2\bar{v}^2 - 2\alpha N\bar{b}\bar{v}\bar{W}_{(n)} - \bar{b}^2)/(\alpha^2 N\bar{\bar{v}}) \tag{A.3}$$

From $\bar{W}_{(n)}$ and $\bar{\bar{W}}_{(n)}$ we computed the standard deviation $\sigma_{W(n)}$ and initialized the weights to

$$W_{(n)ij} = \bar{W}_{(n)} + 2\sqrt{3}\sigma_{W(n)}(rand - 0.5), \tag{A.4}$$

where *rand* had a uniform distribution over the range [0, 1]. Given these values for $W_{(n)ij}$ and assuming the vs in layer $n - 1$ have the desired mean $\bar{v}$ and standard deviation σ_v, it follows that the vs in layer n will have those same statistics. That is, this initialization ensures that at least at the start of the run, the drives in all layers have reasonable values—neither too small nor too large on average, and varying over a reasonable range when the network inputs vary.

Our values for $\bar{v}$ and σ_v imply that E(a) will have a mean of 0.64 and a standard deviation of 0.8. Therefore, we ensured that network inputs had these statistics; for example, in MNIST trials, the input vectors were preprocessed so all 784 pixels had the same mean value of 0.64, across all the training images.

For alignment algorithms, we have to initialize not just the forward weights and biases but the feedback weights as well. For feedback alignment in its original form, those feedback weights, which we will call B_{FA} here, were initialized like the Ws in equation A.4,

$$B_{\mathrm{FA}(n)ij} = \bar{W}_{(n+1)} + 2\sqrt{3}\sigma_{W(n+1)}(rand - 0.5), \tag{A.5}$$

though of course Ws and B_{FA}s were initialized independently (the indices on the right-hand side are $(n + 1)$ because $B_{(n)ik}$ plays the same role in equation 2.6 as $W_{(n+1)ki}$ does in equation 2.5). This formula ensured that the feedback signals were scaled as they would have been with backpropagation; they were about the same size on average.

For broadcast alignment, we first computed the $\boldsymbol{B}_{\mathrm{FA}}$s using equation A.5 and then set each layer's feedback matrix $\boldsymbol{B}_{\mathrm{BA}(n)}$ equal to $\boldsymbol{B}_{\mathrm{FA}(n)}$ times all the downstream $\boldsymbol{B}_{\mathrm{FA}}$s,

$$\boldsymbol{B}_{\mathrm{BA}(n)} = \gamma^D \Pi_{i=n}^{n+D} \boldsymbol{B}_{\mathrm{FA}(i)}, \tag{A.6}$$

where γ is a scalar constant to be explained below and D is the number of downstream layers; e.g., if $n = 4$ and there are seven layers in the net then $D = 3$. The rationale here is that $\boldsymbol{B}_{\mathrm{BA}(n)}$ plays the same role in broadcast alignment as does the whole sequence of downstream $\boldsymbol{B}_{\mathrm{FA}}$s in feedback alignment. That is, in feedback alignment the output-layer error signals pass back through this whole series of $\boldsymbol{B}_{\mathrm{FA}}$s to create the teaching signals

for layer n, whereas in broadcast alignment $\boldsymbol{B}_{\mathrm{BA}(n)}$ alone conveys the output error to layer n, as shown in equation 2.7. Therefore equation A.6 ensured that each $\boldsymbol{B}_{\mathrm{BA}(n)}$ had the right numbers of rows and columns; for example, if the output layer of a network had 10 neurons and layer 2 had 1000, then $B_{\mathrm{BA}(2)}$ had 1000 rows and 10 columns. The scheme also scaled the feedback weights in a useful way, so they kept the feedback signals about the same size as those of the backpropagation and feedback alignment methods if the constant γ was chosen appropriately. This γ compensated for the fact that feedback alignment multiplies its feedback signals $\delta_{(n)i}$ by da/dv in every layer, as shown in equation 2.6, whereas broadcast alignment does not, as shown in equation 2.7. For example if da/dv were, on average, equal to 0.1 in every layer, then after, say, four layers of feedback, the multiplications by da/dv would shrink the feedback signals by about $0.1^4 = 10^{-4}$ in a feedback alignment network. To keep the feedback in broadcast alignment at about the same scale, one can set $\gamma = 0.1$ in equation A.6. So for the MNIST trials in section 3.3, we estimated the mean da/dv, which was about 0.034, and used that value for γ. A simpler approach is to set γ equal to the constant α from the linear approximation to a neuron in equation A.1. This method will usually yield a larger than optimal γ because the neurons are in fact sublinear, but it works well nonetheless and does not require an estimate of da/dv. This simpler approach is what we used in sections 3.1 and 3.2.

Acknowledgments

We thank Sara Scharf for comments. This study was supported by the Natural Sciences and Engineering Research Council of Canada grant 391349-2010.

References

Azghadi, M. R., Iannella, N., Al-Sarawi, S. F., Indiveri, G., & Abbott, D. (2014). Spike-based synaptic plasticity in silicon: Design, implementation, application, and challenges. *Proceedings of the IEEE*, *102*(5), 717–737.

Ba, L. J., & Caruana, R. (2014). Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27* (pp. 2654–2662). Red Hook, NY: Curran.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*, 1–127.

Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms toward AI. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds.), *Large-scale kernel machines* (pp. 321–360). Cambridge, MA: MIT Press.

Beyeler, M., Dutt, N. D., & Krichmar, J. L. (2013). Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule. *Neural Networks*, *48*, 109–124.

Bohte, S. M., Kok, J. N., & La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*, 17–37.

Brader, J. M., Senn, W., & Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, *19*, 2881–2912.

Chinta, L. V., & Tweed, D. B. (2012). Adaptive optimal control without weight transport. *Neural Computation*, *24*, 1487–1518.

Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, *22*(12), 3207–3220.

Crick, F. H. C. (1989). Recent excitement about neural networks. *Nature*, *337*, 129–132.

D'Angelo, E., & De Zeeuw, C. I. (2008). Timing and plasticity in the cerebellum: Focus on the granular layer. *Trends in Neurosciences*, *32*, 30–40.

Diehl, P. U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*. https://dx.doi.org/10.3389%2Ffncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. C., & Pfeiffer, M. (2015). *Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing*. Paper presented at the IEEE International Joint Conference on Neural Networks.

Eliasmith, C., & Anderson, C. H. (2002). *Neural engineering*. Cambridge, MA: MIT Press.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, *338*, 1202–1205.

Farabet, C., Couprie, C., Najman, L., & LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1915–1929.

Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position—Neocognitron. *Electron. & Commun. Japan*, *62*(10), 658–665.

Fukushima, K. (2013). Artificial vision by multi-layered neural networks: Neocognitron and its advances. *Neural Networks*, *37*, 103–119.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science 11*, 23–63.

Henderson, J. A., Gibson, T. A., & Wiles, J. (2015). *Spike event based learning in neural networks*. arXiv:1502.05777

Hinton, G. E. (2016). Stanford Seminars—Can the brain do back-propagation? (Video file). https://www.youtube.com/watch?v=VIRCybGgHts

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, *18*, 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*, 504–507.

Jimenez Rezende, D., & Gerstner, W. (2014). Stochastic variational learning in recurrent spiking networks. *Frontiers in Computational Neuroscience*, *8*. doi:10.3389/fncom.2014.00038

Koch, C. (1999). *Biophysics of computation*. New York: Oxford University Press.

Kolen, J. F., & Pollack, J. B. (1994). Back-propagation without weight transport. *Neural Networks 1994: IEEE World Conference on Computational Intelligence*. Piscataway, NJ: IEEE.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. L. Weinberger (Eds.), *Advances in neural information processing systems*, *25* (pp. 1097–1105). Red Hook, NY: Curran.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Levine, D. S. (2000). *Introduction to neural and cognitive modeling*. Mahwah, NJ: Erlbaum.

Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. (2014). *Random feedback weights support learning in deep neural networks*. arXiv:1411.0247

Lisberger, S. G. (1994). Neural basis for motor learning in the vestibulo-ocular reflex of primates: III. Computational and behavioral analyses of the sites of learning. *Journal of Neurophysiology*, *72*, 974–998.

Liu, W., Príncipe, J., & Haykin, S. (2010). *Kernel adaptive filtering*. Hoboken, NJ: Wiley.

Maass, W., & Markram, H. (2004). On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, *69*(4), 593–616.

Medina, J. F., & Mauk, M. D. (2000). Computer simulation of cerebellar information processing. *Nature Neuroscience*, *3*, 1205–1211.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., & Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, *7*. doi:10.3389/fnins.2013.00272

Neil, D., & Liu, S.-C. (2014). Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *22*(12), 2621–2628.

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., & Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, *7*. doi:10.3389/fnins.2013.00178

Rolls, E., & Deco, G. (2002). *The computational neuroscience of vision*. Oxford: Oxford University Press.

Sakurai, M. (1987). Synaptic modification of parallel fibre-Purkinje cell transmission in in vitro guinea-pig cerebellar slices. *Journal of Physiology*, *394*, 463–480.

Saxe, A., McClelland, J., & Ganguli, S. (2013). Learning hierarchical category structure in deep neural networks. In M. Knauff, M. Paulen, N. Sebanz, & I. Wachsmuth (Eds.), *Proceedings of the 35th Annual Meeting of the Cognitive Science Society*. Austin, TX: Cognitive Science Society.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). *OverFeat: Integrated recognition, localization and detection using convolutional networks*. arXiv:1312.6229

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Stork, D. (1989). Is backpropagation biologically plausible? In *1989 IEEE INNS International Joint Conference on Neural Networks* (pp. 241–246). San Diego: IEEE TAB Neural Network Committee.

Sutskever, I. (2013). *Training recurrent neural networks.* PhD diss., University of Toronto.

Yamins, D. L. K., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., & DiCario, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *PNAS*, *111*, 8619–8624.

Received May 2, 2016; accepted October 26, 2016.