BACKPROPAGATION

BORIS HANIN, POKEY RULE

The goal of this note is to explain weight update by backpropagation for a neural network. Formally, a neural network is the following data

$$\mathcal{N} := \{(V, E),\ \mathcal{L} : V \to \mathbb{R},\ \sigma : V \to \mathrm{NL}\}.$$

Here, (V, E) is the directed acylic graph whose vertices (resp. edges) represent the neurons (resp. connections), $\mathcal{L}$ is the loss function, and NL is a collection of possibly non-linear activations (including $\mathcal{L}$) so that σ_v is the non-linearity at neuron v.

In addition to the data of $\mathcal{N}$, we are given a state (i.e initialization of all weights)

$$W(\mathcal{N}) = \{w : E \to \mathbb{R}\}$$

as well as the incoming signals strengths

$$\mathrm{IN}(\mathcal{N}) = \{in : V \to \mathbb{R}\}$$

of all nodes after seeding $\mathcal{N}$ with (say, a batch of) training data. We denote by

$$\mathrm{OUT}(\mathcal{N}) = \{out(v) := \sigma_v(in(v))\}_{v \in V}$$

the resulting outgoing signals.

The Goal of backpropagation is to update wieghts by gradient descent on the loss function, which requires computing

$$\frac{\partial \mathcal{L}}{\partial w(e)}, \qquad \text{for each } e \in E. \tag{1}$$

Moreover, we require an algorithm with complexity $\lambda(\mathcal{N})$, the runtime of $\mathcal{N}$ on a batch of training data.

It is convenient to augment $\mathcal{N}$ to include the loss function as a node by replacing

$$V \mapsto \widetilde{V} := V \cup \{\mathrm{Loss}\}$$

Let us write $\mathcal{N}_{\mathrm{loss}}$ for the minimal collection of nodes so that $\mathcal{L}$ depends only on their activations $y(v)$. Then we replace

$$E \mapsto \widetilde{E} := E \cup \{v \to \mathrm{Loss}\}_{v \in \mathcal{N}_{\mathrm{Loss}}}$$

and set

$$in(\mathrm{Loss}) = \left(\{out(v)\}_{v \in \mathcal{N}_{\mathrm{Loss}}}\right).$$

so that $out(\mathrm{Loss}) = \mathcal{L}$.. We write

$$\widetilde{\mathcal{N}} = \{(\widetilde{V}, \widetilde{E}), \mathrm{OUT} : \widetilde{V} \to \mathbb{R},\ \mathrm{IN} : \widetilde{E} \to \mathbb{R}, \sigma : \widetilde{V} \to \mathrm{NL}\}$$

for the augmented network and state. Because $(\widetilde{V}, \widetilde{E})$ inherits being an acyclic directed graph from $\mathcal{N}$, there exists an enumeration

$$\widetilde{V} = \{v_0 = \text{Loss}, v_1, \ldots, v_{|V|}\}$$

so that for each $j = 0, \ldots, |V|$

$$v_j \quad \text{is a sink after removing } \{v_0, \ldots, v_{j-1}\} \text{ and } \{N_{v_i}\}_{i=0}^{j-1}.$$

Observe that $v_0 = \text{Loss}$. Such an enumeration is simple to write by hand for many popular neural nets and be calculated during training. The essence of backpropagation is the observation that $\mathcal{L}$ depends on a weight w attached to an edge $v' \to v$ only via $OUT(v)$. Hence, by the chain rule,

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \text{OUT}(v)} \cdot \frac{\partial \text{OUT}(v)}{\partial \text{IN}(v)} \cdot \frac{\partial \text{IN}(v)}{\partial w}.$$

Note that

$$\frac{\partial \text{OUT}(v)}{\partial \text{IN}(v)} = \frac{d}{dz}\Big|_{z=\text{IN}(v)} \sigma_v(z) \qquad \text{and} \qquad \frac{\partial \text{IN}(v)}{\partial w} = \text{OUT}(v') \tag{2}$$

are given quantities. Computing the expression in (1) thus reduces to comuting

$$\frac{\partial L}{\partial out(v)} \qquad \text{for all } v \in V.$$

This this done by the following algorithm:

\# back[v] denotes the set of vertices with an edge into v

in_grad = zeros($|V|$)
out_grad = zeros($|V|$)

\# initialize out_grad

for v_n in back[v_0]: out_grad[j] = $\frac{\partial \mathcal{L}}{\partial \text{out}(v_n)}$

for $n = 1, \ldots, |V|$ and $v \in$ back(v_{n-1}):

  in_grad[v] = out_grad[v] $\cdot \frac{d}{dz}\big|_{z=\text{in}(v)} \sigma_{v_j}(z)$ # update in_grad

  for $v_k \in$ back[v_j]:

    out_grad[v_k]+ = in_grad[v_j] $\cdot\, w(v_k \to v_j)$ # pass back out_grad

return out_grad

Department of Mathematics, MIT, Cambridge, MA 02139
E-mail address, B. Hanin: bhanin@mit.edu

??
E-mail address, P. Rule: pokey.rule@gmail.com