Back Propagation

ISTD 50.035

Computer Vision

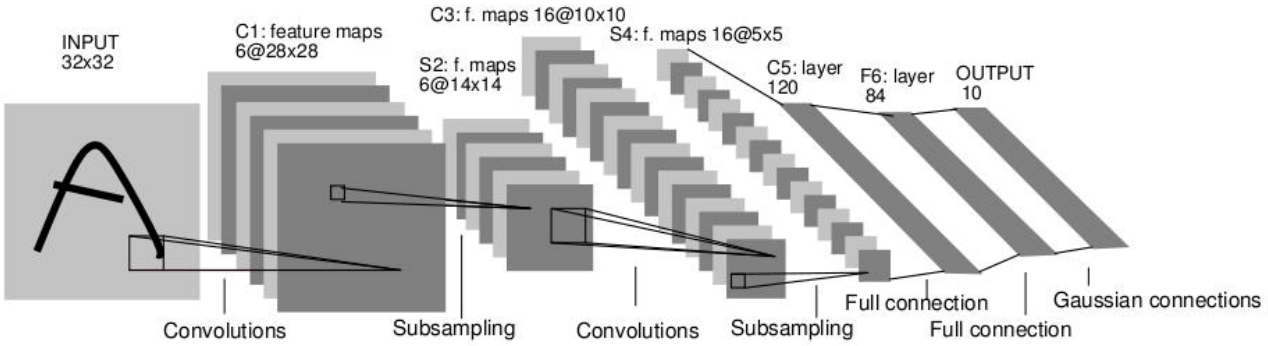
Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, etc.

Learn W using loss function L(W)

- Determine W with the min loss function

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

N training samples



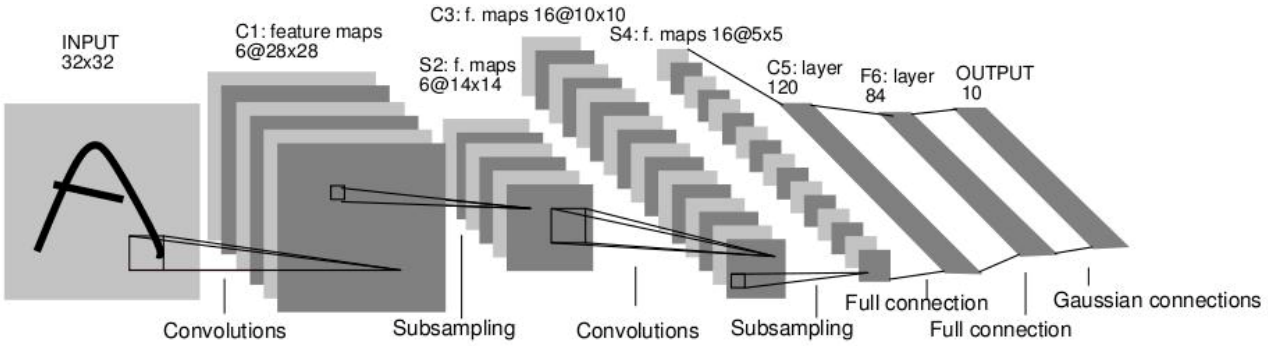
- Start from a random W, iteratively improve W (reduce L(W)): Gradient descent
- Note: L(W) = L(W; (x_1,y_1), (x_2,y_2), ...(x_i,y_i)...(x_N,y_N))

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$L(W) = L(w_1, w_2, ...w_l...)$$

$$W' = W - \gamma \nabla L$$ Gradient descent



$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

Find $\frac{\partial L}{\partial w_l}$ for every w_l

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

$$\frac{\partial L}{\partial w_l} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

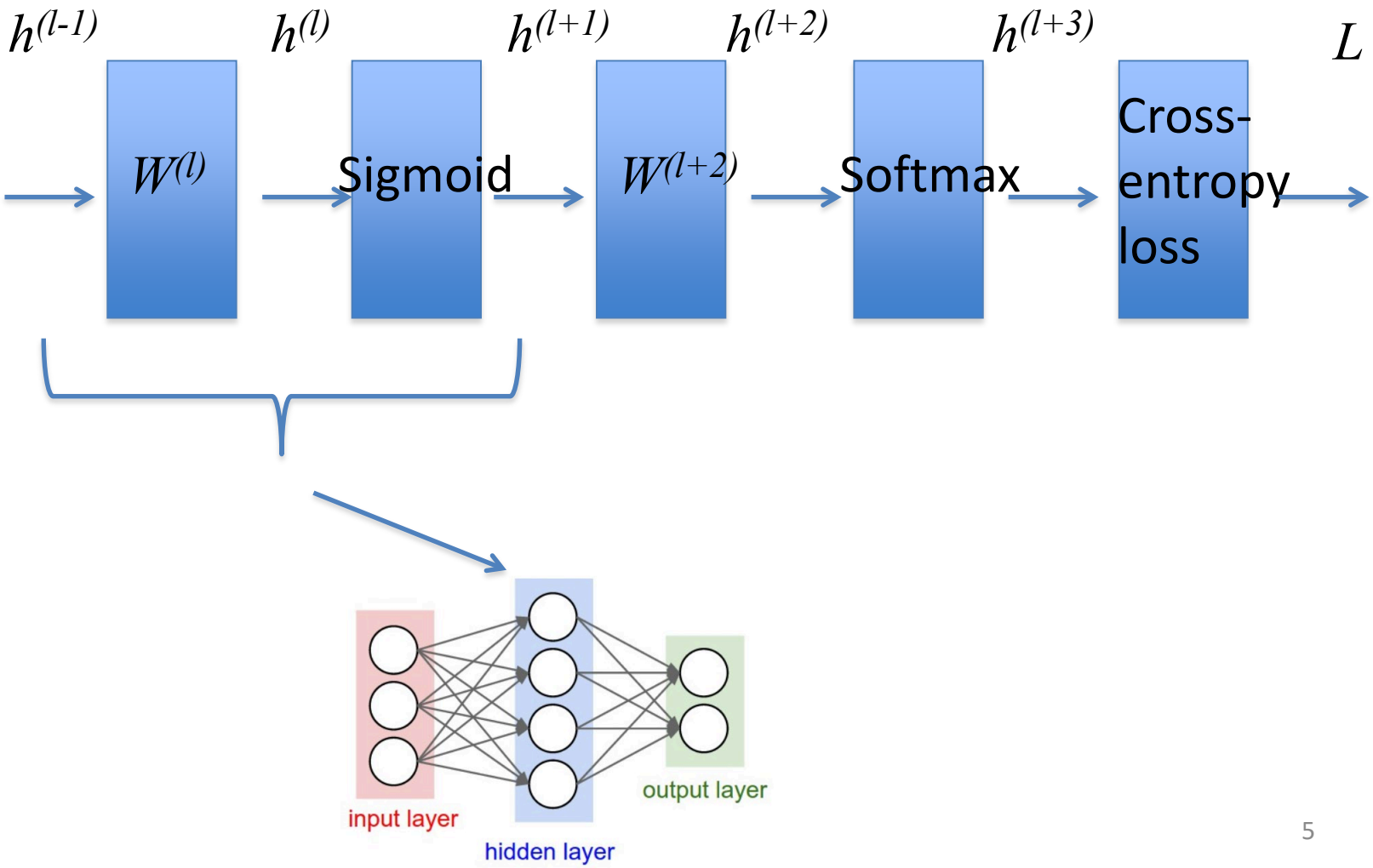
-Sum gradients for all (partial) training samples for one w_l

-Make one update of W once we have **all the gradients**

From now, refer to gradient of one training sample

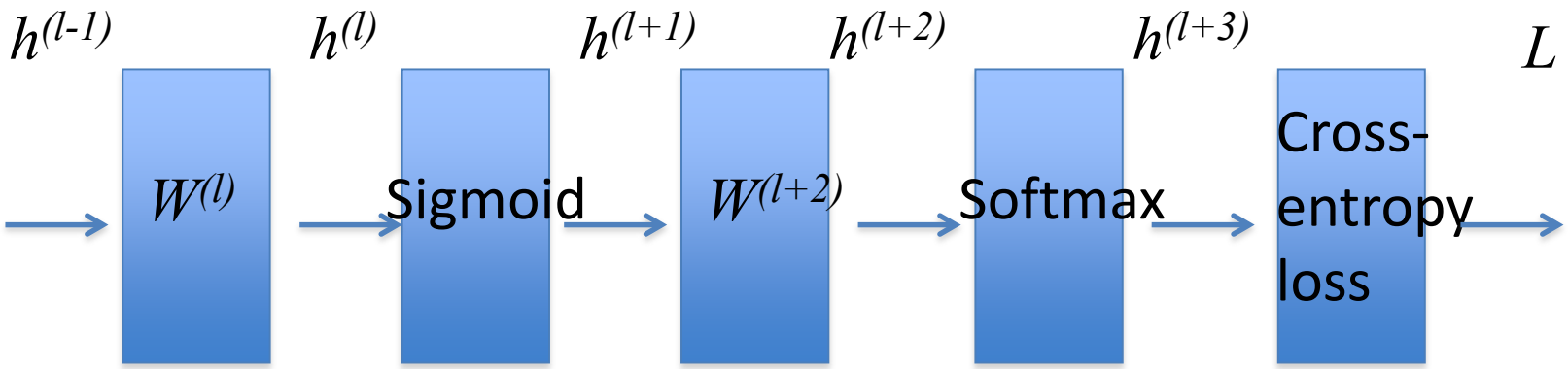
Compute the gradient

- Example



Compute the gradient

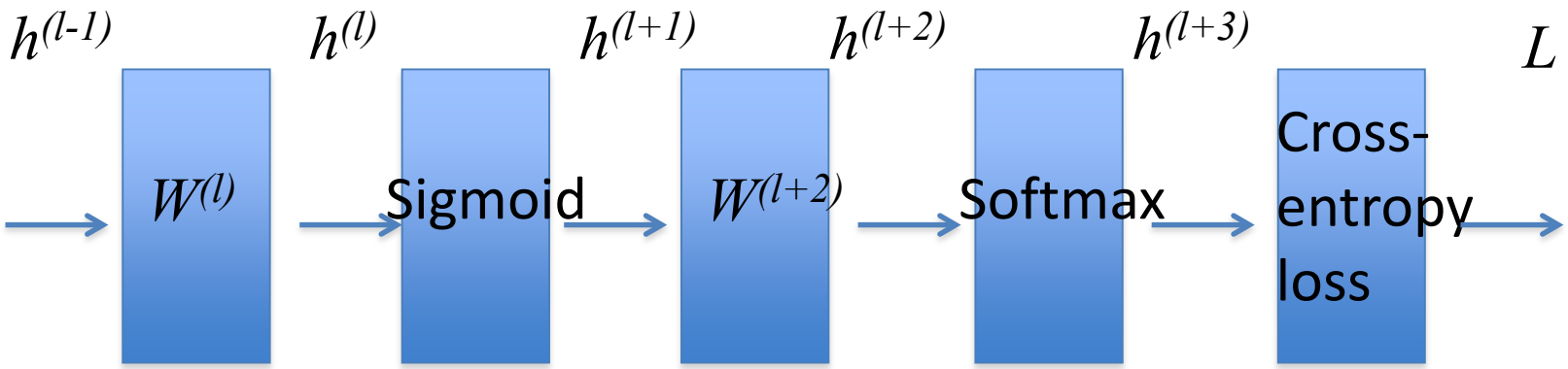
- Example



Approach 1: compute $L(W^{(l)})$, then differentiate

Compute the gradient

- Example

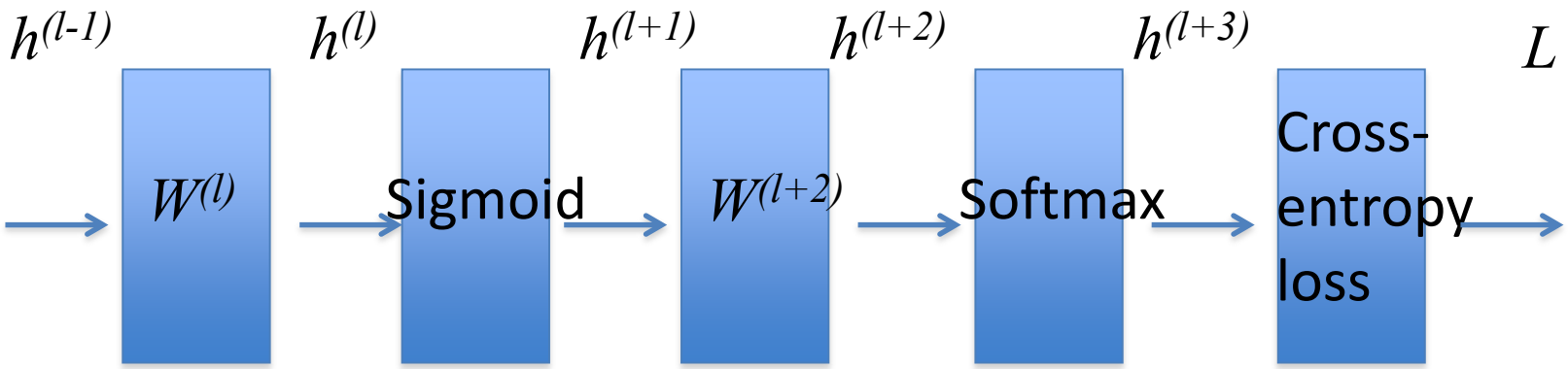


Approach 1: compute $L(W^{(l)})$, then differentiate

-Tedious task, e.g. for deep NN
-Not flexible; if some layer changes (Sigmoid -> ReLU), need to re-derive again

Compute the gradient

- Example



Approach 2: back propagation

-Assume we have $\frac{\partial L}{\partial h^{(l)}}$

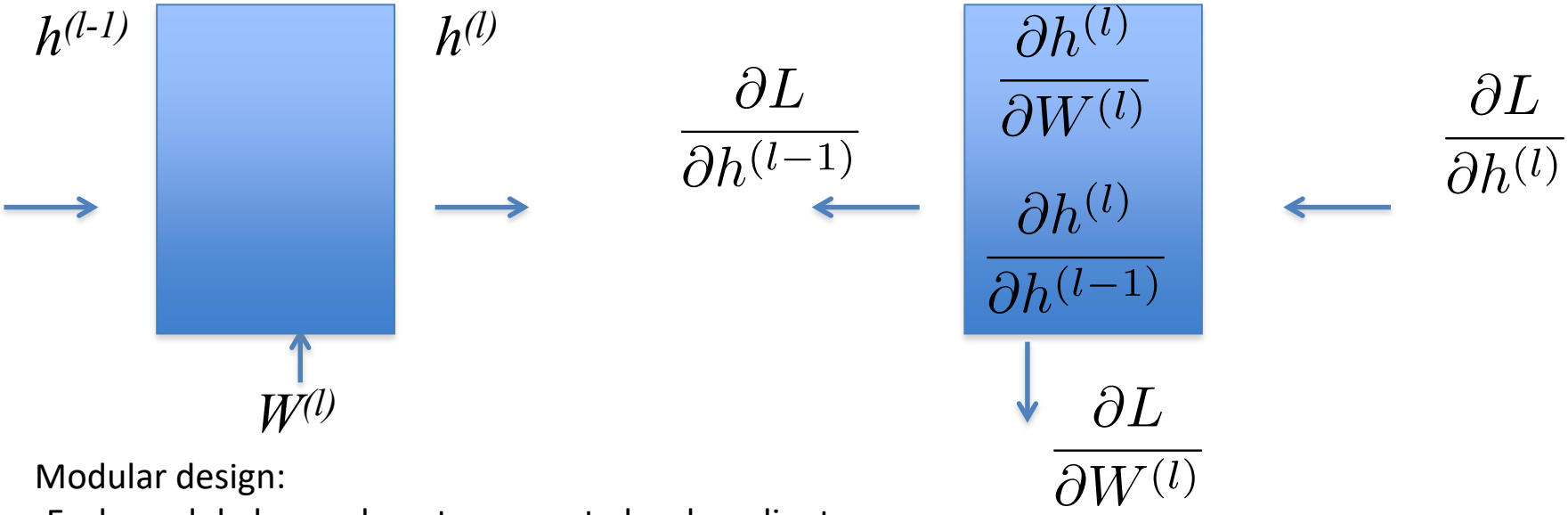
$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial W^{(l)}}$$

For grad descent

$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

Back prop



Modular design:
-Each module knows how to compute local gradients
-Multiply by the global gradient from upstream
-Generalize to other modules, e.g. ReLU
-Obtain exactly same gradient as Approach 1

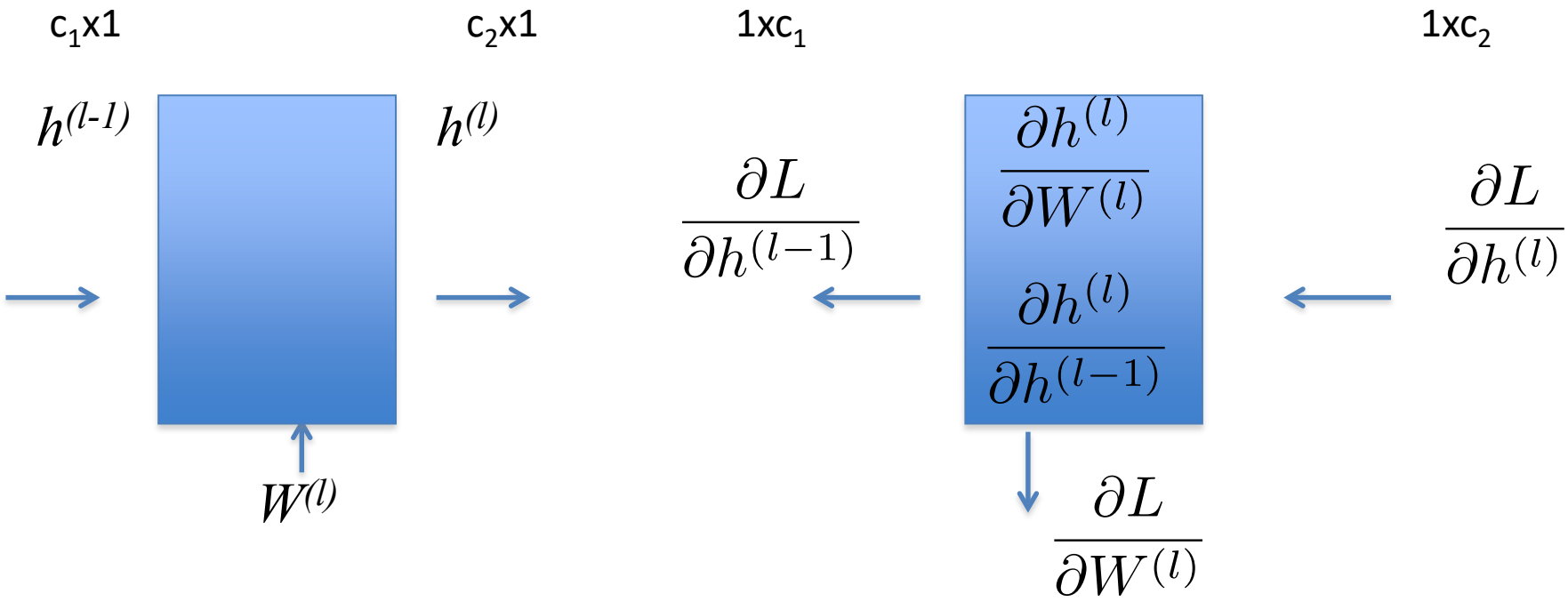
$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial W^{(l)}}$$

For grad descent

$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

Back prop: fully connected

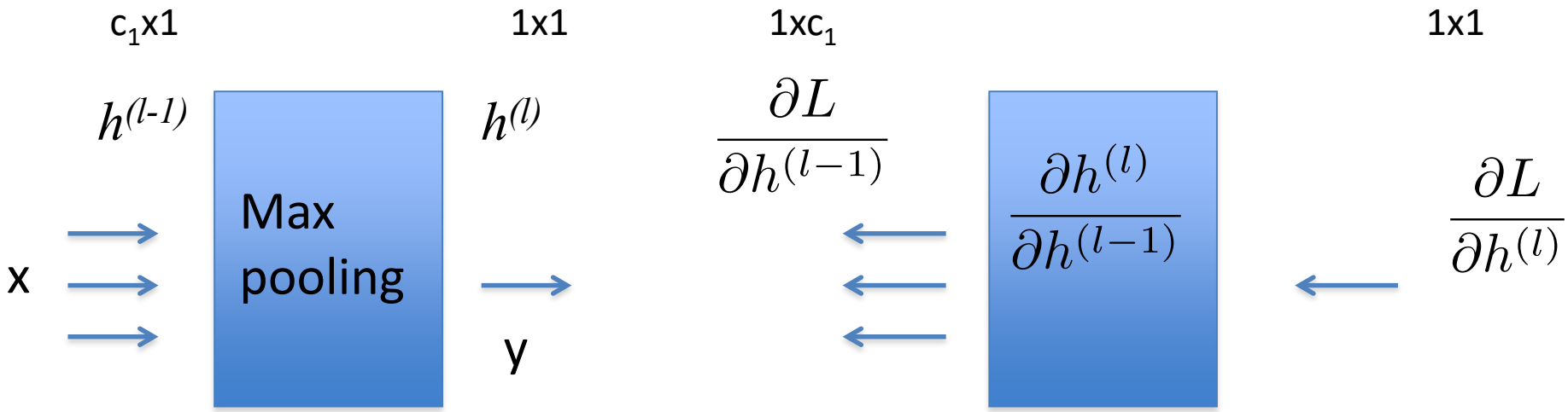


$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} W^{(l)}$$

$$\frac{\partial L}{\partial \mathbf{w}_i^{(l)}} = \frac{\partial L}{\partial h_i^{(l)}} [h^{(l-1)}]^T$$

$\mathbf{w}_i^{(l)}$: i-th row of $W^{(l)}$

Back prop: max pooling



If x is max, y=x, dy/dx = 1

If x is not max, y and x are independent, dy/dx=0

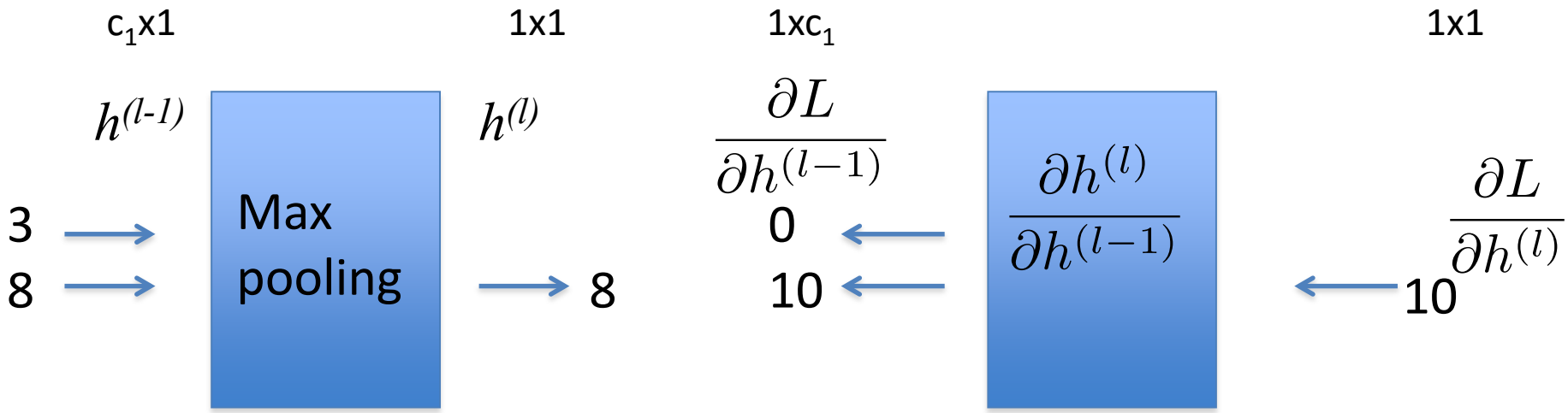
$$\frac{\partial h^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} \text{ is } \max) \qquad \frac{\partial L}{\partial h_i^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \mathbf{I}(h_i^{(l-1)} \text{ is } \max)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Only 1 branch has gradient, and
gradient goes to the max input branch

Intuition: If $h_i^{(l-1)}$ is not max, no impact to the loss, hence: $\frac{\partial L}{\partial h_i^{(l-1)}} = 0$

Back prop: max pooling



If x is max, y=x, dy/dx = 1

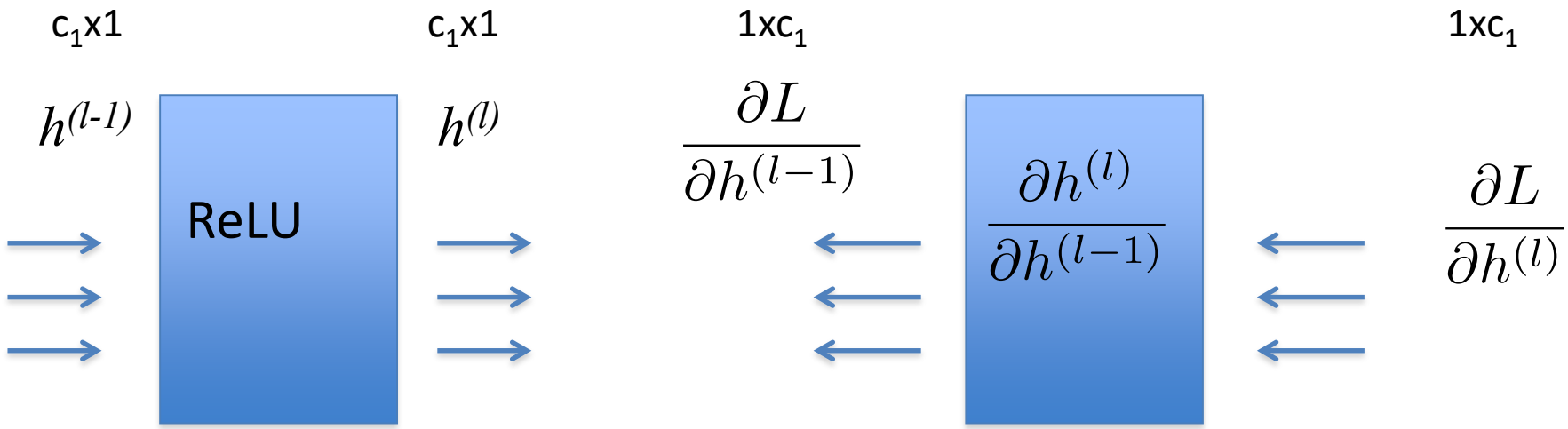
If x is not max, y and x are independent, dy/dx=0

$$\frac{\partial h^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} \text{ is max}) \qquad \frac{\partial L}{\partial h_i^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \mathbf{I}(h_i^{(l-1)} \text{ is max})$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Only 1 branch has gradient, and gradient goes to the max input branch

Back prop: ReLU



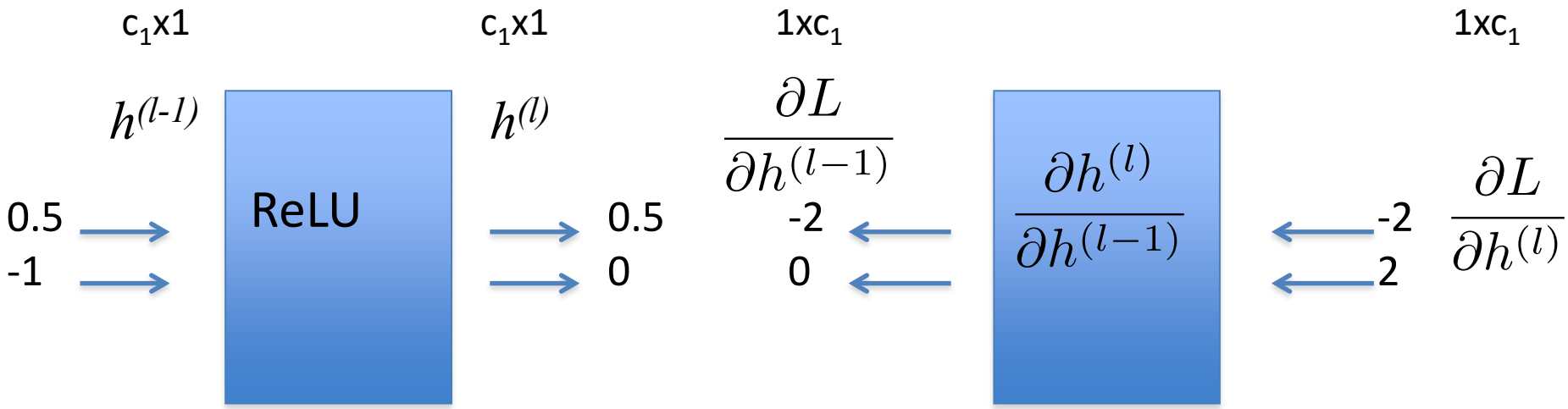
$$h_i^{(l)} = \max(0, h_i^{(l-1)})$$

$$\frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} > 0)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Gate: gradient can or cannot pass through

Back prop: ReLU

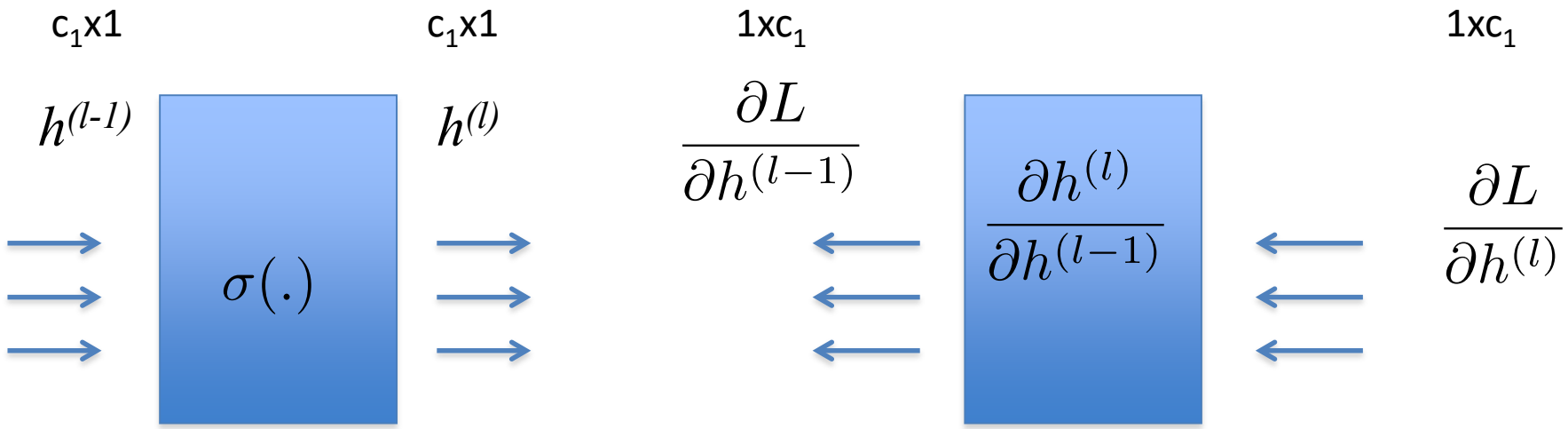


$$h_i^{(l)} = \max(0, h_i^{(l-1)}) \qquad \frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} > 0)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Gate: gradient can or cannot pass through

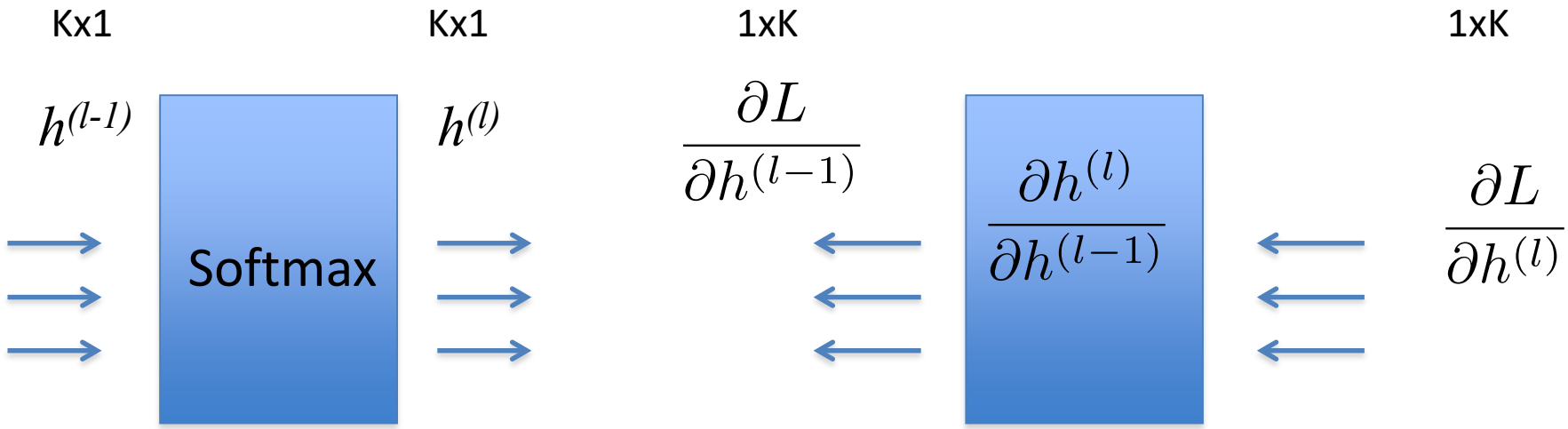
Back prop: Sigmoid



$$h_i^{(l)} = \sigma(h_i^{(l-1)})$$

$$\frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \sigma(h_i^{(l-1)})(1 - \sigma(h_i^{(l-1)}))$$

Back prop: Softmax

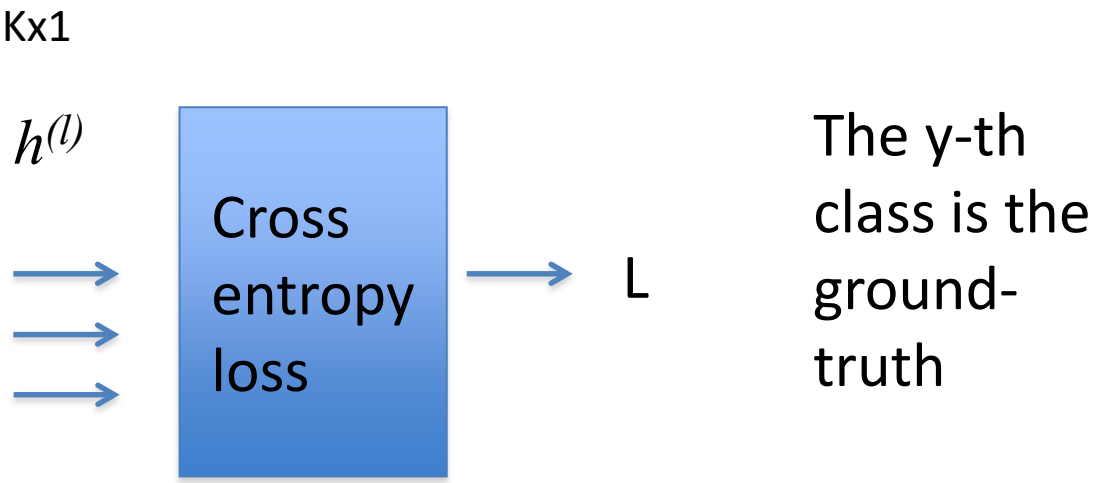


$\frac{\partial h^{(l)}}{\partial h^{(l-1)}}$ is a KxK Jacobian matrix

$$\frac{\partial h_i^{(l)}}{\partial h_j^{(l-1)}} = h_i^{(l)}(1 - h_j^{(l)}) \qquad i = j$$

$$\frac{\partial h_i^{(l)}}{\partial h_j^{(l-1)}} = -h_i^{(l)} h_j^{(l)} \qquad i \neq j$$

Back prop: Cross entropy loss

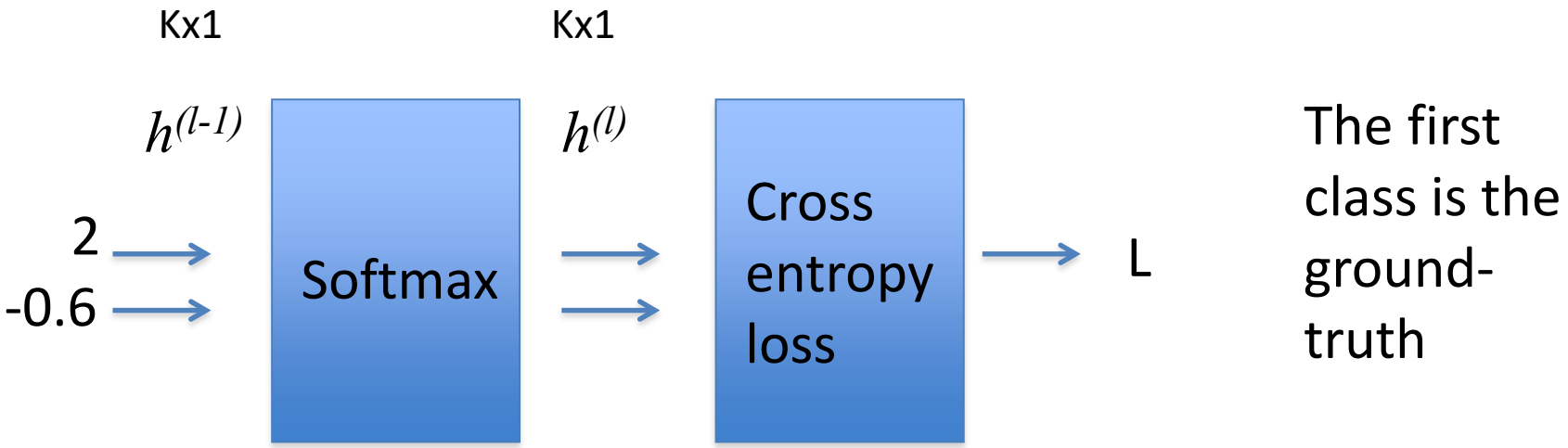


$$L = -\log(h_y^{(l)})$$

$$\frac{\partial L}{\partial h^{(l)}} = [0, 0, ..., \frac{-1}{h_y^{(l)}}, ..., 0]$$

Back prop starts from the loss function

Back prop: Exercise



Compute:

(i) L

(ii) $\frac{\partial L}{\partial h^{(l-1)}}$

(i) 0.07

(ii) [-0.0687 0.0687]

Gradient exploding and vanishing

- Gradients from deeper layers have to go through multiple multiplication until they reach earlier layers

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial W^{(l)}}$$

For grad descent

$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

- Gradients shrink exponentially until they vanish: not possible for the model to learn
- Gradients have large value and get larger and eventually blow up
- Not only CNN problem, but also RNN, GAN, and other gradient based learning approaches

Gradient exploding

- Gradient clipping: use a predefined threshold
- Does not change direction, only the length

$$\text{if } \|g\| > \ threshold$$

$$g \leftarrow \frac{threshold \times g}{\|g\|}$$

$$where: \ g \ is \ the \ gradient \ and$$

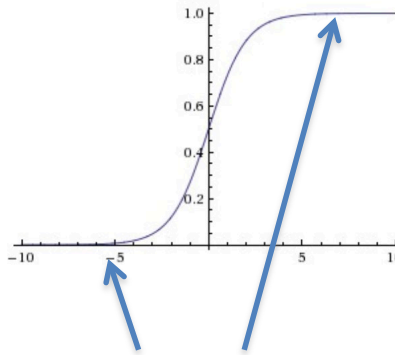
$$\|g\| \ is \ the \ norm \ of \ the \ gradient$$

Gradient vanishing

- Use non saturating nonlinearity

Sigmoid:

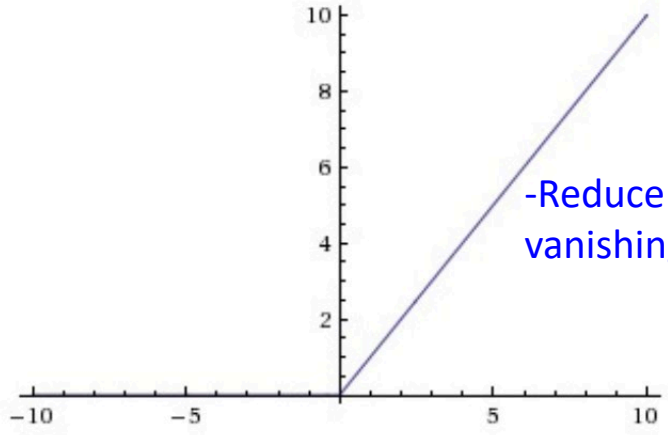
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

Rectified linear unit (ReLU):

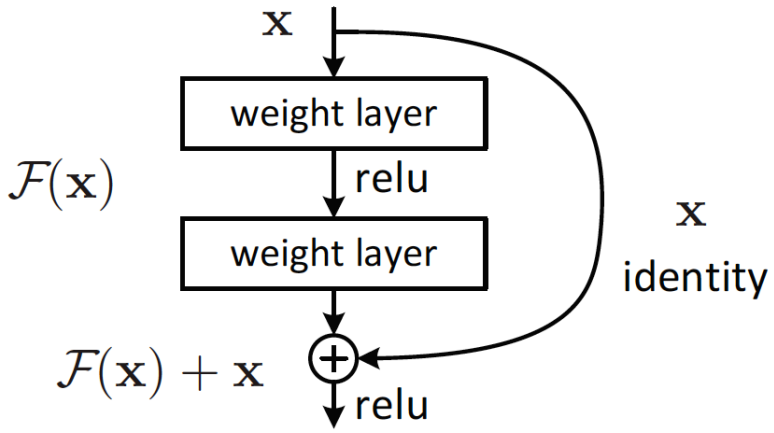


$$f(z) = max(0, z)$$

-Reduce issue of vanishing gradient

Gradient vanishing

- Recurrent NN -> Long short-term memory (LSTM)
- Shortcut connection (ResNet)



$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$
$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

Gradient vanishing

Residual networks with 54 modules:
1 path through all module (length = 54)
54 paths that go through a single module (length = 1)
… -> binomial distribution

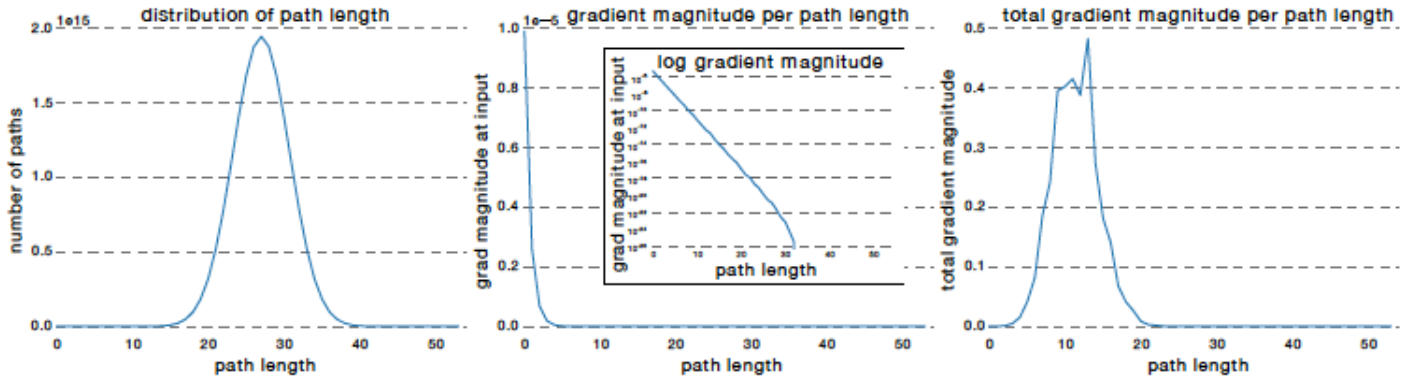
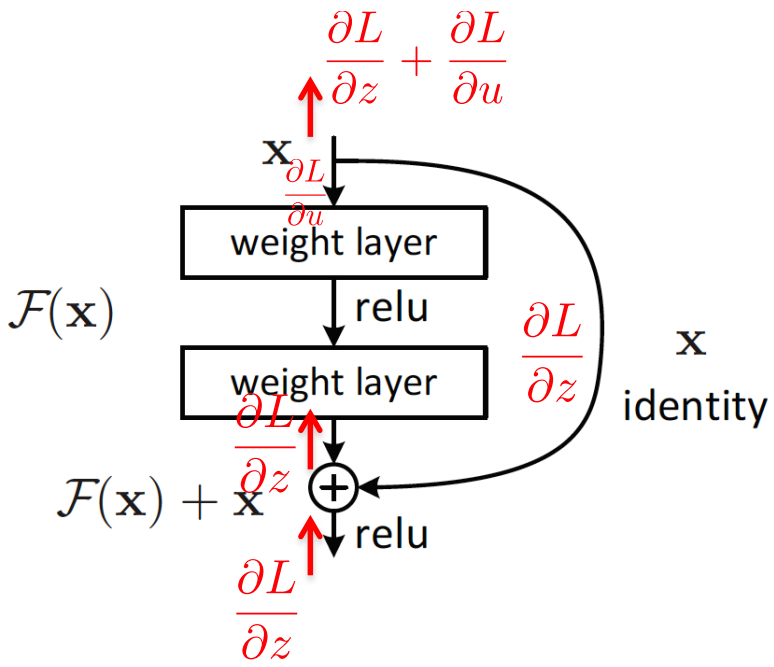


Figure 6: How much gradient do the paths of different lengths contribute in a residual network? To find out, we first show the distribution of all possible path lengths (a). This follows a Binomial distribution. Second, we record how much gradient is induced on the first layer of the network through paths of varying length (b), which appears to decay roughly exponentially with the number of modules the gradient passes through. Finally, we can multiply these two functions (c) to show how much gradient comes from all paths of a certain length. Though there are many paths of medium length, paths longer than ∼20 modules are generally too long to contribute noticeable gradient during training. This suggests that the effective paths in residual networks are relatively shallow.

Gradient vanishing

- Recurrent NN -> Long short-term memory (LSTM)
- Shortcut connection (ResNet)



$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$
$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$