JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM

331.1-95-047
November 20, 1995

To: Distribution

From: Dave Watola

Subject: Overview of Downlink Analyzer Neural Network Technology

REFERENCES:

1. J. B. Hampshire II, *A Differential Theory of Learning for Efficient Statistical Pattern Recognition*. PhD. thesis, Carnegie Mellon University, Department of Electrical & Computer Engineering, September 1993.

2. Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, ed., pp. 598-605, Morgan Kaufmann, San Mateo, CA, 1990.

3. B. A. Pearlmutter, "Fast Exact Multiplication by the Hessian," *Neural Computation* 6(1), January 1994.

4. B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal Brain Surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., pp. 164-171, Morgan Kaufmann, San Mateo, CA, 1993.

1. Purpose

This document is the third in a series of five memoranda detailing the background theory, implementation, and issues surrounding the Downlink Analyzer task. The focus of this memo is the neural network used to model the target system. In particular, the architecture, network topology, and learning schema employed by John Hampshire's ***dlaNet*** tool are described here. Also presented is some discussion of unresolved problems and potential solutions.

2. Neural Networks

A neural network can be characterized as a directed graph containing N nodes, representing *neurons* or *processing units*, indexed with integers on $[1, N]$. The first N_I units are designated as inputs, and may not have edges entering them; these input nodes perform no computation, instead behaving as placeholders for introducing external stimulus to the rest of the network.

Remaining nodes are computational units, with the final N_O neurons designated as the observable network outputs.

The graph and neural network architecture are described by an N×N interconnection matrix, **C**,

$$\mathbf{C} = \mathbf{C}_{N\times N} = [c_{i,j}] \quad \forall\,(i,j){:}\ 1 \le i,j \le N \tag{1}$$

where the elements $c_{i,j}$ of **C** are indicator variables denoting the presence or absence of a connection to neuron i from neuron j:

$$c_{i,j} = \begin{cases} 1 & \text{if a connection exists to neuron } i \text{ from neuron } j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Associated with each edge is a finite connection weight, $w_{i,j}$. Elements of the N×N weight matrix **W** are zero wherever the corresponding element of **C** is zero. There are a total of N_w weights in the network, with

$$N_w = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{i,j}. \tag{3}$$

Discussion here is restricted to strictly feedforward multilayer networks. In these architectures, the sets of input and output neurons comprise the input and output layers, respectively. Any intervening nodes are organized into zero or more discrete "hidden" layers. Connections are permitted only between neurons in adjacent layers, and must be oriented in the direction of computation (input to output). Thus, the graph is acyclic and **C** is strictly lower triangular:

$$c_{i,j} = 0\ \forall\ i \le j. \tag{4}$$

Note that these two conditions also preclude connections originating on the output nodes.

3. Neuron Computation

Except for input units, each neuron i produces an output y_i based on its total input (or *net activation*) x_i,

$$x_i = g_i\left(\{w_{i,j}\}, \{y_j\}\right) \forall\ j{:}\ c_{i,j} = 1, \tag{5}$$

which is a function of the output values from neurons in the previous layer and the weights decorating the connections from those outputs. $g_i(\cdot)$ may vary from neuron to neuron, but is nearly always a simple weighted sum of previous layer outputs,

$$x_i = g_i\left(\{w_{i,j}\}, \{y_j\}\right) = \sum_{j:c_{i,j}=1} w_{i,j} y_j. \tag{6}$$

Normally, each (non-input) unit also possesses a bias input $w_{i,0}$ driven by a virtual input with value y_0=1; this can be accommodated in the above characterization by augmenting **C** and **W** with another column.

The neuron transforms its activation via a nonlinear activation function, $F_i(\cdot)$, such that

$$y_i = F_i(x_i) = F_i\left(g_i\left(\{w_{i,j}\}, \{y_j\}\right)\right). \tag{7}$$

Again, $F_i(\cdot)$ may vary from neuron to neuron. In ***dlaNet***, output units employ the logistic sigmoid function

$$F(x) = (1 + e^x)^{-1} \tag{8}$$

in order to constrain the output between zero and unity. All other units except input units use the hyperbolic tangent as an activation function. Both functions are common choices for activation functions since they possess several desirable properties, *viz*. boundedness, continuous differentiability, and a “soft saturation” characteristic.

The overall network computation maps length-N_I input vectors **X** to length-N_O output vectors **Y**, proceeding layerwise from inputs to outputs.

4. Network Training

Because input/output relationships are typically complex, the weights necessary to effect a desired mapping from input vectors to output vectors are not usually determined through explicit computation. ***dlaNet*** uses *supervised learning*, wherein appropriate weights are determined by applying a feedback-driven iterative optimization procedure. During repeated exposure to input vectors from a carefully selected “training set” of ordered triplets (**X**, **Y**, **T**), the network uses an objective function to assess current network performance as a function of the output vector **Y** and the corresponding desired target vector **T**. Changes in the objective function drive weight adaptation toward a local optimum over the weight space.

The optimization procedure used by ***dlaNet*** is a gradient search. If the objective function is to be minimized, the search is a gradient descent; otherwise it is a gradient ascent. The general optimization strategy is simple: follow the local gradient, taking small steps in weight space until the gradient vanishes. Thus, at timestep (*epoch*) *n* the adaptation proceeds according to

$$\Delta \mathbf{w}^{(n)} = \pm\,\alpha \cdot \frac{\nabla_w \eta\left(\mathbf{w}^{(n)}\right)}{\left\|\nabla_w \eta\left(\mathbf{w}^{(n)}\right)\right\|} + \beta \cdot \Delta \mathbf{w}^{(n-1)} \tag{9}$$

where $\mathbf{w}^{(n)}$ is the current vector containing all N_w weights in the network, $\Delta\mathbf{w}^{(n)}$ is the latest weight update vector, α is a small positive learning rate constant determining the length of the step in the direction of the gradient, β is a momentum constant for accelerating the search in regions with negligible higher-order derivatives, and the gradient vector components are given by

$$[\nabla_w \eta(\mathbf{w})]_i = \frac{\partial \eta}{\partial w_i} \quad i = 1, \ldots, N_w. \tag{10}$$

Both α and β can be adaptive (and in ***dlaNet***, they are). The sign of the first term is negative for gradient descent, and positive otherwise. In ***dlaNet***, the gradient referenced in (9) above is actually an average gradient accumulated over all patterns (and all objective function scores) in the training set, but adaptation on a pattern-by-pattern basis is also viable.

Finally, note that this procedure is only guaranteed to find a locally optimum solution. Enhancements to improve chances of finding a global optimum include simulated annealing and Monte Carlo simulation with different random initial weights. ***dlaNet*** does not currently support either of these options.

5. Objective Functions

The most common objective function driving supervised learning is the mean-squared-error (MSE) criterion,

$$\eta_{MSE}(\mathbf{Y}, \mathbf{T}) = \frac{1}{N_O} \sum_{i=1}^{N_O} (\mathbf{Y}_i - \mathbf{T}_i)^2, \tag{11}$$

which is driven to a local minimum over the collection of training vectors { ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{T}$) } by the adaptation procedure in order to make the ensemble of $\mathbf{Y}$s closely approximate the ensemble of $\mathbf{T}$s. MSE is suited to neural networks with a chief purpose of approximating continuous functions.

By contrast, the ***dlaNet*** network performs a much simpler task: pattern classification. Here, each input vector is associated with one of N_O classes. The target vector components are binary indicator variables with

$$T_i = \begin{cases} 1 & \text{if } \mathbf{X} \text{ belongs to the } i\text{th class} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

and the neuron with the largest output is interpreted as designating the class determined by the network. Mean squared error is not well suited to this situation because the functions are intrinsically discontinuous, and because η_{MSE} does not always decrease monotonically with improved classification performance [1].

dlaNet and all of Downlink Analyzer adopt the classification-figure-of-merit (CFM) objective function,

$$\eta_{CFM}(\mathbf{Y}) = \begin{cases} \sigma_\delta(Y_{(1)} - Y_{(2)}) & \text{if the classification is correct} \\ \sigma_\delta(Y_{(*)} - Y_{(1)}) & \text{otherwise,} \end{cases} \tag{13}$$

where $\sigma_\delta(\cdot)$ is the synthetic CFM function with confidence parameter δ [1], $Y_{(i)}$ is the ith order statistic of the components in $\mathbf{Y}$, and $Y_{(*)}$ denotes the output matching the correct classification. The argument to $\sigma_\delta(\cdot)$ is the discriminant differential, defined as the difference between the output from the correct neuron and the largest other neuron output. It is positive when the classification is correct, negative when incorrect, and generally bounded on [-1, +1].

CFM is a continuously differentiable sigmoidal approximation to a counting function, with the property that $\sigma_\delta(\cdot)\to 1$ as the discriminant differential approaches δ from below and $\sigma_\delta(\cdot)\to 0$ as the differential becomes increasingly negative. Over the ensemble of training patterns, average CFM attains its maximum value of unity when all patterns are correctly classified with a differential exceeding the CFM confidence parameter.

Learning driven by CFM is *differential learning*, seeking only to ensure correct classification rather than tackling the more complex task of modelling *a posteriori* distribution functions the way MSE-based *probabilistic learning* does. The less rigorous nature of CFM as a cost/benefit measurement is reflected in (13), which only depends on two outputs. Further discussion of CFM and its merits relative to MSE is beyond the scope of this document. John Hampshire's thesis [1] contains an entire chapter replete with excruciating details about CFM and its properties.

6. The Backpropagation Algorithm

Backpropagation, or "reverse accumulation", is essentially a procedure for computing the derivatives of η with respect to neuron outputs or weights that are not directly associated with the output layer. Nearly all developments of the backpropagation equations for multilayer feedforward networks stress the derivation of an error signal δ_j for each hidden neuron j, with δ_j being analogous to the error signal Y_i-T_i of each output unit. This approach deemphasizes the difference between output and hidden units by synthesizing an equivalent "target value" for each hidden node. Although this unification has some intuitive appeal, it is unnecessarily complicated as well as conceptually inappropriate for pure classification tasks where distance-based metrics have little meaning.

Completely general backpropagation equations for determining the gradient of η over the weight space or node output space can be obtained by inspection. Consider the network fragment in Figure 1, where units a and b belong to a hidden layer, and units x, y, and z constitute the output layer.

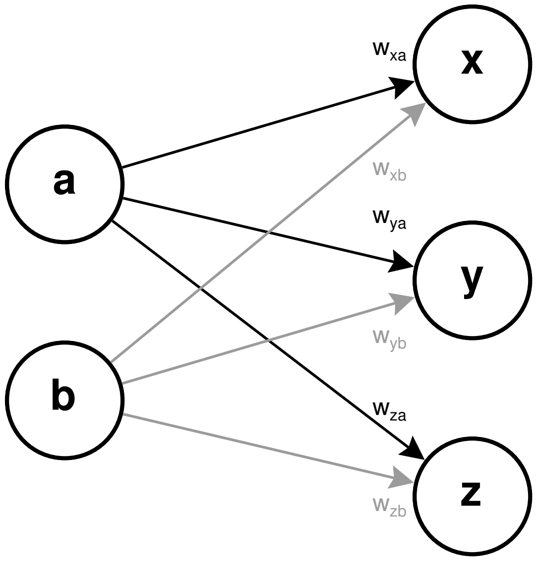

Figure 1. Network fragment illustrating backpropagation equations.

For an output unit i, such as i=x, the gradient with respect to the unit outputs can be derived directly from the objective function $\eta(\mathbf{Y})$:

$$\frac{\partial \eta}{\partial y_i} = \frac{\partial \eta(\mathbf{Y})}{\partial y_i} = \frac{\partial \eta(y_{N-N_O+1}, \ldots, y_N)}{\partial y_i}. \tag{14}$$

For hidden- or input-layer nodes, examination of Figure 1 reveals the general form of the desired derivative:

$$\frac{\partial \eta}{\partial y_i} = \sum_{j:c_{j,i}=1} \frac{\partial \eta}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial y_i} = \sum_{j:c_{j,i}=1} \frac{\partial \eta}{\partial y_j} \cdot F_j'(x_j) \cdot w_{j,i} = \sum_{j:c_{j,i}=1} \frac{\partial \eta}{\partial x_j} \cdot w_{j,i} \tag{15}$$

where

$$\frac{\partial \eta}{\partial x_i} = \frac{\partial \eta}{\partial y_i} \cdot F_i'(x_i). \tag{16}$$

Thus, the gradient components for all neurons are made available by beginning with an explicit computation for the output layer, then propagating these values back through the network toward the input layer and accumulating weighted contributions associated with each forward path. The gradient with respect to the weights is simply

$$\frac{\partial \eta}{\partial w_{i,j}} = \frac{\partial \eta}{\partial x_i} \frac{\partial x_i}{\partial w_{i,j}} = \frac{\partial \eta}{\partial x_i} y_j. \tag{17}$$

7. Differential Backpropagation

Combining CFM-based differential learning with backpropagation produces several interesting consequences.

First, the definition of η_{CFM} above in (13) guarantees that all derivatives of η_{CFM} are zero except for those associated with two of the network outputs. Derivatives of η_{CFM} also vanish for differentials that equal or exceed the confidence parameter, δ, associated with $\sigma_{\delta}(\cdot)$. Accordingly, training samples that produce large enough positive differentials no longer engender weight adaptation. This forms one justification for designating δ as a "confidence;" once the network correctly classifies an exemplar with a differential exceeding δ, it can be said to have learned that pattern with at least confidence δ. The training algorithm no longer needs to allocate portions of its functional complexity to improving discrimination performance on that example, and is instead free to concentrate on more difficult cases. Note however that such patterns may still influence learning if the relevant differential is perturbed when weights are modified during network exposure to other training samples.

Next, note that derivatives of η_{CFM} with respect to a network output y_i may depend on another output y_j in addition to y_i. This is in sharp contrast to most probabilistic supervised learning strategies, wherein each output displays an error derived from strictly local information. In the two-class problem posed by the Downlink Analyzer task, this results in symmetric weight updates—if the initial weights to the output layer are all zero, they will always remain antisymmetric, rendering one output redundant. Symmetry-breaking can be achieved via random initial weights or a few epochs of probabilistic training.

Finally, a subtle aspect of η_{CFM} is found in the fact that the objective function for a given training example depends only on two outputs. More significantly, which two specific outputs are involved varies across the training ensemble, i.e. different exemplars effectively have different objective functions. Furthermore, the two specific outputs can change for a given training pattern as the network adapts. This phenomenon is a further reflection of the nature of differential learning.

8. Regularization (*Entia non sunt multiplicanda praeter necessitatem*)

Excessive functional complexity is known to result in overfitting of network weights during training coupled with subsequent poor generalization performance on previously unseen input vectors. This problem is exacerbated when the training data is very sparse or otherwise not representative of the true underlying spatial probability distribution of the inputs. Methods of reducing network complexity achieve their goal either by reducing weight entropy or reducing the number of weights.

dlaNet supports two techniques for reducing weight entropy: weight decay and weight smoothing. Applying weight decay causes all weights to decay exponentially in magnitude during the training process. Weights to which the classification is insensitive are permitted to decay to zero, while those that are significant contributors are continuously reinforced by (9). In weight smoothing, a lowpass kernel function is convolved with the weights. Weight smoothing is not used in ***dlaNet*** since it is more applicable to retinotopic tasks where adjacent weights originate from highly correlated sources.

In typical weight removal regimes, some objective method determines the relative significance of each weight to the classification of the training ensemble *after* a locally optimum solution has been reached. Those weights that are deemed relatively unimportant are pruned and the network is retrained to reattain an optimum. The pruning procedure may be iteratively applied until performance would degrade beyond an acceptable threshold.

Weight pruning in ***dlaNet*** is achieved using the Optimal Brain Damage (OBD) algorithm [2], described in detail below. Other schemes are known, notably the Optimal Brain Surgeon (OBS) [4] procedure, but few have the computational efficiency of OBD.

9. Optimal Brain Damage

The OBD procedure models weight removal as a perturbation of the weight vector that maps to a perturbation in the objective function score. Each weight is assigned a saliency score that reflects the degradation in performance that would result if the connection were removed from the network. Because OBD is only applied after training has already led to a local optimum, any (small) perturbation is guaranteed to induce a penalty, rather than an improvement, in η.

Consider a Taylor series expansion of the average objective function $\eta(\mathbf{w})$ about the current weight vector $\mathbf{w}_0$ in weight space:

$$\begin{aligned}\eta(\mathbf{w}) = \eta(\mathbf{w}_0) + (\mathbf{w}-\mathbf{w}_0)^T \cdot \nabla_w \eta(\mathbf{w}_0) \\ + \frac{1}{2}(\mathbf{w}-\mathbf{w}_0)^T \cdot \nabla_w^2 \eta(\mathbf{w}_0) \cdot (\mathbf{w}-\mathbf{w}_0) + O\left(\|\mathbf{w}-\mathbf{w}_0\|^3\right).\end{aligned} \tag{18}$$

The $N_w \times N_w$ Hessian matrix has components

$$\left[\nabla_w^2 \eta(\mathbf{w}_0)\right]_{i,j} = \frac{\partial^2 \eta}{\partial w_i \partial w_j} \quad i = 1, \ldots, N_w;\ j = 1, \ldots, N_w. \tag{19}$$

The change in $\eta(\mathbf{w})$ due to a perturbation $\Delta\mathbf{w}\mathbf{=w}\text{-}\mathbf{w}_0$ is

$$\begin{aligned}\Delta\eta(\Delta\mathbf{w}) = \Delta\mathbf{w}^T \cdot \nabla_w \eta(\mathbf{w}_0) \\ + \frac{1}{2}\Delta\mathbf{w}^T \cdot \nabla_w^2 \eta(\mathbf{w}_0) \cdot \Delta\mathbf{w} + O\left(\|\Delta\mathbf{w}\|^3\right).\end{aligned} \tag{20}$$

With OBD, $\mathbf{w}_0$ represents a local optimum for $\eta(\mathbf{w})$, hence the gradient is zero. For simplicity, the local objective function surface is assumed to be well-approximated by a quadratic function, and hence all third- and higher-order terms are also negligible. Under these conditions, the penalty associated with $\Delta\mathbf{w}$ is

$$\Delta\eta(\Delta\mathbf{w}) = \frac{1}{2}\Delta\mathbf{w}^T \cdot \nabla_w^2 \eta(\mathbf{w}_0) \cdot \Delta\mathbf{w}. \tag{21}$$

Rewriting this using only scalars gives

$$\Delta\eta(\Delta\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N_w} \Delta w_i \left(\sum_{j=1}^{N_w} \frac{\partial^2 \eta(\mathbf{w}_0)}{\partial w_i \partial w_j} \Delta w_j \right) = \sum_{i=1}^{N_w} \Psi_i. \tag{22}$$

where Ψ_i is the degradation due to the perturbation in ith weight:

$$\Psi_i = \frac{1}{2}\Delta w_i \left(\sum_{j=1}^{N_w} \frac{\partial^2 \eta(\mathbf{w}_0)}{\partial w_i \partial w_j} \Delta w_j \right). \tag{23}$$

For the specific case of OBD, all perturbations have the form

$$\Delta\mathbf{w} = -\mathbf{M}\mathbf{w}_0 \tag{24}$$

where $\mathbf{M}$ is an $N_w \times N_w$ diagonal matrix with binary indicator variable elements

$$[\mathbf{M}]_{i,i} = m_{i,i} = \begin{cases} 0 & \text{if } w_i \text{ is not to be removed by OBD} \\ 1 & \text{if } w_i \text{ is to be removed by OBD.} \end{cases} \tag{25}$$

When $m_{i,i}$=1, Ψ_i is saliency of the ith weight.

10. Diagonal-Hessian OBD

Consider the saliency of the ith weight,

$$\Psi_i = \frac{1}{2} w_i \left(\sum_{j=1}^{N_w} \frac{\partial^2 \eta(\mathbf{w})}{\partial w_i \partial w_j} w_j \right), \tag{26}$$

which is based on the product of one row of the $N_w \times N_w$ Hessian and the perturbation vector -$\mathbf{M}\mathbf{w}_0$. In general, it is impractical to compute or even store the complete Hessian. The seminal OBD paper [2] adopts a diagonal-Hessian assumption and discards all cross terms, resulting in the simplified expression

$$\Psi_i = \frac{1}{2}\frac{\partial^2\eta(\mathbf{w})}{\partial w_i^{\,2}} w_i^2 \tag{27}$$

for the weight saliencies. This approximation is convenient because the diagonal elements of the Hessian are easily derived by augmenting the backward pass of the standard backpropagation algorithm with two new equations:

$$\frac{\partial^2\eta}{\partial y_i^{\,2}} = \begin{cases} \dfrac{\partial}{\partial y_i}\left(\dfrac{\partial\eta(\mathbf{Y})}{\partial y_i}\right) = \dfrac{\partial^2\eta(y_{N-N_O+1}, \ldots, y_N)}{\partial y_i^{\,2}} & \text{for } i > N - N_O \\ \dfrac{\partial}{\partial y_i}\left(\displaystyle\sum_{j:c_{j,i}=1} \dfrac{\partial\eta}{\partial x_j}\dfrac{\partial x_j}{\partial y_i}\right) = \displaystyle\sum_{j:c_{j,i}=1} w_{j,i}^2\left(\dfrac{\partial\eta}{\partial y_j}F_j''(x_j) + \dfrac{\partial^2\eta}{\partial y_j^{\,2}}F_j'(x_j)^2\right) & \text{for } i \le N - N_O \end{cases} \tag{28}$$

and

$$\frac{\partial^2\eta}{\partial w_{i,j}^2} = \frac{\partial}{\partial w_{i,j}}\left(\frac{\partial\eta}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial w_{i,j}}\right) = y_j^2\left(\frac{\partial\eta}{\partial y_i}F_i''(x_i) + \frac{\partial^2\eta}{\partial y_i^2}F_i'(x_i)^2\right). \tag{29}$$

Of course, these quantities are required only during the regularization phase and not during each epoch of training.

By assuming a diagonal Hessian, this form of OBD ignores the pairwise contributions to $\Delta\eta$ of eliminating each 2-tuple of weights. This is not an approximation if only a single weight is masked at a time. In practice, however, the method works well even if moderate quantities of low-saliency parameters are simultaneously removed. In fact, due to the isolation of saliencies by the diagonal approximation, computing all individual saliencies is tantamount to setting $\Delta\mathbf{w}$=-$\mathbf{w}_0$ in (21) above—a rather dramatic perturbation!

The OBS reference [4] submits that OBD can sometimes remove the wrong weight, but this appears to be a problem only in low-dimensionality cases; John Hampshire asserts that this misfeature is only associated with η_{MSE} in pattern recognition tasks. [Furthermore, the OBS paper strongly invites an invocation of *Watola's First Maxim of Neural Networks*, which firmly states "Never trust any reference that cites the XOR problem as evidence."]

11. Full-Hessian OBD

Full-Hessian OBD resolves the saliency of each weight according to (26) without directly computing or storing the Hessian elements. It accomplishes this by exploiting a numerically stable algorithm for computing the product of the Hessian matrix and an arbitrary vector of compatible dimensions, *viz.*, a perturbation vector. The lynchpin of this procedure is the definition of an operator $R_{v;w}\{\cdot\}$ such that

$$R_{v;w}\left\{f(\mathbf{w})\right\} = \frac{\partial}{\partial r} f(\mathbf{w} + r\mathbf{v})\bigg|_{r=0} \tag{30}$$

as described by Pearlmutter [3]. Because $R_{v;w}\{\cdot\}$ is a linear differential operator, it possesses the following useful properties, all of which are easily derived:

$$\begin{aligned} R_v\left\{c \cdot f(\mathbf{w})\right\} &= c \cdot R_v\left\{f(\mathbf{w})\right\} \\ R_v\left\{f(\mathbf{w}) + g(\mathbf{w})\right\} &= R_v\left\{f(\mathbf{w})\right\} + R_v\left\{g(\mathbf{w})\right\} \\ R_v\left\{f(\mathbf{w}) \cdot g(\mathbf{w})\right\} &= g(\mathbf{w}) \cdot R_v\left\{f(\mathbf{w})\right\} + f(\mathbf{w}) \cdot R_v\left\{g(\mathbf{w})\right\} \\ R_v\left\{f(g(\mathbf{w}))\right\} &= f'(g(\mathbf{w})) \cdot R_v\left\{g(\mathbf{w})\right\} \\ R_v\left\{\frac{d}{dx} f(\mathbf{w})\right\} &= \frac{d}{dx} R_v\left\{f(\mathbf{w})\right\} \\ R_v\left\{\mathbf{w}\right\} &= \mathbf{v} \end{aligned} \tag{31}$$

Note that the w subscript is deliberately omitted above and for the remainder of this discussion since OBD (and $R_v\{\cdot\}$) always operate with respect to the weights. A further propitious consequence of definition (30) is

$$R_v\left\{\nabla_w \eta(\mathbf{w})\right\} = \nabla_w^2 \eta(\mathbf{w}) \cdot \mathbf{v} \tag{32}$$

which implies that the desired saliencies are

$$\Psi_i = \frac{1}{2} w_i \cdot R_v\left\{\frac{\partial \eta}{\partial w_i}\right\}. \tag{33}$$

Once again, an augmented backpropagation procedure can be used to collect these quantities. This time, however, both the forward and backward passes must be supplemented. The forward pass requires the following pair of additional equations for non-input nodes:

$$R_v\{x_i\} = \sum_{j: c_{i,j} = 1} \left(w_{i,j} \cdot R_v\{y_j\} + v_{i,j} y_j \right) \tag{34}$$

and

$$R_v\{y_i\} = R_v\left\{F_i(x_i)\right\} = F'(x_i) \cdot R_v\{x_i\}. \tag{35}$$

The extra backward pass equations are:

$$R_v\left\{\frac{\partial\eta}{\partial y_i}\right\} = \begin{cases} \displaystyle\sum_{j=N-N_O+1}^{N} \frac{\partial}{\partial y_j}\left(\frac{\partial\eta}{\partial y_i}\right)\cdot R_v\{y_j\}, & N-N_O < i \le N \\ \displaystyle\sum_{j:c_{j,i}=1} \left(w_{j,i}\cdot R_v\left\{\frac{\partial\eta}{\partial x_j}\right\} + v_{j,i}\cdot\frac{\partial\eta}{\partial x_j}\right), & 1 \le i \le N-N_O \end{cases} \tag{36}$$

$$R_v\left\{\frac{\partial\eta}{\partial x_i}\right\} = R_v\left\{\frac{\partial\eta}{\partial y_i}\frac{\partial y_i}{\partial x_i}\right\} = F_i'(x_i)\cdot R_v\left\{\frac{\partial\eta}{\partial y_i}\right\} + \frac{\partial\eta}{\partial y_i}\cdot F_i''(x_i)\cdot R_v\{x_i\} \tag{37}$$

$$R_v\left\{\frac{\partial\eta}{\partial w_{i,j}}\right\} = R_v\left\{\frac{\partial\eta}{\partial x_i}\frac{\partial x_i}{\partial w_{i,j}}\right\} = \frac{\partial\eta}{\partial x_i}\cdot R_v\{y_j\} + y_j\cdot R_v\left\{\frac{\partial\eta}{\partial x_i}\right\}. \tag{38}$$

Note that for simple objective functions like η_{MSE}, (36) simplifies to

$$R_v\left\{\frac{\partial\eta}{\partial y_i}\right\} = \frac{\partial^2\eta(y_{N-N_O+1}, \ldots, y_N)}{\partial y_i^2}\cdot R_v\{y_i\} \tag{39}$$

for the case of output units.

Unfortunately, full-Hessian OBD cannot be used as a drop-in replacement for diagonal-Hessian OBD. Because the second-order terms account for penalties associated with pairwise removal of weights, setting $\mathbf{v}=\Delta\mathbf{w}=-\mathbf{w}_0$ would force every possible pairwise pruning to be considered in computing saliency scores, including those associated with masking high-saliency parameters. Clearly, this is undesirable.

One possible compromise is to use both the Diagonal-Hessian and Full-Hessian techniques in conjunction. The approximate method can be used to select a candidate set of expendable weights without great need to be conservative. A better estimate of saliencies can then be obtained for just these weights using the complete Hessian.

Note however that the diagonal-Hessian OBD method works very well in practice despite the approximation it entails. Full-Hessian OBD is not critical to Downlink Analyzer or ***dlaNet***. It is only described in such painstaking detail here because it serves as the basis for the Discriminative Diagnosis procedure used by ***DDD***.

12. Summary of Dual-class CFM-based Differential Backpropagation Equations

Standard forward pass for adaptation:

$$x_i = \begin{cases} \sum_{j:\, c_{i,j}=1} w_{i,j} y_j & \text{for } i > N_I \\ 0 & \text{otherwise} \end{cases} \tag{40}$$

$$y_i = \begin{cases} F_i(x_i) & \text{for } i > N_I \\ X_i & \text{otherwise} \end{cases} \tag{41}$$

$$\frac{\partial y_i}{\partial x_i} = \begin{cases} F_i'(x_i) & \text{for } i > N_I \\ 0 & \text{otherwise} \end{cases} \tag{42}$$

$$\frac{\partial^2 y_i}{\partial x_i^2} = \begin{cases} F_i''(x_i) & \text{for } i > N_I \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

Augmented forward pass for full-Hessian OBD:

$$R_{v;w}\{x_i\} = \begin{cases} \sum_{j:c_{i,j}=1} \left(w_{i,j} \cdot R_{v;w}\{y_j\} + v_{i,j} y_j \right) & \text{for } i > N_I \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

$$R_{v;w}\{y_i\} = F_i'(x_i) \cdot R_{v;w}\{x_i\} \tag{45}$$

Standard backward pass for adaptation:

$$\frac{\partial \eta}{\partial y_i} = \begin{cases} \sum_{j:c_{j,i}=1} w_{j,i} \frac{\partial \eta}{\partial x_j} & \text{for } i \le N - N_O \\ +\sigma_\delta'\left(Y_{(1)} - Y_{(2)}\right) & \text{for } i = (1) \\ -\sigma_\delta'\left(Y_{(1)} - Y_{(2)}\right) & \text{for } i = (2) \end{cases} \tag{46}$$

$$\frac{\partial \eta}{\partial x_i} = \frac{\partial \eta}{\partial y_i} \cdot F_i'(x_i) \tag{47}$$

Augmented backward pass for diagonal-Hessian OBD:

$$\frac{\partial^2\eta}{\partial y_i^2} = \begin{cases} \sum_{j:c_{j,i}=1} w_{j,i}^2 \left(\frac{\partial\eta}{\partial y_j} \cdot F_j''(x_j) + \frac{\partial^2\eta}{\partial y_j^2} \cdot F_j'(x_j)^2 \right), & 1 < i \le N - N_O \\ \sigma_\delta''(Y_{(1)} - Y_{(2)}), & N - N_O < i \le N \end{cases} \tag{48}$$

Augmented backward pass for full-Hessian OBD:

$$R_{v;w}\left\{\frac{\partial\eta}{\partial y_i}\right\} = \begin{cases} \sum_{j:c_{j,i}=1} \left(w_{j,i} \cdot R_{v;w}\left\{\frac{\partial\eta}{\partial x_j}\right\} + v_{j,i} \cdot \frac{\partial\eta}{\partial x_j} \right) & \text{for } i \le N - N_O \\ + \sigma_\delta(Y_{(1)} - Y_{(2)}) \cdot (R_{v;w}\{y_{(1)}\} - R_{v;w}\{y_{(2)}\}) & \text{for } i = (1) \\ - \sigma_\delta(Y_{(1)} - Y_{(2)}) \cdot (R_{v;w}\{y_{(1)}\} - R_{v;w}\{y_{(2)}\}) & \text{for } i = (2) \end{cases} \tag{49}$$

$$R_{v;w}\left\{\frac{\partial\eta}{\partial x_i}\right\} = F_i'(x_i) \cdot R_{v;w}\left\{\frac{\partial\eta}{\partial y_i}\right\} + \frac{\partial\eta}{\partial y_i} \cdot F_i''(x_i) \cdot R_{v;w}\{x_i\} \tag{50}$$

$$R_{v;w}\left\{\frac{\partial\eta}{\partial w_{i,j}}\right\} = \frac{\partial\eta}{\partial x_i} \cdot R_{v;w}\{y_j\} + y_j \cdot R_{v;w}\left\{\frac{\partial\eta}{\partial x_i}\right\} \tag{51}$$

Distribution:

M. Belongie	238-420
F. Pollara	238-420
S. Townes	238-420
D. Watola	238-420
N. Xiao	238-420