

Distributed bottlenecks for improved generalization in back-propagation networks

John K. Kruschke

Department of Psychology, University of California, Berkeley, CA 94720, USA

Abstract: *The primary goal of any adaptive system that learns by example is to generalize from the training examples to novel inputs. Empirically, back-propagation networks can sometimes generalize better when they contain a hidden layer that has significantly fewer units than previous layers. The functional properties of such hidden layer bottlenecks are described, and a method for dynamically producing them, concurrent with back-propagation learning, is explicated. The method does not remove hidden units; rather, it forms clusters of covariant units in a low-dimensional space. The result is a functional bottleneck distributed across many units. The method is a gradient descent procedure, using local computations on simple lateral, Hebbian connections between hidden units.*

1. Bottlenecks improve generalization

The primary goal of any scheme for learning by example is to generalize from the training examples to novel inputs. In particular, the back-propagation learning algorithm [1] for feed-forward neural networks has enjoyed great popularity for its simplicity and for its landmark successes at generalization (e.g. NETtalk, see [2,3]). It has been observed that generalization in back-propagation networks can sometimes be improved when a hidden layer bottleneck is imposed; i.e., a layer with relatively fewer units than previous layers (see [4-12]). There may be many ways besides bottlenecks to constrain the possible types of mappings from input to output, in order to improve generalization. The point of this paper is not to argue that bottlenecks are always the best constraint. Rather, the premise is that bottlenecks are beneficial in at least some cases. This paper analyzes their functional properties, and proposes a new method for creating functional bottlenecks distributed across a large hidden layer.

2. Local bottlenecks.

Insofar as bottlenecks are desirable to improve generalization, one might simply initialize the network with a small hidden layer. But such an approach is undesirable for three reasons. First, for most applications, one simply does not know, in advance, the minimal number of hidden units necessary to reach criterial error. If one initializes the network with too few units, it will not achieve sufficiently small asymptotic error. Second, even if one does know the minimal number of hidden units needed and initializes the network with that number, it is more likely that gradient-descent learning will encounter local minima or plateaux on the error surface (e.g. the XOR problem with two hidden units). Third, bottlenecks increase noise and damage sensitivity. The fewer units there are, the greater is the demand for high accuracy and precision in each unit and in the input patterns.

The first two problems can be addressed by initializing the network with many hidden units, and progressively excising units as learning proceeds. One method using that approach was introduced by Kruschke [13], and improved in Kruschke and Rodriguez-Movellan [14]. In that method, hidden units compete for the right to participate in the hidden layer representation. The more similar or redundant two hidden units are, the more strongly they compete. Computations are local, using simple lateral, Hebbian connections. The result is a hidden layer in which only a few, relatively uncorrelated units remain. By linking the competition rate to the rate at which error decreases, it was also found that new hidden units could be recruited from the pool of unused units. If the error was not decreasing (i.e. learning had stalled), competition was relaxed, allowing new units to participate. When error again began to decrease, stronger competition would suppress all but a few new, dissimilar units.

Other methods for excising hidden units have been independently suggested by Chauvin [15], Hanson and Pratt [5], Mozer and Smolensky [7], Rumelhart [9], and Sietsma and Dow [10]. Only one of those methods (Sietsma and Dow), is explicitly designed to excise redundant units, but it gives no method for deciding when a set of units contains sufficient redundancy for removing some units, nor a method for deciding which of the redundant units to remove.

All of those methods are motivated by the goal of minimal hardware. They all create a bottleneck by de-activating units, i.e. pieces of hardware. The resulting bottleneck is localized in the few remaining units. But as more information must be passed through fewer units, each unit must make finer discriminations and carry information complementary to other units. Thus, these localized hardware bottlenecks do not address the third problem mentioned above, the increase in noise and damage sensitivity.

3. Distributed bottlenecks

An alternative motivation is to reproduce the functional

properties of a bottleneck without reproducing its hardware properties [13]. To do that, we must decide just what functional properties are important. Explanations as to why bottlenecks improve generalization usually reflect one of three functional properties: complexity reduction, dimensionality reduction, or reduction in the number of possible mappings from input to output.

The first functional property, regarding complexity, is nicely expressed by Wieland and Leighton [11]. They suggest that 'viewing learning as curve fitting then allows us to see that generalization ... is simply the effect of a good non-linear interpolation of the data' (p.III-389). They argue that better generalization comes from smoother network mappings, and that smoother network mappings come from fewer units. Therefore, better generalization results from fewer units, to the same extent that better interpolation results from smoother data-fitting curves. Hanson and Pratt [5] corroborate: 'as a result of too many hidden units the underlying feature relations determining the output surface and category separation are arbitrary, *more complex than necessary*, and may result in anomalous generalizations' (italics added).

The second functional property, regarding dimensionality reduction, has been mentioned by many researchers (see [6, 12, 16-19]). Each unit of a hidden layer is, potentially, an independent dimension of variation on which to represent the (transformed) input. With fewer units in a bottleneck, some dimensions of variation in the previous layer are lost; only those dimensions most important for reducing error are retained in the bottleneck layer.

The third functional property, regarding the number of possible mappings from input to output, is described by Wittner *et al* [4]. They remind us that the training data usually leave the network underconstrained, and that there are usually a huge number of possible generalizations from a given training set. The more hidden units we supply to the network, the

more possible generalizations there are. In order to train a large network to yield a particular generalization, we must either supply it with a huge training set, or reduce the number of hidden units. (Related ideas are expressed by Pavel *et al* [8], and by Psaltis and Neifeld [20].)

These properties can be better understood by considering two layers of a feed-forward network (see Figure 1). Suppose that layer s had N_s units, layer $s-1$ has N_{s-1} units, and that the $N_s \times N_{s-1}$ weight matrix connecting them, W , has rank R . (Here we make the usual assumptions that activation flows forward from layer $s-1$ to layer s , that the activation of a unit is a non-linear, sigmoidal function of its net input, and that the net input of a unit is the dot product of its weight vector with the activation vector of the previous layer.) Let V be an $R \times N_{s-1}$ matrix with rows that form a basis for the row space of W . Then there exists a $N_s \times R$ matrix U such that $W = UV$. This is equivalent to supplying a 'virtual basis layer' of R linear units between layer s and layer $s-1$, such that the weight matrix from layer $s-1$ to the basis layer is V , and the weight matrix from the basis layer to layer s is U . Activation vectors in layer s all lie on an R -dimensional manifold in N_s -dimensional space. The manifold is generally not a hyperplane, because of the non-linear activation functions.

DEFINITIONS: We say layer s forms a bottleneck if $R < N_{s-1}$. We say layer s forms a local bottleneck if $N_s = R < N_{s-1}$. We say layer s forms a distributed bottleneck if $N_s > R < N_{s-1}$. Note that we could have $N_s > N_{s-1}$, yet still have a bottleneck.

The important point is that generalization is affected by both small N_s and small R , in different ways. When $R < N_{s-1}$, the (N_{s-1}) -dimensional activation vectors of $s-1$ are projected into an R -dimensional subspace. We then have complete generalization over the complementary $N_{s-1} - R$ dimensions, because no variation in the complementary dimensions can lead to any variation in the R -dimensional representation. In terms of curve fitting, the curve is perfectly smooth (flat) in those com-

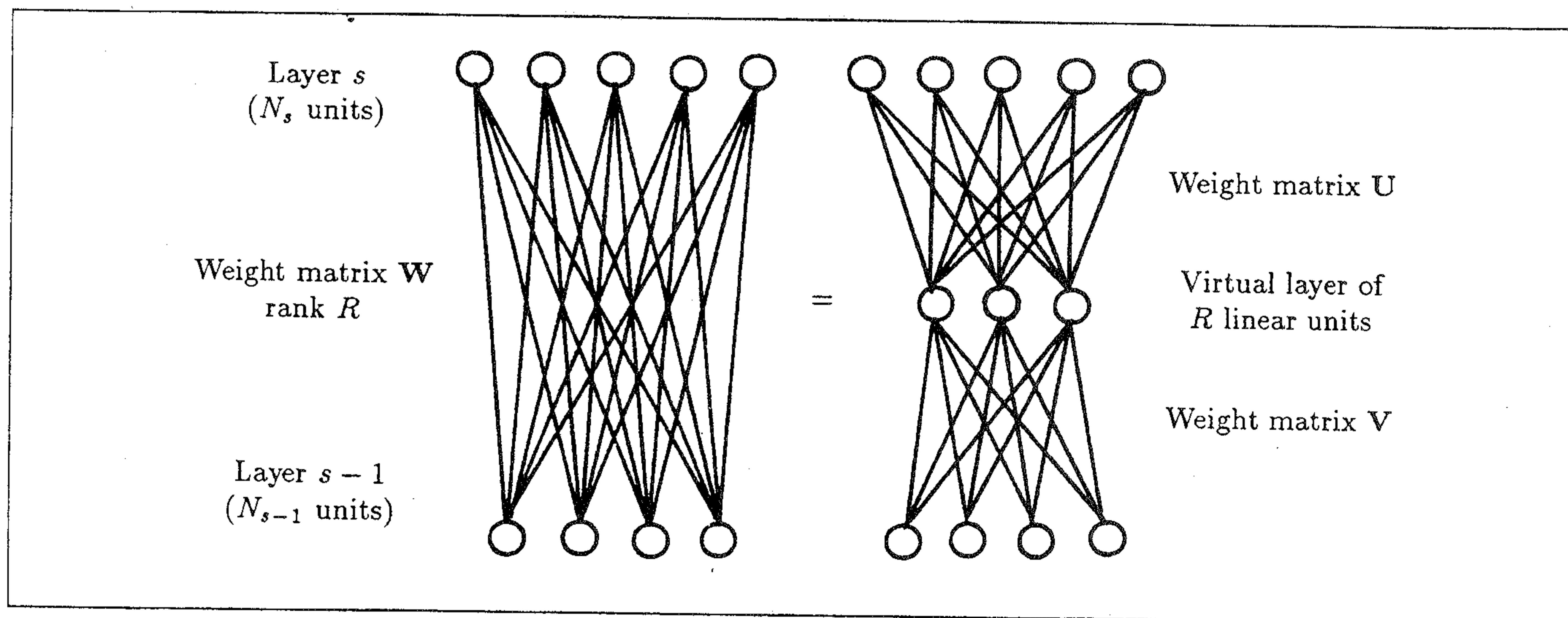


Figure 1. The weight matrix W connecting two layers can be decomposed into two matrices of rank R . Generalization is affected in different ways by small N_s and small R . See text for details.

plementary directions. The number of units, N_s , in layer s determines the maximal complexity of regions that can be distinguished in the remaining R dimensions. In terms of curve fitting, a smaller N_s means fewer possible 'bumps' in the mapping from the R dimensions. Moreover, the maximal number of possible mappings from layer $s-1$ to layer s is reduced when R is small, because the weights in the mapping matrix \mathbf{W} are constrained.

So, to improve generalization by creating a functional bottleneck, we want to do two things: (1) decrease the functional dimensionality R of the weight matrix \mathbf{W} ; and (2) decrease the functional number of units in layer s . These goals do not necessarily imply excising units, as done by the methods for creating local bottlenecks described earlier. Rather, the goals imply that we should (1) dynamically compress the weight vectors (rows of \mathbf{W}) into a low dimensional space and (2) cluster the vectors within that low dimensional space.

There are potentially many ways of achieving those goals. I have worked under two constraints on the choice of method: it should be describable as gradient descent on some objective function, and it should be locally computable in a network.

4. Shepard's method.

A method for compressing the dimensionality of a set of vectors, and simultaneously clustering them, was described by Shepard [21]. He motivated his algorithm by arguing that '... the way to flatten a configuration into a space of smaller dimensionality is to increase the variance of the distances by further stretching those distances that are already large and by further shrinking those distances that are already small' (p. 131). Let $\mathbf{w}_i^s = [w_{i1}^s \dots, w_{iN}^s]^T$ be the column vector of fan-in weights connecting the N units of layer $s-1$ to the i^{th} unit of layer s . (While the typographical difference between \mathbf{w}_i^s and w_{ij}^s is slight, it is important. The first denotes a vector of fan-in weights; the second denotes the j^{th} component of that vector.) Denote the distance between vectors by $d_{ij}^s \equiv \|\mathbf{w}_i^s - \mathbf{w}_j^s\|$. The mean distance is denoted \bar{d}^s . Then for each vector \mathbf{w}_i^s , we shift it by an amount

$$\Delta \mathbf{w}_i \propto \sum_j^B (d_{ij} - \bar{d}) \frac{(\mathbf{w}_i - \mathbf{w}_j)}{d_{ij}}. \quad (1)$$

(Here and throughout the remainder of this paper, the superscript denoting the layer will be suppressed whenever possible. Layer s is assumed unless otherwise marked.) Thus if $d_{ij} > \bar{d}$, then \mathbf{w}_i tends to be shifted *away* from \mathbf{w}_j , and if $d_{ij} < \bar{d}$, then \mathbf{w}_i tends to be shifted *toward* \mathbf{w}_j . After each shift, the vectors are re-centered so that their centroid is zero, and they are re-scaled so that their mean separation, \bar{d} , is a pre-defined constant. Without re-scaling, the vectors will 'explode', i.e. become infinitely long.

A slight variant of equation (1) was applied directly, in conjunction with back-propagation learning, by Kruschke [13]. While the results were encouraging, the method was unsatisfactory for two reasons: It was not explicitly gradient descent, so its stability was questionable, and it was not locally computable (although some parts of it were), so its network implementation was doubtful. Those two problems are resolved presently.

While Shepard never explicitly described his algorithm as gradient descent on the variance of the distances, he might have had in mind something like the following. Let

$$D \equiv -\frac{1}{4} \sum_{i,j}^{B,B} (d_{ij} - \bar{d})^2 \quad (2)$$

where

$$d_{ij} \equiv \|\mathbf{w}_i - \mathbf{w}_j\| \quad \text{and} \quad \bar{d} \equiv \frac{1}{B^2} \sum_{k,l}^{B,B} d_{kl}.$$

(Here $\|\mathbf{x}\| = \sqrt{\sum_n x_n^2}$ denotes the Euclidean norm of \mathbf{x} .)
Then

$$\Delta \mathbf{w}_I \propto -\frac{\partial D}{\partial \mathbf{w}_I} = \sum_j^B (d_{Ij} - \bar{d}) \frac{(\mathbf{w}_I - \mathbf{w}_j)}{d_{Ij}}. \quad (3)$$

(See the Appendix for notes on the derivation.) Equations (1) and (3) are, of course, the same. Maximal variance techniques are further discussed in Cunningham & Shepard [22].

Even though we have been able to re-express Shepard's method as a gradient descent procedure, it remains unclear how to compute it in a network. The algorithm demands computing the global mean distance, \bar{d} , and some way of multiplying corresponding d_{Ij} and $w_{In} - w_{jn}$. We seek something like Shepard's method that not only is nicely expressed as gradient descent, but also is locally computable in a network.

To achieve that goal it is helpful to abstract the essential properties of the method, so that they might be realized other ways. The key property of the algorithm is that it increases the variance of the separations of the vectors. The vectors are then re-centered and re-scaled so that they do not explode. We should retain those essentials, but we are then free to define the separation of vectors any way that is sensible, and free to keep the vectors from exploding in any reasonable way.

5. A new method

Define the 'distance' between two weight vectors \mathbf{w}_i^s and \mathbf{w}_j^s as follows:

$$d_{ij}^s \equiv \sum_p \text{net}_{ip}^s \text{net}_{jp}^s \quad (4)$$

where net_{ip}^s is the net input to unit i on pattern p , and the sum is taken over all training patterns in an epoch. It is convenient to express the net input as the inner product of the unit's fan-in weight vector with the activation vector of the layer below, $\text{net}_{ip}^s = \langle \mathbf{w}_i^s, \mathbf{a}_p^{s-1} \rangle$, where $\mathbf{a}_p^{s-1} = [a_{1p}^{s-1}, \dots, a_{Np}^{s-1}]^T$ is the column vector of activations of the units of layer $s-1$ for pattern p . The newly defined d_{ij} , essentially the covariance of the net inputs, does not satisfy all the distance axioms. It is symmetric, but it is not positive definite and does not satisfy the triangle inequality. However, it does describe the *similarity* of the units.

We define D as in equation (2), where the sum is taken over all units in the bottleneck layer. Then (see the Appendix for notes on derivation)

$$\frac{\partial D}{\partial \mathbf{w}_I} = - \sum_j (d_{Ij} - \bar{d}) \sum_p \text{net}_{jp} \mathbf{a}_p^{s-1}. \quad (5)$$

That is a bit of an improvement over equation (3), but we still must compute the global mean covariance \bar{d} . However, if we can force \bar{d} to be zero, then the expression is significantly simplified. (It makes no sense to force the average *distance* between vectors to be zero, for then all the weight vectors would have zero length. But with the new definition of d_{ij} as covariance in equation (4), some 'distances' are negative, so forcing \bar{d} to be zero is quite acceptable.)

As a step toward the goal of forcing the global mean distance to zero, note that if

$$\bar{\mathbf{w}} \equiv \frac{1}{B} \sum_i \mathbf{w}_i = \mathbf{0},$$

then $\bar{d} = 0$, as follows:

$$\begin{aligned} \bar{d} &\equiv \frac{1}{B^2} \sum_{k,l} \sum_p \langle \mathbf{w}_k, \mathbf{a}_p^{s-1} \rangle \langle \mathbf{w}_l, \mathbf{a}_p^{s-1} \rangle \\ &= \sum_p \langle \bar{\mathbf{w}}, \mathbf{a}_p^{s-1} \rangle^2 \\ &= 0. \end{aligned}$$

So now the goal is to make $\bar{\mathbf{w}} = \mathbf{0}$, and that is easy. Simply do very fast gradient descent on

$$M \equiv \|\bar{\mathbf{w}}\|^2. \quad (6)$$

The change in the weight w_{IJ} due to gradient descent on M is proportional to the negative of the average fan-out weight from unit J in layer $s-1$:

$$\Delta_M w_{IJ} \propto - \frac{\partial M}{\partial w_{IJ}} \propto - \frac{1}{B} \sum_k w_{kJ}. \quad (7)$$

The average fan-out weight is locally computable, since each unit has direct access to all its fan-out weights. By equation (7), each weight tends toward a value that makes the average fan-out weight closer to zero. If we do very rapid gradient descent, using a proportionality constant of 1, we get to zero in one update (since trajectories are linear in this case).

With $\bar{d} = 0$, equation (5) becomes the following simple expression for weight change due to dimensional compression:

$$\Delta_D \mathbf{w}_I \propto \sum_p \left(\sum_j d_{Ij} \text{net}_{jp} \right) \mathbf{a}_p^{s-1}. \quad (8)$$

Now that requires only simple lateral connections between units with the value d_{ij} as their connection strength (see Figure 2). The value of d_{ij} is obtained by simple Hebbian learning during an epoch, as in equation (4). The lateral connections do not participate in the forward propagation of activation, nor in the backward propagation of error.

Like the back-propagation learning rule, the weight change rule in equation (8) conforms to the 'principle of minimal disturbance' [23]. Weight vectors are changed only parallel to the current activation vector. If two units have positive covariance, then their weight vectors tend to move in the same direction (both parallel or antiparallel to \mathbf{a}_p^{s-1}). If two units have negative covariance, then their weight vectors tend to move in opposite directions (one parallel, the other antiparallel, to \mathbf{a}_p^{s-1}).

We have only one more step. So far, the value of D can be made infinitely negative by increasing the magnitudes of the weights infinitely. We can prevent that by including standard weight decay. But we have already included a component of weight decay, the term M , in equations (6) and (7). M makes the mean weight vector tend toward zero. We must now also put a cost on the remaining component of weight magnitude, the variance of the weight vectors, V . The decomposition of weight decay is made explicit in the following formula:

$$\begin{aligned} \frac{1}{B} \sum_i \|\mathbf{w}_i\|^2 &= \frac{1}{B} \sum_i \|\mathbf{w}_i - \bar{\mathbf{w}}\|^2 + \|\bar{\mathbf{w}}\|^2 \\ &= V + M. \end{aligned} \quad (9)$$

Gradient descent on V is also locally computable:

$$\Delta_V w_{IJ} \propto - \frac{\partial V}{\partial w_{IJ}} \propto \frac{1}{B} \sum_k w_{kJ} - w_{IJ}. \quad (10)$$

Each weight is pushed toward the average fan-out weight, which in this case is zero because we have done fast descent on M . Note that by adding equations (7) and (10) we get $\Delta w_{IJ} \propto w_{IJ}$, the standard form of weight decay.

