Backpropagation

CS 510

Lecture #20

March 23th, 2018

Colorado State University

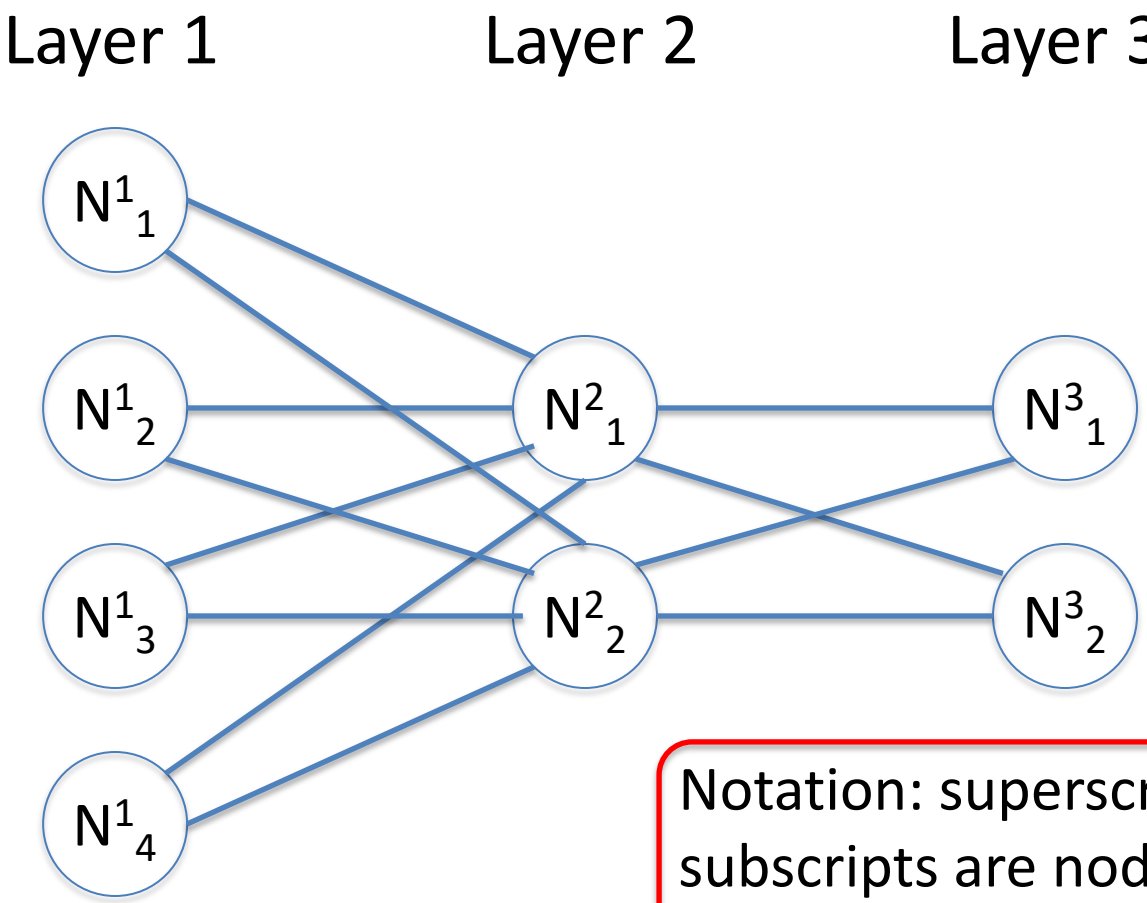
Backpropagation

- Backpropagation is the algorithm that describes how we update weights in a network, given
  - Training samples
  - Training labels
  - A cost function
- Its used for (almost) all networks
- Network nodes may be
  - Non-linear perceptrons (the most common)
  - Convolutional units
  - Pooling units
  - Batch normalization units
  - …

Goals

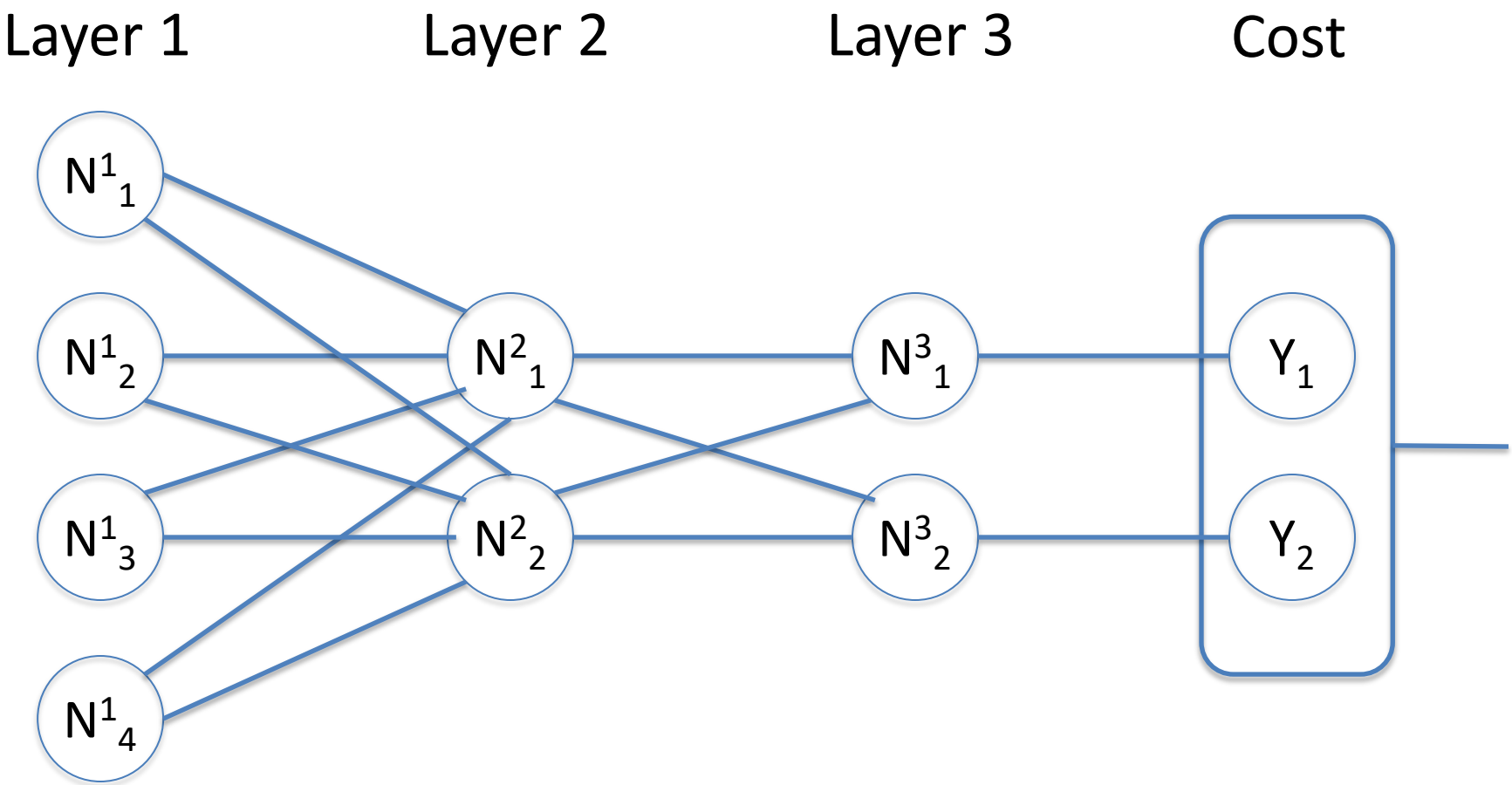
- Walk you through the math of backpropagation
  - Complicated, but just calculus
  - Almost universal : modifiable for different node types (see previous slide)
- Today's derivation assumes multi-layer perceptrons
  - $z(x) = wx + b$
  - $a(x) = \sigma\big(z(x)\big) = \sigma(wx + b)$
- Remember the chain rule from calculus:
  - $f(x) = g\big(h(x)\big) \rightarrow f'(x) = g'\big(h(x)\big)h'(x)$

Simple Neural Network



Notation: superscripts are layers, subscripts are node numbers

Adding a Cost Function



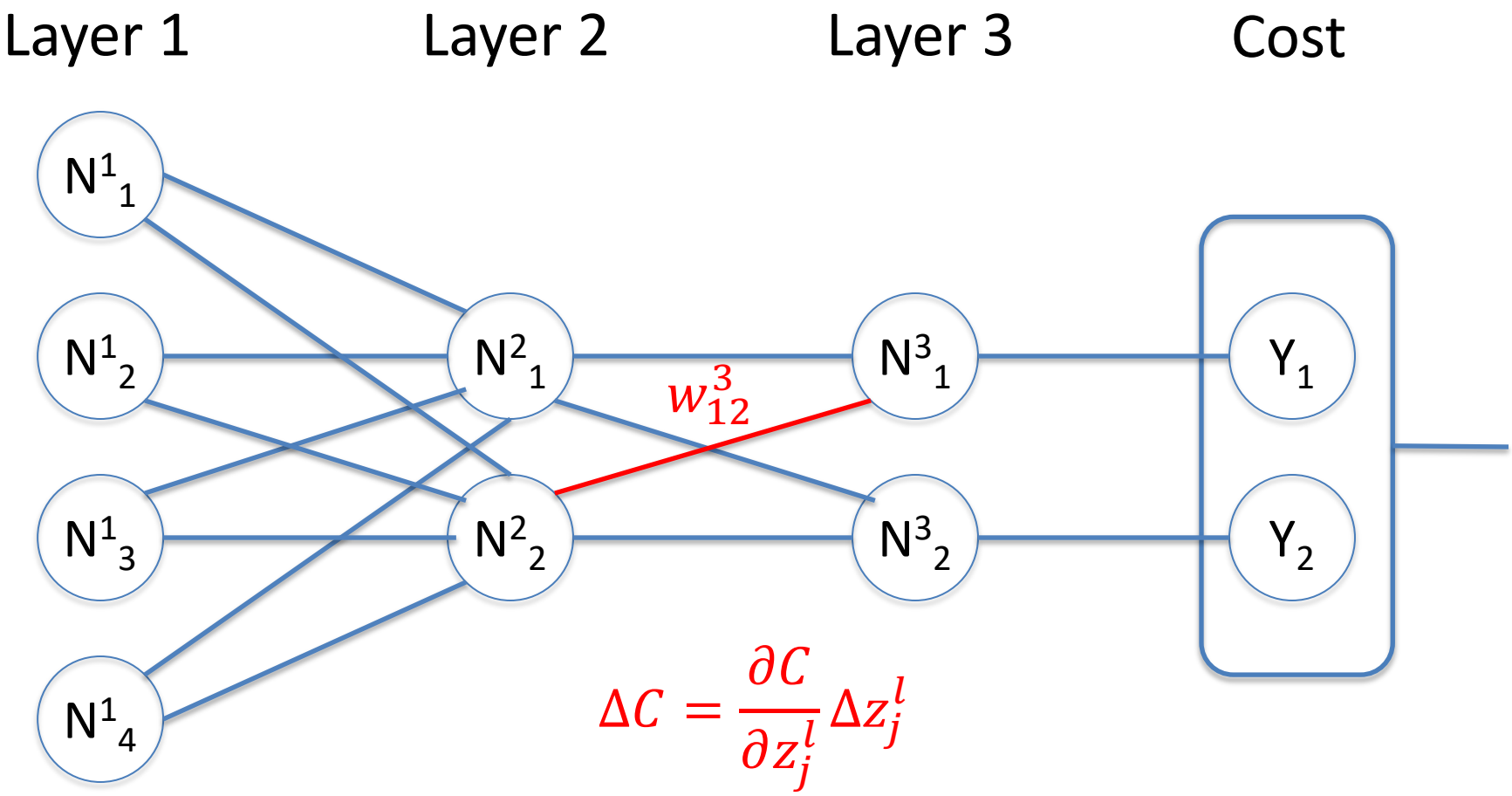
Cost Functions

- Cost functions measure the gap between the network output and the ideal output
- Two necessary properties
  1. An average over samples: $C = \frac{1}{n}\sum_x C_x$
  2. Function of output activations: $C = C(a_l)$
- Example: mean squared error
  - $C = \frac{1}{2n}\sum_x \|y(x) - a^L(x)\|^2$

Allows us to optimize per sample

Allows us to initialize the partial derivative computations

δs : local derivatives as error measures



$$\Delta C = \frac{\partial C}{\partial z_j^l} \Delta z_j^l$$

Partial derivatives as error measures

- Imagine you want to change the output z^l_j, by Δz^l_j
- Then $\Delta C = \frac{\partial C}{\partial z_j^l} \Delta z_j^l$
- If $\left|\frac{\partial C}{\partial z_j^l}\right|$ is large, then C becomes smaller by giving Δz_j^l the opposite sign
- But if $\left|\frac{\partial C}{\partial z_j^l}\right|$ is near zero, then Δz_j^l doesn't matter.
  - $\frac{\partial C}{\partial z_j^l}$ is already optimal!
  - $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$

So where are we?

- We can optimize on a per-sample basis
  - Because the cost function is an average
- Minimizing the δs optimizes the net
  - The δs depend on the data samples
- But how do we minimize the δs?

- We will assume that nodes have non-linear functions, so $a_j^l = \sigma\left(z_j^l\right)$

Output Layer

- $\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$ by the chain rule
- $\frac{\partial C}{\partial a_j^L}$ is the partial derivative of C with respect the activation of output unit j
  - If C is LMS (slide #6)
    - $\frac{\partial C}{\partial a_j^L} = a_j^L(x) - y(x)$
    - The difference between the output & desired output
      - Explains the ½ in LMS

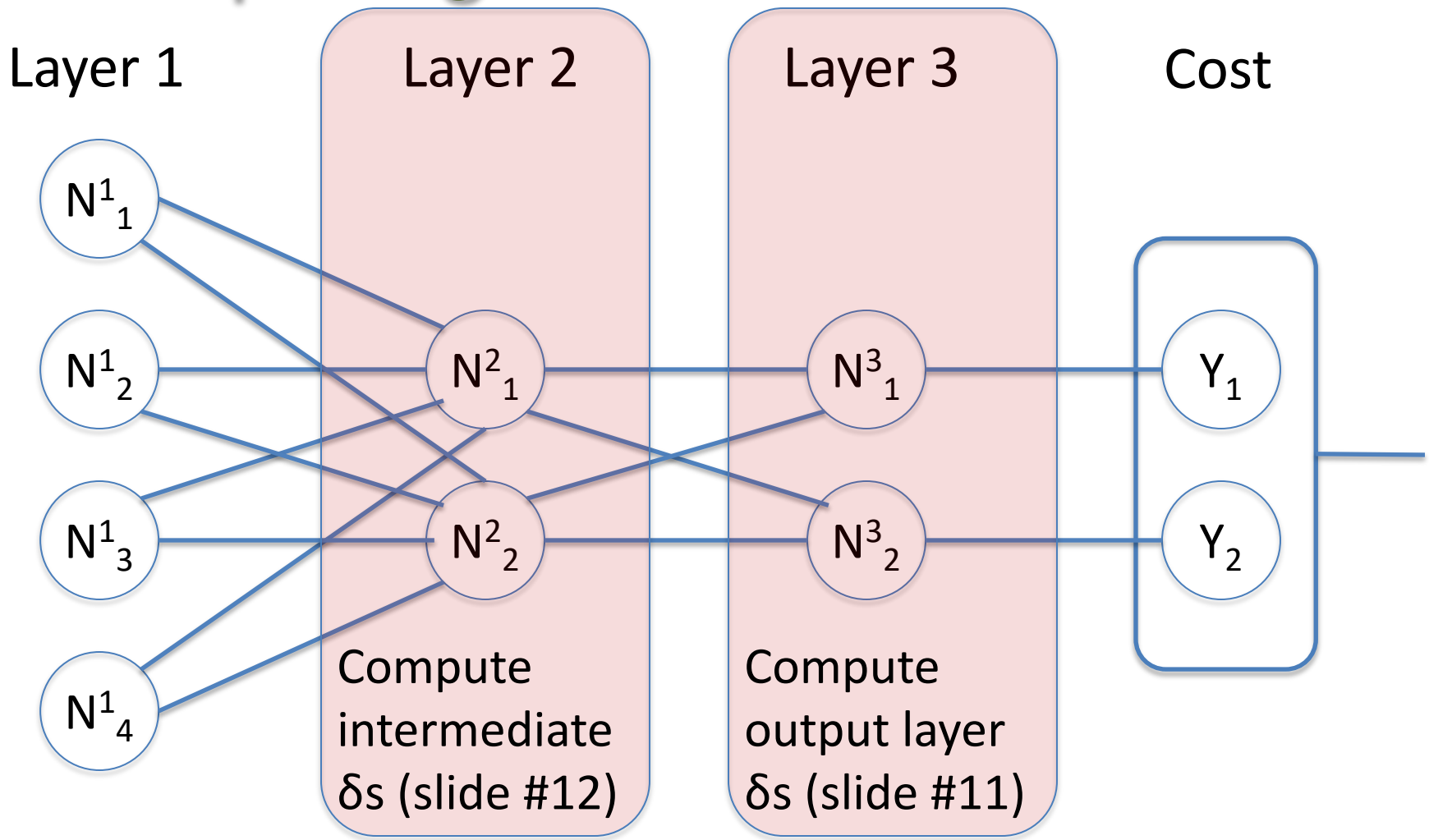
Output Layer (cont.)

- $\sigma'\left(z_j^L\right)$ is the derivative of the non-linear transfer function at z_j^L
- If $\sigma(x) = tanh(x)$, $\sigma'(x) = 1 - tanh^2(x)$
- If $\sigma(x) = (1 + e^{-x})^{-1}$,
$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$
- $\delta_j^L = \left(a_j^L(x) - y(x)\right)\left(1 - tanh^2\left(z_j^L(x)\right)\right)$ <u>or</u>
- $\delta_j^L = \left(a_j^L(x) - y(x)\right)\left(a_j^L(x)\left(1 - a_j^L(x)\right)\right)$

δ^L given δ^{L+1}

- $\delta_j^l = \sigma'(z_j^l) \sum_k w_{kj}^{l+1} \delta_k^{l+1}$
- σ' is computed as on previous slide
- The RHS is just the sum of the impacts
- This is where *backpropagation* comes from
  - Calculate δs for output layer
  - Then recursively compute δs for previous layers

Computing δs…



So…

- Given an input x and output y:
  - We can compute δ^l_j for every node j at every level l
  - Minimizing the δs will optimize the network
    - Relative to this sample
  - So we need to adjust the weights w_i and b to reduce the δs
    - But just a little for each input/output pair
    - So we can optimize across all samples

Adjusting b

- Remember that $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$ (slide #8)
- And that $z_j^l = w_j^l x + b$
- So $\frac{\partial C}{\partial b_j^l} = \delta_j^l$
- So $b_j^l \leftarrow (1-\alpha) b_j^l - \alpha \delta_j^l$
  - Where α is a learning rate
  - Regulates how much you react to each sample

Adjusting w's

- $\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1}\delta_j^l$
- So $w_{jk}^{l} \leftarrow (1-\alpha)w_{jk}^{l} - \alpha a_k^{l-1}\delta_j^l$
  - Where α is the same learning rate as before
  - We are collectively minimizing the deltas by heading downhill in the k+1 dimensional space defined by w & b

Backpropagation (redux)

- Backpropagation updates weights in a network, given
  - Training samples
  - Training labels
  - An cost function
- Network nodes may be
  - Non-linear perceptrons (the most common)
  - Convolutional units
  - Pooling units
  - Batch normalization units
  - …