April 2, 2019

Conservative updates by backward-forward cycles

These notes describe an iterative scheme for computing the exact batch-conservative update for standard neural networks with arbitrary activation function f. The algorithm is in the spirit of Newton's root finding algorithm, where solutions are fixed points of the iteration scheme. However, these solutions are exactly conservative only in a local sense. When large parameter changes are required to accommodate a batch of data, the minimizer found by the algorithm may differ from the global minimizer.

The training task is to make the smallest 2-norm changes to the network parameters so that a batch of input data is exactly mapped to given output data. Such regression problems arise, for example, when batch-conservatively training an autoencoder.

For simplicity we consider networks without bias parameters.

Notation and overview

We use indices i and j for nodes and the notation $i \to j$ for an edge between nodes. A layered architecture is not assumed, and in fact the formulas are just as simple (or even simpler) without making reference to layers. However, the update computation does distinguish five kinds of nodes that every feed-forward network has, layered or not. First there are the input nodes I that hold the input data values. When successfully trained, the propagated input data will match corresponding output data that resides in the output nodes O. All nodes not in I or O comprise the hidden nodes H. We single out a subset H_- of the hidden nodes by the property that all nodes in this subset get input only from nodes in I. Similarly, the subset H_+ of hidden nodes sends outputs only to nodes in O. We use E for the set of all edges in the network.

Post and pre-activation node values are denoted x and y respectively, and the network weights are w. The equations for neuron $j \in H$ is the pair

$$y_j^k = \sum_{i \to j} x_i^k \, w_{i \to j} \tag{1a}$$

$$x_j^k = f(y_j^k), \tag{1b}$$

where $k \in K$ is the index for data items in the training batch. There is an activation function only at the hidden nodes, not the output nodes. Thus there are no x variables at the output nodes, just as there are no y variables at the input nodes. When trained, the network maps x_i^k, $i \in I$ to y_j^k, $j \in O$ for all data $k \in K$.

The initial network weights are denoted $w_{i\to j}^0$. These are updated in an "outer" optimization loop as

$$w_{i\to j}^0 \to w_{i\to j}^0 + \Delta w_{i\to j} \coloneqq w_{i\to j}. \tag{2}$$

Conservative learning aims to solve the batch-regression problem while minimizing

$$\sum_{i\to j\in E} \|\Delta w_{i\to j}\|^2. \tag{3}$$

The weight increments Δw are updated as

$$\Delta w_{i\to j} + \delta w_{i\to j} \to \Delta w'_{i\to j} \tag{4}$$

upon completion of an "inner" loop. The inner loop solves a linearized optimization problem wherein $\delta w_{i\to j}$ participates as a variable. Most of the work occurs in this inner loop, and as we will see, involves cycles of backward and forward propagation.

The inner loop is initialized by feeding-forward all items in the training batch K using the current weights $w_{i\to j}$ set by the outer loop. No approximations are made in the initialization. In the inner loop the initialized (feed-forward) node values, denoted $\tilde{x}$ and $\tilde{y}$, are treated as constants and the neuron equations (1) are linearized about these values:

$$\delta y_j^k = \sum_{i\to j} \left(\delta x_i^k \, w_{i\to j} + \tilde{x}_i^k \, \delta w_{i\to j}\right) \tag{5a}$$

$$\delta x_j^k = f'(\tilde{y}_j^k)\delta y_j^k. \tag{5b}$$

The derivatives $f'(\tilde{y}_j^k) \coloneqq df_j^k$ are also treated as constants in the inner loop.

Equation (5a) has two cases owing to the fact that $\delta x_i^k = 0$ for $i \in I$:

$$\forall\, (j \in H_-, k \in K): \quad \delta y_j^k = \sum_{i\to j} \tilde{x}_i^k \, \delta w_{i\to j} \tag{6a}$$

$$\forall\, (j \notin H_-, k \in K): \quad \delta y_j^k = \sum_{i\to j} \left(\delta x_i^k \, w_{i\to j} + \tilde{x}_i^k \, \delta w_{i\to j}\right) \tag{6b}$$

At nodes $j \in O$ we know what changes δy_j^k are required for the network to produce the given output data. Let these changes, determined by feed-forward of data in the outer optimization loop, be ϵ_j^k. Equation (5b) is then replaced by

$$\forall\, (j \in O, k \in K): \quad \epsilon_j^k = \delta y_j^k. \tag{7}$$

The linear equations in the inner optimization loop treat the output discrepancies ϵ_j^k as small. When the discrepancies are indeed small, the variables δx, δy and δw will be small as well and the the succession of linearizations performed by the outer loop have a chance of converging. Only in this case do we expect the net update Δw to be properly conservative. At the start of training, when the discrepancies often are not small, the updates might be "neo-conservative", that is, only a local minimum of (3). However, we can hope this will just be transient behavior and after a certain stage of training the discrepancies will always be sufficiently small so the updates Δw are conservative in a global sense.

Lagrangian

The constrained optimization for the batch-conservative update in the ("inner-loop") linear approximation is compactly defined by the stationary points of the following Lagrangian:

$$\mathcal{L} = \frac{1}{2} \sum_{i \to j \in E} \left(\Delta w_{i\to j} + \delta w_{i\to j}\right)^2 \tag{8}$$

$$+ \sum_{\substack{k\in K\\ i\in H}} \beta_i^k \left(\delta x_i^k - df_i^k \, \delta y_i^k\right) \tag{9}$$

$$+ \sum_{\substack{k\in K\\ j\in H_-}} \gamma_j^k \left(\delta y_j^k - \sum_{i\to j} \tilde{x}_i^k \, \delta w_{i\to j}\right) \tag{10}$$

$$+ \sum_{\substack{k\in K\\ j\notin H_-\cup O}} \gamma_j^k \left(\delta y_j^k - \sum_{i\to j} \left(\delta x_i^k \, w_{i\to j} + \tilde{x}_i^k \, \delta w_{i\to j}\right)\right) \tag{11}$$

$$+ \sum_{\substack{k\in K\\ j\in O}} \gamma_j^k \left(\epsilon_j^k - \sum_{i\to j} \left(\delta x_i^k \, w_{i\to j} + \tilde{x}_i^k \, \delta w_{i\to j}\right)\right). \tag{12}$$

The variables in the unconstrained optimization are δw, δx, δy and the Lagrange multipliers β and γ. Stationarity with respect to the latter just reproduce equations (5b) and (6). For δx we obtain

$$\forall\, (i \in H, k \in K): \quad \beta_i^k = \sum_{i\to j} w_{i\to j} \, \gamma_j^k, \tag{13}$$

and stationarity with respect to δy implies

$$\forall\, (i \in H, k \in K): \quad \gamma_i^k = df_i^k \, \beta_i^k. \tag{14}$$

We can eliminate β between the last two equations:

$$\forall\,(i \in H, k \in K): \quad \gamma_i^k = df_i^k \sum_{i\to j} w_{i\to j}\,\gamma_j^k. \tag{15}$$

The final set of equations follow from stationarity with respect to δw:

$$\forall\,(i \to j \in E): \quad \Delta w_{i\to j} + \delta w_{i\to j} = \sum_{k\in K} \tilde{x}_i^k \gamma_j^k \tag{16}$$

$$\coloneqq \tilde{x}_i \cdot \gamma_j. \tag{17}$$

A special case of this applies to $j \in H_-$. Substituting $\delta w_{i\to j}$ for this case from (16) into (6a) we obtain

$$\forall\,(j \in H_-, k \in K): \quad \delta y_j^k = \sum_{i\to j} \tilde{x}_i^k\,\left(\tilde{x}_i \cdot \gamma_j - \Delta w_{i\to j}\right). \tag{18}$$

Making the same substitution but now into (6b) and using (5b), we obtain

$$\forall\,(j \notin H_-, k \in K): \quad \delta y_j^k = \sum_{i\to j} \left(df_i^k \delta y_i^k\, w_{i\to j} + \tilde{x}_i^k(\tilde{x}_i \cdot \gamma_j - \Delta w_{i\to j})\right). \tag{19}$$

Inner loop initialization

Each cycle of the linear optimization is initialized at the output layer:

$$\forall\,(j \in O, k \in K): \quad \epsilon_j^k = \sum_{i\to j} \left(df_i^k\,\delta y_i^k\,w_{i\to j} + \tilde{x}_i^k\,\delta w_{i\to j}\right). \tag{20}$$

Introducing an iteration counter $n = 0, 1, \ldots$ for the inner loop and the matrix notation

$$\mathcal{E}(n)_{kj} \coloneqq \epsilon_j^k - \sum_{i\to j} df_i^k\,w_{i\to j}\,\delta y(n)_i^k \tag{21}$$

$$\widetilde{X}_{ik} \coloneqq \tilde{x}_i^k \tag{22}$$

$$\delta W(n)_{ij} \coloneqq \delta w(n)_{i\to j}, \tag{23}$$

we can rewrite (20) in matrix form:

$$\mathcal{E}(n) = \widetilde{X}^T \delta W(n). \tag{24}$$

In the first cycle we set $\delta y(0) = 0$, in effect ignoring the accumulated effects of changes to the node values when resolving the output discrepancy. Since $\mathcal{E}(0)$ is just the known discrepancy ϵ_j^k from the feed-forward of data in the outer loop, by inverting (24) we can get an initial estimate of the weight changes $\delta W(0)$ on edges to the output nodes.

In the generic case of (24) we can only solve for $\delta W(n)$ when the dimensions of the $|H_+| \times |K|$ matrix $\widetilde{X}$ satisfy

$$|K| \leq |H_+|. \tag{25}$$

When the inequality is strict we get a unique solution by imposing, additionally, that $\delta W(n)$ has minimum 2-norm. But that is exactly what we seek in conservative learning. Assuming the batch size and network architecture satisfy (25), equation (24) is inverted by applying the pseudo-inverse:

$$\delta W(n) = \widetilde{X}(\widetilde{X}^T\widetilde{X})^{-1}\mathcal{E}(n). \tag{26}$$

Computing the inverse of the $|K| \times |K|$ matrix $\widetilde{X}^T\widetilde{X}$ is the only matrix inverse required by the algorithm and it is needed only once per iteration of the outer optimization loop.

Defining two more matrices, for $j \in O$,

$$\Delta W_{ij} \coloneqq \Delta w_{i\to j} \tag{27}$$

$$\Gamma(n)_{kj} \coloneqq \gamma(n)_j^k, \tag{28}$$

we can rewrite (16), for $j \in O$, in matrix form:

$$\Delta W + \delta W(n) = \widetilde{X}\,\Gamma(n). \tag{29}$$

Multiplying this by $(\widetilde{X}^T\widetilde{X})^{-1}\widetilde{X}^T$, and using (26), we obtain a formula for the γ_j^k Lagrange multipliers for $j \in O$:

$$\Gamma(n) = (\widetilde{X}^T\widetilde{X})^{-1}\left(\widetilde{X}^T\Delta W + \mathcal{E}(n)\right). \tag{30}$$

Backward-forward cycle

One cycle of the inner optimization loop comprises four steps:

1. Initialize γ's at the output nodes.
2. Back-propagate γ's down to all the hidden nodes.

3. Initialize δy's at nodes adjacent to the input nodes.

4. Forward-propagate δy's up to the nodes adjacent to the output nodes.

Step 1 for cycle n is given by equation (30). Recall that the data for the first cycle, $\mathcal{E}(0)$, is just the set of discrepancies ϵ_j^k at the output nodes after the forward pass of the data in the training batch.

Step 2 is the evaluation of (15) by back-propagation. Propagation terminates at nodes $i \in H_-$ that only receive inputs from input nodes.

Step 3 is given by equation (18).

Step 4 is the evaluation of (19) by forward-propagation. Note that the γ's in this equation were determined in step 2 and propagation extends all the way to δy_i^k with $i \in H_+$. This defines $\mathcal{E}(n+1)$ via $\delta y(n+1)_i^k$ in equation (21) to start the next cycle.

We can use the matrix $\Gamma(n)$ that starts each cycle both as a criterion for terminating iterations and as a means for stabilizing/boosting convergence. For the former we monitor the norm

$$\|\Gamma(n+1) - \Gamma(n)\| \tag{31}$$

and terminate iterations when it falls below some threshold. For the latter we simply make the replacement

$$\Gamma(n+1) \leftarrow (1-r)\Gamma(n+1) + r\Gamma(n), \tag{32}$$

where r is a relaxation parameter and $0 < r < 1$ enhances stability. If stability is not a concern, one can try to accelerate convergence by over-relaxation, or $r < 0$.

By (16), from the converged γ's we obtain the weight changes of the inner (linear optimization) loop by

$$\forall\ (i \to j \in E): \quad \delta w_{i\to j} = \tilde{x}_i \cdot \gamma_j - \Delta w_{i\to j}. \tag{33}$$

When the inner loop is exited, the weights are updated by (4) and the same training data is fed back through the network to define $\tilde{x}$ and $\tilde{y}$ for the next round of backward-forward cycles. We can use the final $\mathcal{E}$ matrix of the inner loop to decide when to terminate the outer loop. By (26), when the converged $\|\mathcal{E}\|$ is small, so are the δw needed to fix the network outputs on the training batch.

Comparison with SGD

Because the inner optimization begins with backward propagation initialized by output discrepancies, one might think that limiting the inner loop to a single half-cycle is equivalent to the stochastic gradient descent (SGD) algorithm. However, as we now show, this is not the case.

Here is a summary of the weight update when we perform just a single outer-loop iteration and a half-cycle in the inner loop. In the first iteration of the outer loop, Δw (accumulated weight updates) is zero and (30) reduces for $n = 0$ (first inner-loop iteration) to

$$\forall\ (j \in O, k \in K): \quad \gamma_j^k = \sum_{k' \in K} (\widetilde{X}^T \widetilde{X})^{-1}_{kk'}\ \epsilon_j^{k'}. \tag{34}$$

We then use the back-propagation equation (15), again with $\Delta w = 0$, to determine the γ's on the hidden nodes:

$$\forall\ (i \in H, k \in K): \quad \gamma_i^k = f'(\tilde{y}_i^k) \sum_{i \to j} w^0_{i \to j}\, \gamma_j^k. \tag{35}$$

Finally, from (16) we get the weight updates:

$$\forall\ (i \to j \in E): \quad \delta w_{i \to j} = \sum_{k \in K} \tilde{x}_i^k \gamma_j^k. \tag{36}$$

Only (35) resembles the back-propagation of SGD, where information at the output nodes is propagated, independently for all items in the "mini-batch", to all the hidden nodes. Already in the initialization (34) we see that there is "mixing" of the mini-batch discrepancies, and the weight updates in (36) are not a simple average but a weighted average of the propagated variables (γ's) over the mini-batch.

A more direct contrast with SGD comes from the fact that the propagated variables in SGD are defined on edges rather than nodes. To facilitate that comparison, here is a derivation of the SGD update in our notation.

For our regression problem SGD computes the gradient of the loss

$$L = \frac{1}{2} \sum_{i \in O} (y_i - y_i^*)^2, \tag{37}$$

where the y^* are the given output values and we have simplified the notation for the case of a single training item. As before, $\tilde{x}$ and $\tilde{y}$ are post- and pre-activation node values when

the data is fed into the network with the current weights, w^0. The output discrepancies therefore satisfy

$$\forall\, (i \in O): \quad \tilde{y}_i + \epsilon_i = y_i^*. \tag{38}$$

By repeated application of the chain rule,

$$g_{i\to j} \coloneqq \frac{\partial L}{\partial w_{i\to j}} \tag{39}$$

$$= \frac{\partial L}{\partial y_j}\,\frac{\partial y_j}{\partial w_{i\to j}} \tag{40}$$

$$= \frac{\partial L}{\partial y_j}\,\tilde{x}_i \tag{41}$$

$$= \frac{\partial L}{\partial x_j}\,\frac{\partial x_j}{\partial y_j}\,\tilde{x}_i \tag{42}$$

$$= \frac{\partial L}{\partial x_j}\,f'(\tilde{y}_j)\,\tilde{x}_i. \tag{43}$$

For our loss function, equation (41) at the output nodes becomes

$$\forall\, (j \in O): \quad g_{i\to j} = -\epsilon_j\,\tilde{x}_i, \tag{44}$$

and initializes the back-propagation.

Continued application of the chain rule to (43) gives

$$g_{i\to j} = f'(\tilde{y}_j)\,\tilde{x}_i \left(\sum_{j\to l} \frac{\partial L}{\partial y_l}\,\frac{\partial y_l}{\partial x_j}\right) \tag{45}$$

$$= f'(\tilde{y}_j)\,\tilde{x}_i \sum_{j\to l} \frac{\partial L}{\partial y_l}\,w^0_{j\to l}. \tag{46}$$

Multiplying by $\tilde{x}_j$ and using (41) we arrive at the back-propagation equation for the gradients:

$$\tilde{x}_j\,g_{i\to j} = f'(\tilde{y}_j)\,\tilde{x}_i \sum_{j\to l} w^0_{j\to l}\,g_{j\to l}. \tag{47}$$

This is clearly different from the γ-recursion (35) in conservative learning. Another difference, of course, is that in SGD the gradient step size is an empirical parameter η, the learning rate:

$$\delta w_{i\to j} = -\eta\,g_{i\to j}. \tag{48}$$