Training Neural Networks

Erhard Hinrichs

Seminar für Sprachwissenschaft
Eberhard-Karls Universität Tübingen

Loss function for a single training example x

1. Binomial (binary) case

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})] \quad (23)$$

Remark: (23) is the cross-entropy loss function L_{CE} for binary logistic regression; see formula (5.23) in Chapter 5.

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_c \quad \text{(where c is the correct class)} \qquad (25)$$

Remark: (25) is called the *negative log likelihood loss*. (25) is equivalent to (23) for the correct class $y_c = 1$ since the first summand in (25) equals $-\log \hat{y}_i$, and the second summand in (23) equals 0.

Loss function for a single training example x

2. Multinomial case with K classes

$$L_{CE}(\hat{\boldsymbol{y}},\boldsymbol{y}) = -\sum_{k=1}^{K} \boldsymbol{y}_k \log \hat{\boldsymbol{y}}_k = -\sum_{k=1}^{K} \mathbb{1}\{\boldsymbol{y}_k = 1\} \log \hat{\boldsymbol{y}}_k \tag{24}$$

$$L_{CE}(\hat{\boldsymbol{y}},\boldsymbol{y}) = -\log \frac{exp(\boldsymbol{z}_c)}{\sum_{j=1}^{K} exp(\boldsymbol{z}_j)} \quad \text{(where c is the correct class)} \tag{26}$$

Remarks:

1. This is the cross-entropy loss function L_{CE} for multinomial logistic regression; see formulae (5.45) to (5.47) in Chapter 5.
2. $\mathbb{1}\{\}$ is an indicator function. It is equal to 1 iff k is the correct class, and is equal to 0 for all other classes. In other words: The indicator function equals 1 only for the correct class, and the CE loss is simply the *negative log likelihood loss* of the output probability corresponding to the correct class.

Computing the Gradient

- requires the partial derivative of the loss function with respect to each parameter.
- for a network with one layer and sigmoid output (that is: a logistic regression model), the formula in (27) can be used. For the derivation of this formula, see Section 5.10.

$$\begin{aligned} \frac{\partial L_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w_j} &= -(\hat{y} - y)\boldsymbol{x}_j \\ &= \sigma(\mathbf{w} \cdot \mathbf{x} + b) - y)\boldsymbol{x}_j \end{aligned} \tag{27}$$

Computing the Gradient

- for a network with one layer and softmax output (that is: a multinominal logistic regression model), the formula in (28) can be used.

$$\begin{aligned}\frac{\partial L_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}_{k,i}} &= -(\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{x}_j \\ &= -(\boldsymbol{y}_k - p(\boldsymbol{y}_k = 1|\boldsymbol{x}))\boldsymbol{x}_j \\ &= -\left(\boldsymbol{y}_k - \frac{exp(\boldsymbol{w}_k \cdot \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} exp(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)}\right)\boldsymbol{x}_i\end{aligned} \tag{28}$$

Product and Quotient Rule

Product Rule

$$\frac{d}{dx}(f(x) * (g(x)) = \frac{d}{dx}(f(x)) * g(x) + f(x) * \frac{d}{dx}(g(x)) \tag{29}$$

Quotient Rule

$$\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{\frac{d}{dx}(f(x)) * g(x) - f(x) * \frac{d}{dx}(g(x))}{[g(x)]^2} \tag{30}$$

Chain Rule

$$\frac{d}{dx}\left[f(g(x))\right] = f'\left[g(x)\right] * g'(x) \tag{31}$$

Remarks:

- $f(u)$ is is called *the outside function*
- $g(x)$ is is called *the inside function*
- The chain rule can be stated in words as follows: $\frac{df}{dx}$ equals the product of the derivatives of the outside function $\frac{df}{du}$ and of the inside function $\frac{du}{dx}$.
- The chain rule is often helpful when taking derivatives of functions such as: $\frac{d}{dx}(5x-4)^6$, $\frac{d}{dx}\sqrt{x^2-1}$, $\frac{d}{dx}\frac{1}{x^2-4x+5}$

Chain Rule – Example

$$\frac{d}{dx}\left[f(g(x))\right] = f'\left[g(x)\right] * g'(x) \tag{32}$$

Find the derivative of $f(x) = (x^2 + 1)^3$

$$\begin{aligned} f'(x) &= 3(x^2+1)^{3-1} * 2x^{2-1} \\ &= 3((x^2+1)^2(2x) \\ &= 6x((x^2+1)^2 \end{aligned} \tag{33}$$

Derivative of Sigmoid Function

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\ &= \frac{(0)(1+e^{-x}) - (-e^{-x})(1)}{(1+e^{-x})^2} \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}}\end{aligned} \tag{34}$$

Derivative of Sigmoid Function (continued)

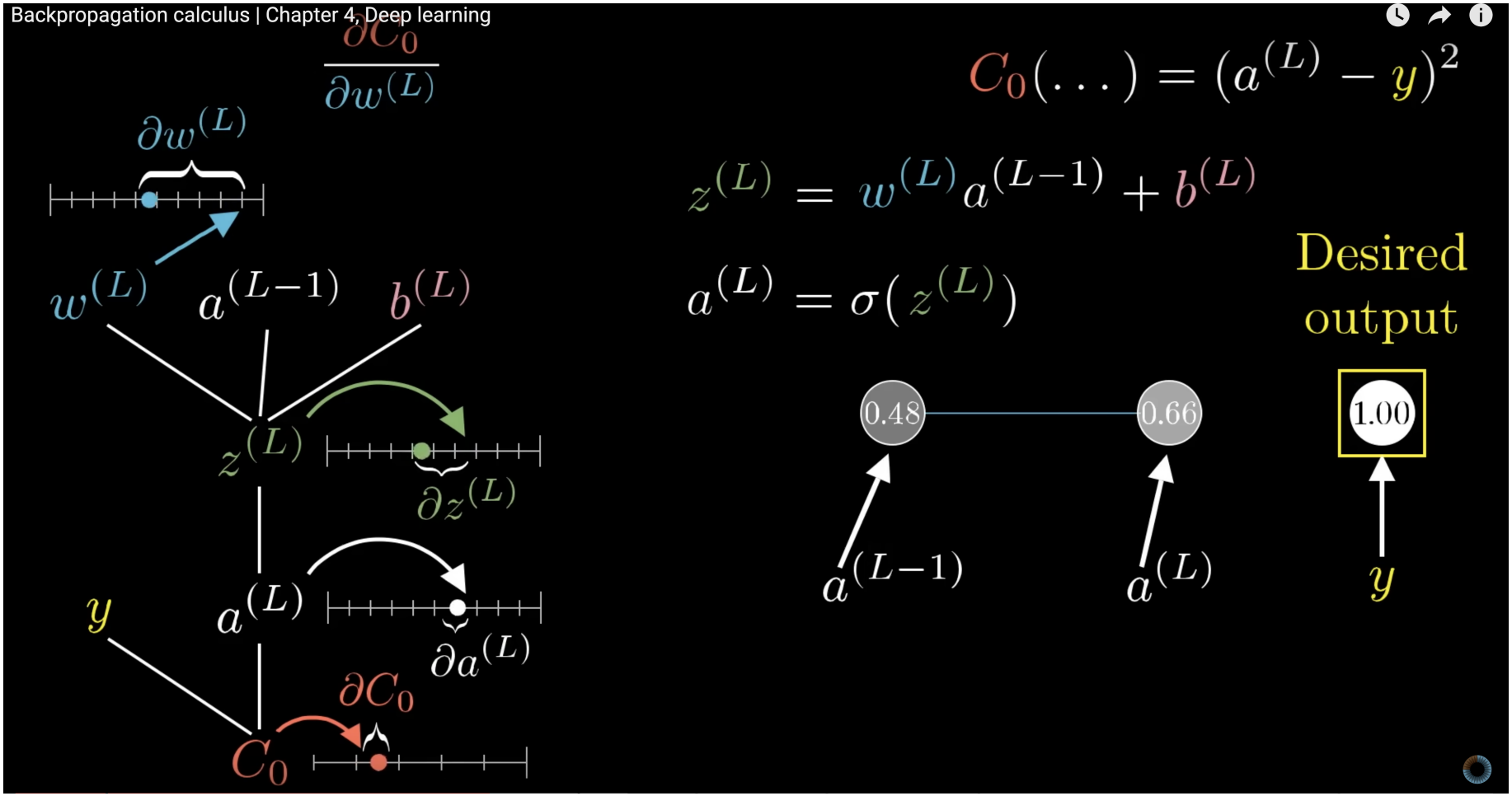
$$
\begin{aligned}
&= \frac{1}{1+e^{-x}} \frac{e^{-x}+(1-1)}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \frac{1+e^{-x}-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \left[\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right] \\
&= \frac{1}{1+e^{-x}} \left[1 - \frac{1}{1+e^{-x}} \right] \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}
\tag{35}
$$

Derivatives of ReLU and tanh

$$\frac{dReLU(z)}{dz} = \begin{cases} 0 & \text{for z} < 0 \\ 1 & \text{for z} \geq 0 \end{cases} \tag{36}$$

$$\frac{dtanh(z)}{dz} = 1 - tanh^2(z) \tag{37}$$

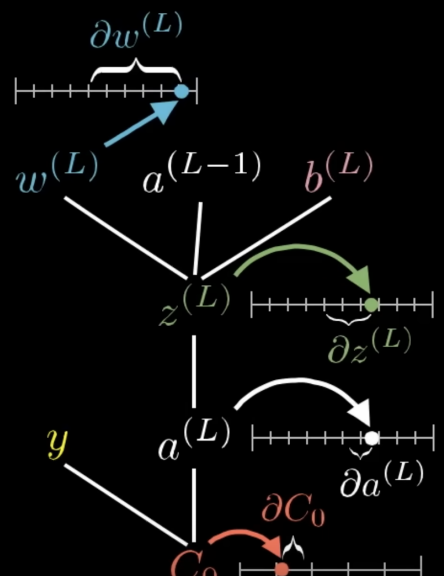
Backprop in a Simple FFN



Applying the Chain Rule

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

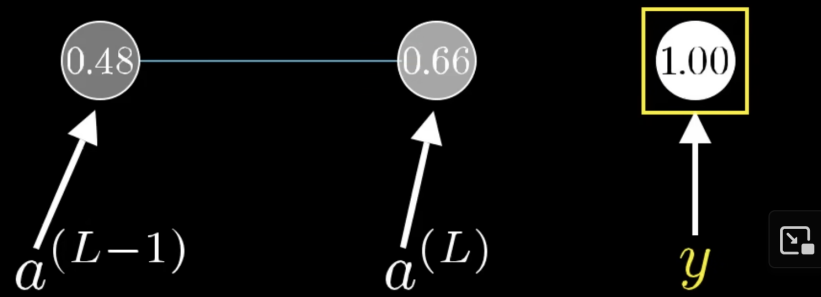
Chain rule



$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Computing Relevant Derivatives

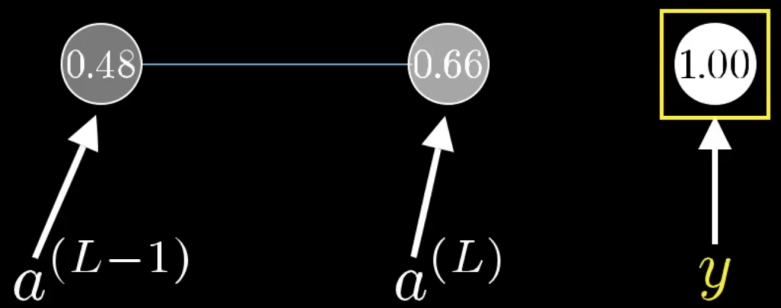
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

$$C_0 = (a^{(L)} - y)^2$$

$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$



$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

Average of All Training Examples

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$C_0 = (a^{(L)} - y)^2$$

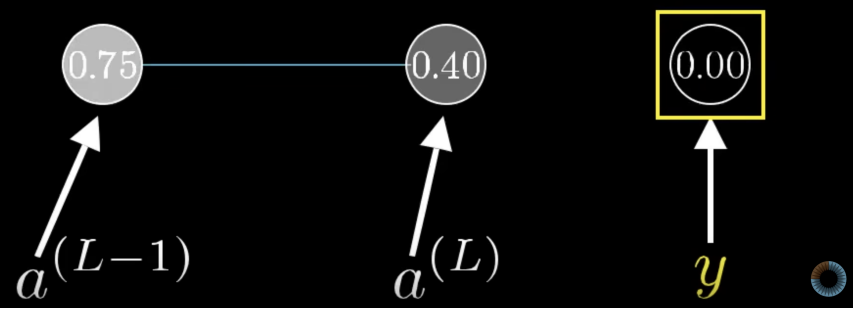
Average of all training examples

$$\underbrace{\frac{\partial C}{\partial w^{(L)}}} = \overbrace{\frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}}$$

Derivative of full cost function

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Gradient

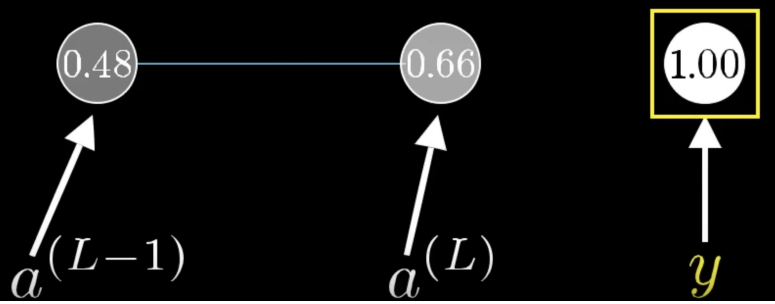
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

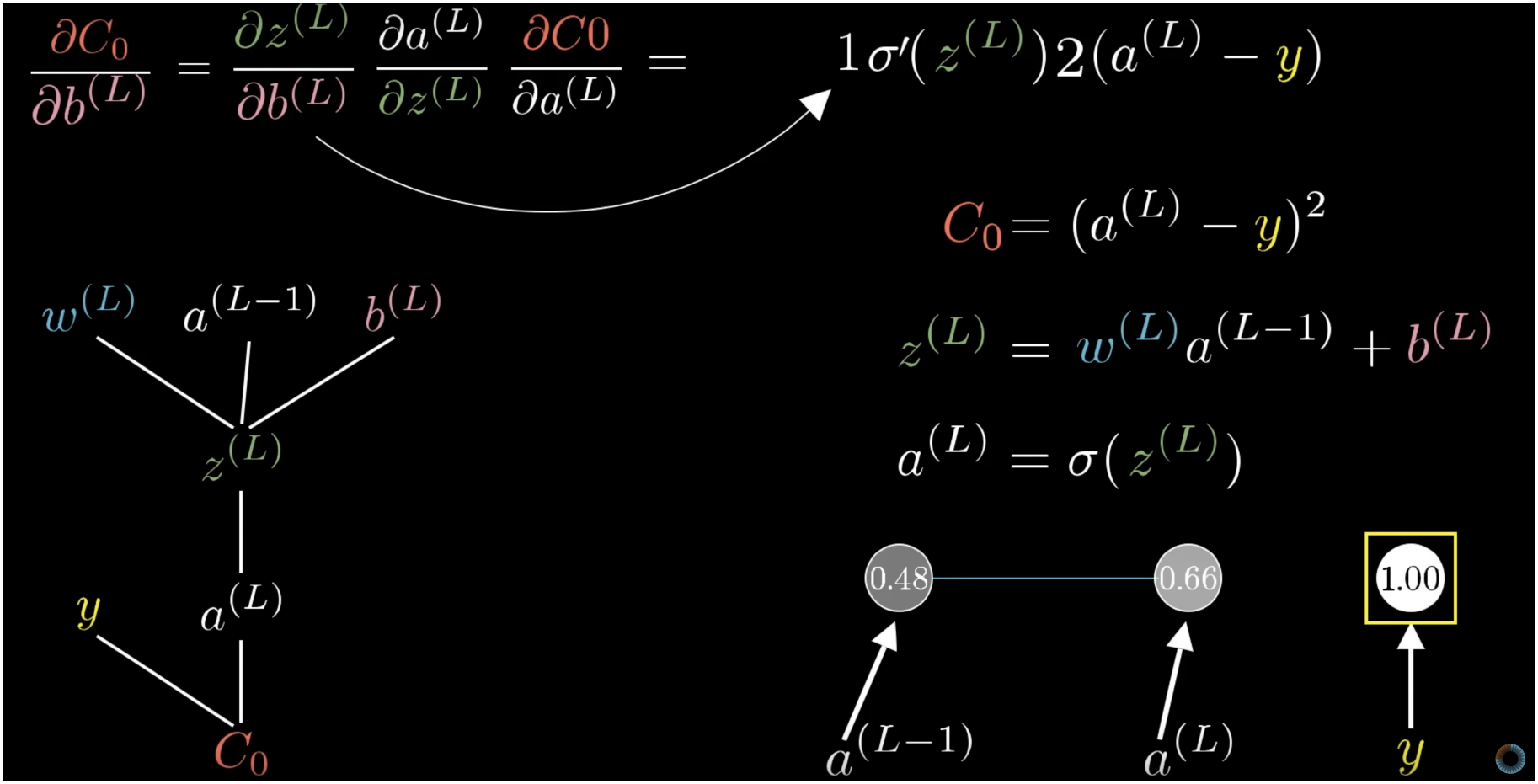
$$C_0 = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

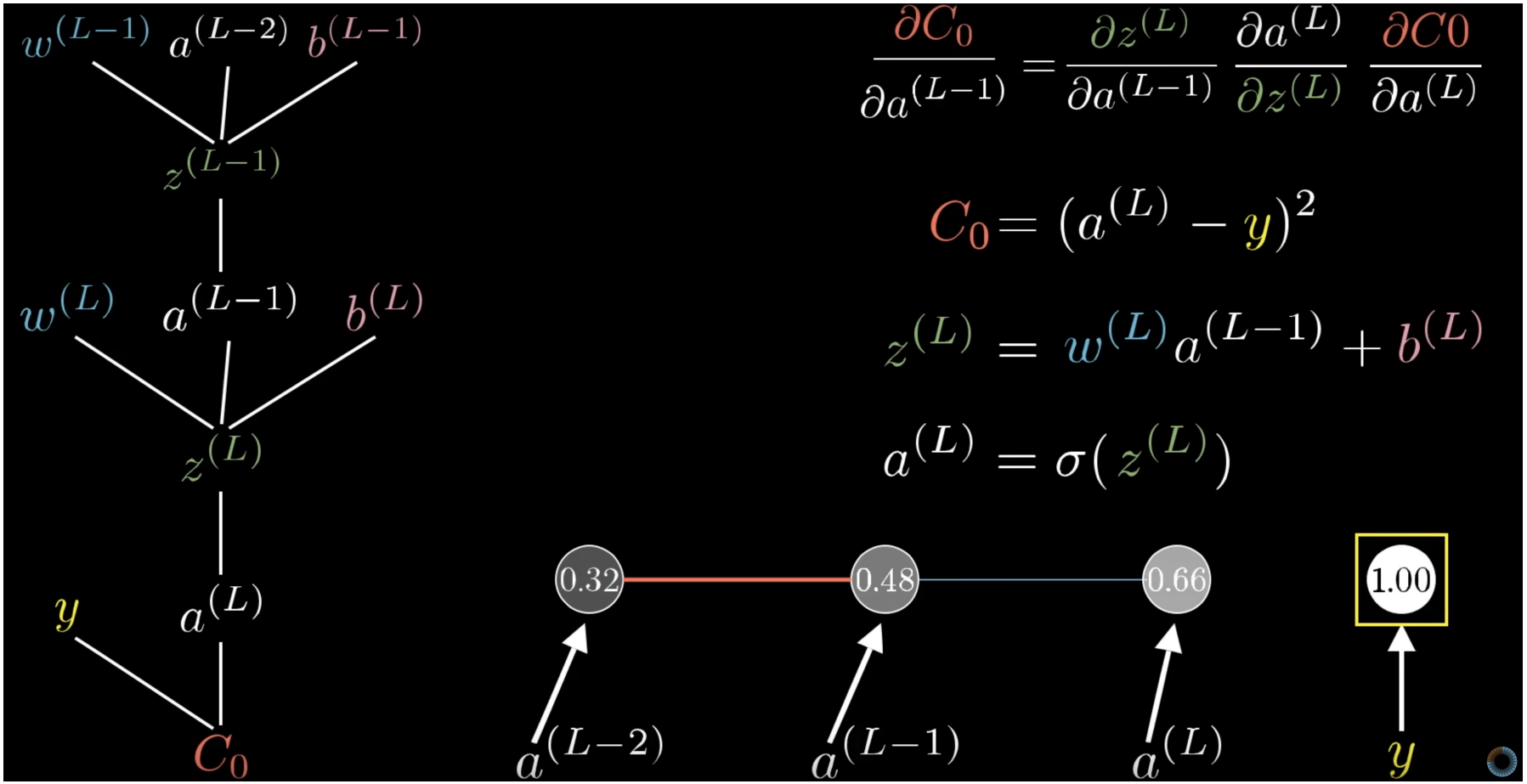
$$a^{(L)} = \sigma(z^{(L)})$$



Derivative of Bias



Adding Layers



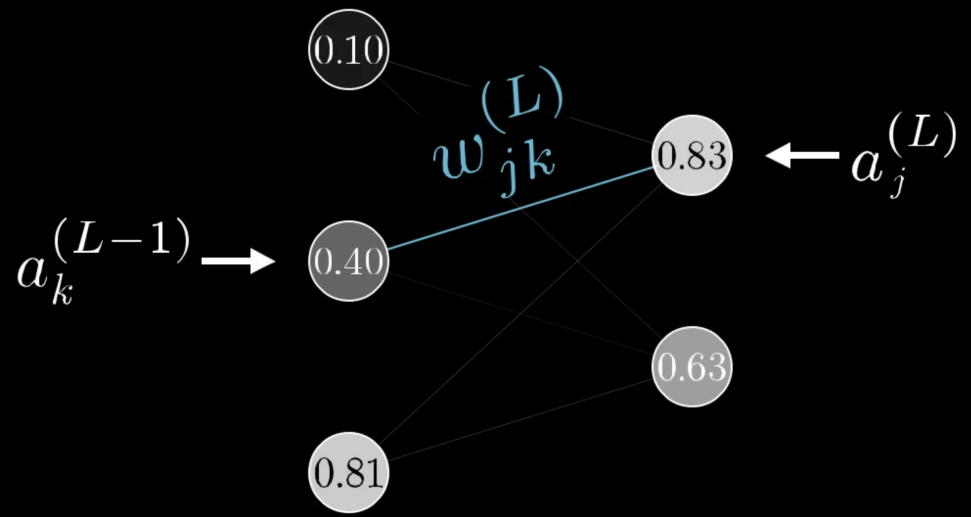
Adding Nodes Per Layer

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$



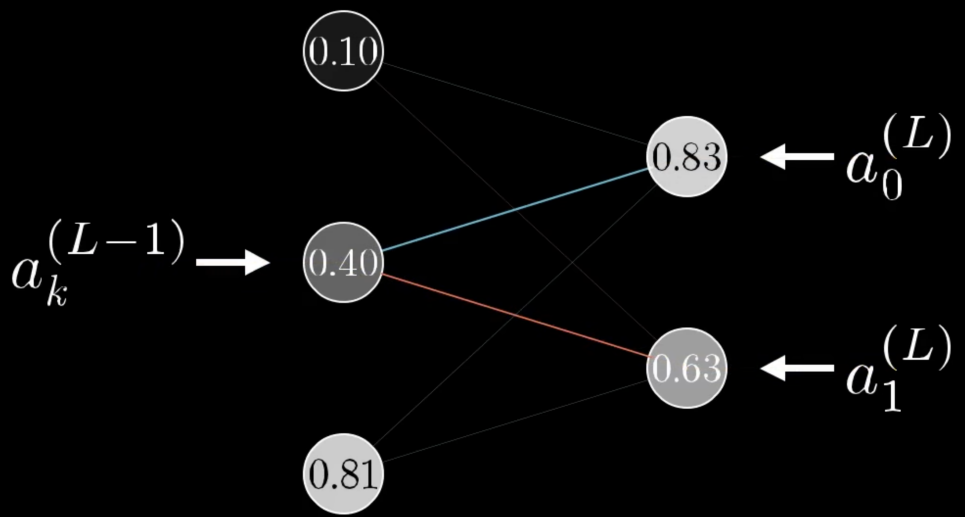
Summing Over Nodes Per Layer

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}}$$

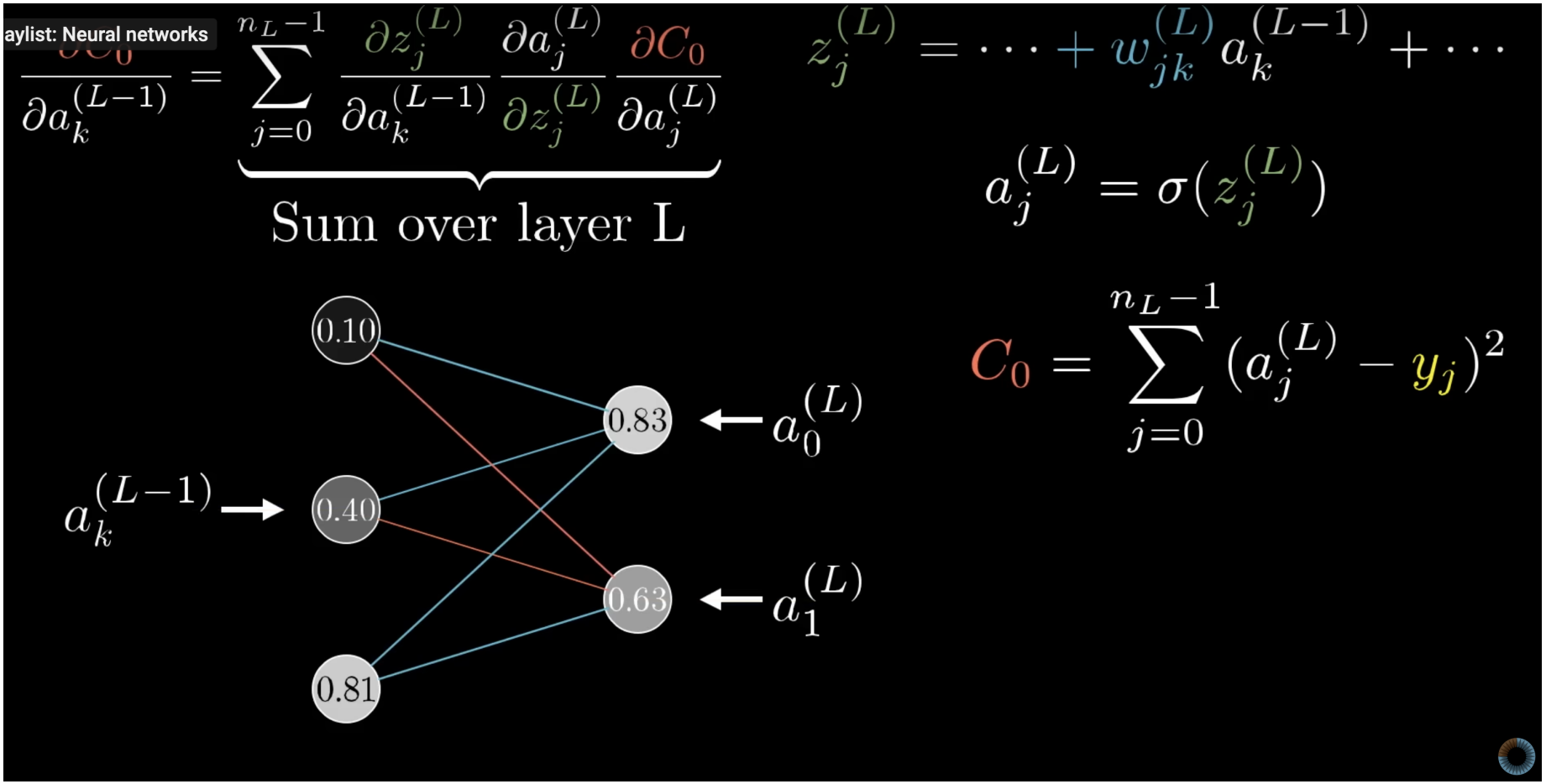
$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

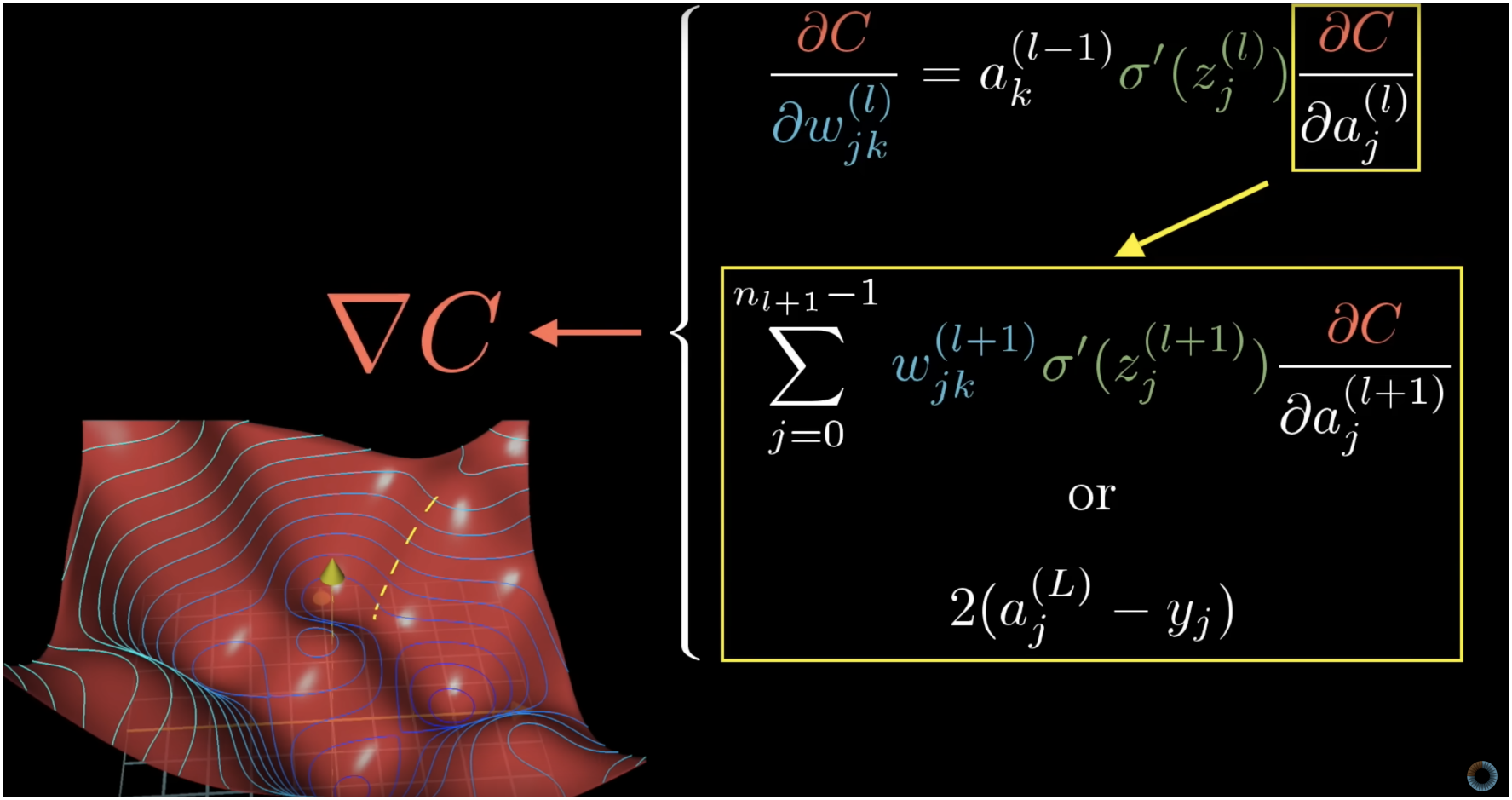
$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$



Summing Over Layers



Summary


$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$
$\nabla C \leftarrow$
$\sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}}$
or
$2(a_j^{(L)} - y_j)$

What is Gradient Descent

- an algorithm for minimizing the cost function by optimizing the model parameters
- the algorithm iteratively updates the parameters in the opposite direction of the gradient of the cost function
- the size of the step at each iteration is determined by the learning rate

Gradient Descent for Logistic Regression

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)}; \theta), y^{(i)}) \tag{38}$$

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y) \tag{39}$$

$$\nabla_\theta L(f(x;\theta), y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta), y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta), y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

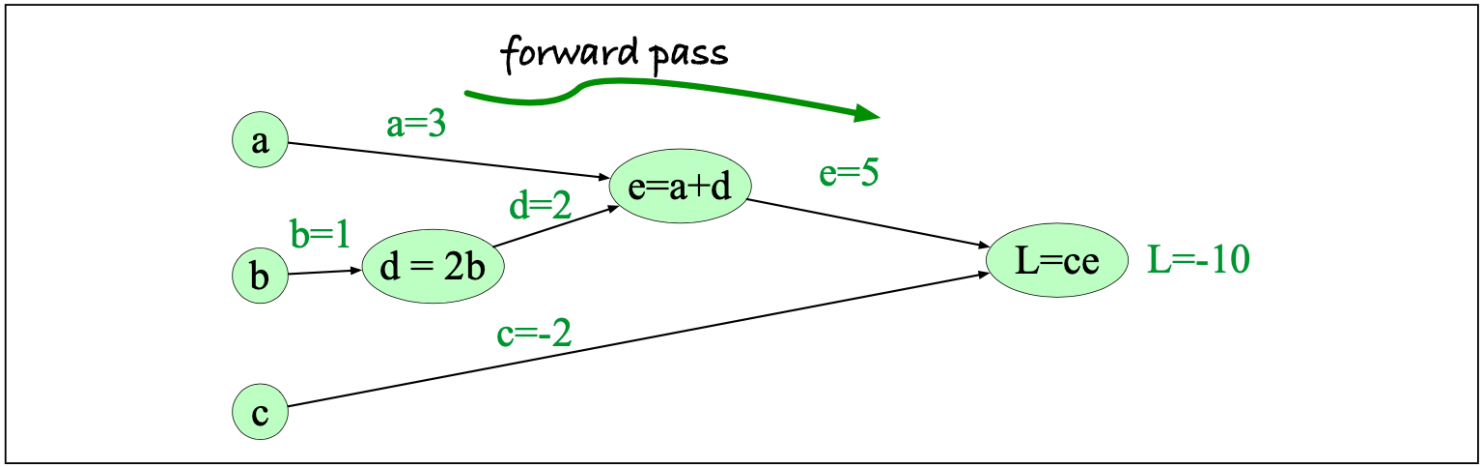
$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y) \tag{40}$$

where η is the learning rate and $\theta = [\mathbf{w}, b]$ in logistic regression

Types of Gradient Descent

- batch gradient descent: at each iteration, the gradient for all training instances is computed, the average of the gradients for all training examples is computed, and the average values are used for updating all parameters.
- Stochastic gradient descent (SGD): updates the parameters for a single (randomly selected) training example.
- mini-batch gradient descent: computes the gradient using a small subset, or mini-batch, of training examples

Computation Graph – A Simple Example



Backward differentiation on computation graphs

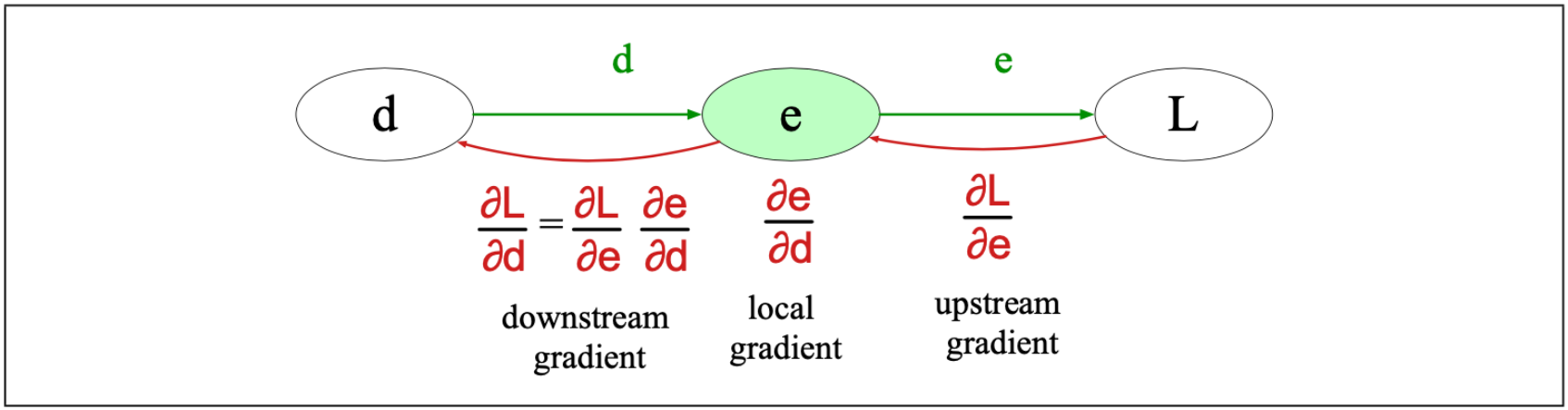
Chain rule:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \tag{41}$$

The chain rule extends to more than two functions. If computing the derivative of a composite function $f(x) = u(v(w(x)))$, the derivative of $f(x)$ is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx} \tag{42}$$

Backward differentiation on sample computation graph



Backward differentiation on sample computation graph

$$\frac{\partial L}{\partial c} = e \tag{43}$$

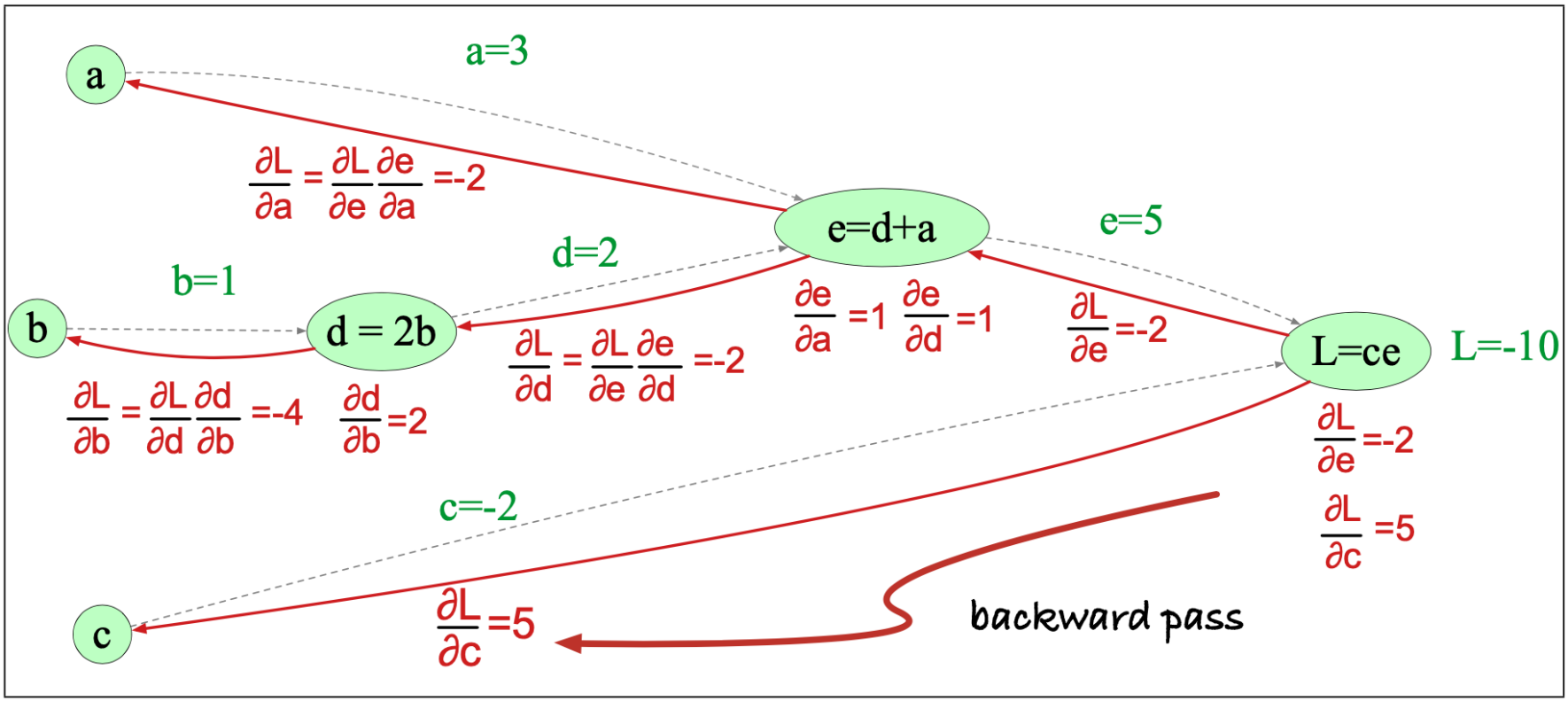
For the other two, we'll need to use the chain rule:

$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned} \tag{44}$$

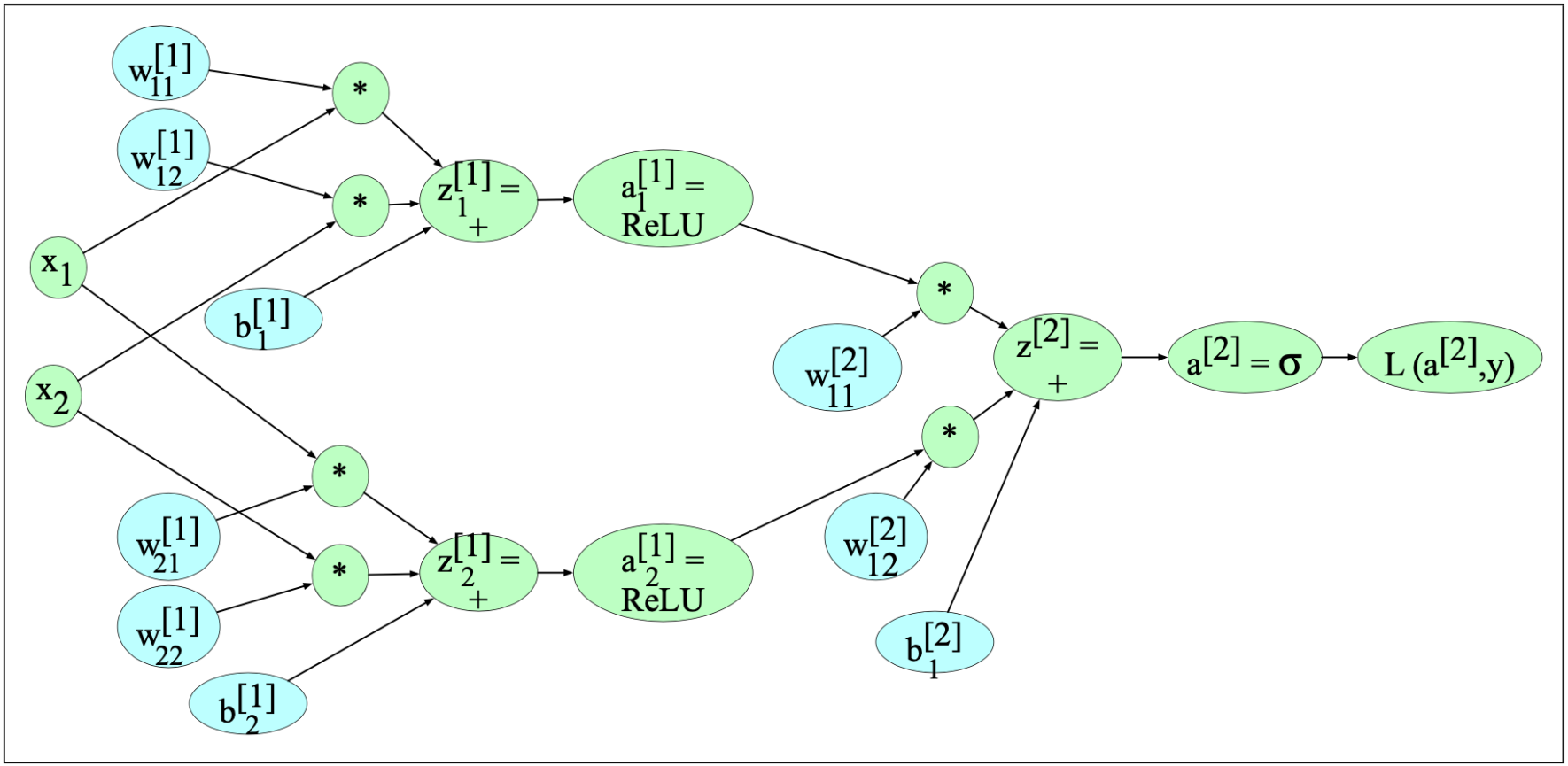
Backward differentiation on sample computation graph

$$
\begin{aligned}
L = ce: \quad & \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\
e = a + d: \quad & \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1 \\
d = 2b: \quad & \frac{\partial d}{\partial b} = 2
\end{aligned}
$$

Backward differentiation for a neural network



Backward differentiation for a neural network



Backward differentiation for a neural network

The function that the computation graph is computing is:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = ReLU(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$
$$\hat{y} = a^{[2]}$$

Backward differentiation for a neural network

The derivative of the sigmoid σ is:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \tag{45}$$

The derivative of the tanh is:

$$\frac{d\tanh(z)}{dz} = 1 - tanh^2(z) \tag{46}$$

The derivative of the ReLU is:

$$\frac{d\text{ReLU}(z)}{dz} = \begin{cases} 0 & for \quad z < 0 \\ 1 & for \quad z \geq 0 \end{cases} \tag{47}$$

Derivatives of log functions

Derivative of Common Logarithm

$$\frac{d}{dx} log_a x = \frac{1}{x * ln(a)} \tag{48}$$

Derivative of Natural Logarithm

$$\frac{d}{dx} ln\ x = \frac{1}{x} \tag{49}$$

Derivative of the CE Loss Function with Respect to z

$$L_{CE}(a^{[2]}, y) = -\left[y \ log \ a^{[2]} + (1-y) log(1-a^{[2]})\right] \tag{50}$$

$$\begin{aligned}
\frac{\partial L}{\partial a^{[2]}} &= -\left(\left(y\frac{\partial log(a^{[2]})}{\partial a^{[2]}}\right) + (1-y)\frac{\partial log(1-a^{[2]})}{\partial a^{[2]}}\right) \\
&= -\left(\left(y\frac{1}{a^{[2]}}\right) + (1-y)\frac{1}{1-a^{[2]}}(-1)\right) \qquad (51) \\
&= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right)
\end{aligned}$$

Derivative of the Sigmoid and the Chain Rule

$$\frac{\partial a^{[2]}}{\partial z} = a^{[2]}(1 - a^{[2]}) \tag{52}$$

$$\begin{aligned}\frac{\partial L}{\partial z} &= \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z} \\ &= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right) a^{[2]}(1 - a^{[2]}) \qquad (53) \\ &= a^{[2]} - y\end{aligned}$$

Automatic Differentiation

- State-of-the-art implementations of backpropagation algorithms obviate the need of having to calculate gradients "by hand".
- Instead, implementations such as TensorFlow and PyTorch use computation graph formalisms that support automatic, efficient algorithms for calculating gradients on GPU hardware.

Neural Language Models

$$P(w_t|w_1, ..., w_{t-1}) \approx P(w_t|w_{t-N+1}, ..., w_{t-1}) \tag{54}$$

Embeddings

Projection layer:

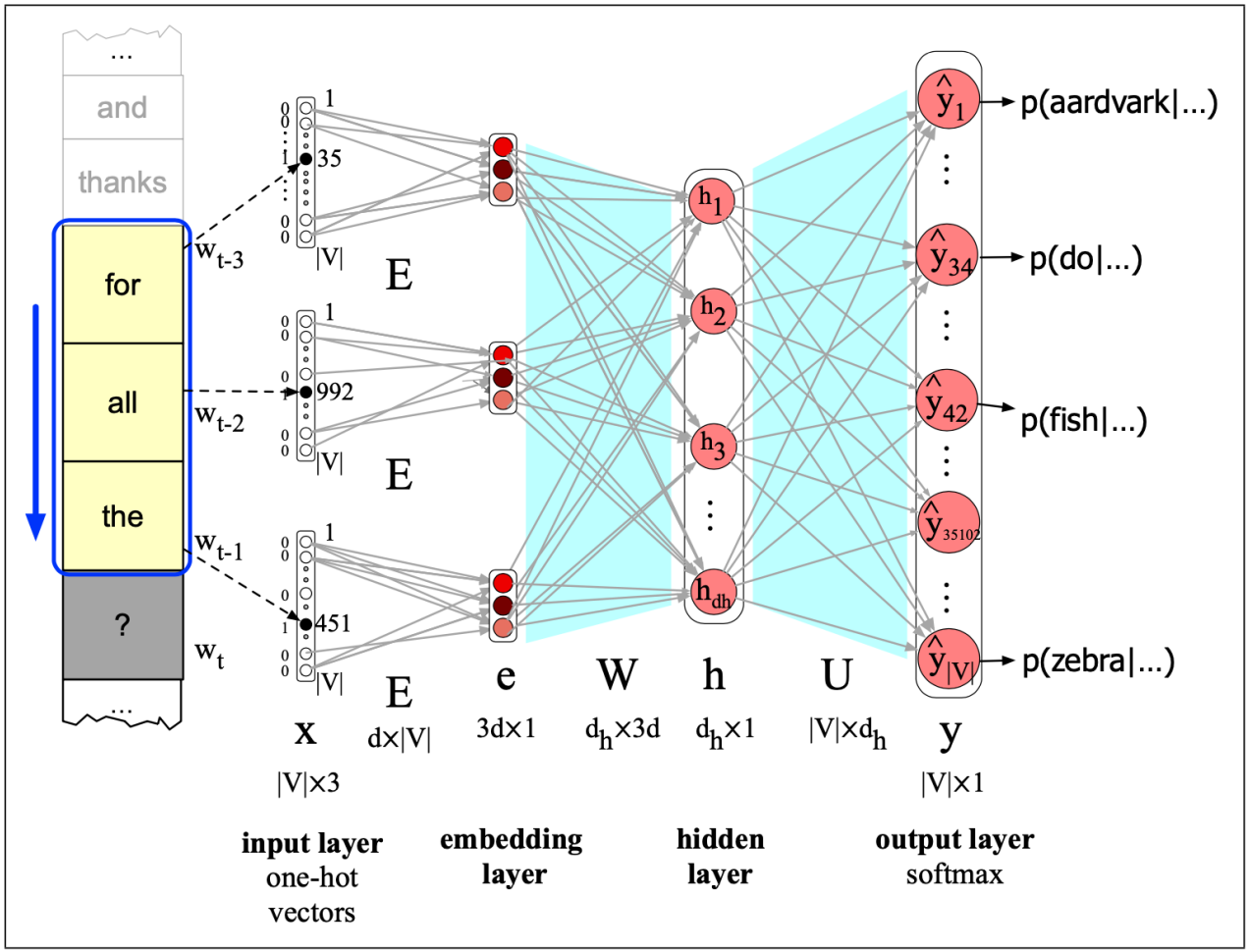
- **Select three embeddings from E**: Given the three previous words, we look up their indices, create 3 one-hot vectors, and then multiply each by the embedding matrix E. Consider w_{t-3} The one-hot vector for 'the' (index 35) is multiplied by the embedding matrix E, to give the first part of the first hidden projection layer layer, called the **projection layer**. Since each row of the input matrix E is just an embedding for a word, and the input is a one-hot column vector x_i for word V_i, the projection layer for input w will be $Ex_i = e_i$, the embedding for word i. We now concatenate the three embeddings for the context words.

Embeddings

Projection layer:

- **Multiply by W**: We now multiply by W (and add b) and pass through the rectified linear (or other) activation function to get the hidden layer h.
- **Multiply by U**: h is now multiplied by U
- **Apply softmax**: After the softmax, each node i in the output layer estimates the probability $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$

Forward inference in the neural language model



Embeddings

In summary, if we use e to represent the projection layer, formed by concatenating the 3 embeddings for the three context vectors, the equations for a neural language model become::

$$e = (Ex_1, Ex_2, ..., Ex) \tag{55}$$

$$h = \sigma(W_e + b) \tag{56}$$

$$z = Uh \tag{57}$$

$$\hat{y} = \text{softmax}(z) \tag{58}$$

Training the neural language model

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad \text{(where i is the correct class)} \tag{59}$$

$$L_{CE} = -\log p(w_t | w_{t-1}, ..., w_{t-n+1}) \tag{60}$$

The parameter update for stochastic gradient descent for this loss from step s to $s+1$ is then:

$$\theta^{s+1} = \theta^s - \eta \frac{\partial - \log p(w_t | w_{t-1}, ..., w_{t-n+1})}{\partial \theta} \tag{61}$$

Backward differentiation for a neural network