Backpropagation: A modular approach (Torch NN)

Nando de Freitas

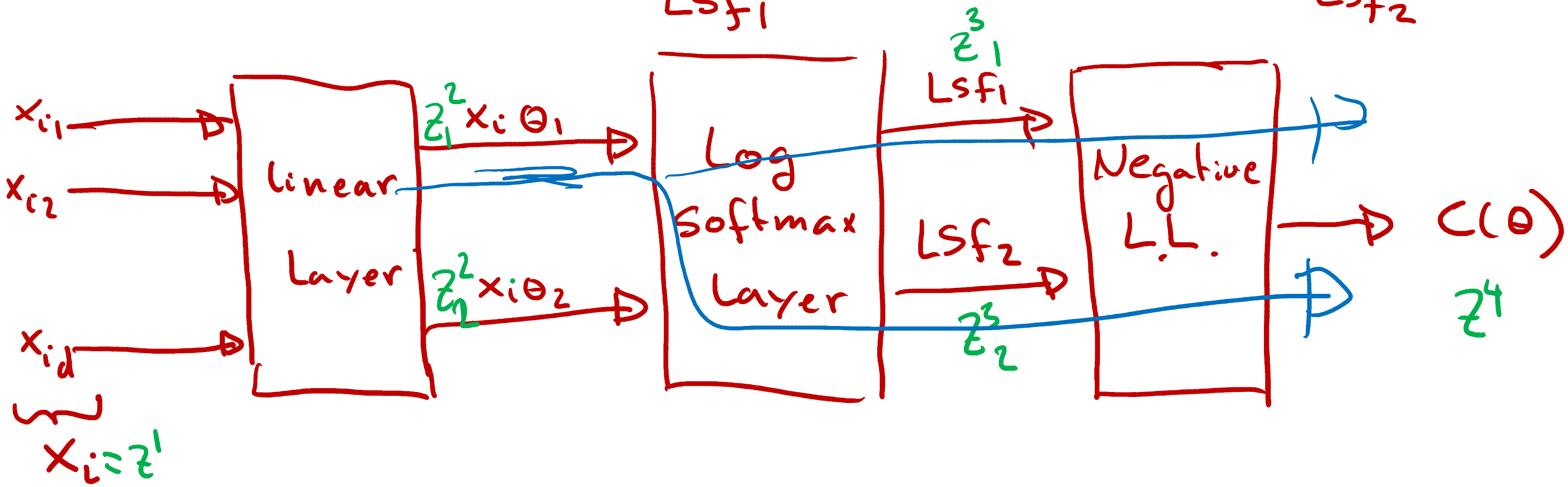
UNIVERSITY OF OXFORD

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.

$$C(\theta) = -\sum_{i=1}^{n} \mathbb{I}_0(y_i)\underbrace{\log\left(\frac{e^{x_i\theta_1}}{e^{x_i\theta_1}+e^{x_i\theta_2}}\right)}_{Lsf_1} + \mathbb{I}_1(y_i)\underbrace{\log\left(\frac{e^{x_i\theta_2}}{e^{x_i\theta_1}+e^{x_i\theta_2}}\right)}_{Lsf_2}$$



$$z_1^2 = x_i\theta_1 \qquad z_2^2 = x_i\theta_2$$

$$z_1^3 = \log\left(\frac{e^{z_1^2}}{e^{z_1^2}+e^{z_2^2}}\right) \qquad z_2^3 = \log\left(\frac{e^{z_2^2}}{e^{z_1^2}+e^{z_2^2}}\right)$$

$$z^4 = \sum_i \mathbb{I}_0(y_i)\, z_1^3 + \mathbb{I}_1(y_i)\, z_2^3$$

Derivative using the chain rule

$$C(\theta) = -\sum_{i=1}^{n} \mathbb{I}_0(y_i)\log\left(\frac{e^{x_i\theta_1}}{e^{x_i\theta_1}+e^{x_i\theta_2}}\right) + \mathbb{I}_1(y_i)\log\left(\frac{e^{x_i\theta_2}}{e^{x_i\theta_1}+e^{x_i\theta_2}}\right)$$

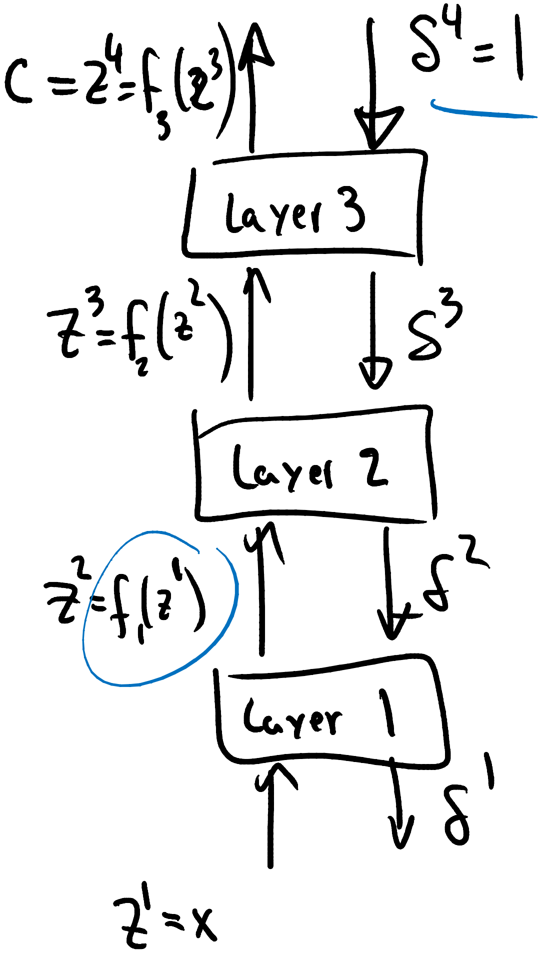
z_{12}

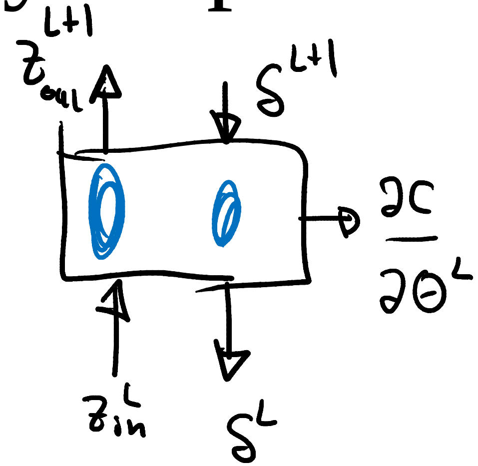
$$C(\theta) = z^4\left\{ z_1^3\left[\boxed{z_1^2(\theta_1, z^1)}, \boxed{z_2^2(\theta_{21}, z^1)}\right], z_2^3\left[\boxed{z_1^2(\theta_1, z^1)}, \boxed{z_2^2(\theta_2, z^1)}\right]\right\}$$

$$\frac{\partial C(\theta)}{\partial \theta_1} = \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}$$

$$+ \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}$$

Layer specification





$$z^{L+1} = f\left(z^L\right)$$

forward pass

backward pass

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \sum_j \frac{\partial C}{\partial z_j^{L+1}} \frac{\partial z_j^{L+1}}{\partial z_i^L} = \sum_j \delta_j^{L+1} \boxed{\frac{\partial z_j^{L+1}}{\partial z_i^L}}$$

$$\frac{\partial C}{\partial \Theta^L} = \sum_j \frac{\partial C}{\partial z_j^{L+1}} \frac{\partial z_j^{L+1}}{\partial \Theta^L} = \sum_j \delta_j^{L+1} \left(\frac{\partial z_j^{L+1}}{\partial \Theta^L}\right)$$

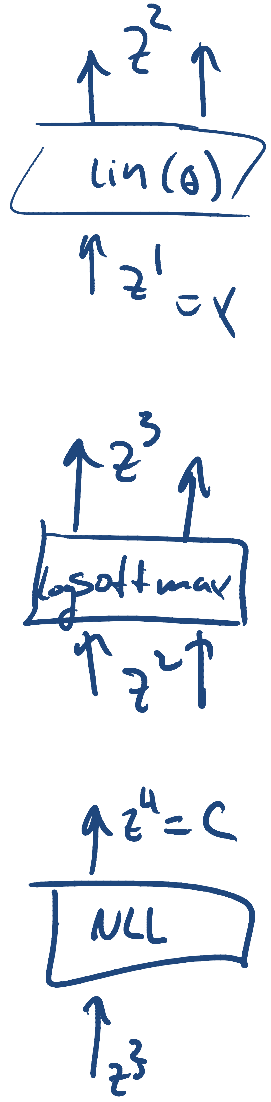
Derivative via layer-specification

$$\frac{\partial c}{\partial \theta_1} = \sum_j \frac{\partial c}{\partial z_j^2} \frac{\partial z_j^2}{\partial \theta_1}$$

$$= \sum_j \left(\sum_k \frac{\partial c}{\partial z_k^3} \frac{\partial z_k^3}{\partial z_j^2} \right) \frac{\partial z_j^2}{\partial \theta_1}$$

$$= \sum_{j=1}^{2} \sum_{k=1}^{2} \overset{1}{\cancel{\frac{\partial c}{\partial z^4}}} \left(\frac{\partial z^4}{\partial z_k^3} \frac{\partial z_k^3}{\partial z_j^2} \frac{\partial z_j^2}{\partial \theta_1} \right)$$

= as before.



Back-propagation algorithm

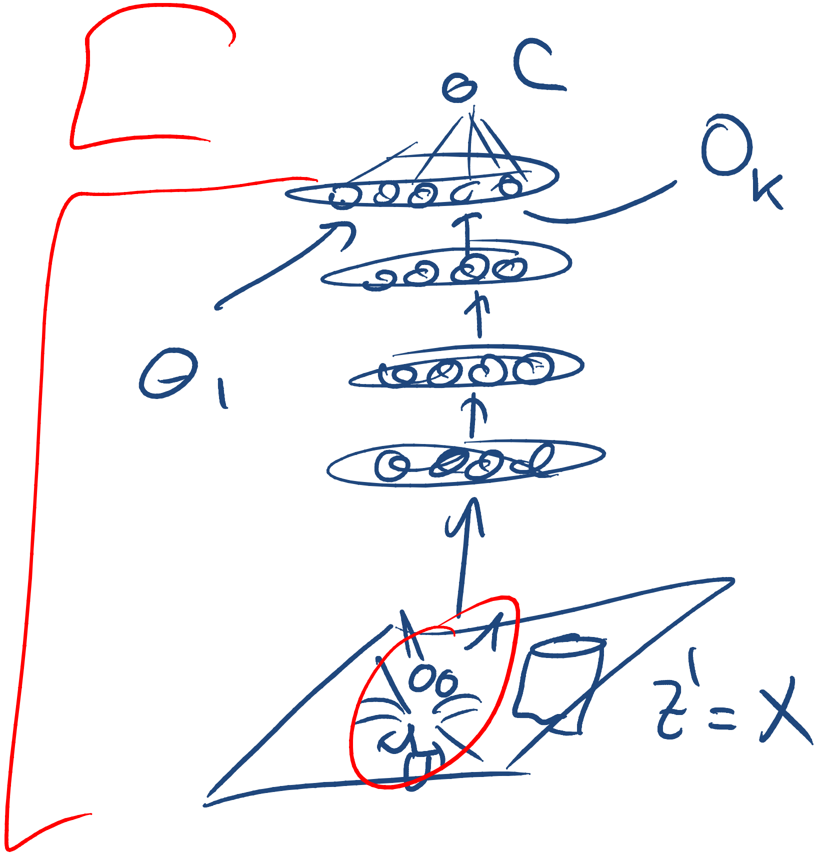
$$\log\left(\frac{e^{x_i\theta_1}}{\Sigma}\right)$$

$$z^1 = x_i \longrightarrow \underbrace{z^2(x_i)}_{}^{x_i\theta_1} \longrightarrow z^3(x_i) \longrightarrow z^4(x_i) = C$$

$$\delta^1 \longleftarrow \delta^2 \longleftarrow \delta^3 \longleftarrow \delta^4 \leftarrow 1$$

Derivatives wrt to the input

imagenet



$$\frac{\partial C}{\partial X}$$

output

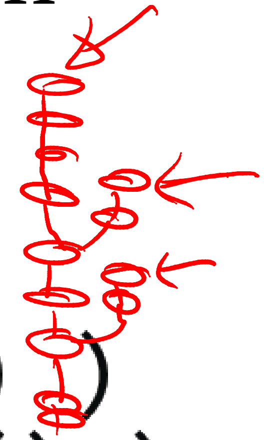
$$\left| \frac{\partial O}{\partial x_i} \right|$$

Misha Denil

Karen Simonyan

Logit Regression Model in Torch

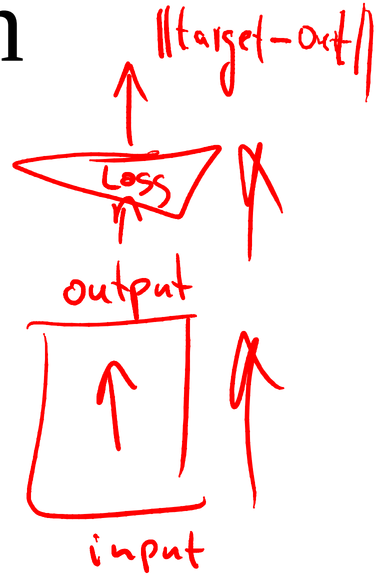
```
model = nn.Sequential()
model:add( nn.Linear(2,2) )
model:add( nn.LogSoftMax() )
```



Loss criterion in Torch

```
criterion = nn.ClassNLLCriterion()
```

Derivatives closure in Torch




```
-- 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.