Data Mining & Machine Learning

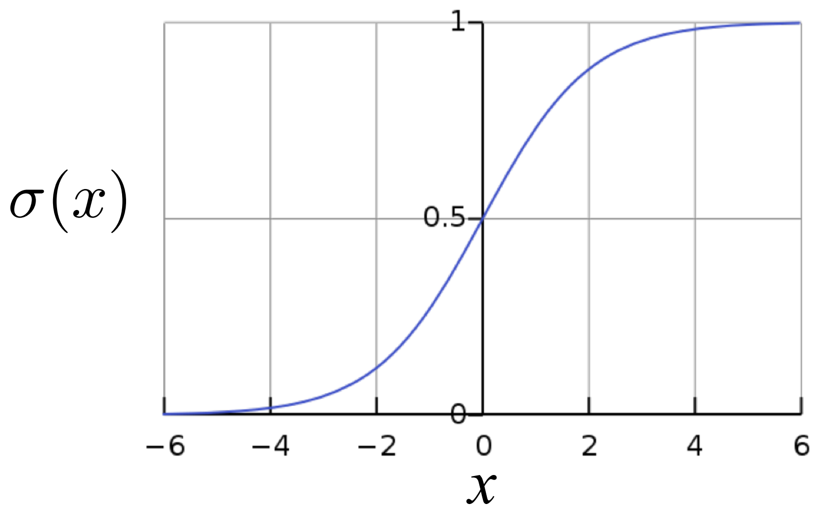
CS57300
Purdue University

February 27, 2017

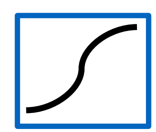
Logistic (neuron) Activation (non-linear filter)

- If input is $x = \mathbf{w}^T\mathbf{x}$, the output will look like a probability $\sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$

- $\mathrm{p}(\mathrm{y} = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$



$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}$$

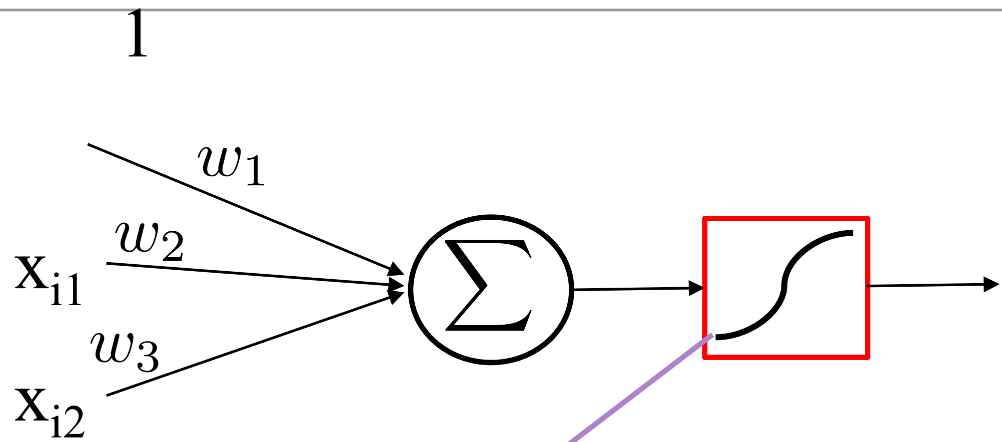
- We will represent the logistic function with the symbol: 

- Very simple derivative:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

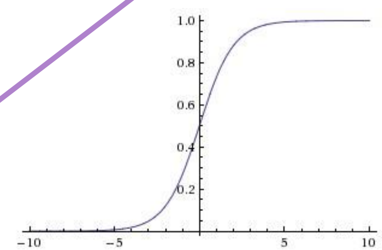
Some Activation Functions Used (two classes)

$G_k(x_i) =$

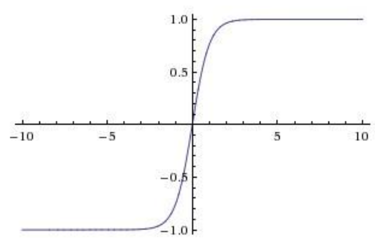


Sigmoid

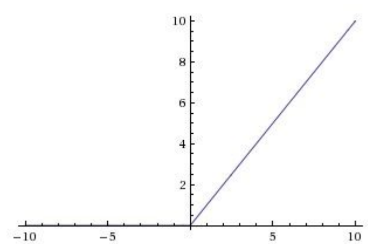
$$\sigma(x) = 1/(1+e^{-x})$$



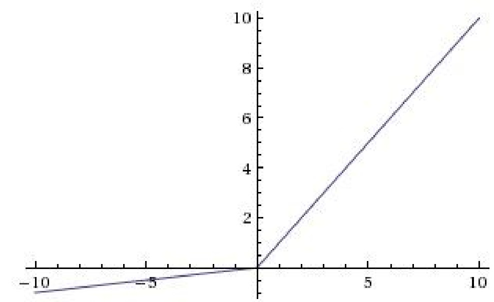
tanh tanh(x)



ReLU max(0,x)



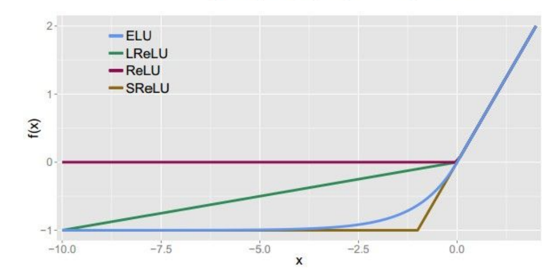
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

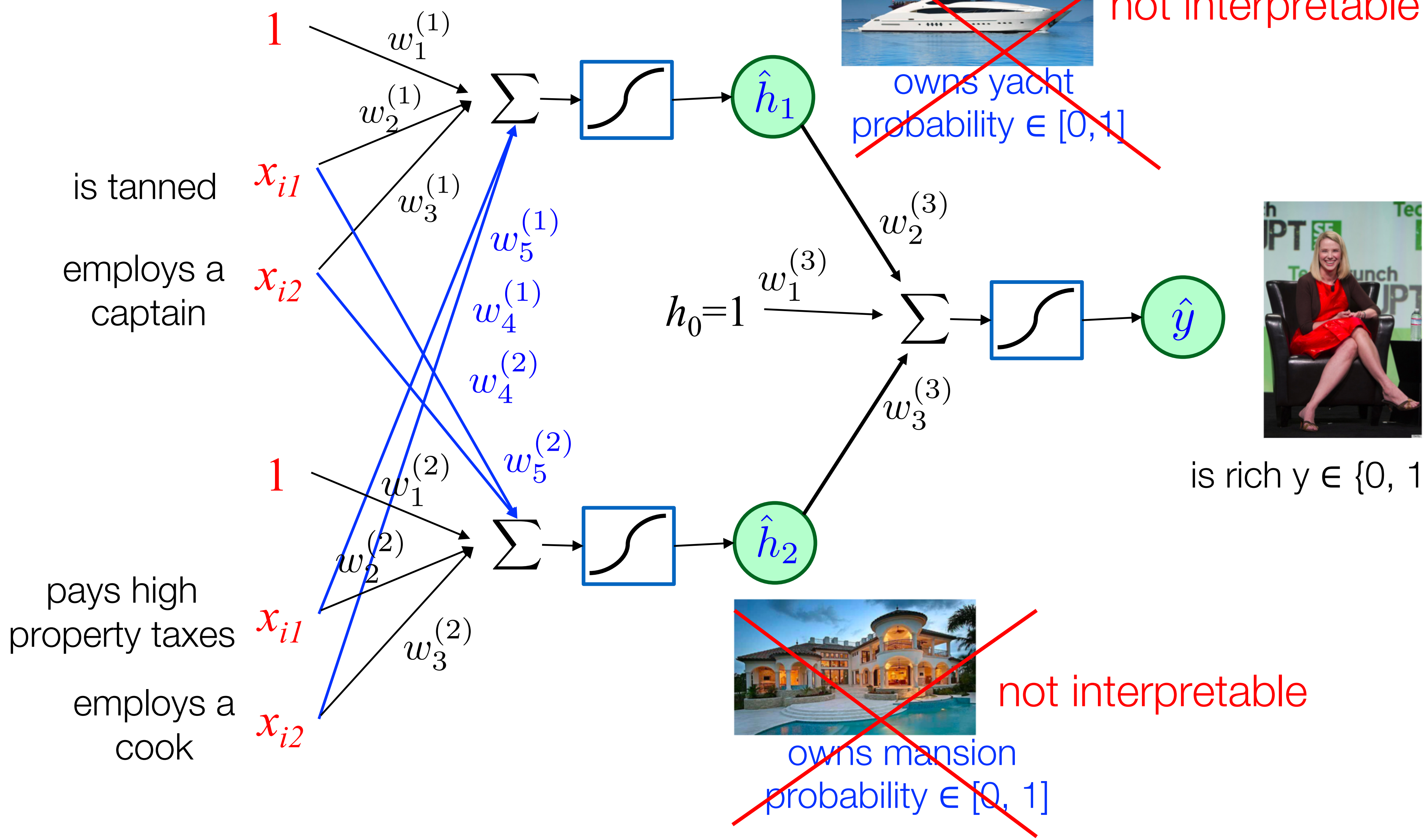
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Ack: Fei-Fei Li & Andrej Karpathy & Justin Johnson

More flexible model: Fully connected allowing inter-connected weights to have any value

Model M3



Model search for our example

- Training data: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- $\mathbf{x} = (1, x_{11}, x_{12}, x_{21}, x_{22})$
- $\mathbf{w}^{(k)} = (b, w_2^{(k)}, w_3^{(k)}, w_4^{(k)}, w_5^{(k)})$, $k = 1, 2$, where $b = w_1^{(1)} + w_1^{(2)}$
- $\mathbf{w}^{(3)} = (w_1^{(3)}, w_2^{(3)}, w_3^{(3)})$

Optimize using maximum likelihood estimation

Characteristics of user *i (tan, captain, cook, …)*

$$\underset{\mathbf{w}^{(1)},\mathbf{w}^{(2)},\mathbf{w}^{(3)}}{\arg\max} \frac{1}{n}\sum_{i=1}^{n} \log p(y = y_i | \mathbf{x}_i; \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$$

User *i* in training data: is rich $y_i \in \{0, 1\}$

$$= \frac{1}{n}\sum_{i=1}^{n} y_i \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))),$$

where $\sigma(x) = \frac{\exp(x)}{1+\exp(x)}$, and $\mathbf{h}(\mathbf{x}) = (1, \hat{h}_1(\mathbf{x}), \hat{h}_2(\mathbf{x}))$,

$$\hat{h}_1(\mathbf{x}) = p(h_1 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(1)})^T \mathbf{x})$$

and

$$\hat{h}_2(\mathbf{x}) = p(h_2 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(2)})^T \mathbf{x})$$

Maximize likelihood via gradient ascent

- Model search via gradient ascent, requires computing the gradient first with respect to all parameters

Let $\mathbf{W} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$. Learning via maximum likelihood estimation

$$\frac{\partial}{\partial \mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} \log p(y = y_i | \mathbf{x}_i; \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$$

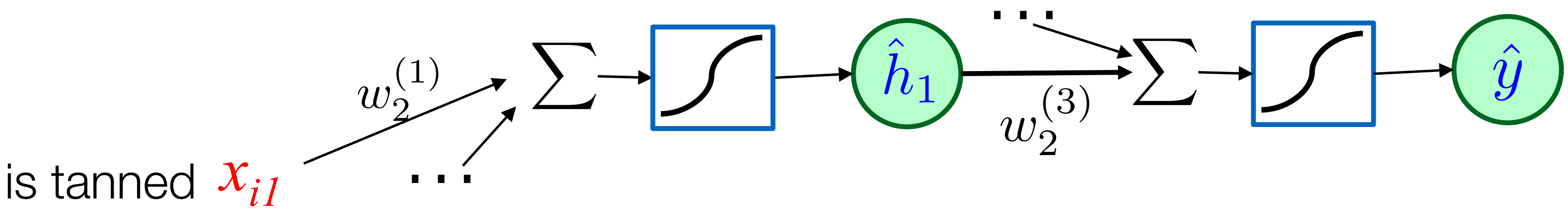
$$= \frac{1}{n} \sum_{i=1}^{n} y_i \frac{\partial}{\partial \mathbf{W}} \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \frac{\partial}{\partial \mathbf{W}} \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))$$

$$= \frac{1}{n} \sum_{i=1}^{n} y_i \frac{\frac{\partial}{\partial \mathbf{W}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))}{\sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))} - (1 - y_i) \frac{\frac{\partial}{\partial \mathbf{W}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))}{(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{W}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) \left(\frac{y_i}{\sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))} - \frac{1 - y_i}{(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))} \right)$$

Computing Gradients of the Lower-Layer Parameters

- In a deep neural network each layer is a composition of previous layers



- The influence of a lower layer parameter in the final error can be recovered by the chain rule, which generally states:

$$\frac{\partial f^l(f^{l-1}(\cdots f^2(f^1(w))))}{\partial w} = \frac{\partial f^l}{\partial f^{l-1}} \cdot \frac{\partial f^{l-1}}{\partial f^{l-2}} \cdots \frac{\partial f^2}{\partial f^1} \cdot \frac{\partial f^1(x)}{\partial w}$$

- Specific to the example given above:

$$\frac{\partial \sigma(\ldots + w_2^{(3)} \sigma(\ldots + w_2^{(1)} x_{i1}))}{\partial w_2^{(1)}} = \frac{\partial \sigma(\ldots + w_2^{(3)} \hat{h}_1)}{\partial \hat{h}_1} \cdot \frac{\partial \sigma(\ldots + w_2^{(1)} x_{i1})}{\partial w_2^{(1)}}$$

where $\hat{h}_1 = \sigma(\ldots + w_2^{(1)} x_{i1})$

How Tensorflow / Pytorch search model parameters

- neural nets will be very large: no hope of writing down gradient formula for all parameters

- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates

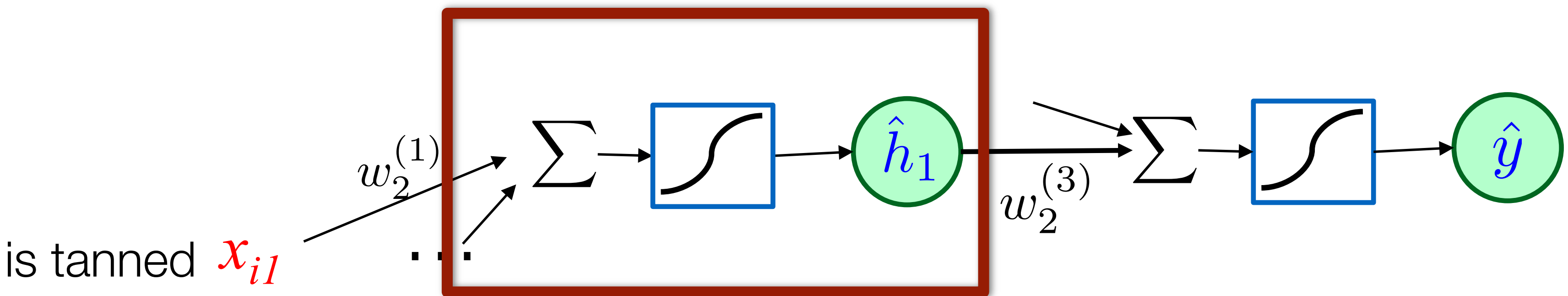
- Widely used implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() functions.

  - **forward:** compute result of an operation and save any intermediates needed for gradient computation in memory

  - **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs.

Neural Network Gradient Ascent

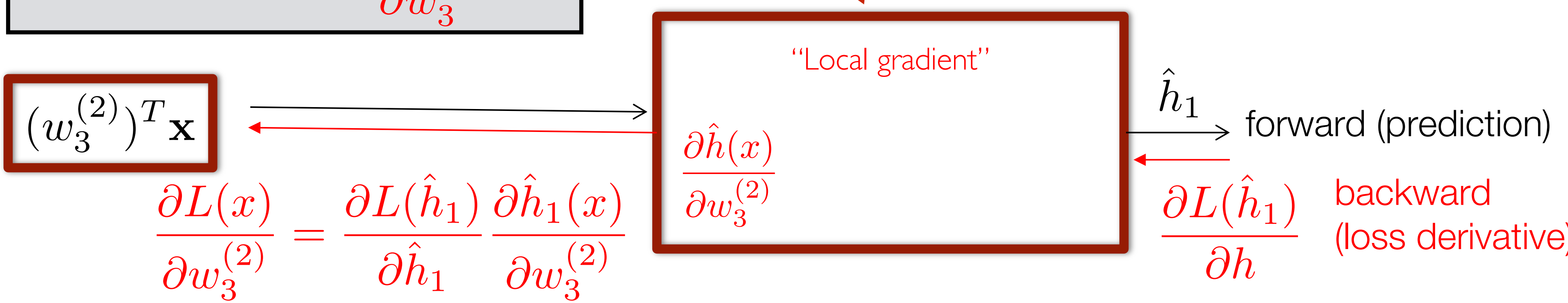
- L is the loss function with respect to final output



Gradient update step
(learning rate $\varepsilon \cong 0$):

$$w_3^{(2)} = w_3^{(2)} + \epsilon \frac{\partial L(x)}{\partial w_3^{(2)}}$$

- The local gradient measures how the output changes with the input



How it works: Forward and backward updates of last layer parameters

loss function
computes prediction error
of every training example i :

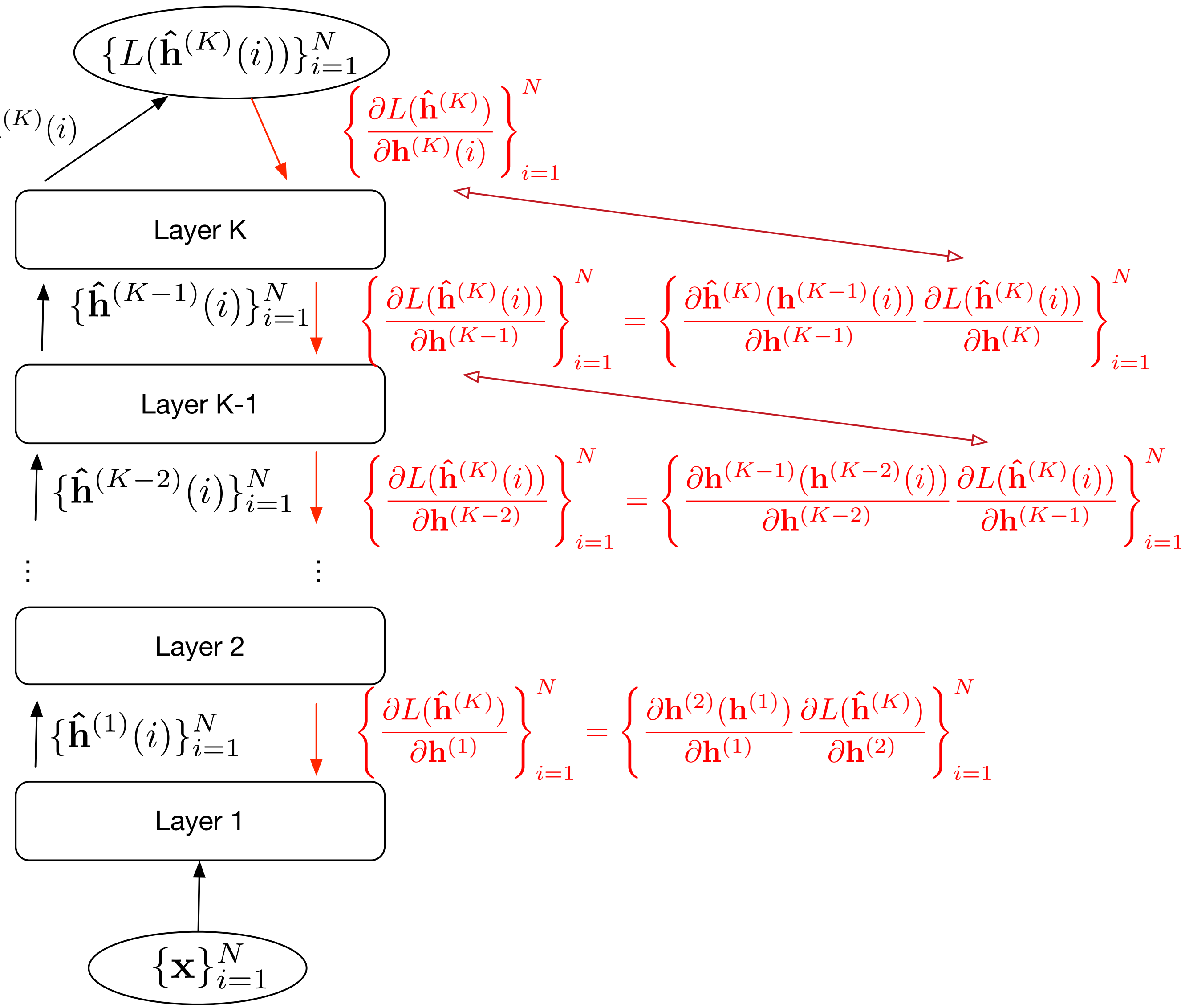
prediction $\hat{\mathbf{y}}(i) = \hat{\mathbf{h}}^{(K)}(i)$

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$



Updates at other layers

- Assume current layer is the ``last'' layer
- Update works the same as in the previous slide
- Updates of all upper layers can be performed together with the update of the lowest layer parameters