Data Mining & Machine Learning

CS37300
Purdue University

October 23, 2017

Kaggle competition update

Public Leaderboard | Private Leaderboard

This leaderboard is calculated with approximately 30% of the test data.
The final results will be based on the other 70%, so the final standings may be different.

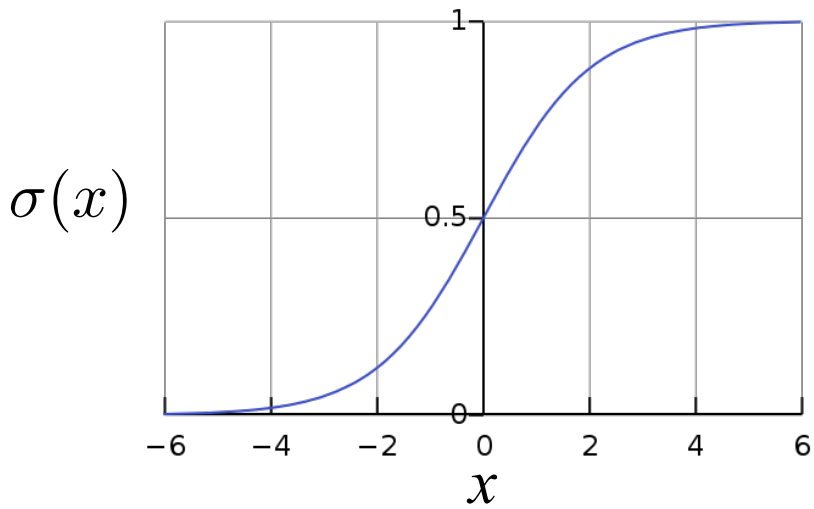
Raw Data | Refresh

#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	—	**Luke Skywalker**			0.83286	8	9d
2	—	**Cad Bane**			0.81303	16	2h
3	—	**Yoda**			0.79522	9	6d
4	—	**Revan**			0.78591	7	5d
5	—	**Ki-Adi-Mundi**			0.76811	1	12d
6	—	**Kyp Durron**			0.73613	4	14d
7	—	**Shaak Ti**			0.70133	2	12d
8	—	**Bossk**			0.67826	2	17d
9	—	**Admiral Thrawn**			0.65520	1	12d
10	—	**General Grievous**			0.58599	3	12d
11	—	**Boba Fett**			0.52407	11	7d
12	—	**Count Dooku**			0.51071	1	10d
13	—	**Darth Maul**			0.49129	2	17d
		Bank_Sample_Submission.csv			0.49008		
14	new	**Zuckuss**			0.49008	1	2d

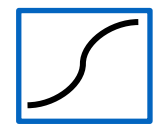
Deep Learning (Model Search)

Logistic (neuron) Activation (non-linear filter)

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



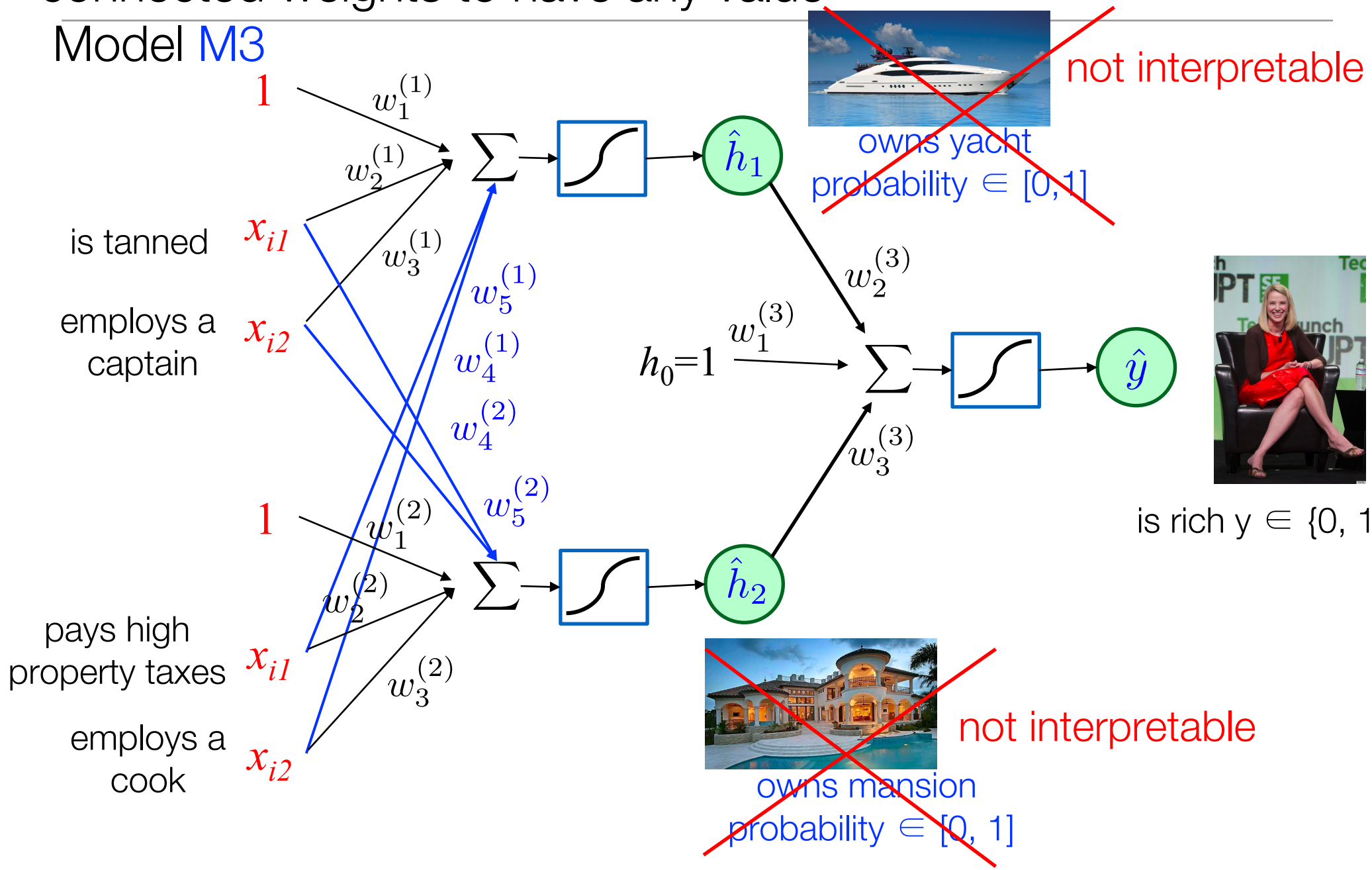
$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}$$

- We will represent the logistic function with the symbol: 
- Very simple derivative:

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

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,...)*

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

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

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

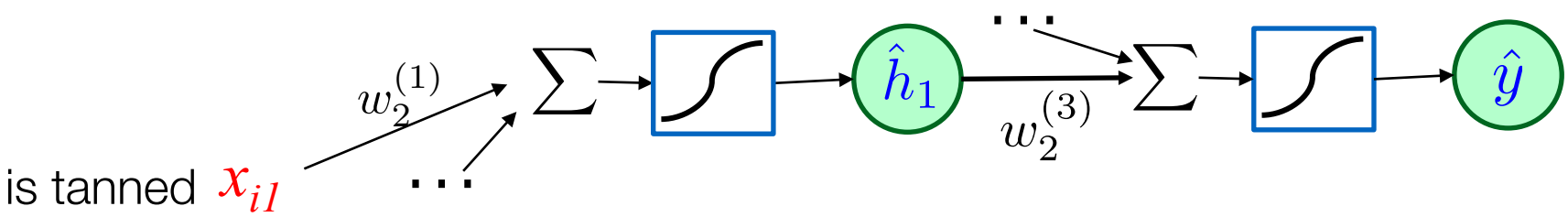
$$= \frac{1}{n} \sum_{i=1}^{n} y_i \frac{\partial}{\partial \mathbf{W}} \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \frac{\partial}{\partial \mathbf{W}} \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))$$

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

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

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:

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

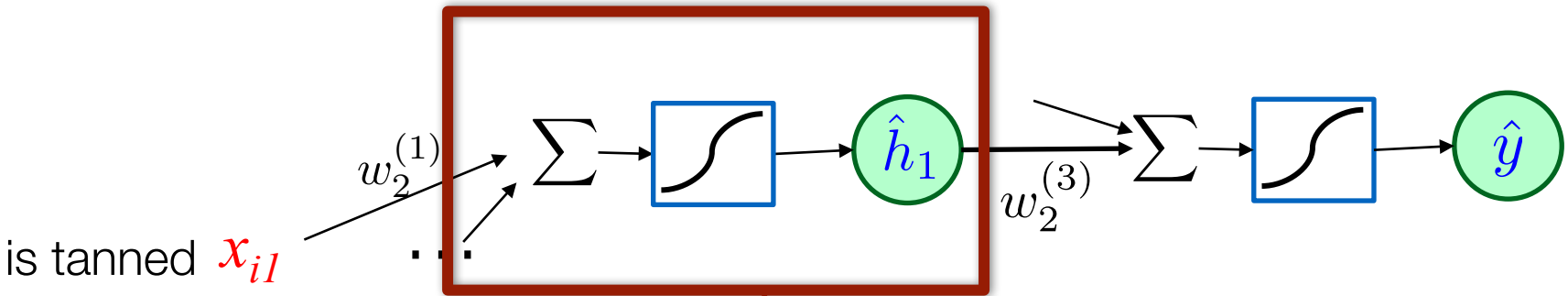
where $\hat{h}_1 = \sigma(\ldots + w_2^{(1)} x_{i1})$

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

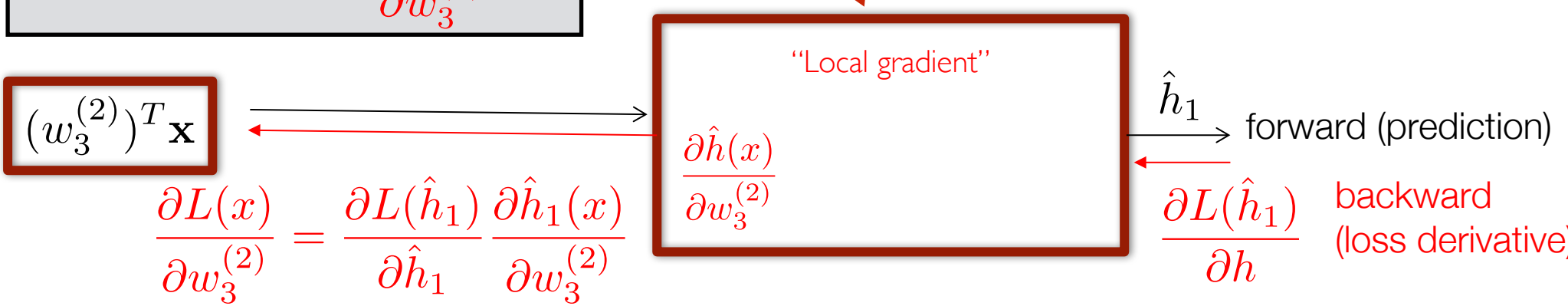
- L is the loss function with respect to final output



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

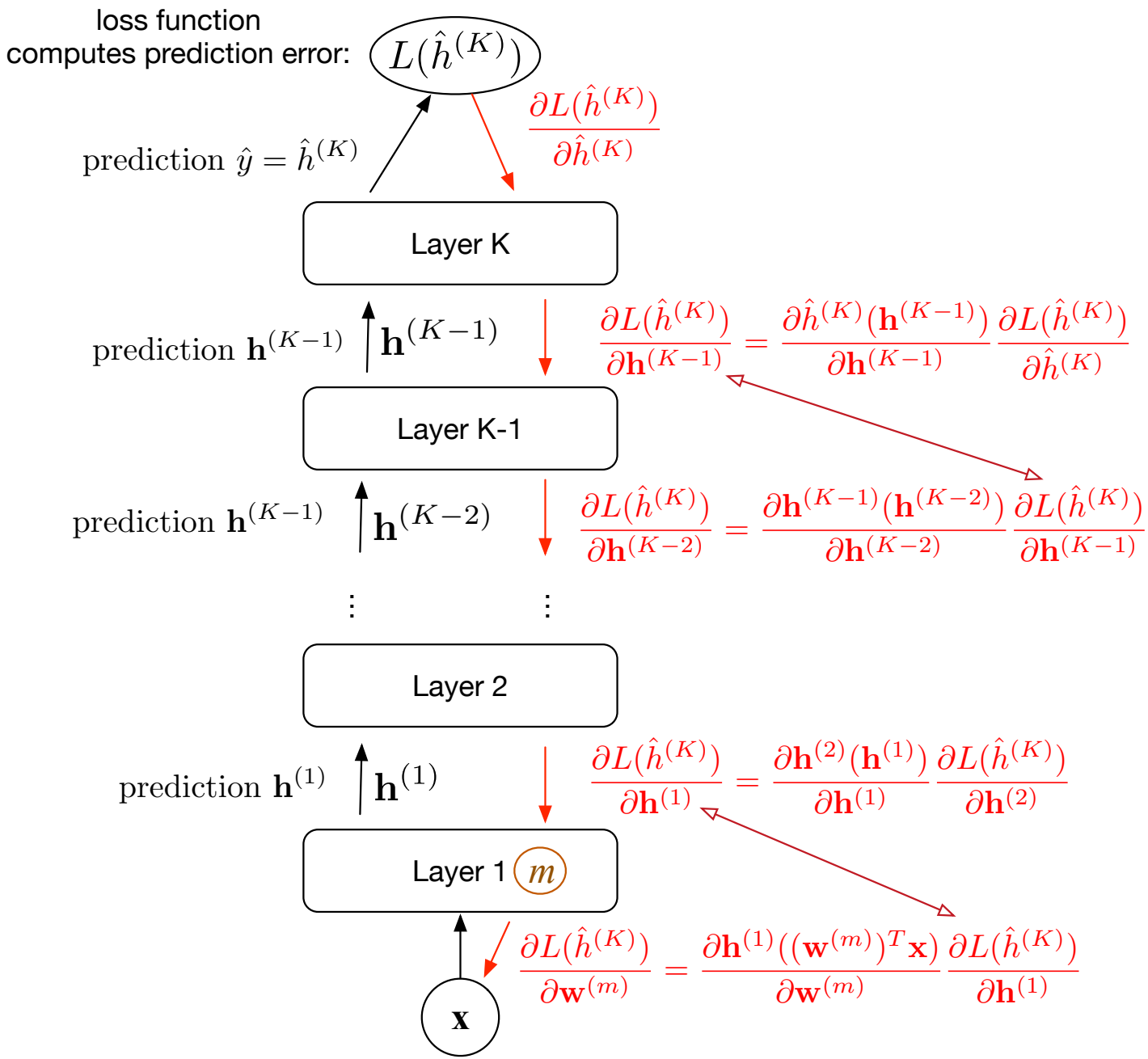
$$w_3^{(2)} = w_3^{(2)} + \epsilon \frac{\partial L(x)}{\partial w_3^{(2)}}$$

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



$$\frac{\partial L(x)}{\partial w_3^{(2)}} = \frac{\partial L(\hat{h}_1)}{\partial \hat{h}_1} \frac{\partial \hat{h}_1(x)}{\partial w_3^{(2)}}$$

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



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