Multi-layer networks

A multi-layer network is a class of functions $f : \mathbb{R}^d \to \mathbb{R}$ based on directed acyclic graph $G = (V, E)$ with $d$ source vertices and a single sink vertex. For such a function $f$, the sources in $G$ are associated with the inputs $x_1, \ldots, x_d$ of $f$; the sink is associated with the output $\hat{y}$ of $f$, and the other vertices are associated with intermediate variables used in the computation of $f$. Each edge $(u, v) \in E$ is associated with a weight $w_{u,v} \in \mathbb{R}$, and each non-source vertex $v \in V$ is associated with a link function $g_v : \mathbb{R} \to \mathbb{R}$. The value of $v$, given the values of its parents $\pi_G(v) := \{ u \in V : (u, v) \in E \}$, is

$$v := g_v(z_v), \quad z_v := \sum_{u \in \pi_G(v)} w_{u,v} \cdot u.$$

This naturally provides a recursive approach to computing the value of the function $f$, given values of the sources $x_1, \ldots, x_d$.

Evaluating a multi-layer network function

An alternative to the recursive process for computing the value of a multi-layer network function $f$ is the forward propagation algorithm. The vertices $V$ are partitioned into layers $V_0, V_1, \ldots$. The source vertices $x_1, \ldots, x_d$ constitute $V_0$. A vertex $v$ is in $V_l$ if the longest path in $G$ from a source vertex to $v$ has $l$ edges. Thus, edges are always directed from one vertex to another vertex in a higher layer. To compute the function $f$ given values for the inputs $x_1, \ldots, x_d$, we compute the values for all vertices in $V_1$, then the values for all vertices in $V_2$, and so on, until the value for the output vertex $\hat{y}$ is computed.

Fitting a multi-layer network to data

Fitting multi-layer networks to data amounts to choosing the weight vectors associated with every vertex based on training examples. Each training example is associated with a loss function that measures how poor the output $\hat{y}$ of a function $f$ is on the given input values $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$. For example, if the training example comes with both the input values $x$ and a target output value $y \in \mathbb{R}$, then the squared loss of the function $f$ on this example is

$$\ell = (\hat{y} - y)^2.$$

Many fitting algorithms (e.g., gradient descent, stochastic gradient method) require a subroutine to compute of the gradient of the loss on an individual training example with respect to the weights defining the multi-layer network function $f$. The most common algorithm used here is the backward propagation algorithm (also called back-propagation and backprop).\(^3\)

---

\(^1\)Extension to multiple sinks (i.e., multiple output functions) is straightforward.

\(^2\)One could also choose the link function for each vertex, but in practice, these are usually fixed along with the graph.

\(^3\)Sometimes “backprop” is used to refer to the gradient descent algorithm combined with this particular efficient algorithm for computing gradients.
Chain rule

Backward propagation is based on the chain rule for derivatives. Suppose a function \(h: \mathbb{R} \to \mathbb{R}\) is defined by

\[
h(t) := h_0(r_1, \ldots, r_n),
\]

\[r_i := h_i(t), \quad i = 1, \ldots, n,
\]

for some functions \(h_0: \mathbb{R}^n \to \mathbb{R}\) and \(h_i: \mathbb{R} \to \mathbb{R}\) for \(i = 1, \ldots, n\). The chain rule gives the derivative \(h'\) of \(h\) in terms of the derivatives \(h_i'\) of \(h_i\), for \(i = 1, \ldots, n\):

\[
h'(t) = \sum_{i=1}^n \frac{\partial h_0(r_1, \ldots, r_n)}{\partial r_i} \cdot h_i'(t).
\]

This is used to derive the backprop algorithm for computing the derivative of \(\ell\) with respect to a weight \(w_{u,v}\) for any \((u,v) \in E\).

Backprop derivation

The derivation begins with a basic application of the chain rule:

\[
\frac{\partial \ell}{\partial w_{u,v}} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v} \cdot \frac{\partial v}{\partial w_{u,v}}.
\]

The first factor on the right-hand side, \(\partial \ell/\partial \hat{y}\), is typically easy to handle. For example, for the squared loss, \(\partial \ell / \partial \hat{y} = 2(\hat{y} - y)\).

This is straightforward to compute after computing the value of \(\hat{y}\) given the input values \(x_1, \ldots, x_d\). The third factor is also easy to handle with a basic application of the chain rule:

\[
\frac{\partial v}{\partial w_{u,v}} = \frac{\partial v}{\partial z_v} \cdot \frac{\partial z_v}{\partial w_{u,v}} = g'_v(z_v) \cdot u,
\]

where \(g'_v\) is the derivative of the link function \(g_v\). This is similar to the computation required in forward propagation, except that \(g'_v\) (and not \(g_v\)) is applied to \(z_v\), and an additional multiplication by \(u\) is needed.

It remains to handle the second factor, \(\partial \hat{y}/\partial v\), for \(v \in V\). If \(v = \hat{y}\), the derivative is \(\partial \hat{y}/\partial \hat{y} = 1\). But what about for \(v \neq \hat{y}\)? This is where the chain rule comes in. Suppose we want to compute \(\partial \hat{y}/\partial v\) for some \(v \in V\). Assume that we have already computed \(\partial \hat{y}/\partial v'\) for all \(v' \in V_{l'}\) where \(l' > l\). Let the edges out of \(v\) in \(G\) be \((v, v_1), \ldots, (v, v_{\text{deg}(v)})\). Then, by the chain rule, we have

\[
\frac{\partial \hat{y}}{\partial v} = \sum_{i=1}^{\text{deg}(v)} \frac{\partial \hat{y}}{\partial v_i} \cdot \frac{\partial v_i}{\partial v}.
\]

The \(i\)-th summand on the right-hand side is the product of two factors. One factor, \(\partial v_i/\partial v\), is straightforward to compute using the chain rule:

\[
\frac{\partial v_i}{\partial v} = \frac{\partial v_i}{\partial z_{v_i}} \cdot \frac{\partial z_{v_i}}{\partial v} = g'_v(z_{v_i}) \cdot w_{v,v_i}.
\]

The other factor, \(\partial \hat{y}/\partial v_i\), has already been computed since edges in \(G\), such as \((v, v_i)\), are always directed towards vertices in higher layers \(l' > l\).

So, the backward propagation algorithm is as follows. First, execute the forward propagation algorithm, saving intermediate values \(z_v\) and \(v\) for all non-source \(v \in V\). Then, in descending order of layers \(l\), compute \(\partial \hat{y}/\partial v\) for all \(v \in V_l\) using the recursion given above, along with \(\partial \ell/\partial w_{u,v}\) for every edge \((u,v) \in E\) using the formulas given above.