Backpropagation: A modular approach (Torch NN)

Nando de Freitas
Outline of the lecture

This lecture describes modular ways of formulating and learning distributed representations of data. The objective is for you to learn:

- How to specify models such as logistic regression in layers.
- How to formulate layers and loss criterions.
- How well formulated local rules results in correct global rules.
- How back-propagation works.
- How this manifests itself in Torch.
\[ C(\theta) = -\sum_{i=1}^{n} \Pi_0(Y_i) \log \left( \frac{e^{x_i\theta_1}}{e^{x_i\theta_1} + e^{x_i\theta_2}} \right) + \Pi_1(Y_i) \log \left( \frac{e^{x_i\theta_2}}{e^{x_i\theta_1} + e^{x_i\theta_2}} \right) \]

\[ z_1^2 = x_i\theta_1 \quad z_2^2 = x_i\theta_2 \]

\[ z_1^3 = \log \left( \frac{e^{z_1^2}}{e^{z_1^2} + e^{z_2^2}} \right) \quad z_2^3 = \log \left( \frac{e^{z_2^2}}{e^{z_1^2} + e^{z_2^2}} \right) \]

\[ z_1^4 = \sum_{i} \Pi_0(Y_i) z_1^3 + \Pi_1(Y_i) z_2^3 \]
Derivative using the chain rule

\[ C(\theta) = -\sum_{i=1}^{n} \Pi_0(Y_i) \log \left( \frac{e^{x_i \theta_1}}{e^{x_i \theta_1} + e^{x_i \theta_2}} \right) + \Pi_1(Y_i) \log \left( \frac{e^{x_i \theta_2}}{e^{x_i \theta_1} + e^{x_i \theta_2}} \right) \]

\[ C(\theta) = \mathbb{E} \left\{ \begin{array}{c} Z_1^3 \left( Z_1^2 \left( \theta_1 Z_1 \right) \right) \left( Z_2^2 \left( \theta_2 Z_2 \right) \right) \\
Z_2^3 \left( Z_1^2 \left( \theta_1 Z_1 \right) \right) \left( Z_2^2 \left( \theta_2 Z_2 \right) \right) \end{array} \right\} \]

\[ \frac{\partial C(\theta)}{\partial \theta_1} = \frac{\partial}{\partial \theta_1} \left( \begin{array}{c} Z_1^4 \frac{e^{Z_1^3}}{e^{Z_1^3} + e^{Z_2^3}} \frac{Z_2^2}{e^{Z_2^3}} \frac{e^{Z_2^2}}{e^{Z_2^3}} \frac{e^{Z_2^2}}{e^{Z_2^3} + e^{Z_2^3}} \\
Z_2^4 \frac{e^{Z_2^3}}{e^{Z_2^3} + e^{Z_2^3}} \frac{Z_1^2}{e^{Z_1^3}} \frac{e^{Z_1^2}}{e^{Z_1^3}} \frac{e^{Z_1^2}}{e^{Z_1^3} + e^{Z_1^3}} \end{array} \right) \]
Layer specification

\[ z^3 = f(x^2) \]
\[ z^2 = f(z^1) \]
\[ z^1 = x \]

\[ C = z^4 = f_3(z^3) \]
\[ S^4 = 1 \]

Layer 3

Layer 2

Layer 1

\[ \frac{\partial C}{\partial \theta^L} = \sum_j \frac{\partial C}{\partial z_j^{L+1}} \frac{\partial z_j^{L+1}}{\partial \theta^L} = \sum_j \delta_j^{L+1} \left( \frac{\partial z_j^{L+1}}{\partial \theta^L} \right) \]

Forward pass

\[ z = f(z) \]

Backward pass

\[ \delta_i^L = \frac{\partial C}{\partial z_i^L} = \sum_j \frac{\partial C}{\partial z_j^{L+1}} \frac{\partial z_j^{L+1}}{\partial z_i^L} = \sum_j \delta_j^{L+1} \left( \frac{\partial z_j^{L+1}}{\partial z_i^L} \right) \]
Derivative via layer-specification

\[ \frac{\partial c}{\partial \theta_1} = \sum_j \frac{\partial c}{\partial z^l_j} \frac{\partial z^l_j}{\partial \theta_1} \]

\[ = \sum_j \left( \sum_k \frac{\partial^2 c}{\partial z^3_k} \frac{\partial z^3_k}{\partial \theta_2} \right) \frac{\partial z^2_j}{\partial \theta_1} \]

\[ = \sum_j \sum_{k=1}^2 \frac{\partial z^2_j}{\partial \theta_1} \left( \frac{\partial^2 z^k}{\partial z^3_k} \frac{\partial z^3_k}{\partial \theta_2} \right) \]

\[ = \text{as before}. \]
Back-propagation algorithm

\[ z^1 = x_i \rightarrow z^2(x_i) \rightarrow z^3(x_i) \rightarrow z^4(x_i) = c \]

\[ \delta^1 \leftarrow \delta^2 \leftarrow \delta^3 \leftarrow \delta^4 \leftarrow 1 \]
Derivatives wrt to the input

Karen Simonyan
Logit Regression Model in Torch

1  model = nn.Sequential()
2  model.add( nn.Linear(2,1) )
3  model.add( nn.LogSoftMax() )
Loss criterion in Torch

```python
1 criterion = nn.ClassNLLCriterion()
```
Derivatives closure in Torch

```python
-- params/gradients
x, dl_dx = model:getParameters()

local loss_x = criterion:forward(model:forward(inputs), target)
model:backward(inputs, criterion:backward(model.output, target))
```
Optimization in Torch

```
_, fs = optim.sgd(feval, x, sgd_params)
```

-- Functions in optim all return two things:
--   + the new x, found by the optimization method (here SGD)
--   + the value of the loss functions at all points that were used by
--     the algorithm. SGD only estimates the function once, so
--   that list just contains one value.
Next lecture

In the next lecture, we consider a generalization of logistic regression, with many logistic units, called multi-layer perceptron (MLP) or feed-forward neural network.