Implementing Neural Networks

High Performance Computing for Science and Engineering

Guido Novati

November 17, 2017
Introduction

• Ex 7-2 is a toy example of a “deep-learning” library
• Code evaluates and optimizes the model given a cost function
• For the exercise we limit ourselves to linear models

• Have a look at deep learning libraries such as caffe: https://github.com/BVLC/caffe

• Core of the library is split in two categories:
  • Optimizers
  • Layers
    • For each layer type, two operations are defined:
      • Forward
      • Backward
Implementing Neural Networks

• NN is composition of operations:
  \[ x \rightarrow h^{(1)} \rightarrow h^{(2)} \rightarrow \ldots \rightarrow h^{(K)} \]

• Arrows denote operations that are differentiable:

  • Linear algebra
    \[ h^{(k+1)} = W^{(k)} h^{(k)} \]

  • Non linear operations:
    \[ h^{(k+2)} = f(h^{(k+1)}) \]

• Cost function, eg mean squared error:
  \[ \mathcal{L} = \frac{1}{2} \| y^* - h^{(K)} \|^2 \]
Forward-propagation

- Computing the output of a network: “prediction” or “forward-propagation”
  \[ h^{(k+1)} = W^{(k)} h^{(k)} \]
  \[ h^{(k)} = \{ h_1^{(k)}, h_2^{(k)}, \ldots, h_{n_k}^{(k)} \} \]

- Layer \( k \) has \( n_{k+1} \) outputs and received \( n_k \) inputs

- Parameters of the layer (to be optimized) constitute a matrix of \( n_{k+1} \) by \( n_k \) elements.
  \[ W^{(k)} = \begin{bmatrix} w_1^{(k)}, 1 & w_1^{(k)}, 2 & \cdots & w_1^{(k)}, n_k \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{k+1}}^{(k)}, 1 & w_{n_{k+1}}^{(k)}, 2 & \cdots & w_{n_{k+1}}^{(k)}, n_k \end{bmatrix} \]

- I will use superscript only to denote index of the layer

- Subscript will denote indices iterating over neurons. For weights, first index is output second is input neuron. If index is omitted, variable refers to the entire set.
Back-propagation (I)

• Back-propagation denotes computing the gradients of:
  • Error with respect of input \( \delta^{(k)} = \left. \frac{d\mathcal{L}}{dh^{(k)}} \right|_{x,W} \) (this is a vector)
  • Error with respect of weights \( \left. \frac{d\mathcal{L}}{dW^{(k)}} \right|_{x,W} \) (this is a matrix)
  • Given a certain input and the current parameters.

• Back-propagation does not change the weights:
  • Weights are updated by averaging multiple gradients \( W \leftarrow W - \frac{\beta}{B} \sum_{i=1}^{B} \left. \frac{d\mathcal{L}}{dW} \right|_{x,W} \)
  • (or quasi-Newton methods)
  • (or user might be just checking gradients)
Back-propagation (II)

\[ x \rightarrow h^{(1)} \rightarrow h^{(2)} \rightarrow \ldots \rightarrow h^{(K)} \]

• With back-propagation we aim to compute all derivatives, with similar computational complexity as for forward-propagation (matrix vector operations).

• The gradient of the loss with respect to the last output:

\[
\delta^{(K)} = \frac{d\|y^* - h^{(K)}\|^2}{dh^{(K)}} = (y^* - h^{(K)})
\]
Back-propagation (III)

- Each layer computes the delta of the input to layer $\delta^{(k)}$
- We assume that we already know delta for the output (in fact, we know the last $\delta^{(K)}$)
- First step is to write the total differential (calculus 1) of $\delta$ in terms of layer outputs

\[
\frac{dL}{dh_i^{(k)}} = \sum_{j=1}^{n_{k+1}} \frac{dL}{dh_j^{(k+1)}} \frac{dh_j^{(k+1)}}{dh_i^{(k)}} = \sum_{j=1}^{n_{k+1}} \delta_j^{(k+1)} \frac{dh_j^{(k+1)}}{dh_i^{(k)}} = \sum_{j=1}^{n_{k+1}} \delta_j^{(k+1)} w_{j,i}^{(k)}
\]

- Derivative of neuron $j$ of upper layer w.r.t. neuron $i$ of lower is the connecting weight!
- Last step is to notice that what we wrote is an other linear operation:

\[
\delta^{(k)} = \left(W^{(k)}\right)^T \delta^{(k+1)}
\]
- This gradient is then available to lower layers, so that continuing back-prop amounts to more matrix-vector multiplications (same as forward-prop).
Back-propagation (IV)

• Each layer also computes the gradient of the weights

• This is easy, if we write total differential of Loss in terms of layer output only one term depends on the weight:

\[
\frac{dL}{dw_{j,i}^{(k)}} = \sum_{l=1}^{n_{k+1}} \frac{dL}{dh_{l}^{(k+1)}} \frac{dh_{l}^{(k+1)}}{dw_{j,i}^{(k)}} = \delta_{j}^{(k+1)} \frac{dh_{j}^{(k+1)}}{dw_{j,i}^{(k)}} = \delta_{j}^{(k+1)} h_{i}^{(k)}
\]

• Again, linear layer: derivative of output neuron wrt to weight is the input neuron

• This operation can be written in vector notation as a pointwise product:

\[
\frac{dL}{dW} = \delta^{(k+1)} \odot h^{(k)} = \begin{bmatrix}
\delta_{1}^{(k+1)} h_{1}^{(k)} & \cdots & \delta_{1}^{(k+1)} h_{n_{k}}^{(k)} \\
\vdots & \ddots & \vdots \\
\delta_{n_{k+1}}^{(k+1)} h_{1}^{(k)} & \cdots & \delta_{n_{k+1}}^{(k+1)} h_{n_{k}}^{(k)}
\end{bmatrix}
\]