Multi-layer Perceptrons and Back-Propagation

EE599 Deep Learning

Keith M. Chugg

Spring 2019

USC University of Southern California

MLP/Back-Prop Topics

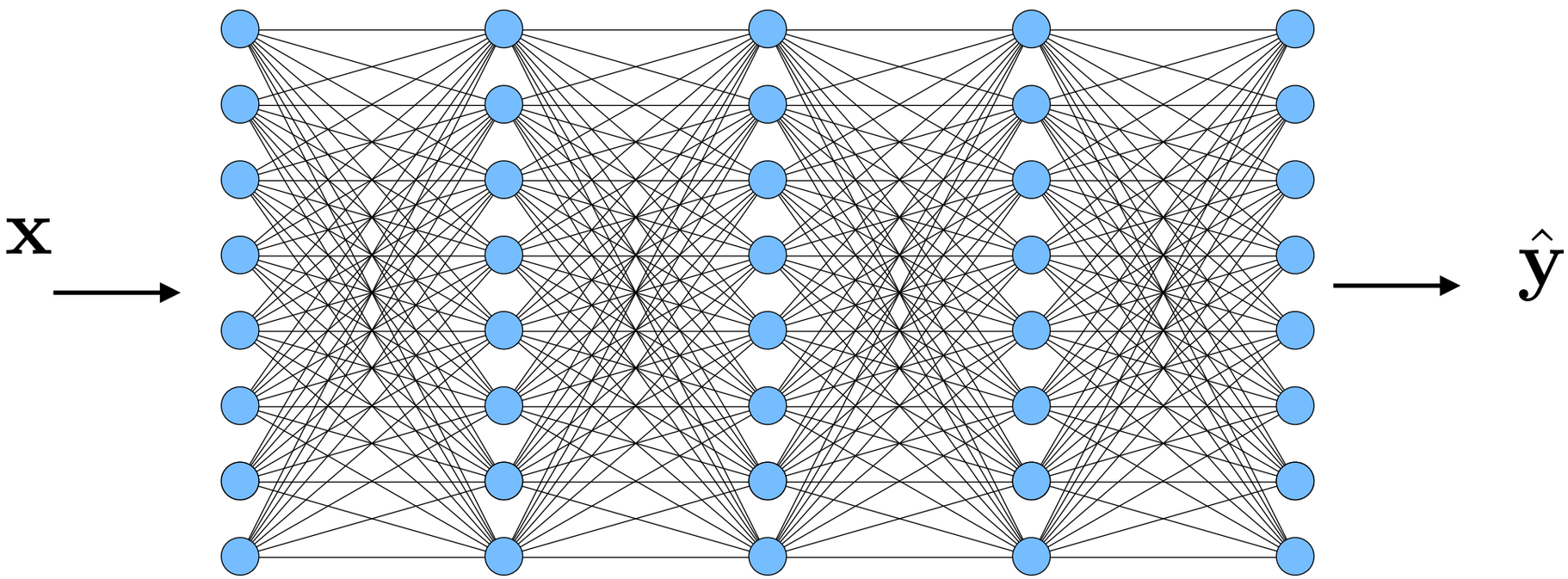
- MLP feedforward (inference) equations
- Back-propagation derivation
  - from scalar to the matrix-vector case
  - General rules/conventions for matrix-vector calculus
  - Variations on back-prop and MLP training
    - activations and their derivatives
    - regularizers, optimizers (e.g., adam) and momentum
    - dropout, batch normalization
- Universal Approximation
- Automatic Speech Recognition (ASR) guest lecture

Sourya will cover these in lecture

MLPs

Forward propagation (inference and training)

$$\mathbf{a}^{(l)} = \underline{h}(\mathbf{W}_l \mathbf{a}^{(l-1)} + \mathbf{b}_l) \qquad \mathbf{\Theta} = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=0}^{L-1}$$ (trainable parameters)



Backward propagation (training)

Learn the trainable parameters using SGD and the chain-rule

MLP Forward Prop Details

vector vs matrix vs scalar views

MLP Backprop Idea

do SGD on all trainable parameters:

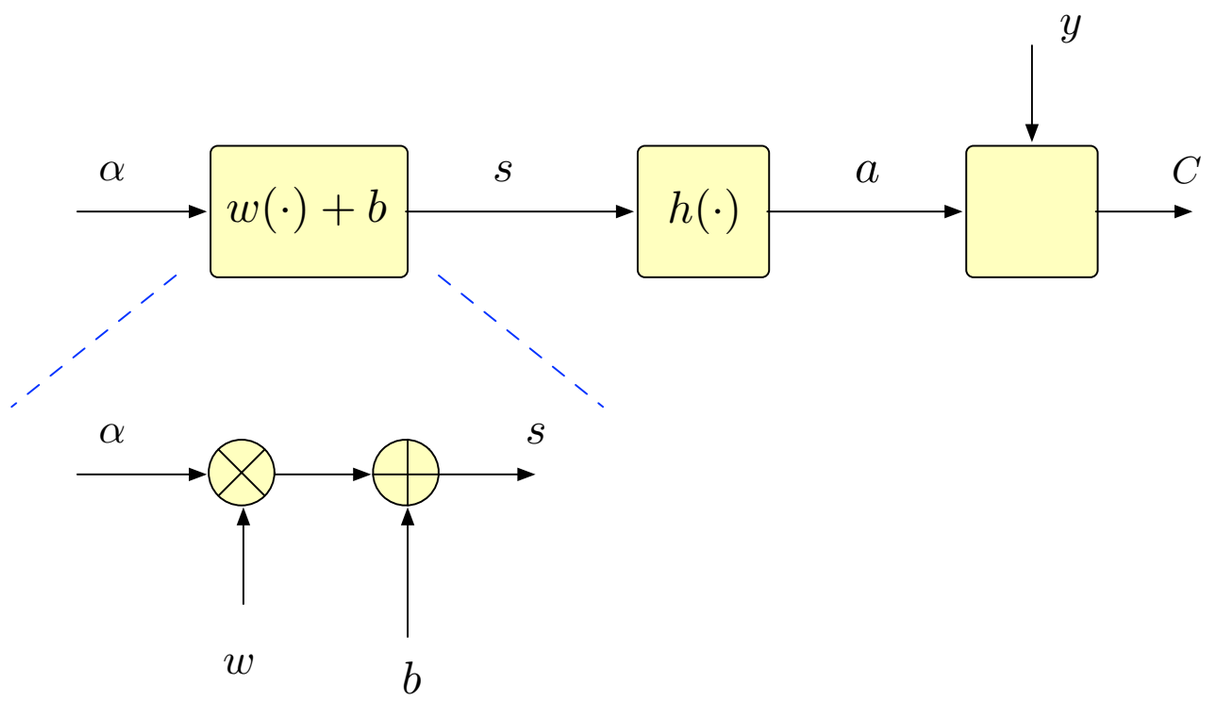
weight updates: $$w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \frac{\partial C}{\partial w_{i,j}^{(l)}}$$

bias updates: $$b_{i}^{(l)} \leftarrow b_{i}^{(l)} - \eta \frac{\partial C}{\partial b_{i}^{(l)}}$$

all layers, all indices: $$\forall\, l \in \{1, 2, \ldots L\}, i, j$$

Backprop is an algorithm for computing these, starting at the neural network output and propagating backward to the input layer using the **chain rule**

MLP-Backprop: scalar example



want to compute: $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$

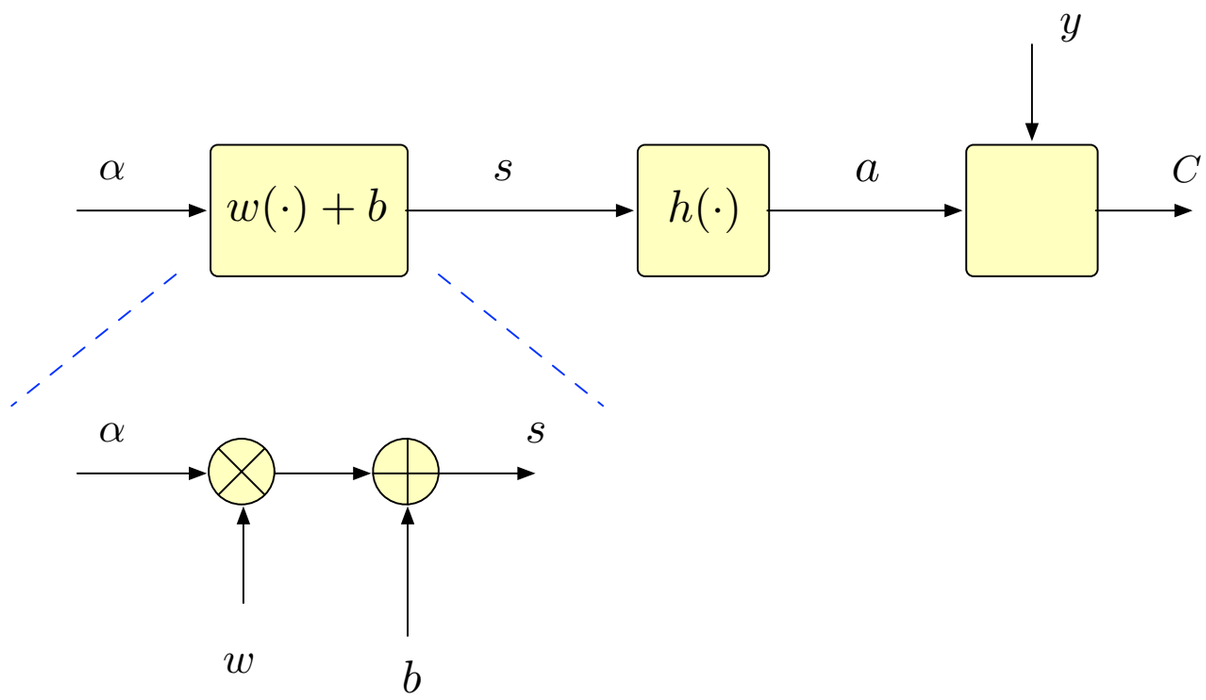
Note that: $\frac{\partial s}{\partial w} = \alpha$ $\frac{\partial s}{\partial b} = 1$

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial s}\frac{\partial s}{\partial w} = \alpha\frac{\partial C}{\partial s}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial s}\frac{\partial s}{\partial b} = \frac{\partial C}{\partial s}$$

so we can find $\frac{\partial C}{\partial s}$ and convert to the desired partials easily

MLP-Backprop: scalar example



$$\frac{\partial C}{\partial s} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial s}$$

$$= \frac{\partial C}{\partial a}\frac{\partial h(s)}{\partial s}$$

$$= \dot{C}(a)\dot{h}(s)$$

shorthand

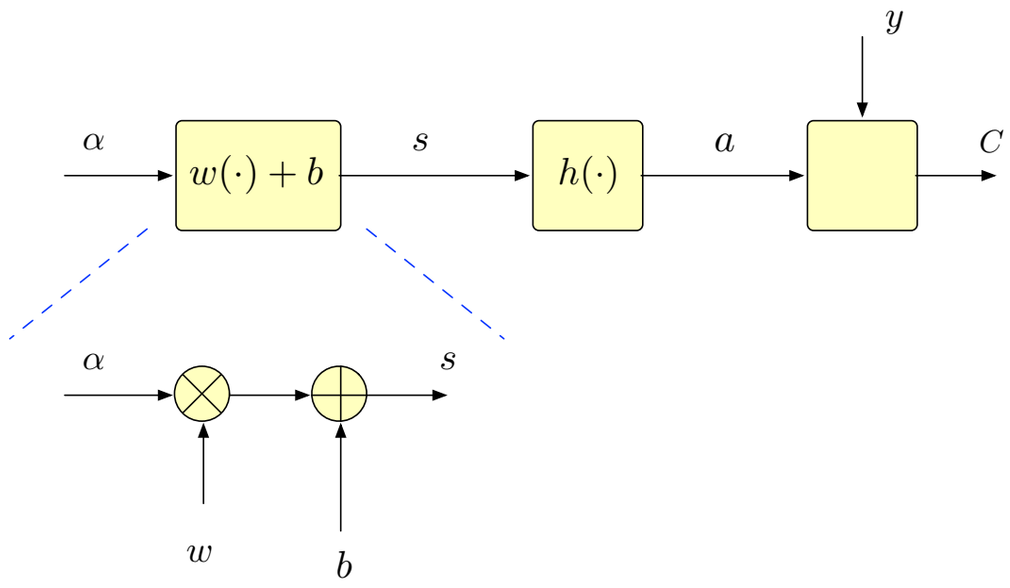
$$\dot{C}(a) = \frac{\partial C}{\partial a}$$

$$\dot{h}(s) = \frac{dh(s)}{ds}$$

$$\frac{\partial C}{\partial w} = \dot{C}(a)\dot{h}(s)\alpha$$

$$\frac{\partial C}{\partial b} = \dot{C}(a)\dot{h}(s)$$

MLP-Backprop: scalar example



$$\frac{\partial C}{\partial s} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial s}$$

$$= \frac{\partial C}{\partial a}\frac{\partial h(s)}{\partial s}$$

$$= \dot{C}(a)\dot{h}(s)$$

$$\frac{\partial C}{\partial w} = \dot{C}(a)\dot{h}(s)\alpha$$

$$\frac{\partial C}{\partial b} = \dot{C}(a)\dot{h}(s)$$

example:

$$C(y,a) = \frac{1}{2}(y-a)^2 \qquad h(s) = \sigma(s) = \frac{1}{1+e^{-s}}$$

$$\dot{C}(a) = \frac{\partial C}{\partial a} = -(y-a) \qquad \dot{h}(s) = \frac{dh(s)}{ds} = \sigma(s)(1-\sigma(s))$$

$$w \leftarrow w + \eta a(1-a)(y-a)\alpha$$

$$b \leftarrow b + \eta a(1-a)(y-a)$$

MLP-Backprop: scalar example

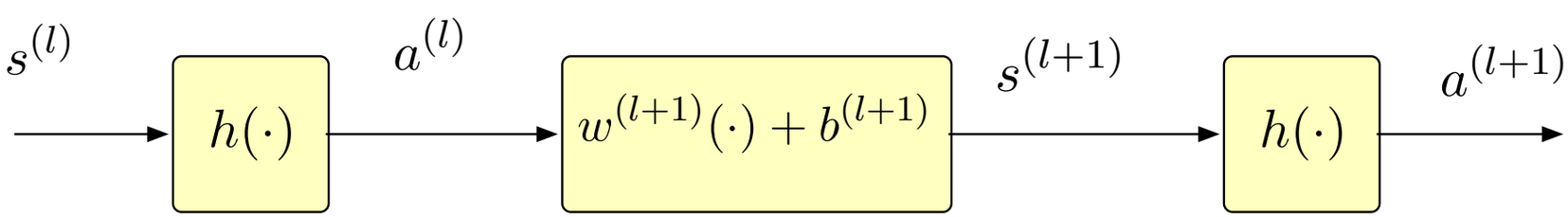
we know how to compute: $\dfrac{\partial C}{\partial s}$

and how to convert to:

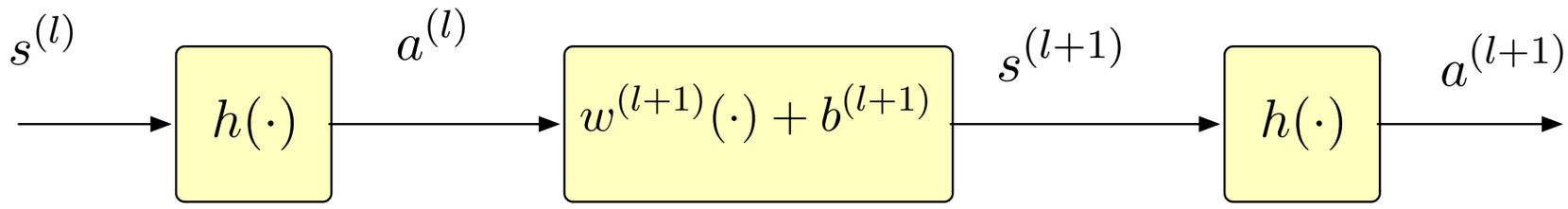
$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial s}\frac{\partial s}{\partial w} = \alpha\frac{\partial C}{\partial s}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial s}\frac{\partial s}{\partial b} = \frac{\partial C}{\partial s}$$

now consider:



MLP-Backprop: scalar example



Goal: get a recursion:

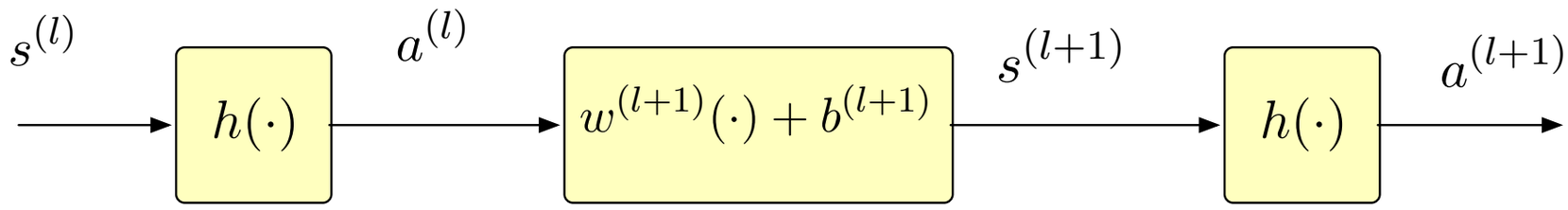
$$\frac{\partial C}{\partial s^{(l)}} \leftarrow \frac{\partial C}{\partial s^{(l+1)}}$$

From previous result:

$$\frac{\partial C}{\partial w^{(l+1)}} = \frac{\partial C}{\partial s^{(l+1)}} \frac{\partial s^{(l+1)}}{\partial w^{(l+1)}} = \frac{\partial C}{\partial s^{(l+1)}} a^{(l)}$$

$$\frac{\partial C}{\partial b^{(l+1)}} = \frac{\partial C}{\partial s^{(l+1)}} \frac{\partial s^{(l+1)}}{\partial b^{(l+1)}} = \frac{\partial C}{\partial s^{(l+1)}}$$

MLP-Backprop: scalar example



Shorthand:

$$\delta^{(l)} \triangleq \frac{\partial C}{\partial s^{(l)}}$$

$$\frac{\partial C}{\partial s^{(l)}} \leftarrow \frac{\partial C}{\partial s^{(l+1)}}$$

$$\boxed{\delta^{(l)} \leftarrow \delta^{(l+1)}}$$

$$\frac{\partial C}{\partial w^{(l)}} = \frac{\partial C}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial w^{(l)}} = \frac{\partial C}{\partial s^{(l)}} a^{(l-1)}$$

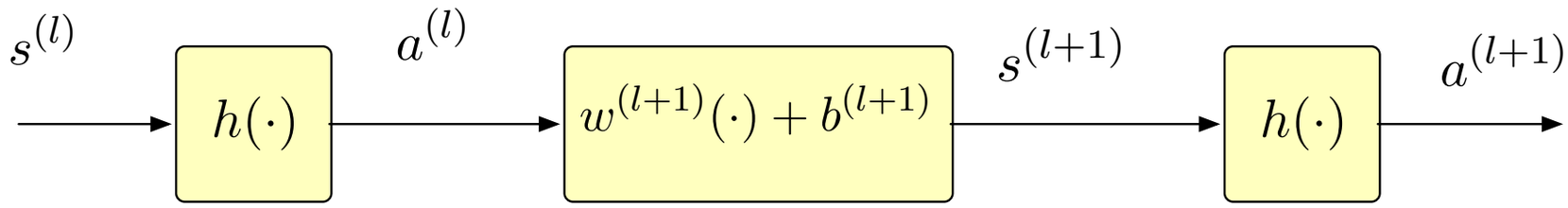
$$\frac{\partial C}{\partial w^{(l)}} = \delta^{(l)} a^{(l-1)}$$

$$\frac{\partial C}{\partial b^{(l)}} = \frac{\partial C}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial b^{(l)}} = \frac{\partial C}{\partial s^{(l)}}$$

$$\frac{\partial C}{\partial b^{(l)}} = \delta^{(l)}$$

$$\boxed{\begin{aligned} w^{(l)} &\leftarrow w^{(l)} - \eta \delta^{(l)} a^{(l-1)} \\ b^{(l)} &\leftarrow b^{(l)} - \eta \delta^{(l)} \end{aligned}}$$

MLP-Backprop: scalar example



$$\delta^{(l)} = \frac{\partial C}{\partial s^{(l)}} = \frac{\partial C}{\partial s^{(l+1)}} \frac{\partial s^{(l+1)}}{\partial s^{(l)}}$$

$$= \delta^{(l+1)} \frac{\partial s^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial s^{(l)}}$$

$$= \delta^{(l+1)} w^{(l+1)} \dot{h}(s^{(l)})$$

$$= \dot{h}(s^{(l)}) w^{(l+1)} \delta^{(l+1)}$$

$$= \dot{a}^{(l)} w^{(l+1)} \delta^{(l+1)}$$

$$\delta^{(l)} \leftarrow \delta^{(l+1)}$$

Shorthand:

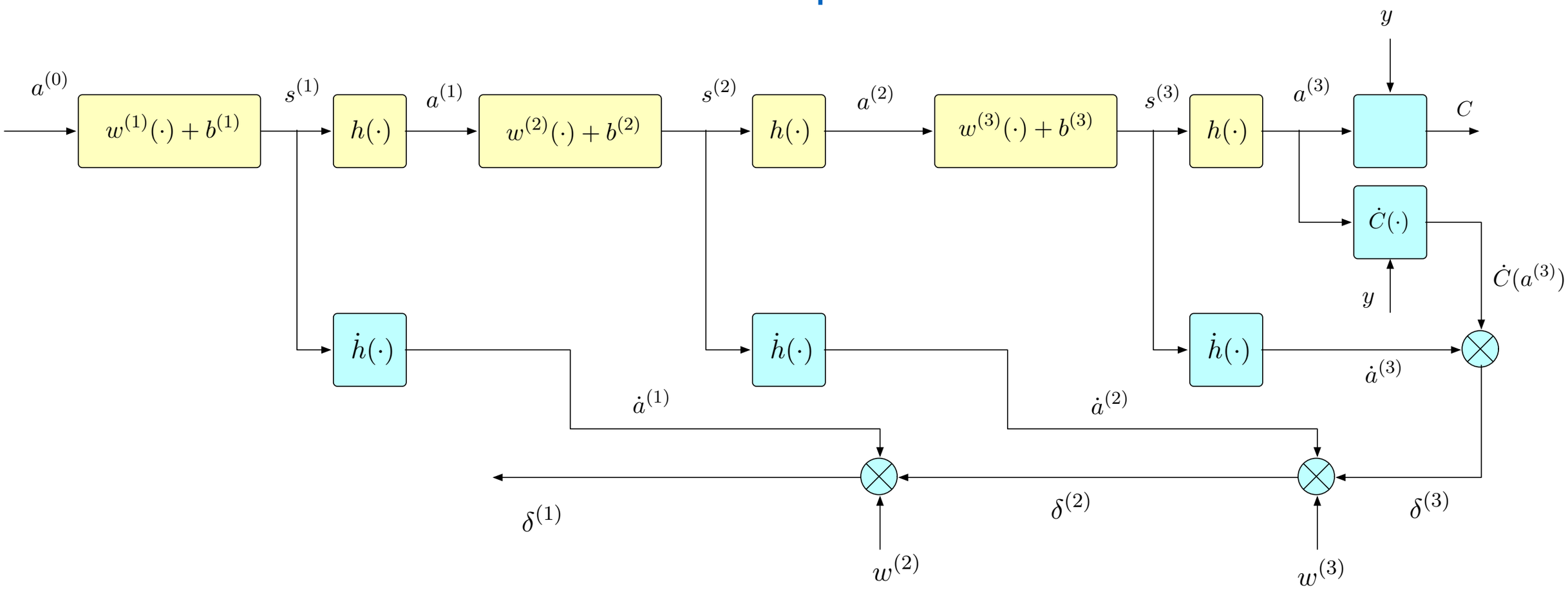
$$\dot{a}^{(l)} \triangleq \dot{h}(s^{(l)})$$

$$w^{(l)} \leftarrow w^{(l)} - \eta \delta^{(l)} a^{(l-1)}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \delta^{(l)}$$

MLP-Backprop: scalar example

L=3 example



Note: you update the weights and biases only after all the deltas are computed

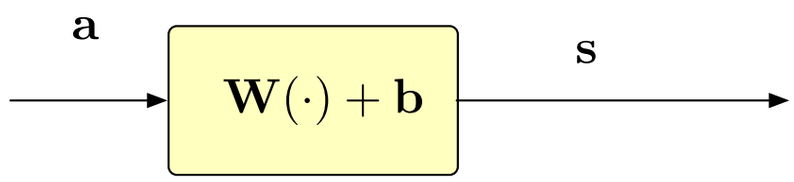
$$w^{(l)} \leftarrow w^{(l)} - \eta \delta^{(l)} a^{(l-1)}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \delta^{(l)}$$

MLP-Backprop

now we just need to handle the vector case…

(let's just repeat what we did)



$$\mathbf{s} = \mathbf{W}\mathbf{a} + \mathbf{b} \qquad \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \qquad s_n = \left(\sum_m w_{nm} a_m \right) + b_n$$

Suppose we know: $\nabla_{\mathbf{s}} C = \boldsymbol{\delta}$

How to convert to: $\dfrac{\partial C}{\partial w_{ij}} \qquad \dfrac{\partial C}{\partial b_i}$

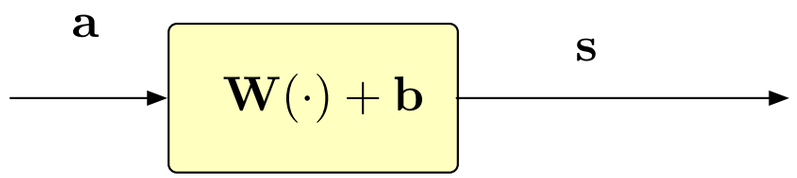
MLP-Backprop

we'll use this repeatedly…

$$\frac{\partial C(s_0(w_0,w_1),s_1(w_0,w_1))}{\partial w_0} = \frac{\partial C(s_0(w_0,w_1),s_1(w_0,w_1))}{\partial s_0}\frac{\partial s_0}{\partial w_0} + \frac{\partial C(s_0(w_0,w_1),s_1(w_0,w_1))}{\partial s_1}\frac{\partial s_1}{\partial w_0}$$

$$\frac{\partial C}{\partial w_0} = \frac{\partial C}{\partial s_0}\frac{\partial s_0}{\partial w_0} + \frac{\partial C}{\partial s_1}\frac{\partial s_1}{\partial w_0}$$

MLP-Backprop



$$\mathbf{s} = \mathbf{W}\mathbf{a} + \mathbf{b} \qquad \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \qquad s_n = \left(\sum_m w_{nm} a_m \right) + b_n$$

Suppose we know: $\nabla_{\mathbf{s}} C = \boldsymbol{\delta}$

$$\frac{\partial C}{\partial w_{ij}} = \sum_m \frac{\partial C}{\partial s_m} \frac{\partial s_m}{\partial w_{ij}}$$

$$s_m = \left(\sum_n w_{mn} a_n \right) + b_i$$

$$\frac{\partial s_m}{\partial w_{ij}} = \begin{cases} 0 & m \neq i \\ a_j & m = i \end{cases}$$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial s_i} a_j \qquad \frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial s_i}$$

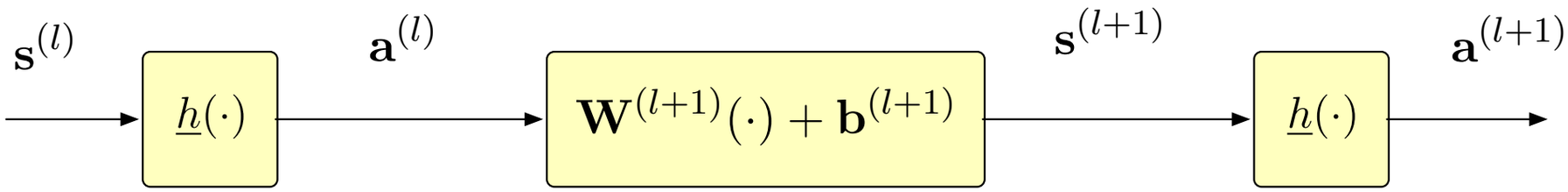
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \boldsymbol{\delta} \mathbf{a}^{\mathrm{t}}$$
$$\mathbf{b} \leftarrow \mathbf{b} - \eta \boldsymbol{\delta}$$

MLP-Backprop: delta recursion

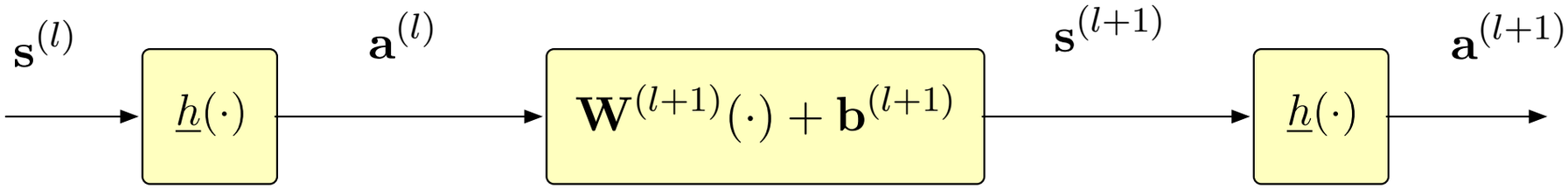
we know how to compute: $\nabla_{\mathbf{s}} C = \boldsymbol{\delta}$

and how to convert to: $\dfrac{\partial C}{\partial w_{ij}} = \dfrac{\partial C}{\partial s_i} a_j$ $\qquad$ $\dfrac{\partial C}{\partial b_i} = \dfrac{\partial C}{\partial s_i}$

now consider:



MLP-Backprop: delta recursion



Goal: get a recursion:

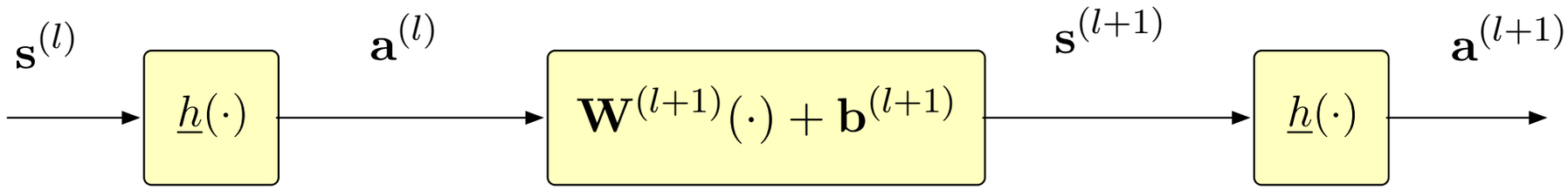
$$\nabla_{\mathbf{s}^{(l)}} C \leftarrow \nabla_{\mathbf{s}^{(l+1)}} C$$

From previous result:

$$\frac{\partial C}{\partial w_{ij}^{(l+1)}} = \frac{\partial C}{\partial s_i^{(l+1)}} a_j^{(l)}$$

$$\frac{\partial C}{\partial b_i^{(l+1)}} = \frac{\partial C}{\partial s_i^{(l+1)}}$$

MLP-Backprop: delta recursion



$$\frac{\partial C}{\partial s^{(l)}} \leftarrow \frac{\partial C}{\partial s^{(l+1)}}$$

Shorthand:

$$\nabla_{\mathbf{s}^{(l)}} C \triangleq \boldsymbol{\delta}^{(l)}$$

$$\frac{\partial C}{\partial s_i^{(l)}} = \delta_i^{(l)}$$

$$= \sum_m \frac{\partial C}{\partial s_m^{(l+1)}} \frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}}$$

$$= \sum_m \delta_m^{(l+1)} \frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}}$$

$$\boldsymbol{\delta}^{(l)} \leftarrow \boldsymbol{\delta}^{(l+1)}$$

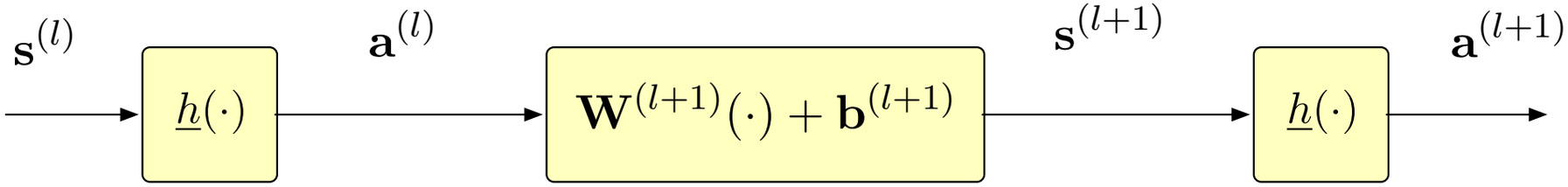
$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

$$\frac{\partial C}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \boldsymbol{\delta}^{(l)} \left[\mathbf{a}^{(l-1)}\right]^{\mathrm{t}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \boldsymbol{\delta}^{(l)}$$

MLP-Backprop: delta recursion



$$\frac{\partial C}{\partial s_i^{(l)}} = \delta_i^{(l)}$$

$$= \sum_m \frac{\partial C}{\partial s_m^{(l+1)}} \frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}}$$

$$= \sum_m \delta_m^{(l+1)} \frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}}$$

$$\frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}} = \sum_n \frac{\partial s_m^{(l+1)}}{\partial a_n^{(l)}} \frac{\partial a_n^{(l)}}{\partial s_i^{(l)}}$$

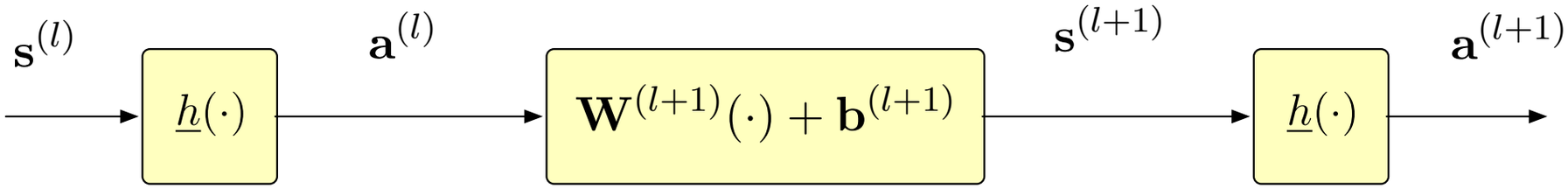
$$\frac{\partial a_n^{(l)}}{\partial s_i^{(l)}} = \begin{cases} \dot{h}(s_i^{(l)}) & i = n \\ 0 & i \neq n \end{cases}$$

assumes $\underline{h}(.)$ is component-wise vector function
i.e., vectorized version of h(.)

$$\frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}} = \frac{\partial s_m^{(l+1)}}{\partial a_i^{(l)}} \dot{h}(s_i^{(l)})$$

$$= w_{mi} \dot{h}(s_i^{(l)})$$

MLP-Backprop: delta recursion



$$\delta_i^{(l)} = \sum_m \delta_m^{(l+1)} \frac{\partial s_m^{(l+1)}}{\partial s_i^{(l)}}$$

$$= \sum_m \delta_m^{(l+1)} w_{mi} \dot{h}(s_i^{(l)})$$

$$= \left(\sum_m \delta_m^{(l+1)} w_{mi} \right) \dot{h}(s_i^{(l)}$$

$$= \left(\sum_m \delta_m^{(l+1)} w_{mi} \right) \dot{a}_i^{(l)}$$

$$= \left[\left(\mathbf{W}^{(l+1)} \right)^{\mathrm{t}} \boldsymbol{\delta}^{(l+1)} \right]_i \dot{a}_i^{(l)}$$

$$\boldsymbol{\delta}^{(l)} = \dot{\mathbf{a}}^{(l)} \odot \left[\left(\mathbf{W}^{(l+1)} \right)^{\mathrm{t}} \boldsymbol{\delta}^{(l+1)} \right]$$

$$\dot{a}_i^{(l)} = \dot{h}(s_i^{(l)})$$

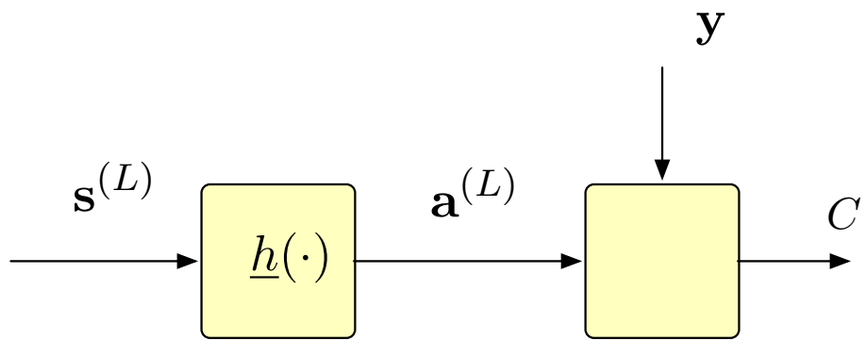
shorthand:

$$\dot{\mathbf{a}}^{(l)} = \underline{\dot{h}}(\mathbf{s}^{(l)})$$

$$\mathbf{v} \odot \mathbf{w} = \begin{bmatrix} v_0 w_0 \\ v_1 w_1 \\ \vdots \\ v_{D-1} w_{D-1} \end{bmatrix}$$

Hadamard product: *. in matlab or * for np.arrays

MLP-Backprop: delta initialization



Assume additive costs across components (very common)

$$C_{\text{tot}}(\mathbf{y}, \mathbf{a}) = \sum_i C_i(y_i, a_i)$$

$$\frac{\partial C_{\text{tot}}}{\partial a_i} = \dot{C}_i(y_i, a_i)$$

$$\frac{\partial C_{\text{tot}}}{\partial s_i^{(L)}} = \sum_m \frac{\partial C}{\partial a_m^{(L)}} \frac{\partial a_m^{(L)}}{\partial s_i^{(L)}}$$

$$= \dot{C}_i(a_i^{(L)}) \dot{h}(s_i^{(L)})$$

$$\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{s}^{(L)}} C_{\text{tot}} = \nabla_{\mathbf{s}^{(L)}} C$$

$$= \underline{\dot{C}}\left(\mathbf{a}^{(L)}\right) \odot \underline{\dot{h}}\left(\mathbf{s}^{(L)}\right)$$

$$= \underline{\dot{C}}\left(\mathbf{a}^{(L)}\right) \odot \dot{\mathbf{a}}^{(L)}$$

MLP-Backprop: summary

$$\mathbf{a}^{(l)} = \underline{h}\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

$$\dot{\mathbf{a}}^{(l)} = \underline{\dot{h}}\left(\mathbf{W}^{(l)}\mathbf{a}_{i-1} + \mathbf{b}_i\right)$$

$$\boldsymbol{\delta}^{(L)} = \dot{\mathbf{a}}^{(L)} \odot \underline{\dot{C}}\left(\mathbf{y}, \mathbf{a}^{(l)}\right)$$

$$\boldsymbol{\delta}^{(l)} = \dot{\mathbf{a}}^{(l)} \odot \left[\left(\mathbf{W}^{(l+1)}\right)^{\mathrm{t}} \boldsymbol{\delta}^{(l+1)}\right]$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta\boldsymbol{\delta}^{(l)}\left[\mathbf{a}^{(l-1)}\right]^{\mathrm{t}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\boldsymbol{\delta}^{(l)}$$

specialized to vectorized output activation and additive cost

MLP-Backprop: summary

$$\mathbf{a}_l = \mathrm{act}(\mathbf{W}_l \mathbf{a}_{l-1} + \mathbf{b}_l) \qquad \text{(activations)}$$

$$\dot{\mathbf{a}}_l = \dot{\mathrm{act}}(\mathbf{W}_l \mathbf{a}_{l-1} + \mathbf{b}_l) \qquad \text{(derivative activations)}$$

$$\boldsymbol{\delta}_L = \dot{\mathbf{a}}_L \odot \dot{\mathrm{cost}}(\mathbf{y}, \mathbf{a}_L) \qquad \text{(delta initialization)}$$

$$\boldsymbol{\delta}_l = \dot{\mathbf{a}}_l \odot \left[\mathbf{W}_{l+1}^{\mathrm{t}} \boldsymbol{\delta}_{l+1}\right] \qquad \text{(delta recursion)}$$

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \boldsymbol{\delta}_l \mathbf{a}_{l-1}^{\mathrm{t}} \qquad \text{(weight SGD update)}$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \boldsymbol{\delta}_l \qquad \text{(bias SGD update)}$$

specialized to vectorized output activation and additive cost

Using Batches

For each data sample: $(\mathbf{x}[n], \mathbf{y}[n])$

Do FF and BP to compute activations, a-dots, and deltas

$$\mathbf{a}_l[n] = \text{act}(\mathbf{W}_l \mathbf{a}_{l-1}[n] + \mathbf{b}_l) \qquad \text{(activations)}$$

$$\dot{\mathbf{a}}_l[n] = \dot{\text{act}}(\mathbf{W}_l \mathbf{a}_{l-1}[n] + \mathbf{b}_l) \qquad \text{(derivative activations)}$$

$$\boldsymbol{\delta}_L[n] = \dot{\mathbf{a}}_L[n] \odot \dot{\text{cost}}(\mathbf{y}[n], \mathbf{a}_L[n]) \qquad \text{(delta initialization)}$$

$$\boldsymbol{\delta}_l[n] = \dot{\mathbf{a}}_l[n] \odot \left[\mathbf{W}_{l+1}^{\text{t}} \boldsymbol{\delta}_{l+1}[n]\right] \qquad \text{(delta recursion)}$$

Do one SGD update after finishing the batch

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \frac{1}{B} \sum_{n=0}^{B-1} \boldsymbol{\delta}_l[n] \mathbf{a}_{l-1}^{\text{t}}[n]$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \frac{1}{B} \sum_{n=0}^{B-1} \boldsymbol{\delta}_l[n]$$

Effect of Regularizers

For typical weight-penalty regularizers (e.g., L1 and L2), these are direct functions of the weights

Example:

$$C_{\text{reg}} = C + \lambda \sum_l \|\mathbf{W}^{(l)}\|^2 = C + \lambda \sum_l \sum_{i,j} \left[\mathbf{W}_{i,j}^{(l)}\right]^2$$

$$\frac{\partial C_{\text{reg}}}{\partial w_{i,j}^{(l)}} = \frac{\partial C}{\partial w_{i,j}^{(l)}} + 2\lambda w_{i,j}^{(l)}$$

Minor change to gradient update:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \left(\boldsymbol{\delta}^{(l)} \left[\mathbf{a}^{(l-1)}\right]^{\text{t}} + 2\lambda \mathbf{W}^{(l)} \right)$$

Vector Calculus

Always remember this is just the chain rule…

But it is tedious…

So there are various conventions of doing derivatives wrt vectors and matrices

e.g., you have seen the Jacobian matrix d**y**/d**x**

There are several conventions for keeping track of all the partial derivatives and storing them in tables (vectors, matrices, tensors)

Sourya has written a handout on the "numerator" convention and also has a general development of BP using this convention

Compact Tensor/Matrix/Vector Calc. Notation

recall how we started with the vector version of BP…

$$\frac{\partial C}{\partial w_0} = \frac{\partial C}{\partial s_0}\frac{\partial s_0}{\partial w_0} + \frac{\partial C}{\partial s_0}\frac{\partial s_1}{\partial w_0}$$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial s_0}\frac{\partial s_0}{\partial w_1} + \frac{\partial C}{\partial s_1}\frac{\partial s_1}{\partial w_1}$$

this is just book-keeping convention for chain rule

This suggests a matrix form:

$$\nabla_{\mathbf{w}} C = \begin{bmatrix} \frac{\partial C}{\partial w_0} \\ \frac{\partial C}{\partial w_1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial s_0}{\partial w_0} & \frac{\partial s_1}{\partial w_0} \\ \frac{\partial s_0}{\partial w_1} & \frac{\partial s_1}{\partial w_1} \end{bmatrix} \begin{bmatrix} \frac{\partial C}{\partial s_0} \\ \frac{\partial C}{\partial s_1} \end{bmatrix}$$

$$\nabla_{\mathbf{w}} C = \frac{d\mathbf{s}}{d\mathbf{w}} \nabla_{\mathbf{s}} C$$

denominator convention

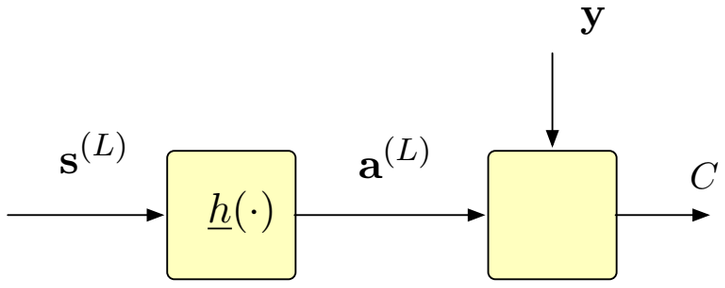
$$[\nabla_{\mathbf{w}} C]^{\mathrm{t}} = [\nabla_{\mathbf{w}} C]^{\mathrm{t}} \begin{bmatrix} \frac{\partial s_0}{\partial w_0} & \frac{\partial s_0}{\partial w_1} \\ \frac{\partial s_1}{\partial w_0} & \frac{\partial s_1}{\partial w_1} \end{bmatrix}$$

$$[\nabla_{\mathbf{w}} C]^{\mathrm{t}} = [\nabla_{\mathbf{s}} C]^{\mathrm{t}} \frac{d\mathbf{s}}{d\mathbf{w}}$$

numerator convention

MLP-Backprop: Non-vectorized output activation

recall, assuming the activation is a vectorized scalar function:



$$\frac{\partial C_{\text{tot}}}{\partial s_i^{(L)}} = \sum_m \frac{\partial C}{\partial a_m^{(L)}} \frac{\partial a_m^{(L)}}{\partial s_i^{(L)}}$$

$$= \dot{C}_i(a_i^{(L)}) \dot{h}(s_i^{(L)})$$

$$\boldsymbol{\delta}^{(L)} = \underline{\dot{C}}\left(\mathbf{a}^{(L)}\right) \odot \dot{\mathbf{a}}^{(L)}$$

if this final output activation is not a vectorized scalar function:

$$\frac{\partial C_{\text{tot}}}{\partial s_i^{(L)}} = \sum_m \frac{\partial C}{\partial a_m^{(L)}} \frac{\partial a_m^{(L)}}{\partial s_i^{(L)}}$$

$$= \sum_m \frac{\partial C}{\partial a_m^{(L)}} \frac{\partial h_m(\mathbf{s}^{(L)})}{\partial s_i^{(L)}}$$

$$= \sum_m \dot{A}_{i,m}^{(L)} \frac{\partial C}{\partial a_m^{(L)}}$$

$$\dot{A}_{i,m}^{(L)} = \left[\dot{\mathbf{A}}^{(L)}\right]_{i,m} = \frac{\partial h_m(\mathbf{s}^{(L)})}{\partial s_i^{(L)}}$$

(denominator convention)

$$\boldsymbol{\delta}^{(L)} = \dot{\mathbf{A}}^{(L)} \underline{\dot{C}}\left(\mathbf{a}^{(L)}\right)$$

MLP-Backprop: Non-vectorized output activation

$$\mathbf{a}^{(l)} = \underline{h}\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

$$\dot{\mathbf{a}}^{(l)} = \underline{\dot{h}}\left(\mathbf{W}^{(l)}\mathbf{a}_{i-1} + \mathbf{b}_i\right)$$

$$\boldsymbol{\delta}^{(L)} = \dot{\mathbf{A}}^{(L)}\underline{\dot{C}}\left(\mathbf{a}^{(L)}\right)$$

$$\boldsymbol{\delta}^{(l)} = \dot{\mathbf{a}}^{(l)} \odot \left[\left(\mathbf{W}^{(l+1)}\right)^{\mathrm{t}}\boldsymbol{\delta}^{(l+1)}\right]$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta\boldsymbol{\delta}^{(l)}\left[\mathbf{a}^{(l-1)}\right]^{\mathrm{t}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\boldsymbol{\delta}^{(l)}$$

Note: notation in Sourya's notes:

$$\mathbf{H}'^{(L)^T} = \dot{\mathbf{A}}^{(L)}$$

$$\mathbf{H}'^{(l)^T} = \mathbf{diag}\left[\dot{\mathbf{a}}^{(l)}\right]$$

similar change could be made if other layers had general activations
(not seen in practice)

LMS vs BP-MLP…

When training an MLP, it is assumed that the target mapping is fixed

— i.e., the probability distributions are not a function of n

$$p_{\text{data}}(\mathbf{y}_n|\mathbf{x}_n) \approx p_{\text{model}}(\mathbf{y}_n|\mathbf{x}_n;\Theta)$$

if you train a MLP using BP and these data statistics vary with n (i.e., non-stationary) you will get junk!

this of the time-varying LMS example

Can a MLP be used in a non-stationary environment — as a nonlinear adaptive filter?

LMS vs BP-MLP…

1. Train an MLP using representative (stationary data)

2. Use on-line learning (batch-size 1) to update the MLP with new data as it becomes available

this could be done on just the last layer if the representative model is good

In this way, an MLP can be used as a direct generalization of the LMS adaptive filter
— an **on-line (adaptive) non-linear regressor!**

MLP/Back-Prop Topics

- MLP feedforward (inference) equations
- Back-propagation derivation
  - from scalar to the matrix-vector case
  - General rules/conventions for matrix-vector calculus

(numerator convention)

- Variations on back-prop and MLP training
  - activations and their derivatives
  - regularizers, optimizers (e.g., adam) and momentum
  - dropout, batch normalization

Sourya will cover these in lecture

- Universal Approximation
- Automatic Speech Recognition (ASR) guest lecture