A Neural Network Gradients

We consider a neural network $f_{\boldsymbol{\theta}}(\cdot)$, with parameters $\boldsymbol{\theta}$. We maximize the likelihood (8), with respect to the parameters $\boldsymbol{\theta}$, using stochastic gradient. By negating the likelihood, we now assume it corresponds to minimize a cost $C(f_{\boldsymbol{\theta}}(\cdot))$, with respect to $\boldsymbol{\theta}$.

Following the classical "back-propagation" derivations (LeCun, 1985; Rumelhart et al., 1986) and the modular approach shown in (Bottou, 1991), any feed-forward neural network with L layers, like the one shown in Figure 3, can be seen as a composition of functions $f^l_{\boldsymbol{\theta}}(\cdot)$, corresponding to each layer l:

$$f_{\boldsymbol{\theta}}(\cdot) = f^L_{\boldsymbol{\theta}}(f^{L-1}_{\boldsymbol{\theta}}(\dots f^1_{\boldsymbol{\theta}}(\cdot)\dots))$$

Partionning the parameters of the network with respect to each layers $1 \leq l \leq L$, we write:

$$\boldsymbol{\theta} = (\boldsymbol{\theta}^1,\, \dots,\, \boldsymbol{\theta}^l,\, \dots,\, \boldsymbol{\theta}^L)\,.$$

We are now interested in computing the gradients of the cost with respect to each $\boldsymbol{\theta}^l$. Applying the chain rule (generalized to vectors) we obtain the classical backpropagation recursion:

$$\frac{\partial C}{\partial \boldsymbol{\theta}^l} = \frac{\partial f^l_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}^l}\frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}} \tag{11}$$

$$\frac{\partial C}{\partial f^{l-1}_{\boldsymbol{\theta}}} = \frac{\partial f^l_{\boldsymbol{\theta}}}{\partial f^{l-1}_{\boldsymbol{\theta}}}\frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\,. \tag{12}$$

In other words, we first initialize the recursion by computing the gradient of the cost with respect to the last layer output $\partial C/\partial f^L_{\boldsymbol{\theta}}$. Then each layer l computes the gradient respect to its own parameters with (11), given the gradient coming from its output $\partial C/\partial f^l_{\boldsymbol{\theta}}$. To perform the backpropagation, it also computes the gradient with respect to its own inputs, as shown in (12). We now derive the gradients for each layer we used in this paper. For simplifying the notation, we denote $\langle A\rangle_i$ the i^{th} column vector of matrix A.

Lookup Table Layer Given a matrix of parameters $\boldsymbol{\theta}^1 = W^1$ and word (or discrete feature) indices $[w]_1^T$, the layer outputs the matrix:

$$f^l_{\boldsymbol{\theta}}([w]_l^T) = \begin{pmatrix} W^1_{w_1} & W^1_{w_2} & \dots & W^1_{w_T}\end{pmatrix}\,.$$

The gradients of the weights W_i are given by:

$$\frac{\partial C}{\partial W^1_i} = \sum_{\{1\leq t\leq T\,/\,w_t = i\}} \langle \frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\rangle_i$$

This sum equals zero if the index i in the lookup table does not corresponds to a word in the sequence. In this case, the i^{th} column of W does not need to be updated. As a Lookup Table Layer is always the first layer, we do not need to compute its gradients with respect to the inputs.

Linear Layer Our convolutional architecture performs a series of "Linear" operations, as described in (3). Given parameters $\boldsymbol{\theta}^l = (W^l, \boldsymbol{b}^l)$, and an input *vector* $f^{l-1}_{\boldsymbol{\theta}}$ the output is given by:

$$f^l_{\boldsymbol{\theta}} = W^l f^{l-1}_{\boldsymbol{\theta}} + \boldsymbol{b}^l\,. \tag{13}$$

The gradients with respect to the parameters are then obtained with:

$$\frac{\partial C}{\partial W^l} = \left[\frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\right]\left[f^{l-1}_{\boldsymbol{\theta}}\right]^{\mathrm{T}} \quad\text{and}\quad \frac{\partial C}{\partial \boldsymbol{b}^l} = \frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\,, \tag{14}$$

and the gradients with respect to the inputs are computed with:

$$\frac{\partial C}{\partial f^{l-1}_{\boldsymbol{\theta}}} = \left[W^l\right]^{\mathrm{T}}\frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\,. \tag{15}$$

Max Layer Given a *matrix* $f^{l-1}_{\boldsymbol{\theta}}$, the Max Layer computes

$$\left[f^l_{\boldsymbol{\theta}}\right]_i = \max_m \left[\langle f^{l-1}_{\boldsymbol{\theta}}\rangle_m\right]_i \text{ and } a_i = \operatorname*{argmax}_t \left[\langle f^{l-1}_{\boldsymbol{\theta}}\rangle_m\right]_i\ \forall i\,,$$

where a_i stores the index of the largest value. We only need to compute the gradient with respect to the inputs, as this layer has no parameters. The gradient is given by

$$\left[\langle \frac{\partial C}{\partial f^{l-1}_{\boldsymbol{\theta}}}\rangle_m\right]_i = \begin{cases} \left[\langle \frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\rangle_m\right]_i & \text{if } m = a_i \\ 0 & \text{otherwise}\end{cases}\,.$$

HardTanh Layer Given a *vector* $f^{l-1}_{\boldsymbol{\theta}}$, and the definition of the HardTanh (4) we get

$$\left[\frac{\partial C}{\partial f^{l-1}_{\boldsymbol{\theta}}}\right]_i = \begin{cases} 0 & \text{if } \left[f^{l-1}_{\boldsymbol{\theta}}\right]_i < -1 \\ \left[\frac{\partial C}{\partial f^l_{\boldsymbol{\theta}}}\right]_i & \text{if } -1 <= \left[f^{l-1}_{\boldsymbol{\theta}}\right]_i <= 1 \\ 0 & \text{if } \left[f^{l-1}_{\boldsymbol{\theta}}\right]_i > 1 \end{cases}\,,$$

if we ignore non-differentiability points.

Sentence-Level Log-Likelihood The network outputs a matrix where each element $[f_{\boldsymbol{\theta}}]_{t,\,n}$ gives a score for tag t at word n. Given a tag sequence $[t]_1^N$ and a input sequence $[\boldsymbol{x}]_1^N$, we maximize the likelihood (8), which corresponds to minimizing the score

$$C(f_{\boldsymbol{\theta}}, A) = \underbrace{\operatorname*{logadd}_{\forall [u]_1^N} s([\boldsymbol{x}]_1^N,\, [u]_1^N,\, \tilde{\boldsymbol{\theta}})}_{C_{logadd}} - s([\boldsymbol{x}]_1^N,\, [t]_1^N,\, \tilde{\boldsymbol{\theta}})\,,$$

with

$$s([\boldsymbol{x}]_1^N,\, [t]_1^N,\, \tilde{\boldsymbol{\theta}}) = \sum_{n=1}^{N}\left([A]_{[t]_{n-1},\,[t]_n} + [f_{\boldsymbol{\theta}}]_{[t]_n,\,n}\right)\,.$$

We first initialize all gradients to zero

$$\frac{\partial C}{\partial \left[f_{\boldsymbol{\theta}}\right]_{t,\,n}} = 0 \ \forall t, n \ \text{ and } \ \frac{\partial C}{\partial \left[A\right]_{t,\,u}} = 0 \quad \forall t, u \,.$$

We then *accumulate* gradients over the second part of the cost $-s([\boldsymbol{x}]_1^N,\, [t]_1^N,\, \tilde{\boldsymbol{\theta}})$, which gives:

$$\left.\begin{array}{r} \dfrac{\partial C}{\partial \left[f_{\boldsymbol{\theta}}\right]_{[t]_n,\, n}} \mathrel{+}= 1 \\ \dfrac{\partial C}{\partial \left[A\right]_{[t]_{n-1},\, [t]_n}} \mathrel{+}= 1 \end{array}\right. \qquad \forall n \,.$$

We now need to accumulate the gradients over the first part of the cost, that is C_{logadd}. We differentiate C_{logadd} by applying the chain rule through the recursion (9). First we initialize our recursion with

$$\frac{\partial C_{logadd}}{\partial \delta_N(t)} = \frac{e^{\delta_N(t)}}{\sum_u e^{\delta_N(u)}} \quad \forall t \,.$$

We then compute iteratively:

$$\frac{\partial C_{logadd}}{\partial \delta_{n-1}(t)} = \sum_u \frac{\partial C_{logadd}}{\partial \delta_n(u)} \frac{e^{\delta_{n-1}(t) + [A]_{t,\,u}}}{\sum_v e^{\delta_{n-1}(v) + [A]_{v,\,u}}} \,,$$

where at each step n of the recursion we accumulate of the gradients with respect to the inputs $f_{\boldsymbol{\theta}}$, and the transition scores $[A]_{t,\,u}$:

$$\frac{\partial C}{\partial \left[f_{\boldsymbol{\theta}}\right]_{t,\,n}} \mathrel{+}= \frac{\partial C_{logadd}}{\partial \delta_n(t)} \, \frac{\partial \delta_n(t)}{\partial \left[f_{\boldsymbol{\theta}}\right]_{t,\,n}} = \frac{\partial C_{logadd}}{\partial \delta_n(t)} \,,$$

and

$$\begin{aligned} \frac{\partial C}{\partial \left[A\right]_{t,\,u}} &\mathrel{+}= \frac{\partial C_{logadd}}{\partial \delta_n(u)} \, \frac{\partial \delta_n(u)}{\partial \left[A\right]_{t,\,u}} \\ &= \frac{\partial C_{logadd}}{\partial \delta_n(u)} \, \frac{e^{\delta_{n-1}(t) + [A]_{t,\,u}}}{\sum_v e^{\delta_{n-1}(v) + [A]_{v,\,u}}} \,. \end{aligned}$$

Initialization and Learning Rate We employed only two classical "tricks" for training our neural networks: the initialization and update of the parameters of each network layer were done according to the "fan-in" of the layer, that is the number of inputs used to compute each output of this layer (Plaut and Hinton, 1987). The fan-in for the lookup table (1) and the l^{th} linear layer (3) (also rewritten in (13)) are respectively 1 and $|f_{\boldsymbol{\theta}}^{l-1}|$ (where $|z|$ is the number of dimensions of vector z). The initial parameters of the network were drawn from a centered uniform distribution, with a variance equal to the inverse of the square-root of the fan-in. The learning rate in (10) was divided by the fan-in, but stayed fixed during the training.