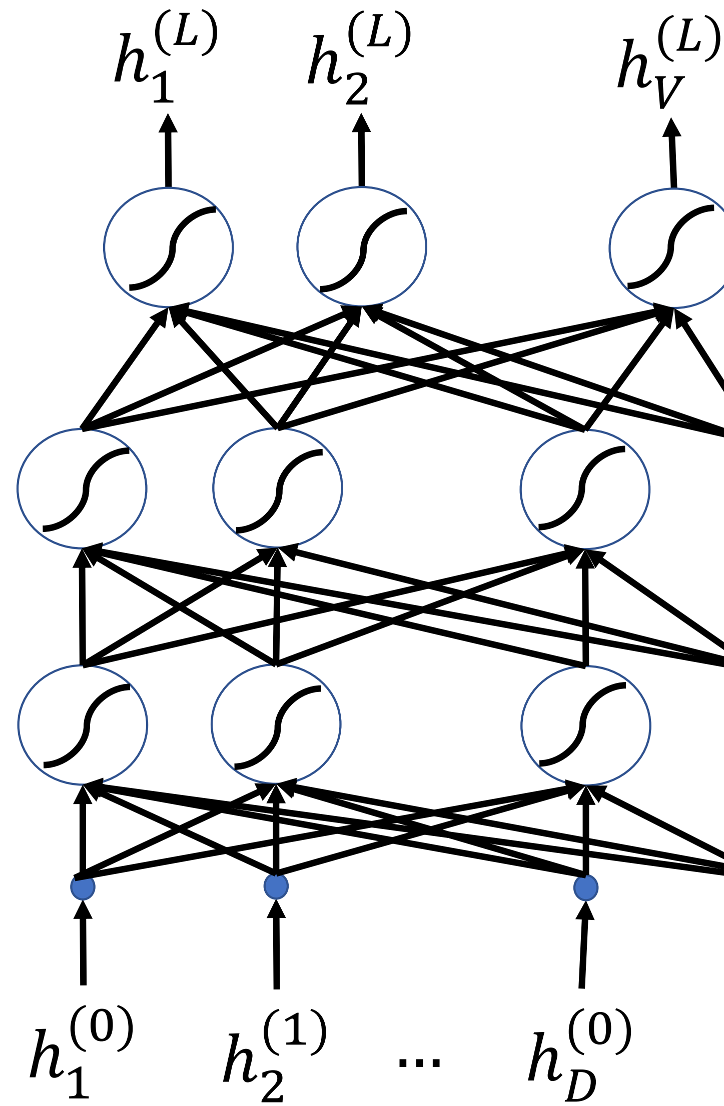
Lecture 11: Back-Propagation

Mark Hasegawa-Johnson

2/2022



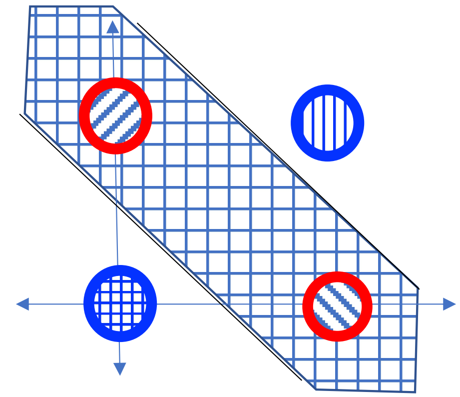
Outline

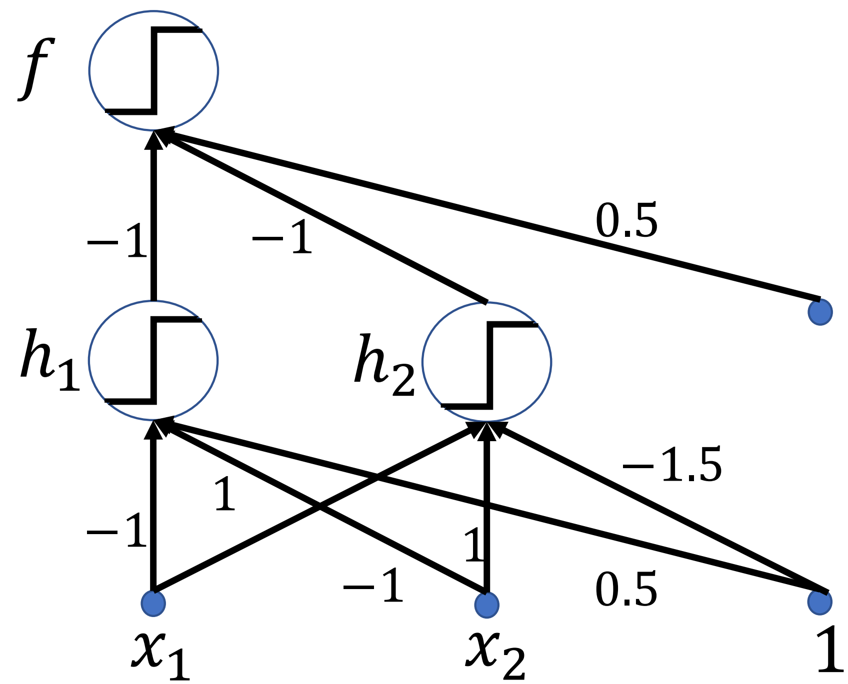
- Review: flow diagrams
- Gradient descent
- The chain rule of calculus
- Back-propagation

Flow diagrams

A flow diagram is a way to represent the computations performed by a neural network.

- Circles, a.k.a. “nodes,” a.k.a. “neurons,” represent scalar operations.
  - The circles above x_1 and x_2 represent the scalar operation of “read this datum in from the dataset.”
  - The circles labeled h_1 and h_1 represent the scalar operation of “unit step function.”
- Lines represent multiplication by a scalar.
- Where arrowheads come together, the corresponding variables are added.

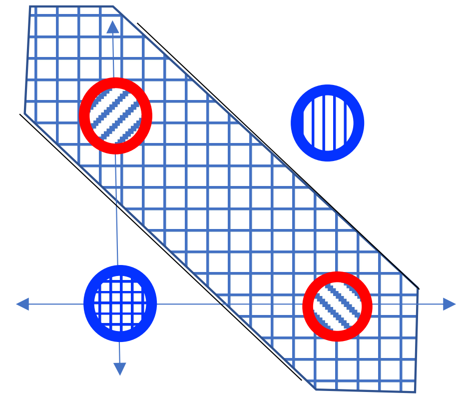


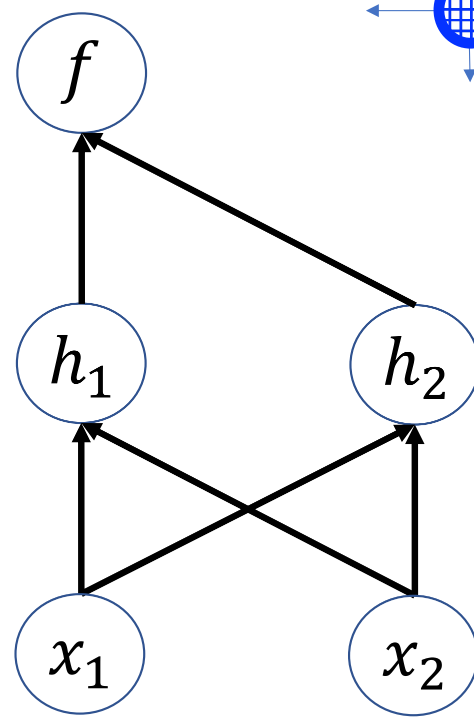


Flow diagrams

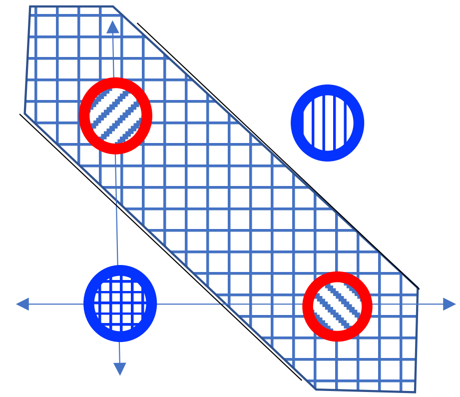
Usually, a flow diagram shows only the activations (including inputs). Excitations, weights, biases, and nonlinearities are implicit. For example, this flow diagram means that there are some nonlinearities $g^{(l)}$, and some weights $w_{j,k}^{(l)}$ and biases $b_j^{(l)}$ such that:

$$h_1 = g^{(1)}\left(b_1^{(1)} + w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2\right)$$
$$h_2 = g^{(1)}\left(b_2^{(1)} + w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2\right)$$
$$f = g^{(2)}\left(b_1^{(2)} + w_1^{(2)} h_1 + w_2^{(2)} h_2\right)$$



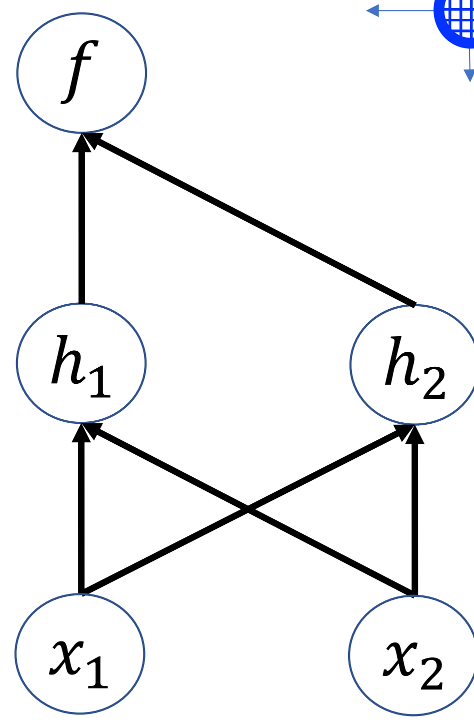


Flow diagrams



The important piece of information shown in a flow diagram is the order of computation. This flow diagram shows that, given x_1 and x_2,

- First, you calculate h_1 and h_2,
- then you can calculate f.



Outline

- Review: flow diagrams
- Gradient descent
- The chain rule of calculus
- Back-propagation

Gradient descent: basic idea

- Suppose we have a training token, x.
- Its target label is y.
- The neural net produces output $f(x)$, which is not y.
- The difference between y and $f(x)$ is summarized by some loss function, $\mathcal{L}(y, f(x))$.
- The output of the neural net is determined by some parameters, $w_{j,k}^{(l)}$.
- Then we can improve the network by setting:

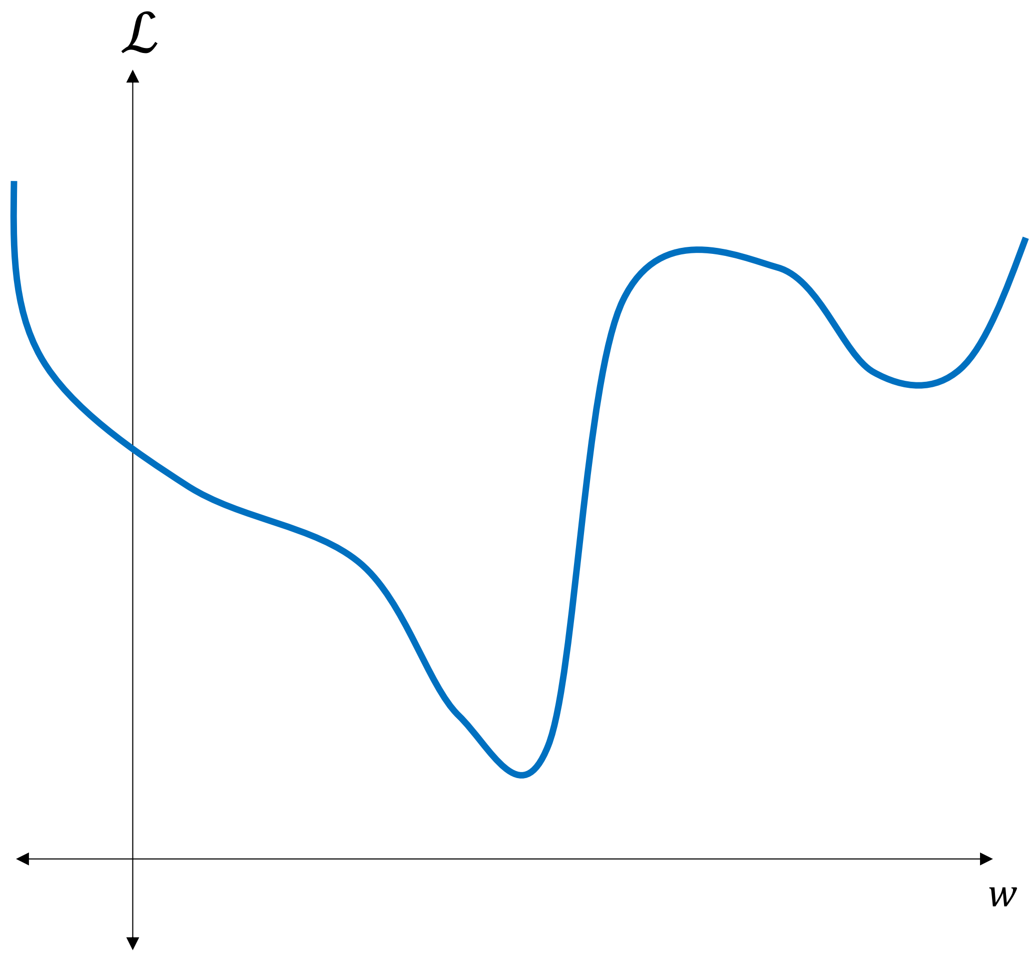
$$w_{j,k}^{(l)} \leftarrow w_{j,k}^{(l)} - \eta \frac{d\mathcal{L}}{dw_{j,k}^{(l)}}$$

Gradient descent in a multi-layer neural net

Just like in linear regression, the MSE loss is still a quadratic function of $f(\vec{x})$:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(f(\vec{x}_i) - y_i)^2$$

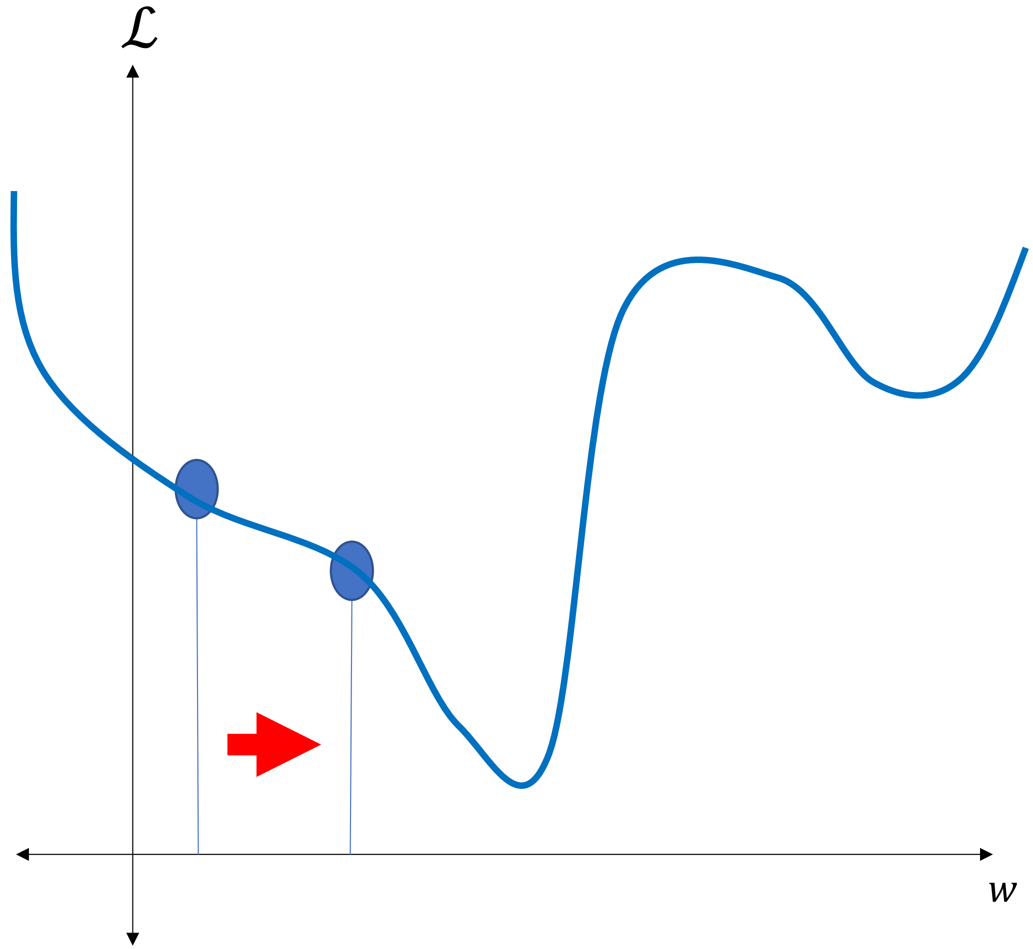
… but now, $f(\vec{x})$ is a complicated nonlinear function of the weights. Therefore, if we draw $\mathcal{L}(w)$, it's no longer a simple parabola.



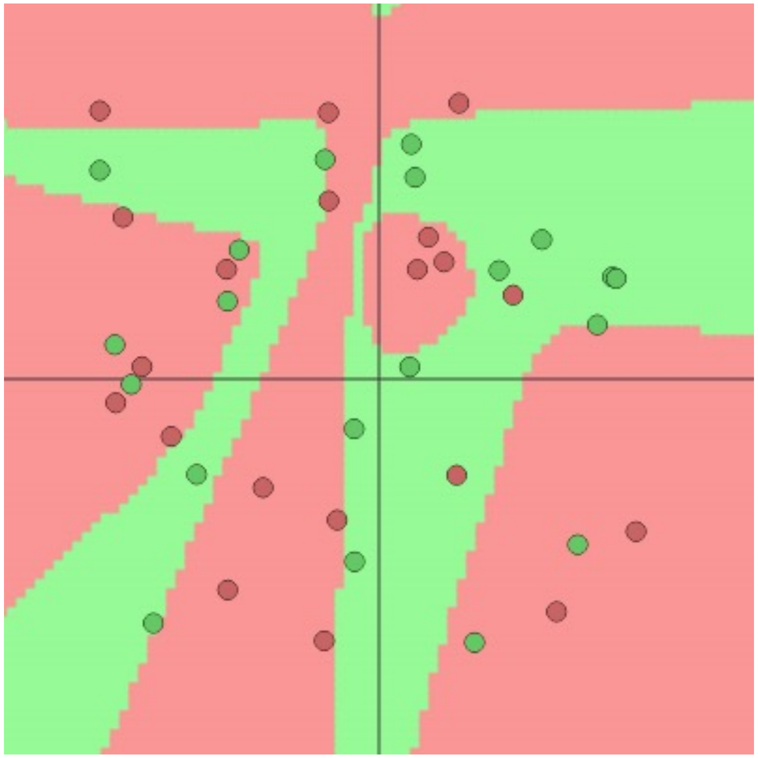
Gradient descent in a multi-layer neural net

Even though $\mathcal{L}(w)$ is complicated, we can still minimize it using gradient descent:

$$\vec{w} \leftarrow \vec{w} - \eta \nabla_{\vec{w}} \mathcal{L}$$



Visualizing gradient descent



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

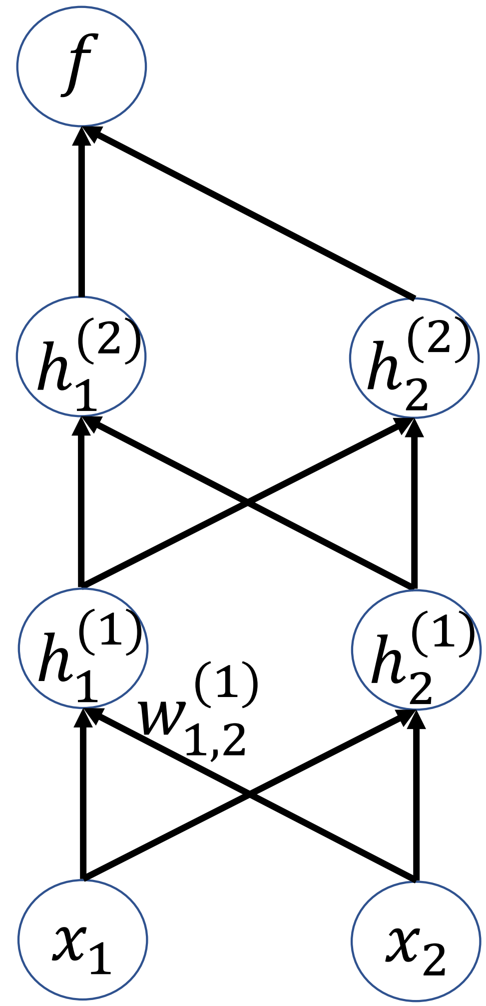
Outline

- Review: flow diagrams
- Gradient descent
- The chain rule of calculus
- Back-propagation

Definition of gradient

The gradient is the vector of partial derivatives:

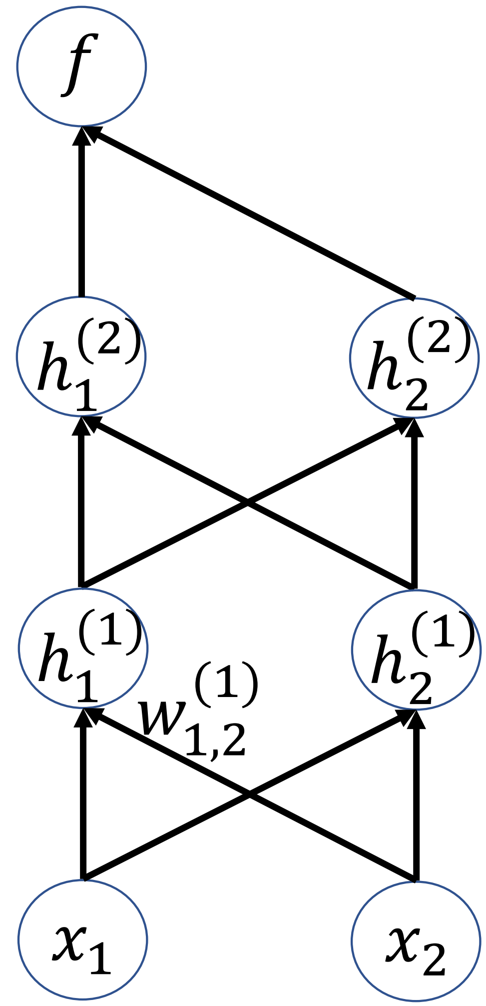
$$\nabla_{\vec{w}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,1}^{(1)}} \\ \vdots \\ \frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} \\ \vdots \\ \frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(3)}} \end{bmatrix}$$



Definition of partial derivative

The partial derivative of $\mathcal{L}(\vec{w})$ with respect to $w_{1,2}^{(1)}$ is what we get if we change $w_{1,2}^{(1)}$ to $w_{1,2}^{(1)} + \delta$, while keeping all of the other weights the same. If we define $\hat{\imath}_{1,2}^{(1)}$ to be a vector that has a 1 in the $w_{1,2}^{(1)}$ place, and zeros everywhere else, then:

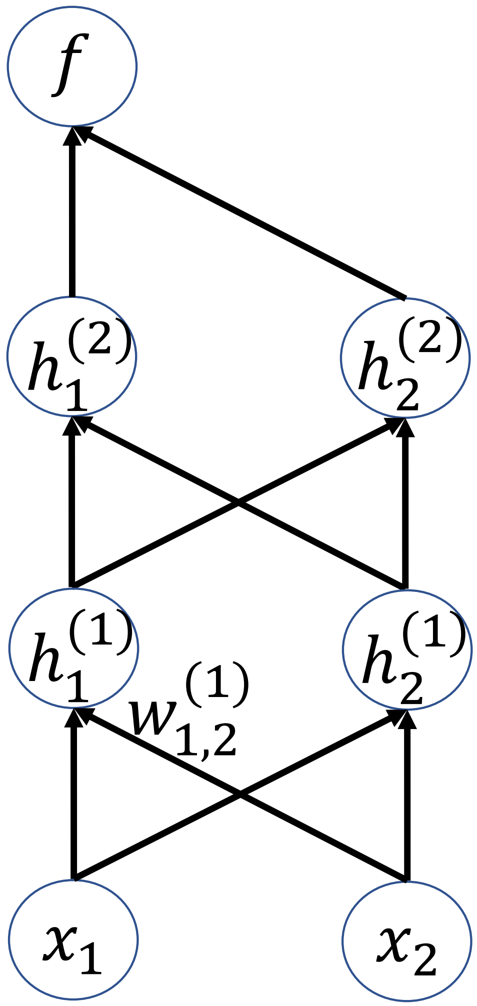
$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} = \lim_{\delta \to 0} \frac{\mathcal{L}\left(\vec{w} + \delta \hat{\imath}_{1,2}^{(1)}\right) - \mathcal{L}(\vec{w})}{\delta}$$



Lots of different partial derivatives

There are a lot of useful partial derivatives we could compute. For example:

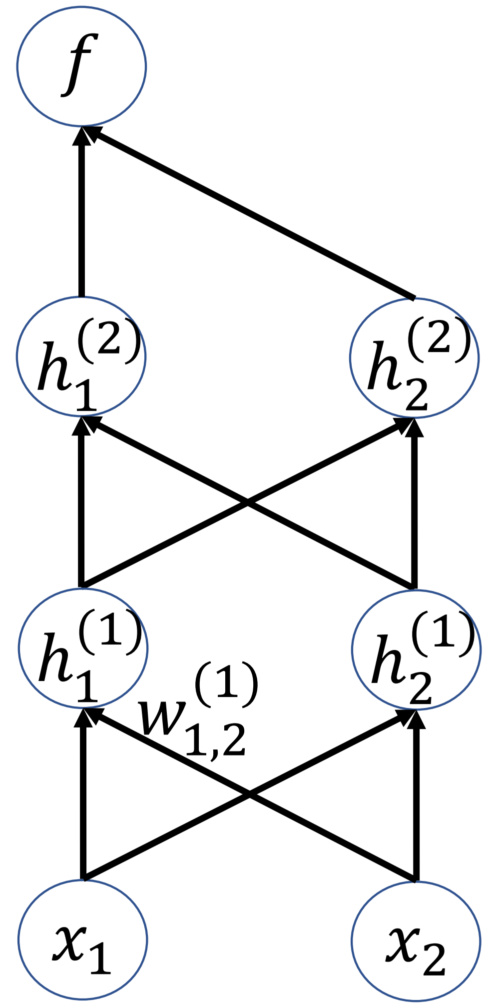
- $\frac{\partial \mathcal{L}(\vec{h}^{(2)})}{\partial h_1^{(2)}}$ is the partial derivative of $\mathcal{L}$ with respect to $h_1^{(2)}$, while keeping all other elements of the vector $\vec{h}^{(2)} = \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \end{bmatrix}$ constant.



The chain rule of calculus

- $\mathcal{L}$ depends on f, which depends on $w_{1,2}^{(1)}$.
- We can calculate $\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}}$ by "chaining" (multiplying) the two partial derivatives along the flow path:

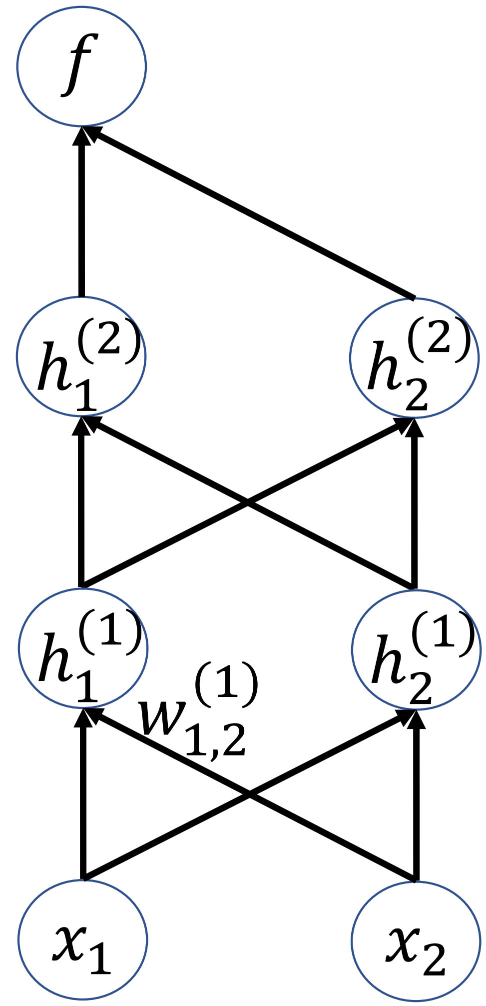
$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} = \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f(\vec{w})}{\partial w_{1,2}^{(1)}}$$



More chain rule

- f depends on $h_1^{(1)}$, which depends on $w_{1,2}^{(1)}$.
- We can calculate $\frac{\partial f(\vec{w})}{\partial w_{1,2}^{(1)}}$ by chaining:

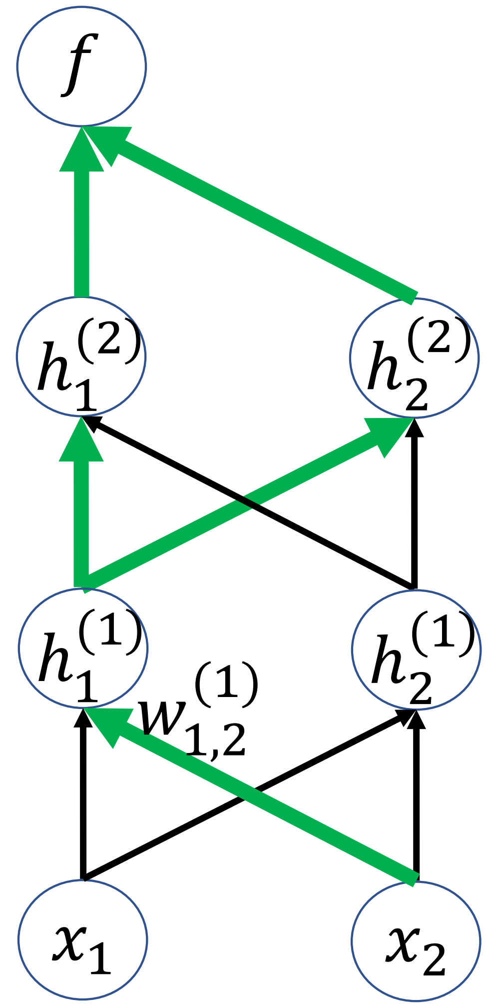
$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} = \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f\left(\vec{h}^{(1)}\right)}{\partial h_1^{(1)}} \cdot \frac{\partial h_1^{(1)}(\vec{w})}{\partial w_{1,2}^{(1)}}$$



More chain rule

- f depends on $h_1^{(2)}$ and $h_2^{(2)}$. Both $h_1^{(2)}$ and $h_2^{(2)}$ depend on $h_1^{(1)}$.
- To apply the chain rule here, we need to sum over both of the flow paths:

$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} = \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f\left(\vec{h}^{(2)}\right)}{\partial h_1^{(2)}} \cdot \frac{\partial h_1^{(2)}\left(\vec{h}^{(1)}\right)}{\partial h_1^{(1)}} \cdot \frac{\partial h_1^{(1)}(\vec{w})}{\partial w_{1,2}^{(1)}} + \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f\left(\vec{h}^{(2)}\right)}{\partial h_2^{(2)}} \cdot \frac{\partial h_2^{(2)}\left(\vec{h}^{(1)}\right)}{\partial h_1^{(1)}} \cdot \frac{\partial h_1^{(1)}(\vec{w})}{\partial w_{1,2}^{(1)}}$$



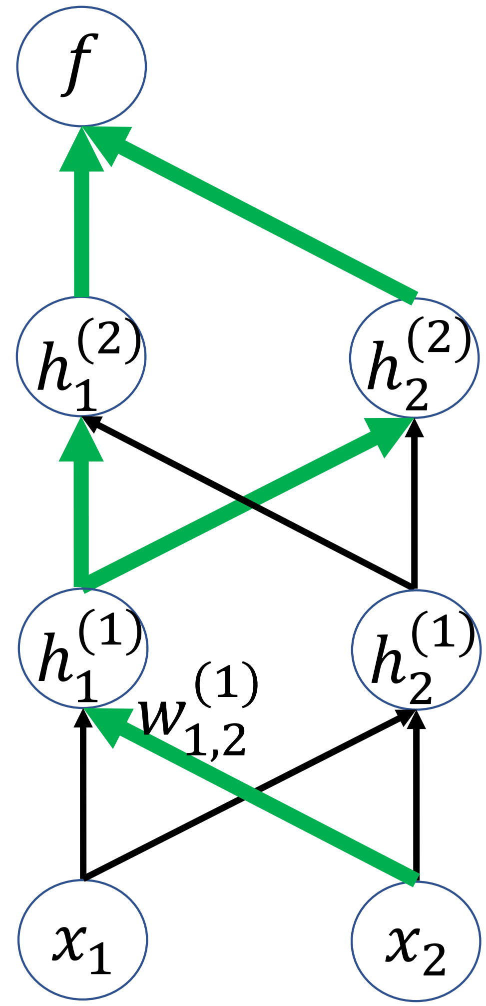
Outline

- Review: flow diagrams
- Gradient descent
- The chain rule of calculus
- Back-propagation

Back-propagation

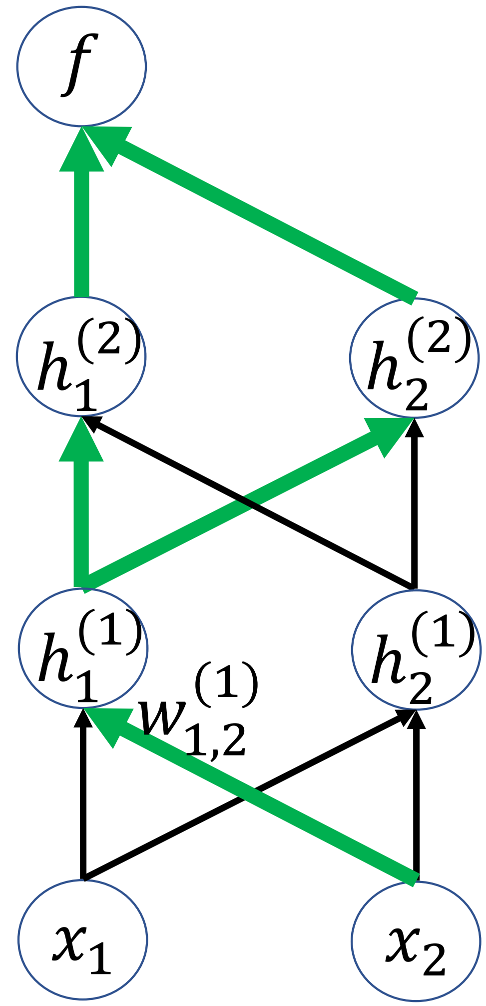
The key idea of back-propagation is to calculate $\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{j,k}^{(l)}}$, for every layer l, for every pair of nodes j and k, as follows:

- Start at the output node.
- Apply the chain rule of calculus backward, layer-by-layer, from the output node backward toward the input.



Back-propagation

- First, calculate $\frac{d\mathcal{L}}{\partial f}$.



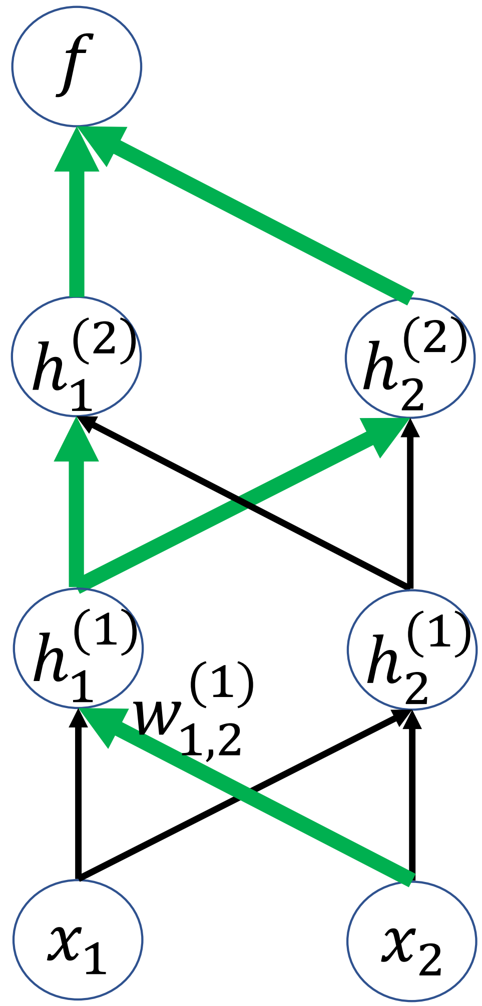
Back-propagation

- First, calculate $\frac{d\mathcal{L}}{\partial f}$.
- Second, calculate

$$\frac{\partial \mathcal{L}\left(\vec{h}^{(2)}\right)}{\partial h_1^{(2)}} = \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f\left(\vec{h}^{(2)}\right)}{\partial h_1^{(2)}}$$

… and …

$$\frac{\partial \mathcal{L}\left(\vec{h}^{(2)}\right)}{\partial h_2^{(2)}} = \frac{d\mathcal{L}}{\partial f} \cdot \frac{\partial f\left(\vec{h}^{(2)}\right)}{\partial h_2^{(2)}}$$



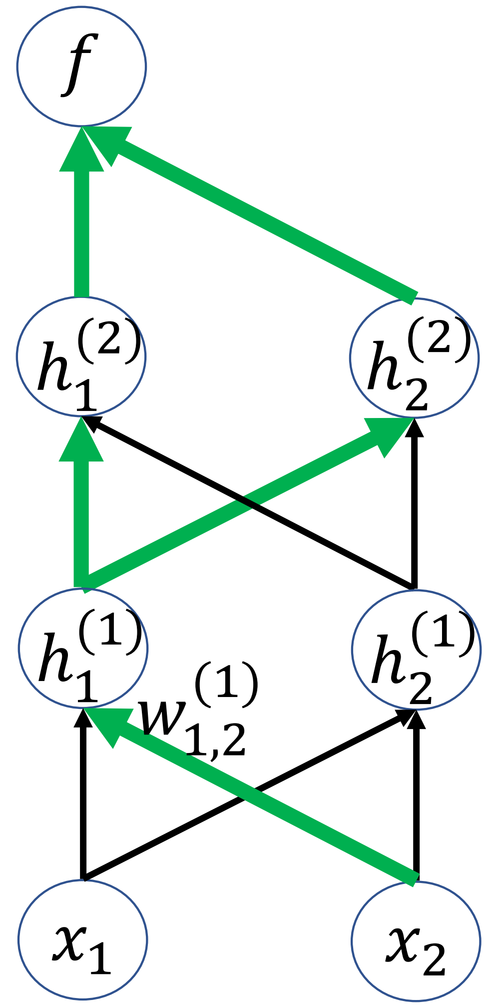
Back-propagation

- Third, calculate

$$\frac{\partial \mathcal{L}\left(\vec{h}^{(1)}\right)}{\partial h_1^{(1)}} = \sum_j \frac{\partial \mathcal{L}\left(\vec{h}^{(2)}\right)}{\partial h_j^{(2)}} \cdot \frac{\partial h_j^{(2)}\left(\vec{h}^{(1)}\right)}{\partial h_1^{(1)}}$$

… and …

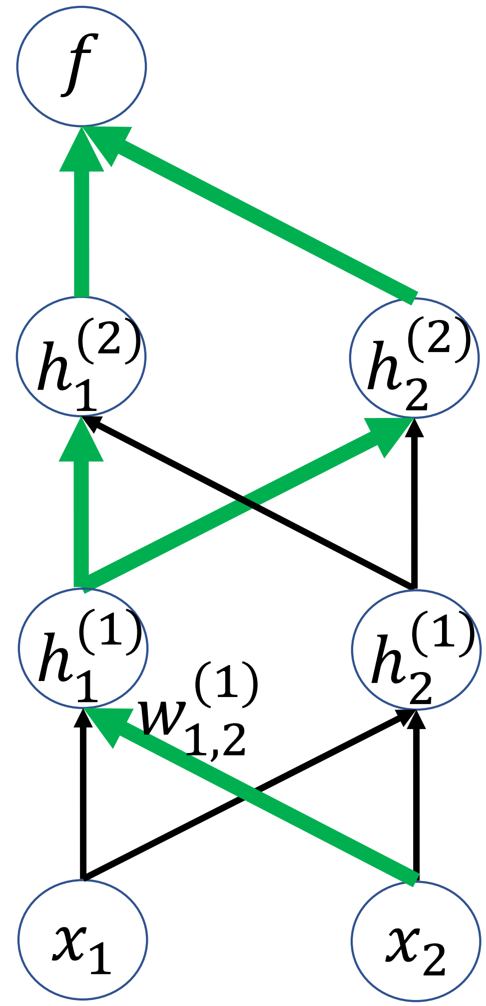
$$\frac{\partial \mathcal{L}\left(\vec{h}^{(1)}\right)}{\partial h_2^{(1)}} = \sum_j \frac{\partial \mathcal{L}\left(\vec{h}^{(2)}\right)}{\partial h_j^{(2)}} \cdot \frac{\partial h_j^{(2)}\left(\vec{h}^{(1)}\right)}{\partial h_2^{(1)}}$$



Back-propagation

- Fourth, calculate

$$\frac{\partial \mathcal{L}(\vec{w})}{\partial w_{1,2}^{(1)}} = \sum_j \frac{\partial \mathcal{L}\left(\vec{h}^{(1)}\right)}{\partial h_j^{(1)}} \cdot \frac{\partial h_j^{(1)}(\vec{w})}{\partial w_{1,2}^{(1)}}$$



Back-propagation: splitting it up into excitation and activation

- **Activation to excitation**: here, the derivative is pre-computed. For example, if g=ReLU, then g'=unit step:

$$h_j^{(l)} = \mathrm{ReLU}\left(\xi_j^{(l)}\right) \Longrightarrow \frac{\partial h_j^{(l)}}{\partial \xi_j^{(l)}} = u\left(\xi_j^{(l)}\right)$$

- **Excitation to activation**: here, the derivative is just the network weight!

$$\xi_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} h_k^{(l-1)} \Longrightarrow \frac{\partial \xi_j^{(l)}}{\partial h_k^{(l-1)}} = w_{j,k}^{(l)}$$

Back-propagation: splitting it up into excitation and activation

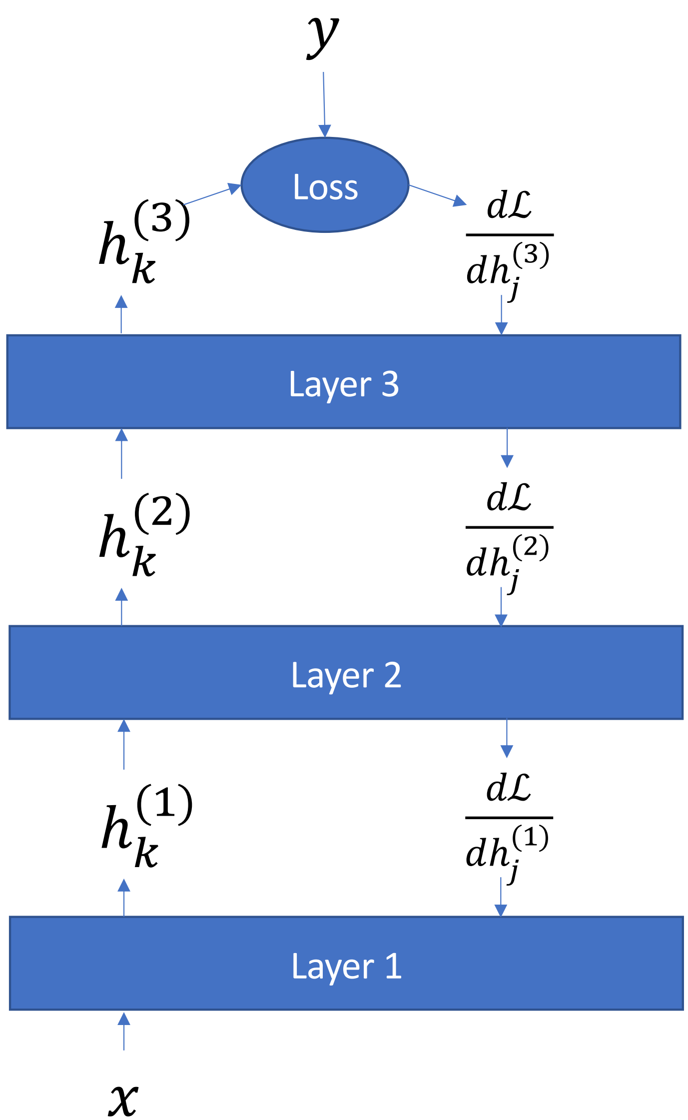
- **<u>Excitation to network weight</u>**: here, the derivative is the previous layer's activation:

$$\xi_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} h_k^{(l-1)} \Longrightarrow \frac{\partial \xi_j^{(l)}}{\partial w_{j,k}^{(l)}} = h_k^{(l-1)}$$

Finding the derivative

- Forward propagate, from x, to find $h_k^{(l-1)}$ in each layer
- Back-propagate, from y, to find $\frac{d\mathcal{L}}{dh_j^{(l)}}$ in each layer
- Multiply them to get $\frac{d\mathcal{L}}{dw_{jk}^{(l)}}$, then

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{d\mathcal{L}}{dw_{jk}^{(l)}}$$



Outline

- Review: flow diagrams
- Gradient descent

$$\vec{w} \leftarrow \vec{w} - \eta \nabla_{\vec{w}} \mathcal{L}$$

- The chain rule of calculus

$$\frac{\partial \mathcal{L}\left(\vec{h}^{(1)}\right)}{\partial h_2^{(1)}} = \sum_j \frac{\partial \mathcal{L}\left(\vec{h}^{(2)}\right)}{\partial h_j^{(2)}} \cdot \frac{\partial h_j^{(2)}\left(\vec{h}^{(1)}\right)}{\partial h_2^{(1)}}$$

- Back-propagation
    - Apply the chain rule of calculus backward, layer-by-layer, from the output node backward toward the input.