General Gradient of Neural Network

tomas.ukkonen@iki.fi, 2017

Backpropagation is a commonly known algorithm for computing the gradient of error function which arises when we know target values and the loss, cost or error function is one dimensional. Generalizing this to general gradient calculation when we seek to find the maximum or minimum value of a neural network (thought often ill-fated because of local optimas produced by an over-fitted neural network) is then important. This is needed, for example, when implementing certain reinforcement learning methods.

Consider a two-layer neural network

$$y(\boldsymbol{x}) = f\big(\boldsymbol{W}^{(2)}\, g\big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\big) + \boldsymbol{b}^{(2)}\big)$$

The gradients of the final layer are (non-zero terms are at the j:th row):

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, w_{ji}^{(2)}} = \mathrm{diag}\Big(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\Big)\begin{pmatrix} 0 \\ g_i \\ 0 \end{pmatrix}$$

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, b_{j}^{(2)}} = \mathrm{diag}\Big(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\Big)\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

The derivation chain-rule can be used to calculate the second (and more deep layers' gradients):

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, w_{ji}^{(1)}} = \mathrm{diag}\Bigg(\frac{\partial f(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(2)}\boldsymbol{g} + \boldsymbol{b}^{(2)})}\Bigg)\frac{\partial(\boldsymbol{W}^{(2)}\,\boldsymbol{g} + \boldsymbol{b}^{(2)})}{\partial \boldsymbol{g}}\,\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}\,\frac{\partial(\boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)})}{\partial\, w_{ji}^{2}}$$

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, w_{ji}^{(1)}} = \mathrm{diag}\Bigg(\frac{\partial f(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(2)}\boldsymbol{g} + \boldsymbol{b}^{(2)})}\Bigg)\boldsymbol{W}^{(2)}\,\mathrm{diag}\Bigg(\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}\Bigg)\begin{pmatrix} 0 \\ x_i \\ 0 \end{pmatrix}$$

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, b_{j}^{(1)}} = \mathrm{diag}\Bigg(\frac{\partial f(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(2)}\boldsymbol{g} + \boldsymbol{b}^{(2)})}\Bigg)\boldsymbol{W}^{(2)}\,\mathrm{diag}\Bigg(\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}\Bigg)\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

By analysing the chain rule we can derive generic backpropagation formula for the full gradient. Let $\boldsymbol{v}^{(k)}$ be a k:th layers local field, $\boldsymbol{v}^{(k)} = \boldsymbol{W}^{(k)} f\big(\boldsymbol{v}^{(k-1)}\big) + \boldsymbol{b}^{(k)}$. Then local gradient matrices $\boldsymbol{\delta}^{(k)}$ are

$$\boldsymbol{\delta}^{(L)} = \mathrm{diag}\Bigg(\frac{\partial \boldsymbol{f}(\boldsymbol{v}^{(L)})}{\partial \boldsymbol{v}^{(L)}}\Bigg)$$

$$\boldsymbol{\delta}^{(k-1)} = \boldsymbol{\delta}^{(k)}\,\boldsymbol{W}^{(k)}\,\mathrm{diag}\Bigg(\frac{\partial \boldsymbol{f}(\boldsymbol{v}^{(k-1)})}{\partial \boldsymbol{v}^{(k-1)}}\Bigg)$$

And network's parameter gradient matrices for each layer are (only j:th element of each row is non-zero):

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, w_{ji}^{(k)}} = \boldsymbol{\delta}^{(k)}\begin{pmatrix} 0 \\ f\big(v_i^{(k-1)}\big) \\ 0 \end{pmatrix}$$

$$\frac{\partial\, y(\boldsymbol{x})}{\partial\, b_{j}^{(k)}} = \boldsymbol{\delta}^{(k)}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

To test that gradient matrix is correctly computed it can be compared with normal squared error calculations (normal backpropagation).

$$\varepsilon(\boldsymbol{x}|\boldsymbol{w}) = \frac{1}{2}\|y_i - y(\boldsymbol{x}|\boldsymbol{w})\|^2$$

$$\frac{\partial \varepsilon(\boldsymbol{x}|\boldsymbol{w})}{\partial \boldsymbol{w}} = (y(\boldsymbol{x}|\boldsymbol{w}) - y_i)^T\,\frac{\partial y(\boldsymbol{x}|\boldsymbol{w})}{\partial \boldsymbol{w}}$$

Sometimes also needs gradient with respect to $\boldsymbol{x}$ and not weights parameters $\boldsymbol{w}$. This can be calculated using the chain rule again. For simplicity, let's consider two-layer case initially.

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{f}\big(\boldsymbol{W}^{(2)}\,\boldsymbol{h}\big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\big) + \boldsymbol{b}^{(2)}\big)$$

The gradient is:

$$\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}(\boldsymbol{v}^{(2)})}{\partial(\boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)})}\,\frac{\partial(\boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)})}{\partial \boldsymbol{h}}\,\frac{\partial\,\boldsymbol{h}(\boldsymbol{v}^{(1)})}{\partial(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}\,\frac{\partial(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}{\partial\,\boldsymbol{x}}$$

$$\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}(\boldsymbol{v}^{(2)})}{\partial \boldsymbol{v}^{(2)}}\,\boldsymbol{W}^{(2)}\,\frac{\partial\,\boldsymbol{h}(\boldsymbol{v}^{(1)})}{\partial \boldsymbol{v}^{(1)}}\,\boldsymbol{W}^{(1)}$$

This results into following formula (diag() entries are square matrices which diagonal is nonzero):

$$\nabla_{\boldsymbol{x}}\,\boldsymbol{g}(\boldsymbol{x}|\boldsymbol{w}) = \mathrm{diag}\big(\nabla_{\boldsymbol{v}^{(L)}}\boldsymbol{f}\big(\boldsymbol{v}^{(L)}\big)\big)\boldsymbol{W}^{(L)} \ldots \mathrm{diag}\big(\nabla_{\boldsymbol{v}^{(2)}}\boldsymbol{f}\big(\boldsymbol{v}^{(2)}\big)\big)\boldsymbol{W}^{(2)}\,\mathrm{diag}\big(\nabla_{\boldsymbol{v}^{(1)}}\boldsymbol{f}\big(\boldsymbol{v}^{(1)}\big)\big)\boldsymbol{W}^{(1)}$$

Recurrent Neural Networks and Backpropagation Through Time (BPTT)

The basic learning algorithm for recurrent neural networks (RNN) is **backpropagation through time** ($BPTT$). This is done by unfolding neural net in time an computing the gradients. The recurrent neural network is

$$\boldsymbol{z}(n+1) = \begin{pmatrix} \boldsymbol{y}(n+1) \\ \boldsymbol{r}(n+1) \end{pmatrix} = \boldsymbol{f}(\boldsymbol{x}(n), \boldsymbol{r}(n))$$

The error function to minimize is:

$$E(N) = \tfrac{1}{2}\textstyle\sum_{n=0}^{N} \|\boldsymbol{d}(n+1) - \boldsymbol{\Gamma}_{\boldsymbol{y}}\boldsymbol{f}(\boldsymbol{x}(n+1), \boldsymbol{\Gamma}_{\boldsymbol{r}}\boldsymbol{z}(n)|\boldsymbol{w})\|^2$$

In which $\boldsymbol{\Gamma}$ matrices are used to select $\boldsymbol{y}(n)$ and $\boldsymbol{r}(n)$ vectors from generic output vector and the initial input to feedforward neural network is zero $\boldsymbol{z}(0) = \boldsymbol{0}$.

It is possible to calculate gradient of $\boldsymbol{f}$ using the chain rule

$$\frac{\partial E(N)}{\partial \boldsymbol{w}} = \textstyle\sum_{n=0}^{N} (\boldsymbol{\Gamma}_{\boldsymbol{y}}\boldsymbol{f}(\boldsymbol{x}(n+1), \boldsymbol{\Gamma}_{\boldsymbol{r}}\boldsymbol{z}(n)|\boldsymbol{w}) - \boldsymbol{d}(n+1))^T\,\boldsymbol{\Gamma}_{\boldsymbol{y}}\,\nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x}(n+1), \boldsymbol{\Gamma}_{\boldsymbol{r}}\boldsymbol{z}(n)|\boldsymbol{w})$$

To calculate the gradient $\nabla_{\boldsymbol{w}}\boldsymbol{f}$ one must remember that $\boldsymbol{z}(n)$ now also depends on $\boldsymbol{w}$ resulting into eq:

$$\nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x}(n+1), \boldsymbol{\Gamma}_{\boldsymbol{r}}\boldsymbol{z}(n)) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{r}}\,\boldsymbol{\Gamma}_{\boldsymbol{r}}\frac{\partial \boldsymbol{z}(n)}{\partial \boldsymbol{w}}$$

To further compute gradients we get a generic update rule

$$\frac{\partial\,\boldsymbol{z}(n)}{\partial \boldsymbol{w}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{r}}\,\boldsymbol{\Gamma}_{\boldsymbol{r}}\,\frac{\partial \boldsymbol{z}(n-1)}{\partial \boldsymbol{w}}$$

The computation of gradients can be therefore bootstrapped by setting $\frac{\partial \boldsymbol{z}(1)}{\partial \boldsymbol{w}} = \frac{\partial \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{0})}{\partial \boldsymbol{w}}$ and iteratively updating $\boldsymbol{z}$ gradient while computing current error for the timestep.