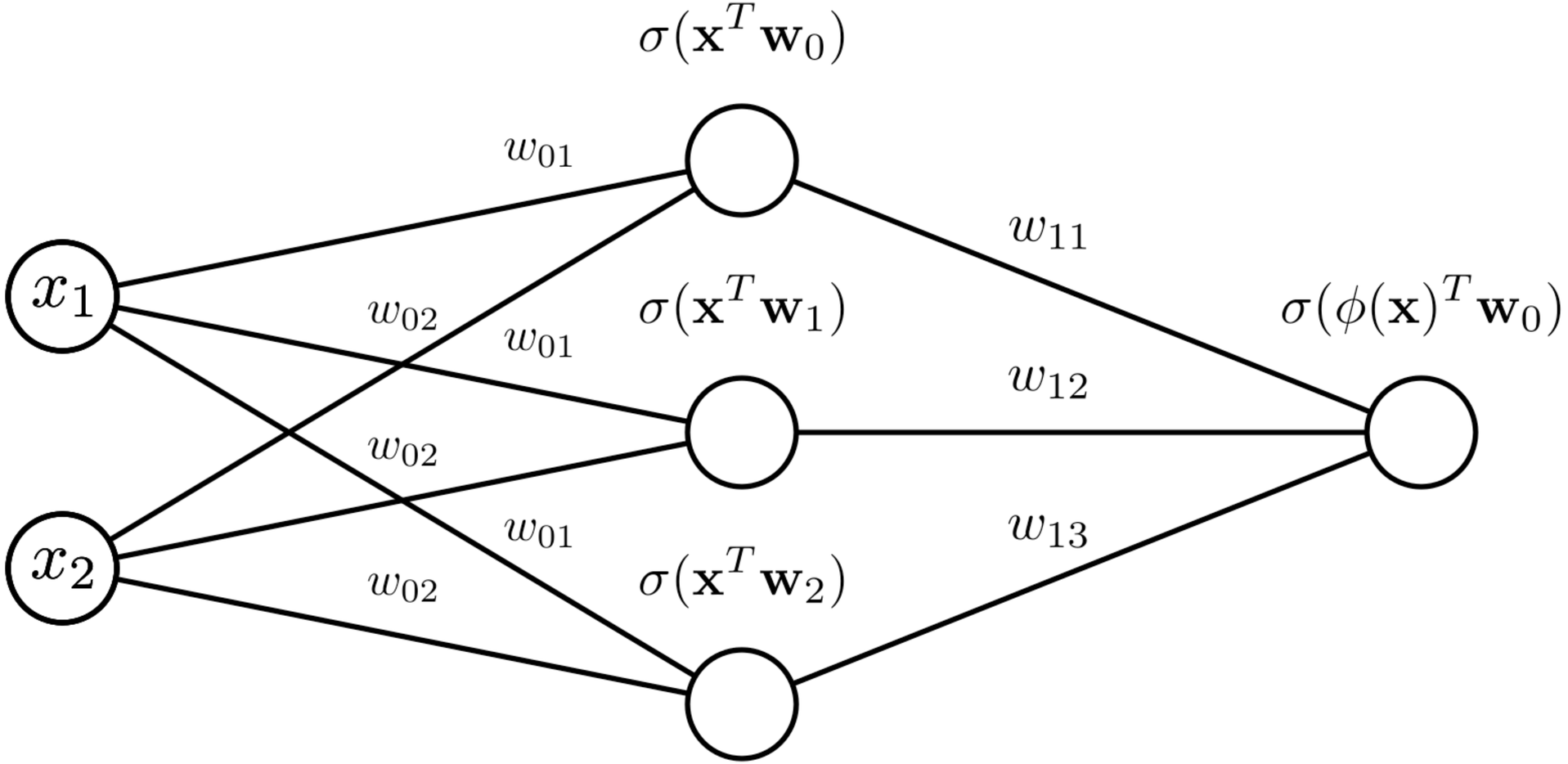
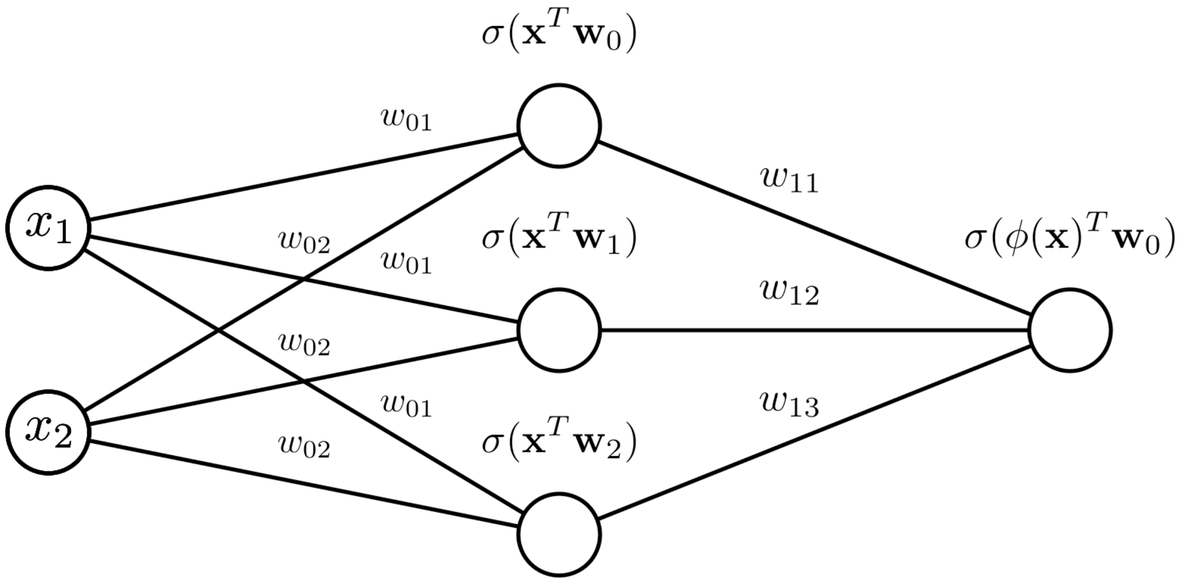
Learning Neural Networks

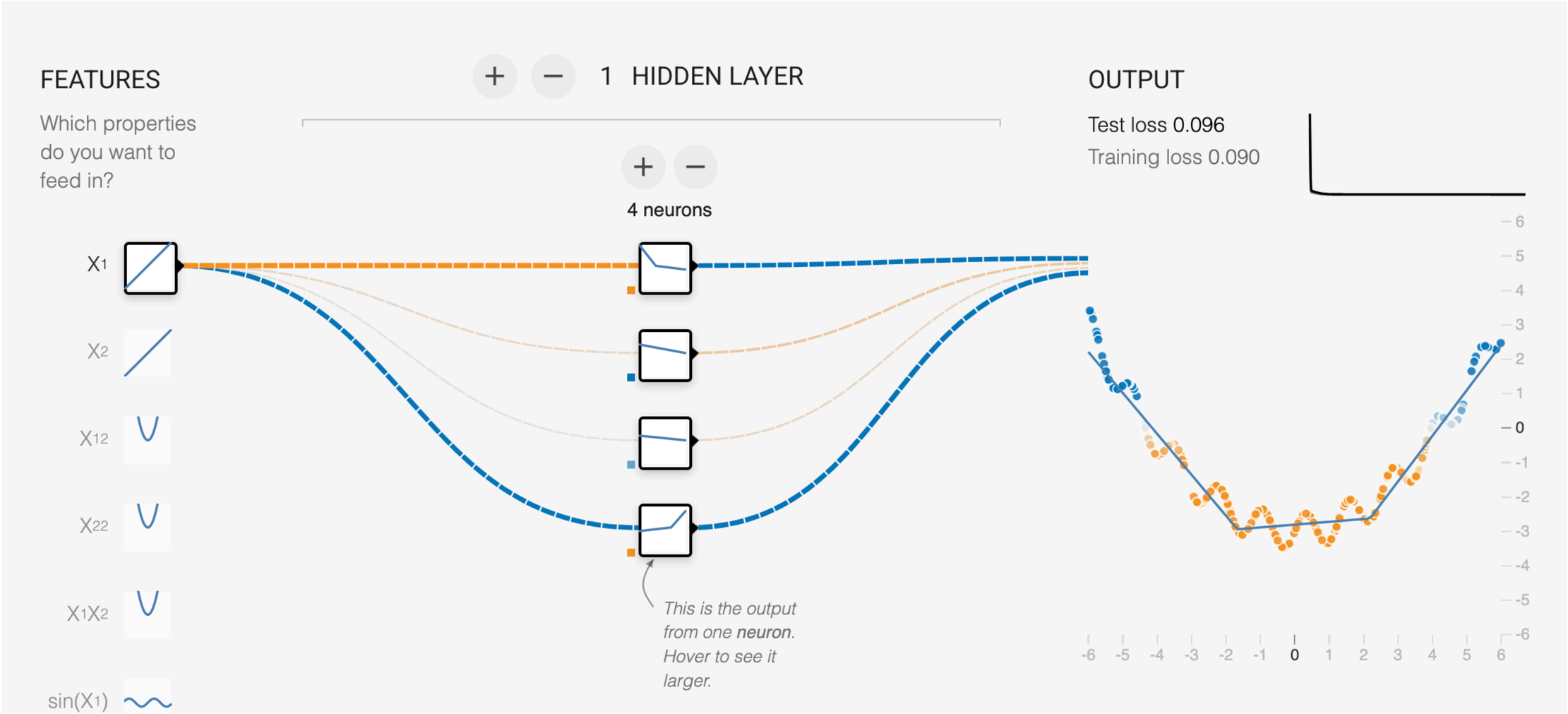
Last time: A Neural Network!



Last time: A Single Neuron



Last time: A Neural Network!

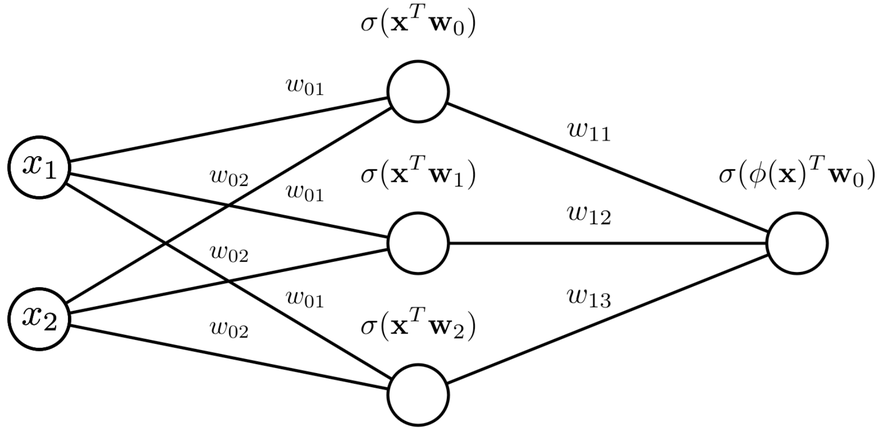


Last time: Neural networks with matrix notation

Our neural network

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{w}_0 = \begin{bmatrix} w_{01} \\ w_{02} \end{bmatrix}$$

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}_0, \quad \phi(\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{x}^T \mathbf{w}_1) \\ \sigma(\mathbf{x}^T \mathbf{w}_2) \\ \sigma(\mathbf{x}^T \mathbf{w}_3) \end{bmatrix} = \begin{bmatrix} \sigma(x_1 w_{11} + x_2 w_{12}) \\ \sigma(x_1 w_{21} + x_2 w_{22}) \\ \sigma(x_1 w_{31} + x_2 w_{32}) \end{bmatrix}$$



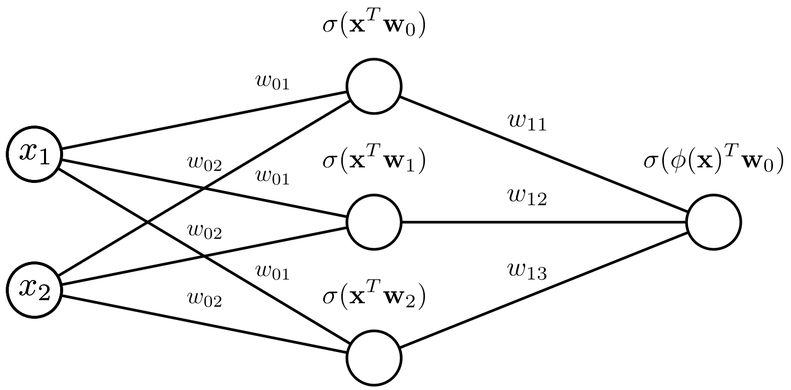
Write the linear function for each neuron as a matrix vector product

$$\begin{bmatrix} \mathbf{x}^T \mathbf{w}_1 \\ \mathbf{x}^T \mathbf{w}_2 \\ \mathbf{x}^T \mathbf{w}_3 \end{bmatrix} = \begin{bmatrix} x_1 w_{11} + x_2 w_{12} \\ x_1 w_{21} + x_2 w_{22} \\ x_1 w_{31} + x_2 w_{32} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{x}^T \mathbf{w}_1 \\ \mathbf{x}^T \mathbf{w}_2 \\ \mathbf{x}^T \mathbf{w}_3 \end{bmatrix} = \mathbf{W}\mathbf{x} = (\mathbf{x}^T \mathbf{W}^T)^T$$

Last time: Neural networks with matrix notation

Write all the weights for a layer in a big matrix



$$\mathbf{W}_1 = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \\ \vdots \end{bmatrix}$$

$$f(\mathbf{x}) = \sigma(\mathbf{W}_1\mathbf{x})^T\mathbf{w_0}$$

Compact prediction function!

Can do the same for the full dataset

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \\ \vdots \end{bmatrix}$$

$$f(\mathbf{x}) = \sigma(\mathbf{X}\mathbf{W}_1)\mathbf{w_0}$$

Neural networks with matrix notation

$$f(\mathbf{x}) = \sigma(\mathbf{X}\mathbf{W}^T)\mathbf{w}_0$$

To summarize:

- $\mathbf{X}$: $N \times d$ matrix of observations
- $\mathbf{W}$: $h \times d$ matrix of network weights
- $\mathbf{w}_0$: h $(\times 1)$ vector of linear regression weights

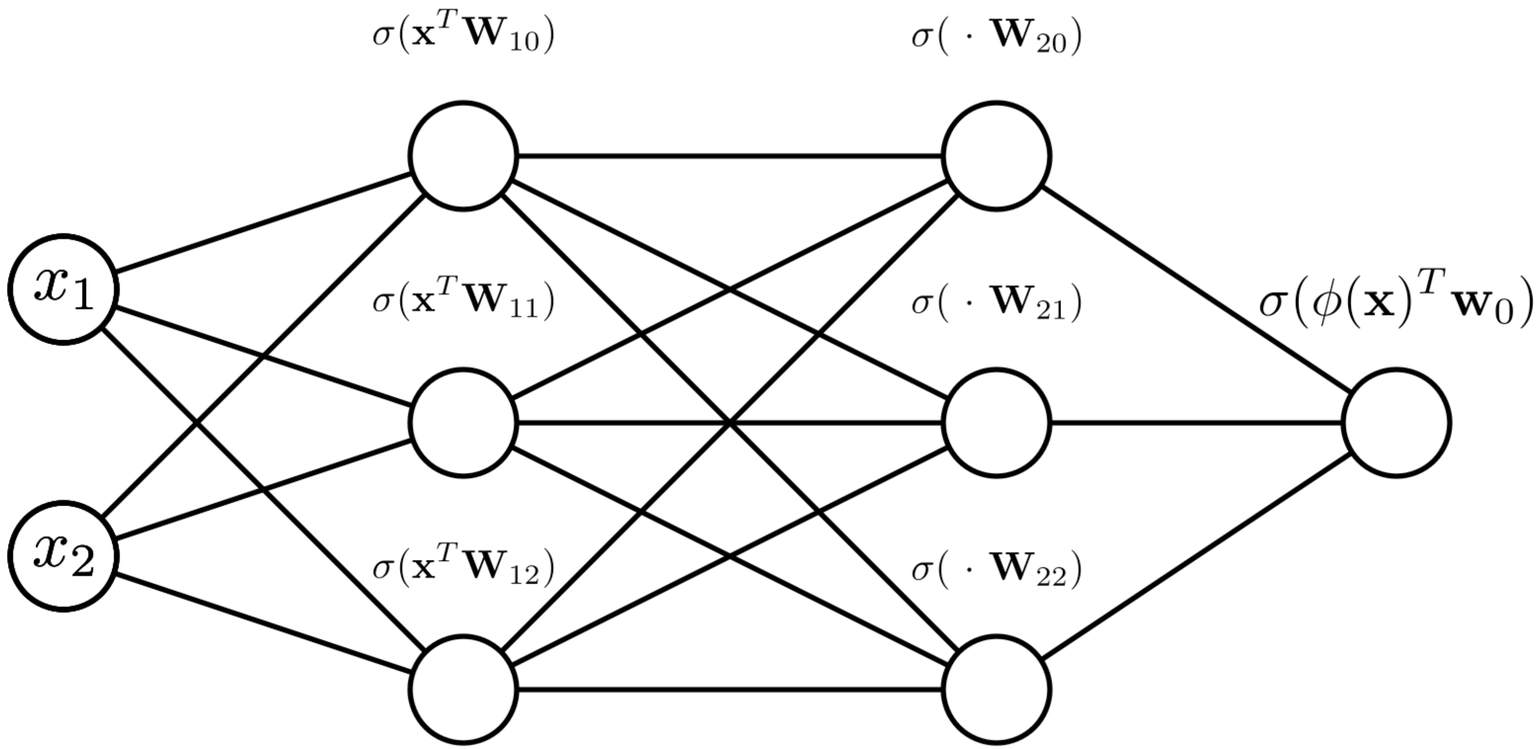
$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \\ \vdots \end{bmatrix} \quad \mathbf{W}_1 = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \\ \vdots \end{bmatrix} \quad \mathbf{w}_0 = \begin{bmatrix} w_{01} \\ w_{02} \end{bmatrix}$$

If we check that our dimensions work for matrix multiplication we see that we get the $N \times 1$ vector of predictions we are looking for!

$$(N \times d)(h \times d)^T(h \times 1) \rightarrow (N \times d)(d \times h)(h \times 1) \rightarrow (N \times h)(h \times 1)$$

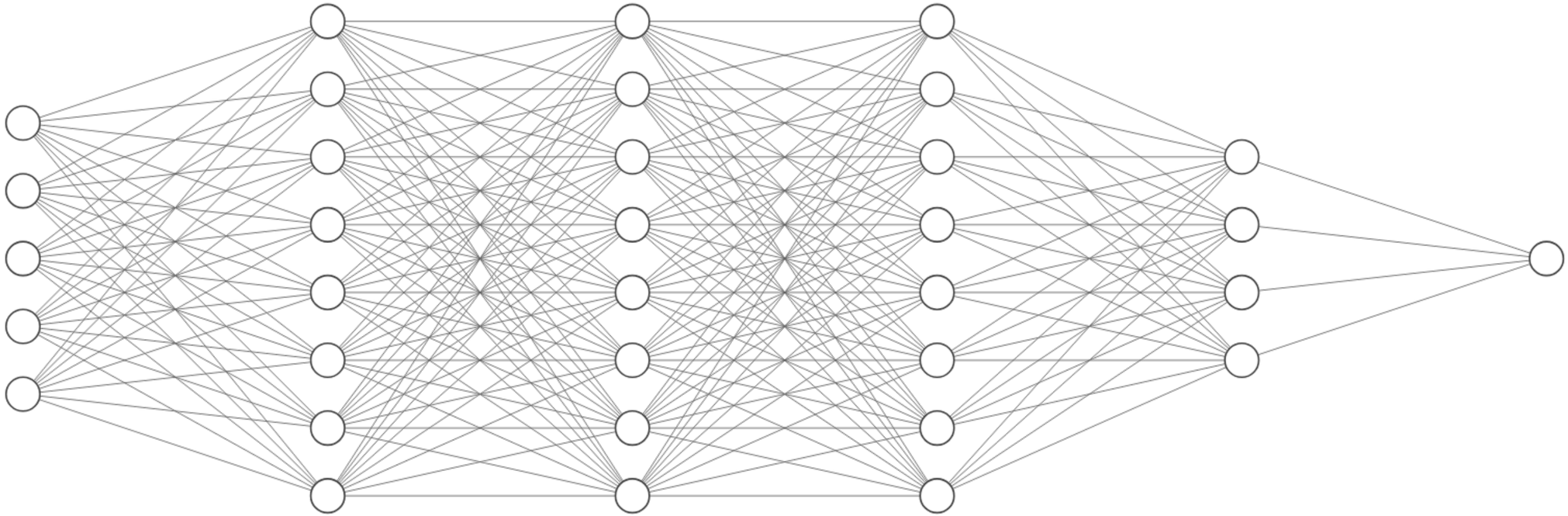
$$\longrightarrow (N \times 1)$$

Applying this recursively



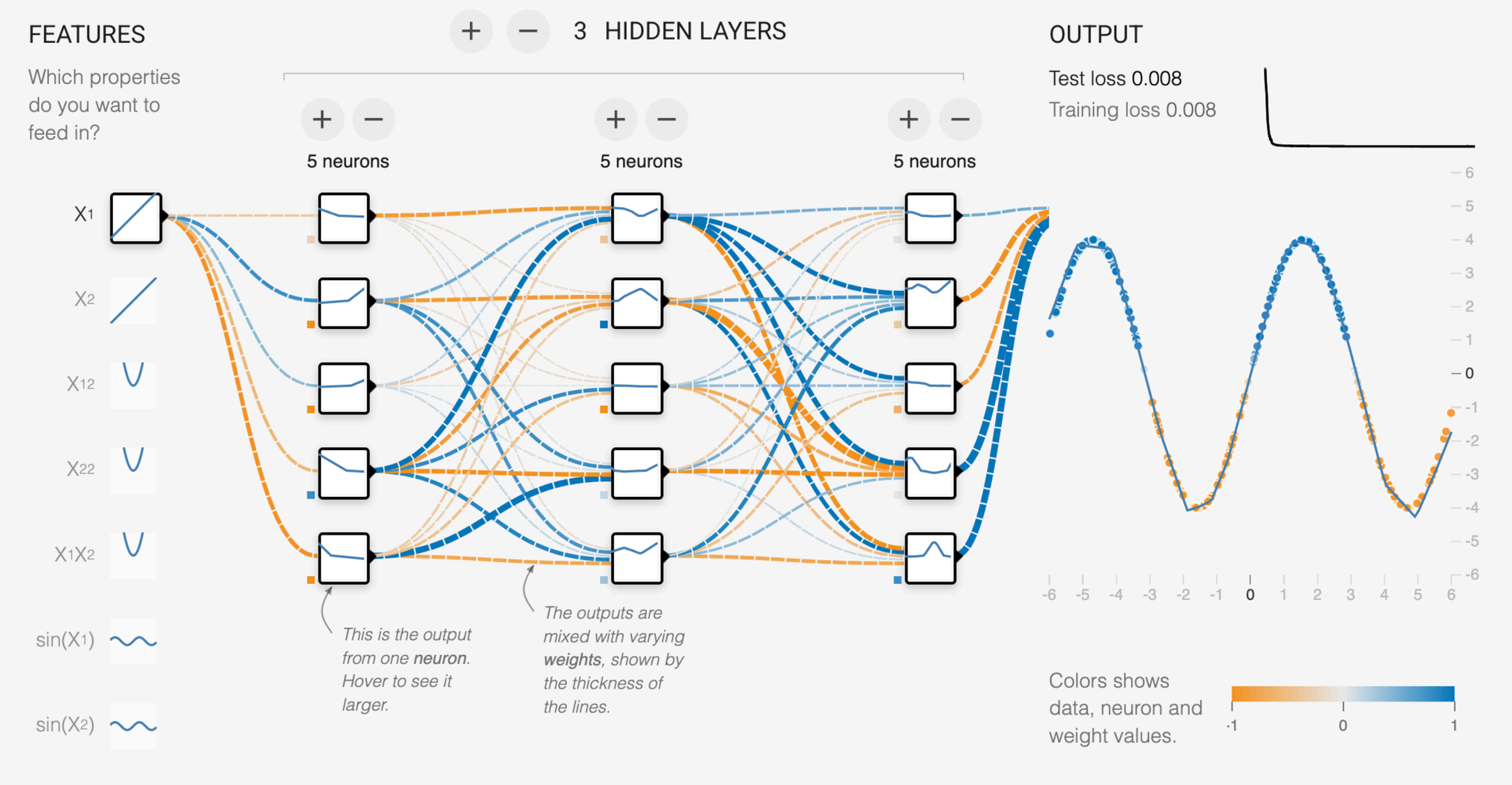
$$\phi(\mathbf{x})_i = \sigma(\mathbf{x}^T\mathbf{w}_i) \qquad \phi(\mathbf{x})_i = \sigma(\sigma(\mathbf{x}^T\mathbf{W}^T)\mathbf{w}_i)$$

Applying this recursively



Input Layer $\in \mathbb{R}^5$ Hidden Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^4$ Output Layer $\in \mathbb{R}^1$

Applying this recursively



Adding Biases

Loss function

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \Big[y_i \log p(y = 1 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) + (1 - y_i) \log p(y = 0 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) \Big]$$

$$= -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T \mathbf{w}\big)$$

Loss function

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \Big[y_i \log p(y = 1 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) + (1 - y_i) \log p(y = 0 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) \Big]$$

$$= -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T \mathbf{w}\big)$$

Substituting in our prediction function

$$p(y = 1 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) = \sigma(\phi(\mathbf{x})^T \mathbf{w}_0) = \sigma(\sigma(\mathbf{x}^T \mathbf{W}^T)\mathbf{w}_0), \quad \phi(\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{x}^T \mathbf{W}_1) \\ \sigma(\mathbf{x}^T \mathbf{W}_2) \\ \sigma(\mathbf{x}^T \mathbf{W}_3) \end{bmatrix}$$

$$= \sigma\big(w_{01} \cdot \sigma(x_1 W_{11} + x_2 W_{12}) + w_{02} \cdot \sigma(x_1 W_{21} + x_2 W_{22}) + w_{03} \cdot \sigma(x_1 W_{31} + x_2 W_{32})\big)$$

Gradient descent

Loss

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \Big[y_i \log p(y = 1 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) + (1 - y_i) \log p(y = 0 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) \Big]$$

Updates

$$\mathbf{w}_0^{(k+1)} \longleftarrow \mathbf{w}_0^{(k)} - \alpha \nabla_{\mathbf{w}_0} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}), \quad \mathbf{W}^{(k+1)} \longleftarrow \mathbf{W}^{(k)} - \alpha \nabla_{\mathbf{W}} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y})$$

Need:

$$\nabla_{\mathbf{w}_0} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = \ ? \qquad \nabla_{\mathbf{W}} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = \ ?$$

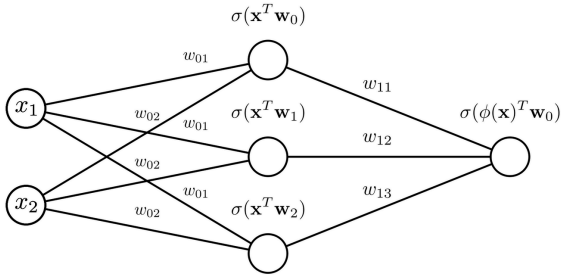
Gradient Shapes

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \Big[y_i \log p(y = 1 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) + (1 - y_i) \log p(y = 0 \mid \mathbf{x}, \mathbf{w}_0, \mathbf{W}) \Big]$$

$$\nabla_{\mathbf{w}_0} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = \begin{bmatrix} \frac{\partial \mathbf{NLL}}{\partial w_{01}} \\ \frac{\partial \mathbf{NLL}}{\partial w_{02}} \\ \frac{\partial \mathbf{NLL}}{\partial w_{03}} \\ \vdots \end{bmatrix}, \quad \nabla_{\mathbf{W}} \mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = \begin{bmatrix} \frac{\partial \mathbf{NLL}}{\partial W_{11}} & \frac{\partial \mathbf{NLL}}{\partial W_{12}} & \cdots & \frac{\partial \mathbf{NLL}}{\partial W_{1d}} \\ \frac{\partial \mathbf{NLL}}{\partial W_{21}} & \frac{\partial \mathbf{NLL}}{\partial W_{22}} & \cdots & \frac{\partial \mathbf{NLL}}{\partial W_{2d}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{NLL}}{\partial W_{h1}} & \frac{\partial \mathbf{NLL}}{\partial W_{h2}} & \cdots & \frac{\partial \mathbf{NLL}}{\partial W_{hd}} \end{bmatrix}$$

Automatic differentiation

Automatic differentiation: Motivation

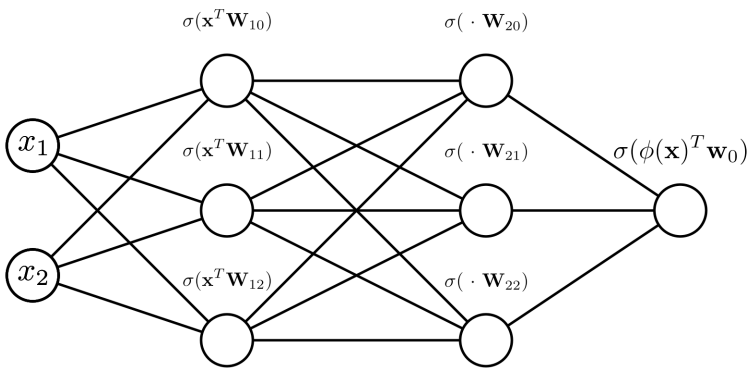


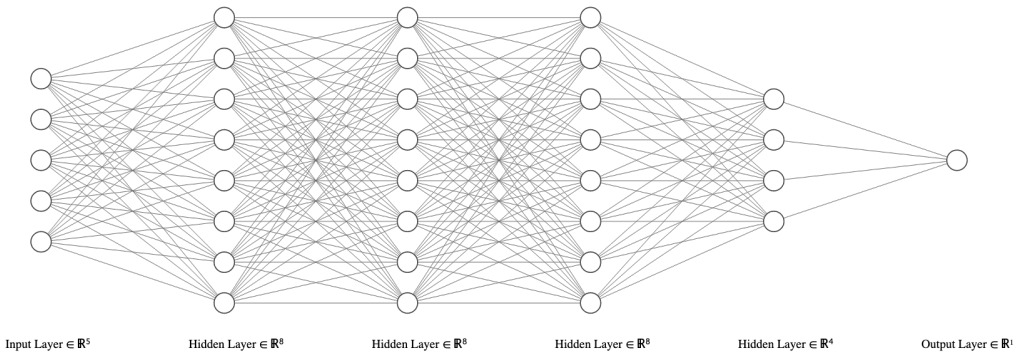
$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T\mathbf{w}\big)$$

$$= -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\sigma\big(w_{01} \cdot \sigma(x_1 W_{11} + x_2 W_{12}) + w_{02} \cdot \sigma(x_1 W_{21} + x_2 W_{22}) + w_{03} \cdot \sigma(x_1 W_{31} + x_2 W_{32})\big)\big)$$

Automatic differentiation: Motivation

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T \mathbf{w}\big)$$





Derivatives revisited

Given a function $f: \mathbb{R} \to \mathbb{R}$, a derivative is the *instantaneous rate of change*

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Derivatives revisited

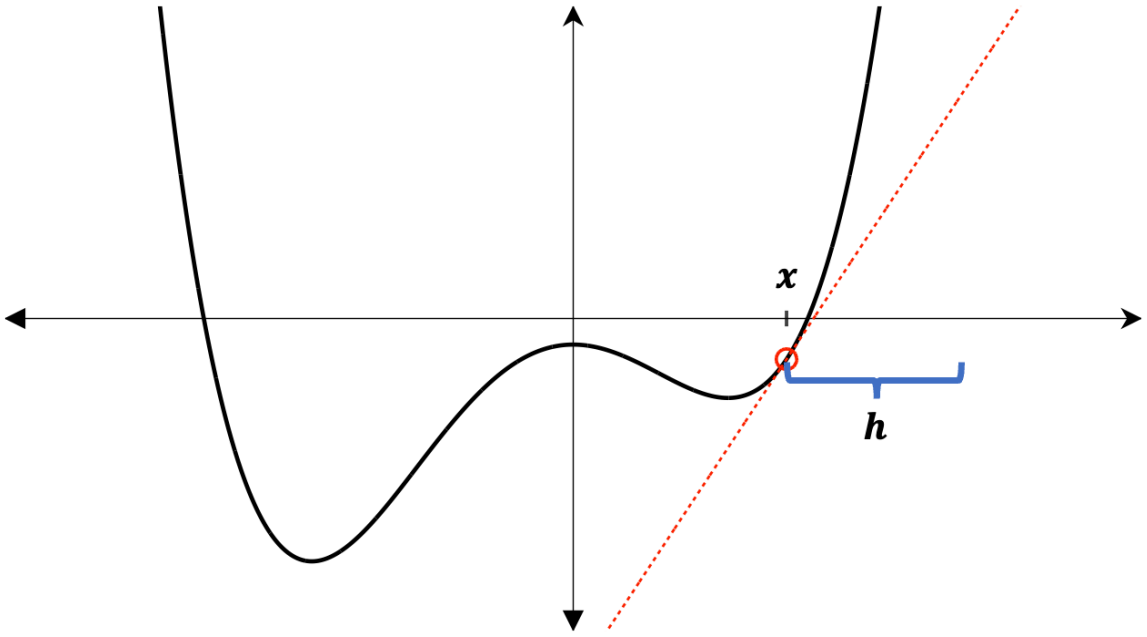
Given a function $f: \mathbb{R} \to \mathbb{R}$, a derivative is the *instantaneous rate of change*

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Equivalently, it's the slope of the *optimal linear approximation* of f near x.

Derivatives revisited

Optimal linear approximation



Consider the question:

"If I perturb x by h, how does the output change?"

The chain rule

Where does this appear?

Consider the chain rule

$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot g'(x) \cdot 1$$

If I perturb x by 1, by how much does $f\big(g(x)\big)$ change?

(under a first-order approx.)

The chain rule

Where does this appear?

Consider the chain rule

$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot \underbrace{g'(x) \cdot 1}$$

If I perturb x by 1, by how much does $g(x)$ change?

The chain rule

Where does this appear?

Consider the chain rule

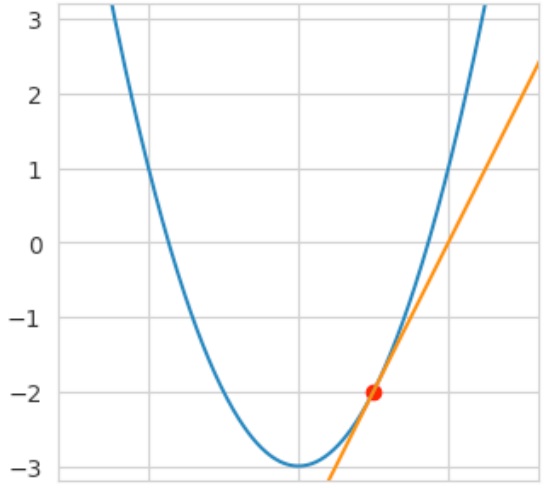
$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot h$$

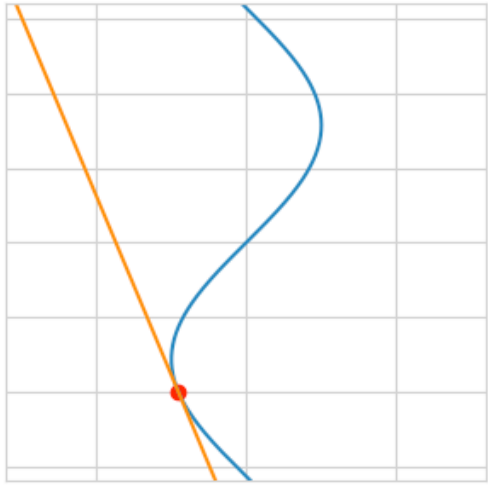
If I perturb $g(x)$ by h, by how much does $f(g(x))$ change?

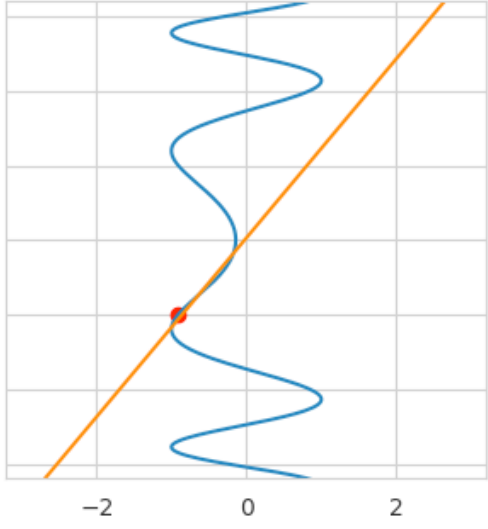
(to the first order)

To find the derivative of a complicated function, we iteratively map input perturbations to output perturbations for each sub-function

Chain rule illustrated







Useful notation

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

$$b = g(x)$$

$$a = f(b)$$

$$\frac{da}{dx} = \frac{da}{db}\frac{db}{dx}$$

Corresponds to code

```
b = x ** 2
a = log(b)
```

Useful notation

```
b = x ** 2
a = log(b)
```

$$b = g(x)$$

$$a = f(b)$$

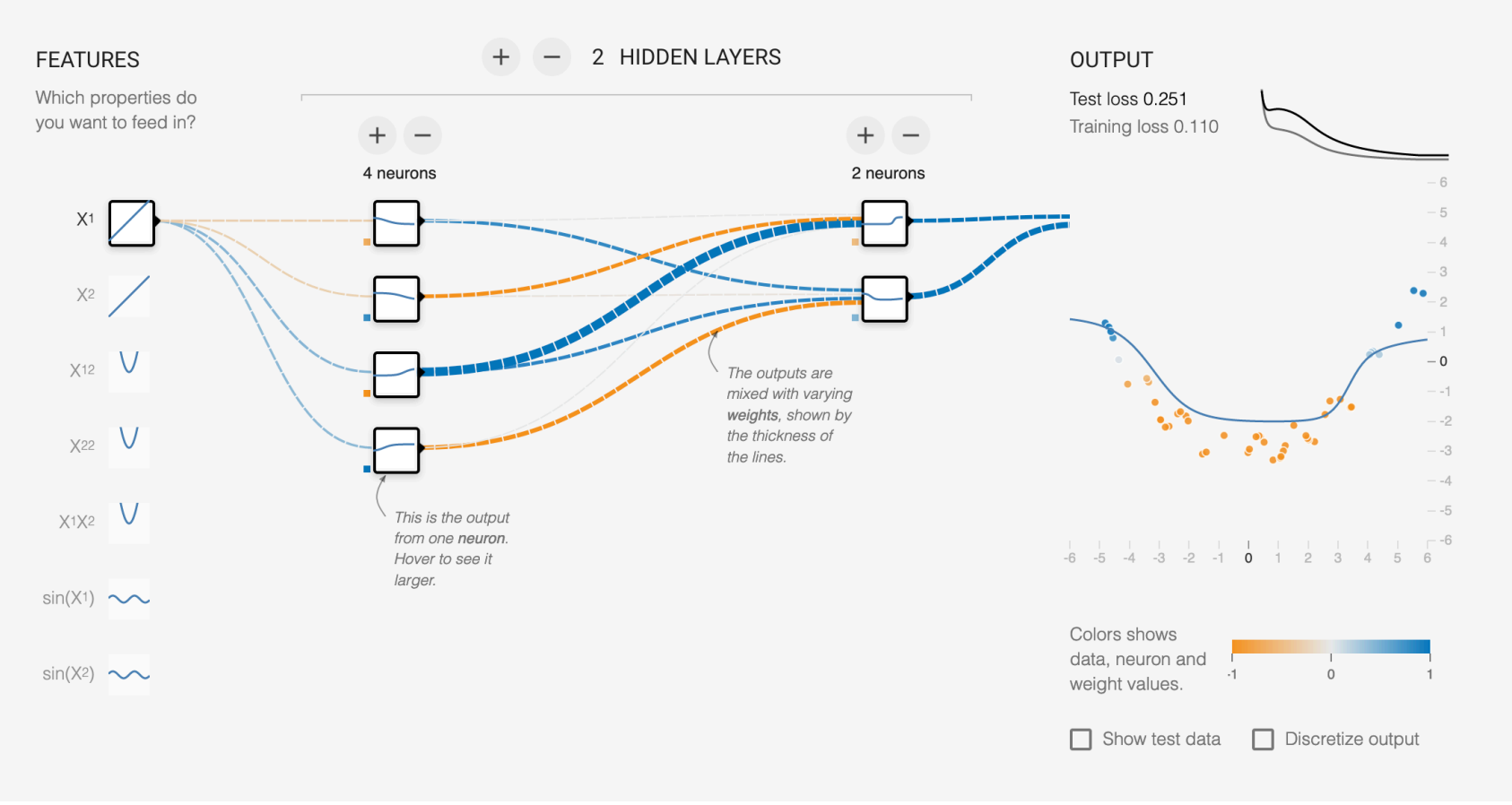
$$a = \log(b), \quad b = x^2$$

$$\frac{da}{dx} = \frac{da}{db}\frac{db}{dx}$$

$$\frac{da}{db} = \qquad \frac{db}{dx} =$$

$$\frac{da}{dx} =$$

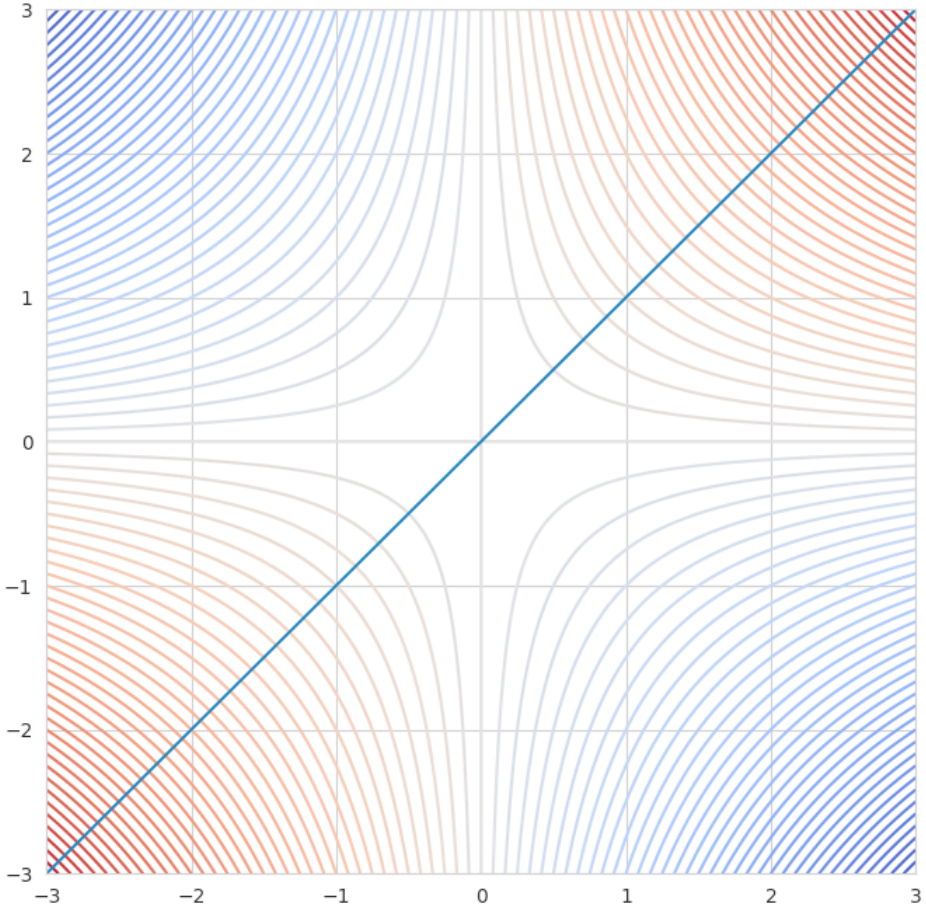
Neural networks



Total derivatives

$$\frac{df(y(x), z(x))}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dz}\frac{dz}{dx}$$

Total derivatives



$$f(x, y) = xy$$

$$\frac{df}{dx} = y \qquad \frac{df}{dy} = x$$

$$\frac{d}{dx} f(x, x) =$$

Total derivatives

$$\frac{df(y(x), z(x))}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dz}\frac{dz}{dx}$$

Total derivatives

$$\frac{df(y(x),x)}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dx}$$

Partial derivatives

$$\frac{df(y(x),x)}{dx} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial x}$$

Composing the chain rule

$$L = -\log \sigma\left(wx^2\right)$$

$$a = x^2$$

$$b = wa$$

$$c = \sigma(b)$$

$$g = \log c$$

$$L = -g$$

Composing the chain rule

$$L = -\log \sigma\left(wx^2\right)$$

$$a = x^2$$

$$b = wa$$

$$c = \sigma(b)$$

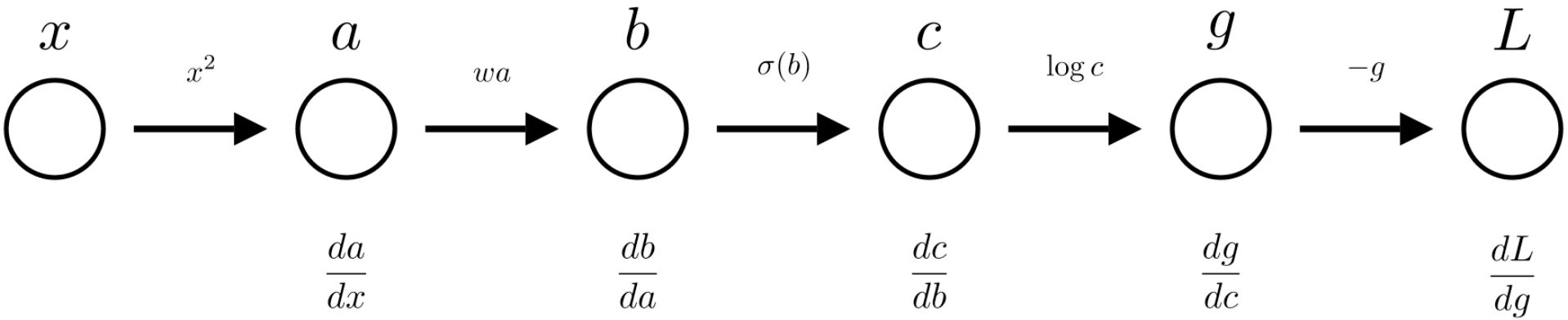
$$g = \log c$$

$$L = -g$$

$$\frac{dL}{dx} = \frac{dL}{dg}\frac{dg}{dc}\frac{dc}{db}\frac{db}{da}\frac{da}{dx}$$

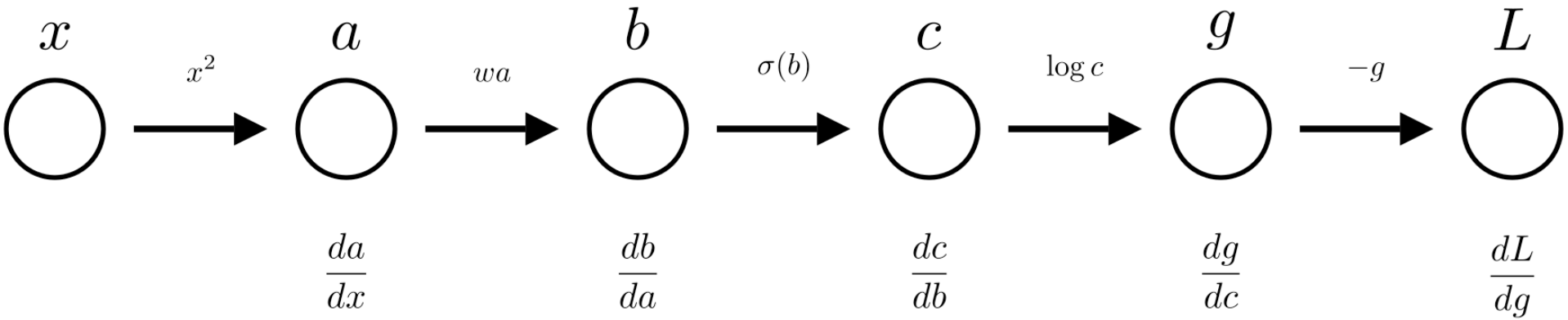
Computational graphs

$$L = -\log \sigma\left(wx^2\right)$$



Forward-mode automatic differentiation

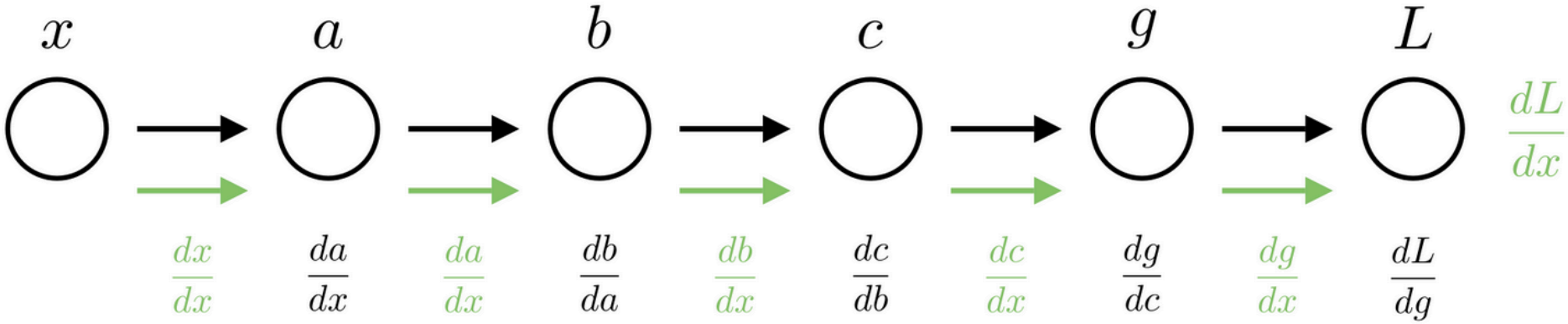
$$L = -\log \sigma\left(wx^2\right)$$



Forward-mode automatic differentiation

$$L = -\log \sigma\left(wx^2\right)$$

Forward mode



Automatic differentiation with multiple inputs

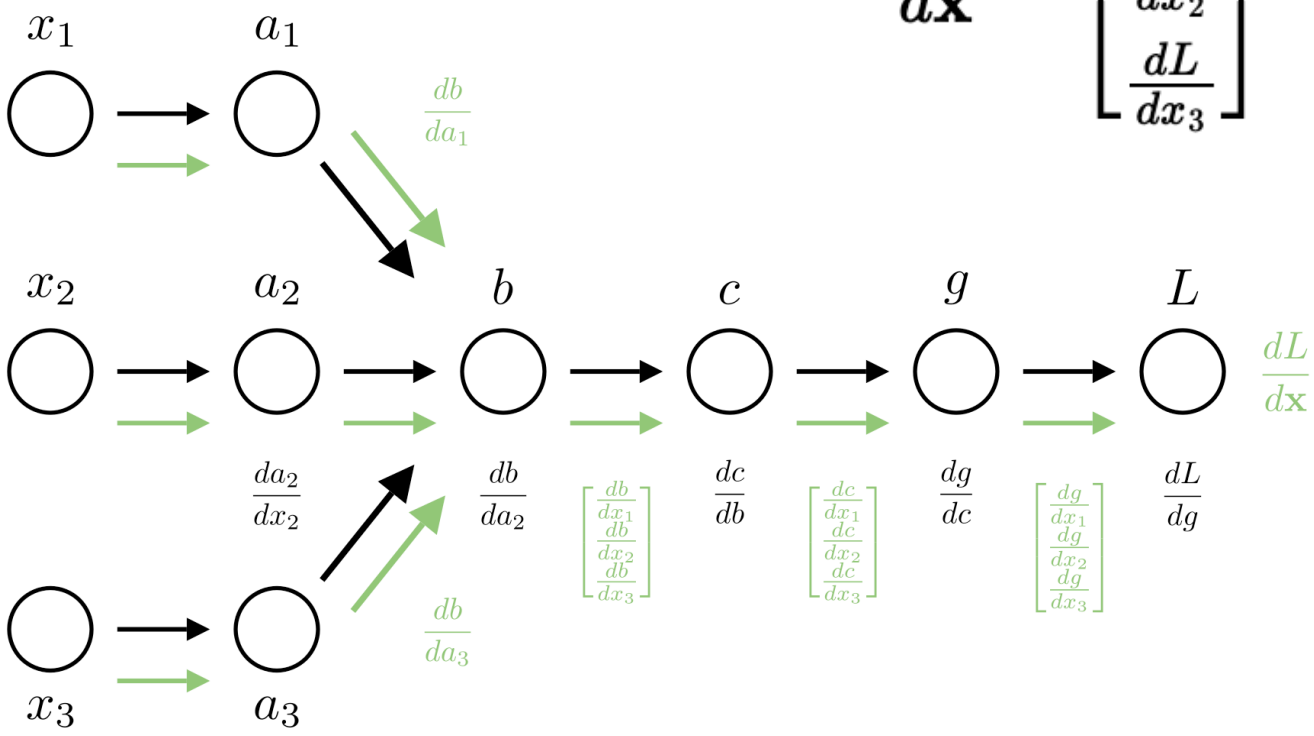
$$-\log \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

$$\frac{dL}{d\mathbf{x}} = \begin{bmatrix} \frac{dL}{dx_1} \\ \frac{dL}{dx_2} \\ \frac{dL}{dx_3} \end{bmatrix}$$

Automatic differentiation with multiple inputs

$$-\log \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

$$\frac{dL}{d\mathbf{x}} = \begin{bmatrix} \frac{dL}{dx_1} \\ \frac{dL}{dx_2} \\ \frac{dL}{dx_3} \end{bmatrix}$$



Reusing values

```
def loss(x):
    a = x ** 2
    b = 5 * a
    c = log(a)
    g = b * c
    L = -g
    return L
```

$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = -b$$

Reusing values

$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = -b$$

Reusing values

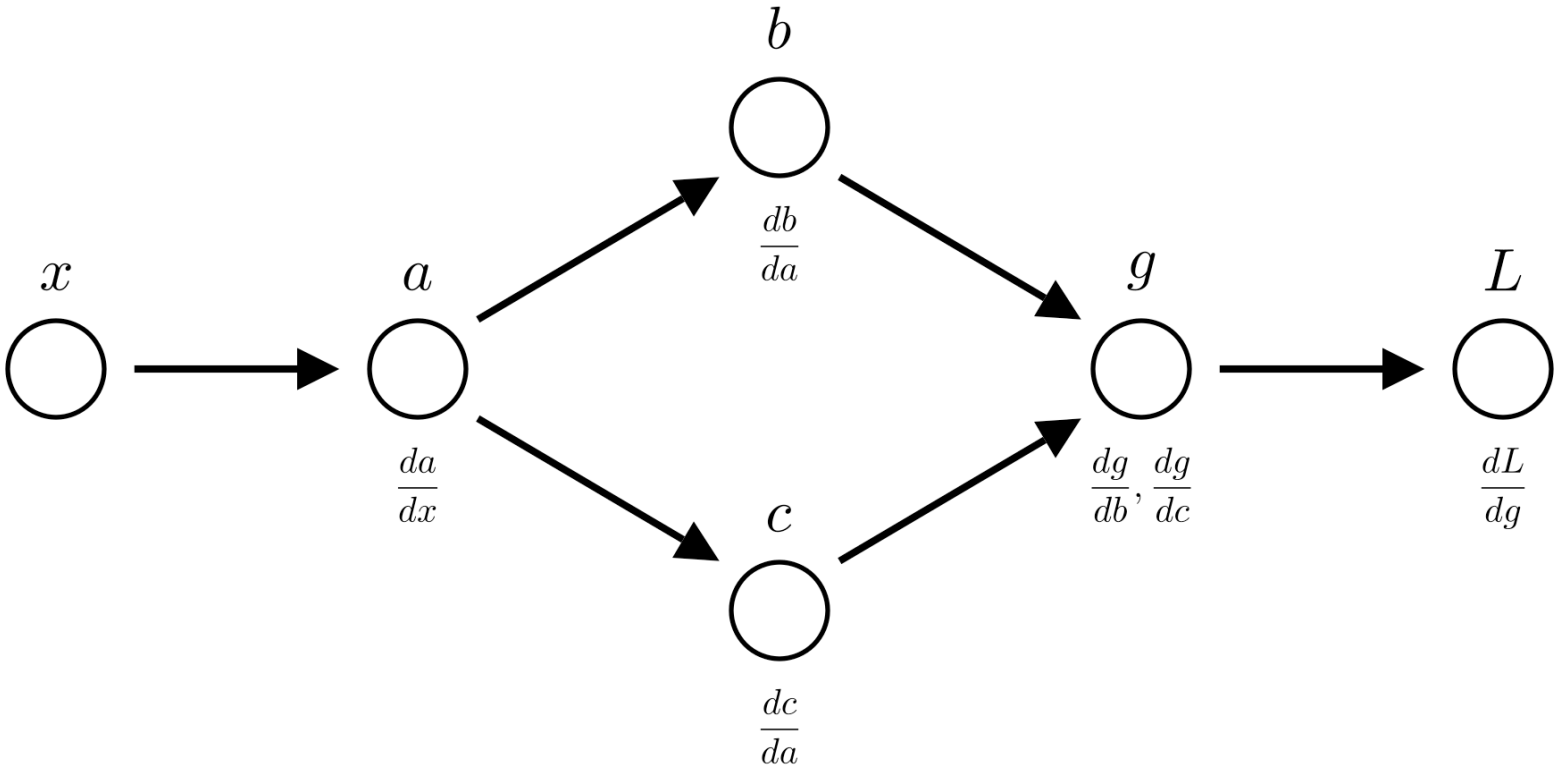
$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = -b$$



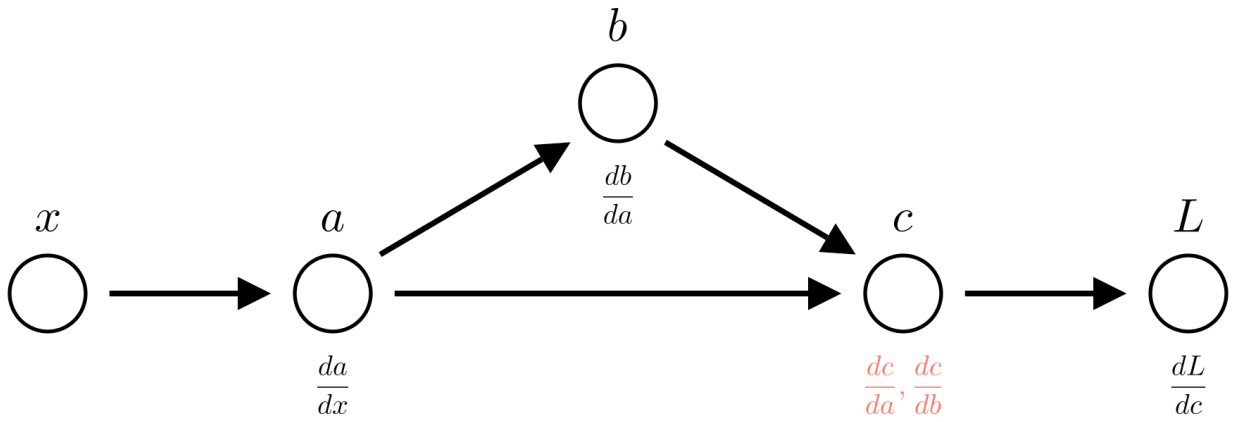
Partial derivatives revisited

$$a = x^2$$

$$b = 5a$$

$$c = ab$$

$$L = -c$$



$$\frac{dc}{da} =$$

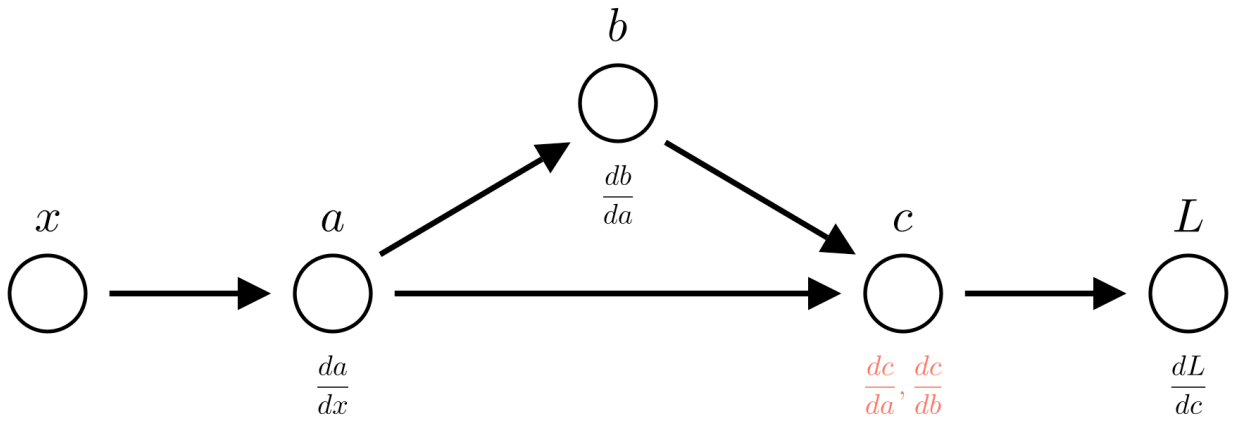
Partial derivatives revisited

$$a = x^2$$

$$b = 5a$$

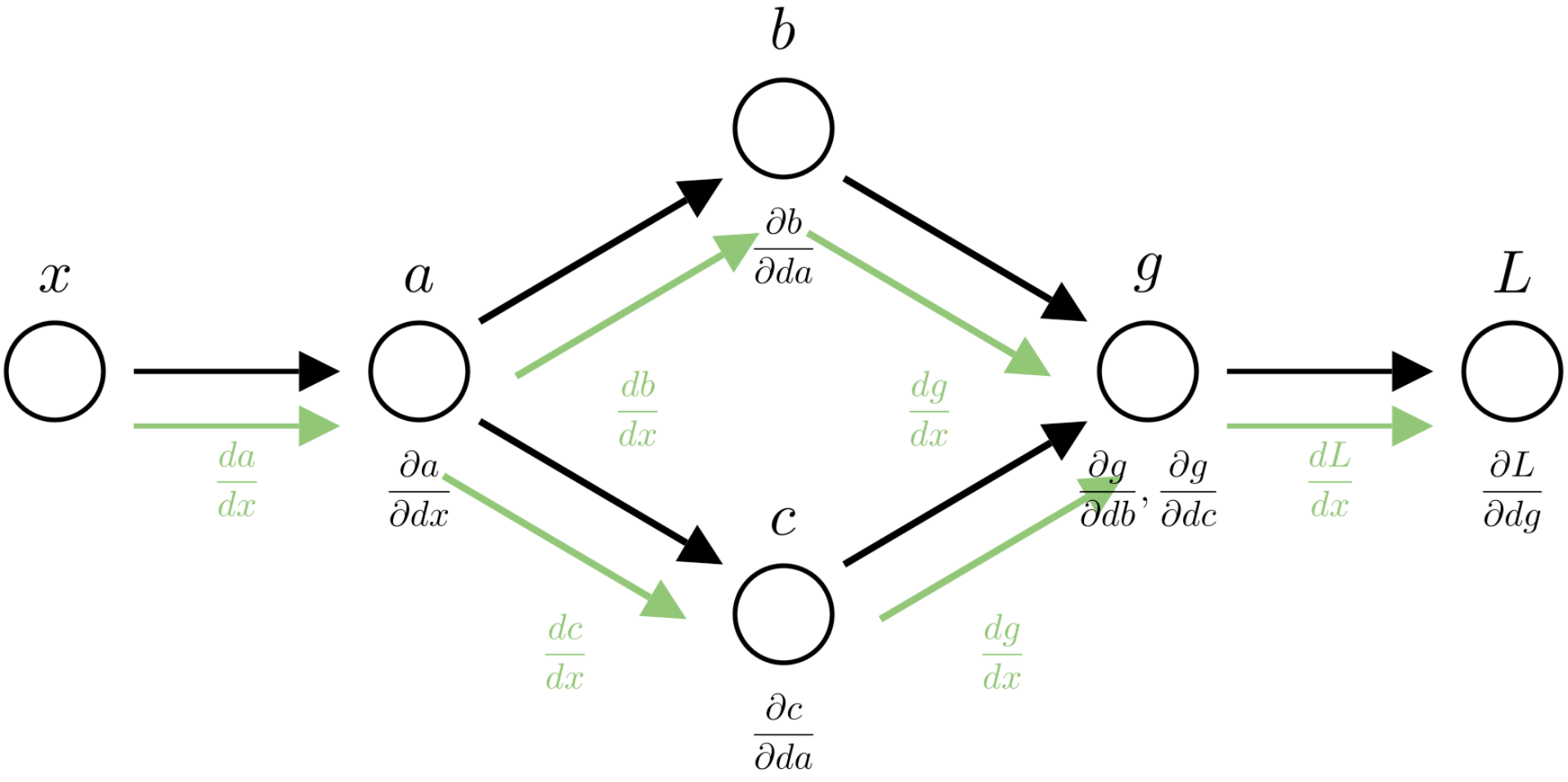
$$c = ab$$

$$L = -c$$

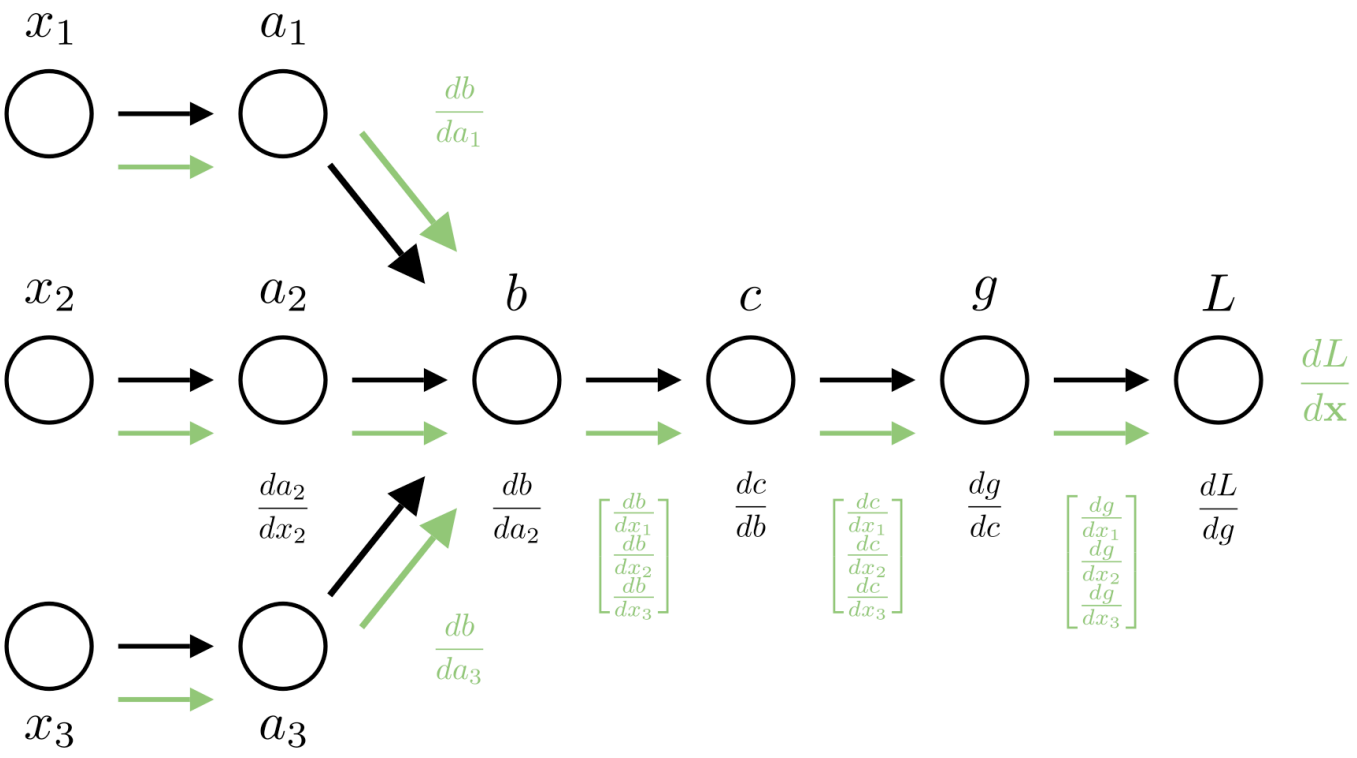


$$\frac{dc}{da} = \frac{\partial c}{\partial a} + \frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = 5a + 5a = 10a$$

Partial derivatives revisited

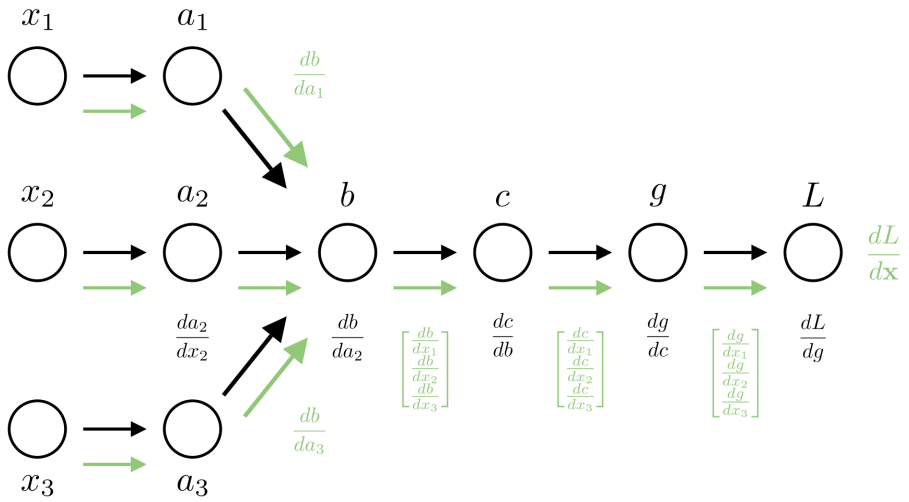


Implementing automatic differentiation



Implementing automatic differentiation

```python
class ForwardValue:
    '''
    Base class for automatic differentiation operations. Represents variable delcaration.
    Subclasses will overwrite func and grads to define new operations.

    Properties:
        parent_values    (list): A list of raw values of each input (as floats)
        value   (float): The value of the result of this operation
        forward_grads (dict): A dictionary mapping inputs to gradients
    '''

    def __init__(self, *args):
        self.parent_values = [arg.value if isinstance(arg, ForwardValue) else arg for arg in args]
        self.value = self.forward_pass(args)

        if len(self.forward_grads.keys()) == 0:
            self.forward_grads = {self: 1}


    def func(self, input):
        '''
        Compute the value of the operation given the inputs.
        For declaring a variable, this is just the identity function (return the input).

        Args:
            input (float): The input to the operation
        Returns:
            value (float): The result of the operation
        '''
        return input

    def grads(self, *args):
        '''
        Compute the derivative of the operation with respect to each input.
        In the base case the derivative of the identity function is just 1. (da/da = 1).

        Args:
            input (float): The input to the operation
        Returns:
            grads (tuple): The derivative of the operation with respect to each input
                           Here there is only a single input, so we return a length-1 tuple.
        '''
        return (1,)

    def forward_pass(self, args):
        # Calls func to compute the value of this operation
        self.forward_grads = {}
        return self.func(*self.parent_values)
```



Implementing automatic differentiation

```python
class _add(ForwardValue):
    # Addition operator (a + b)
    def func(self, a, b):
        return a + b

    def grads(self, a, b):
        return 1., 1.

class _neg(ForwardValue):
    # Negation operator (-a)
    def func(self, a):
        return -a

    def grads(self, a):
        return (-1.,)

class _sub(ForwardValue):
    # Subtraction operator (a - b)
    def func(self, a, b):
        # Your code here

    def grads(self, a, b):
        # Your code here
```