DEEP Q LEARNING AND AUTOMATIC DIFFERENTIATION

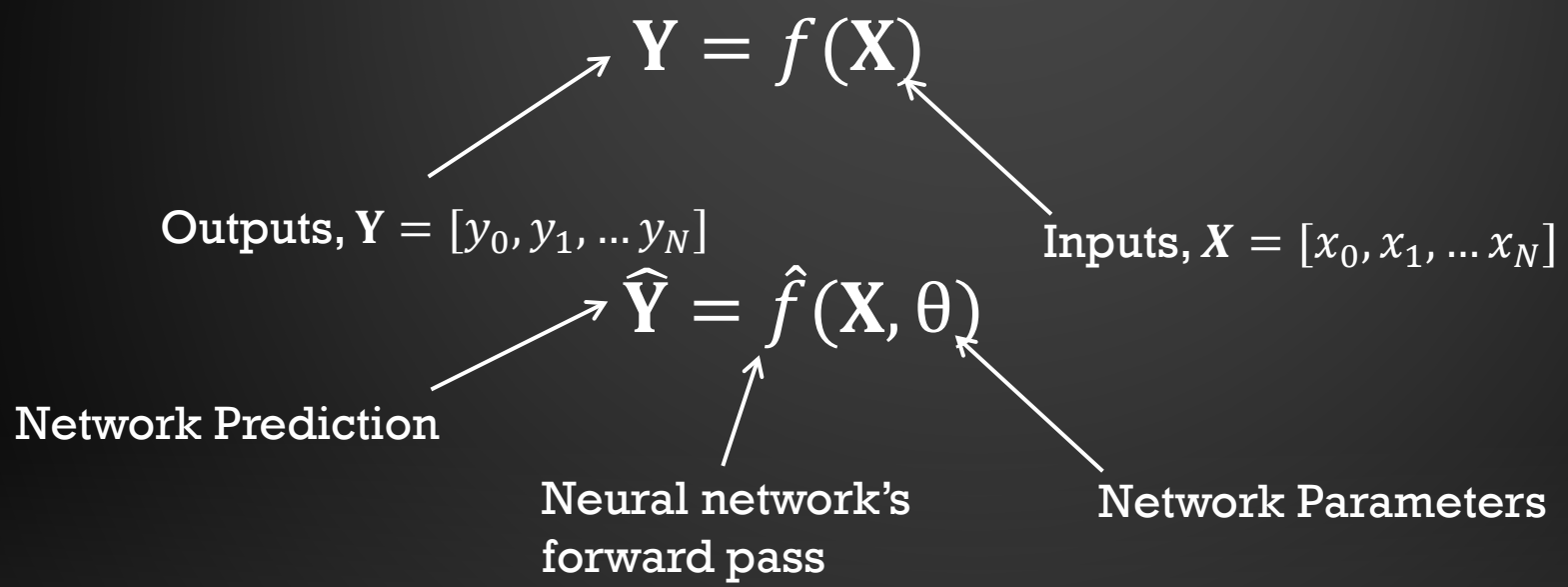
COM3240 GUEST LECTURE

IAN VIDAMOUR

OVERVIEW

- Recap on neural networks as function approximators
- Deep Q learning
- Avoiding training instability
- Automatic differentiation
- Forward-mode AutoDiff
- Reverse-mode AutoDiff

NEURAL NETWORKS AS FUNCTION APPROXIMATORS



NEURAL NETWORKS AS FUNCTION APPROXIMATORS

We would like:

$$\mathbf{Y} = \hat{\mathbf{Y}}$$

Define a loss function:

$$\mathcal{L} \propto (\mathbf{Y} - \hat{\mathbf{Y}})^2$$

Change parameters
to minimise loss:

$$\Delta\theta \propto -\frac{\partial \mathcal{L}}{\partial \theta}$$

NEURAL NETWORKS AND REINFORCEMENT LEARNING

Q learning:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

$$Q(s,a) = f(s,a)$$

Neural Network:

$$Q(s,a) = \hat{f}(s,a;\theta)$$

NEURAL NETWORKS AND REINFORCEMENT LEARNING

Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Target

Prediction

Predicting part of its own output?

Nonstationary target?

Updates biased towards single datapoints?

MANAGING NONSTATIONARY TARGETS

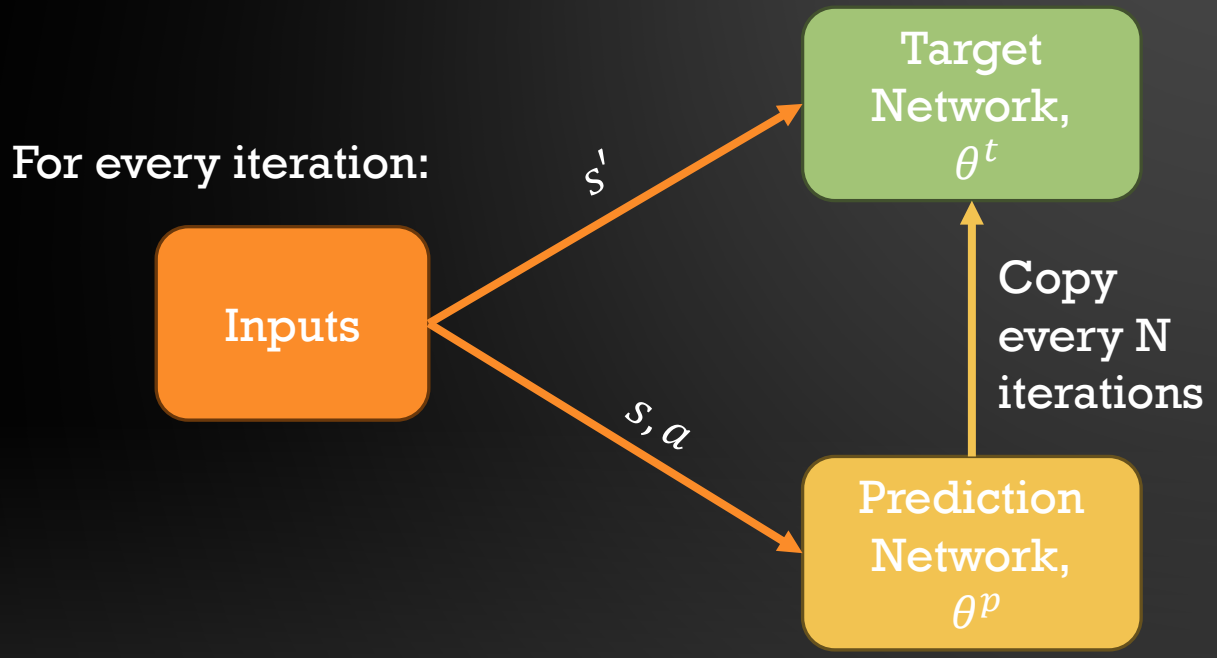
Chasing a moving target causes unstable learning

Can we fix the target?

Double Q learning:

- Clone the network into two networks
- Fix one copy, use for target prediction
- Update parameters of other copy to minimise loss
- Update fixed network sporadically

MANAGING NONSTATIONARY TARGETS



$$\mathcal{L} = [r + \gamma \max_{a'} Q(s', a'; \theta^t) - Q(s, a; \theta^p)]^2$$

Stationary

Changing

EXPLOITING PAST KNOWLEDGE

Neural networks benefit from lots of data

Batch updates smooth learning

Past information is useful!

→ Store experiences to help in future updates

EXPLOITING PAST KNOWLEDGE

Loss function:

$$\mathcal{L} = [r + \gamma \max_{a'} Q(s', \overset{?}{a'}; \theta^t) - Q(s, a; \theta^p)]^2$$

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

AUTOMATIC DIFFERENTIATION

Symbolic Differentiation

+ Software calculates derivatives
+ Provides general solutions
- Problems with expression swell
- Static→ Conditionals, loops, and recursion not suitable

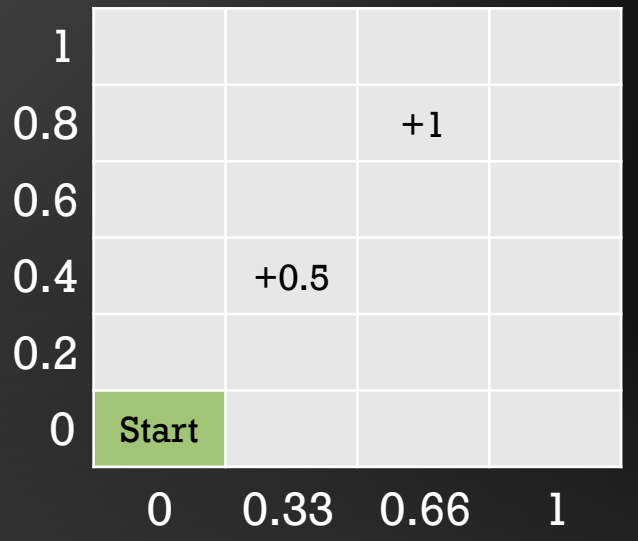
Numerical Differentiation

+ Conceptually simple
- Errors dependent upon step size
- Number of calculations scales poorly

BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

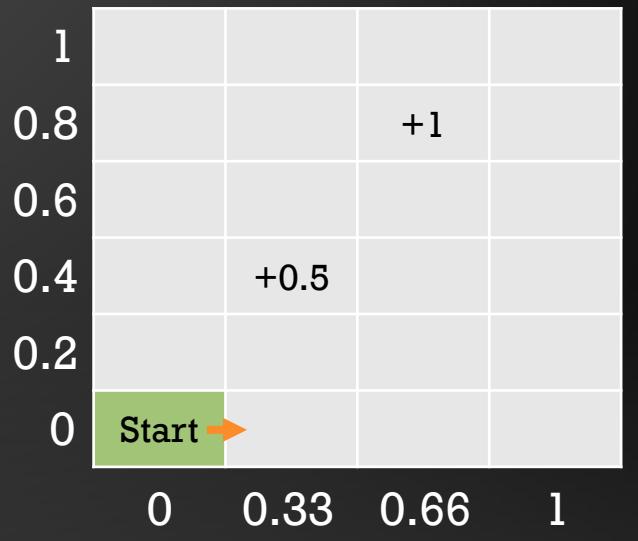
s	a	r	s’



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

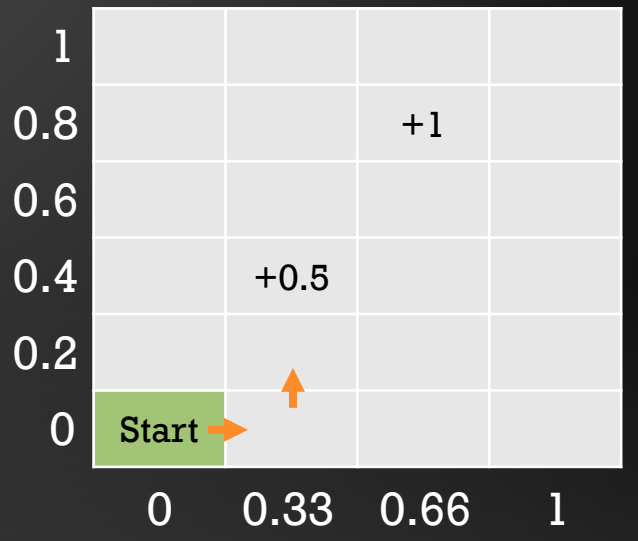
s	a	r	s’
(0,0)	(0,1,0,0)	0	(0.33,0)



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

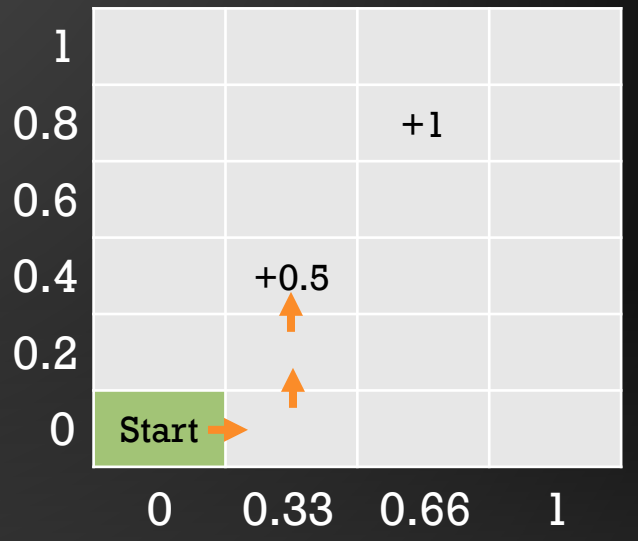
s	a	r	s'
(0,0)	(0,1,0,0)	0	(0.33,0)
(0.33,0)	(1,0,0,0)	0	(0.33,0.2)



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

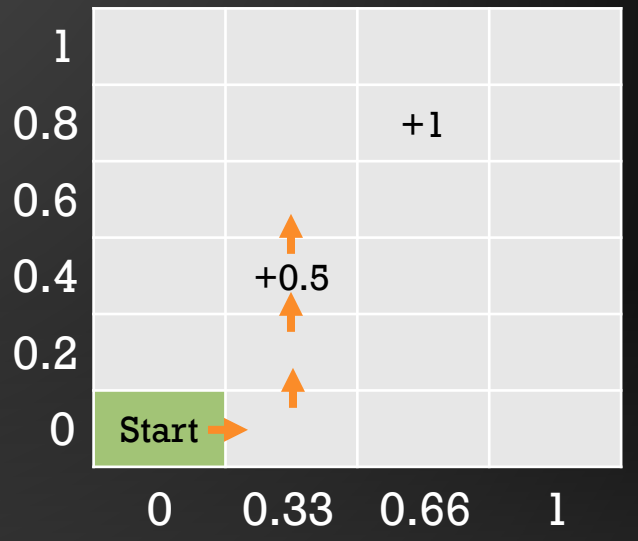
s	a	r	s'
(0,0)	(0,1,0,0)	0	(0.33,0)
(0.33,0)	(1,0,0,0)	0	(0.33,0.2)
(0.33, 0.2)	(1,0,0,0)	0.5	(0.33,0.4)



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

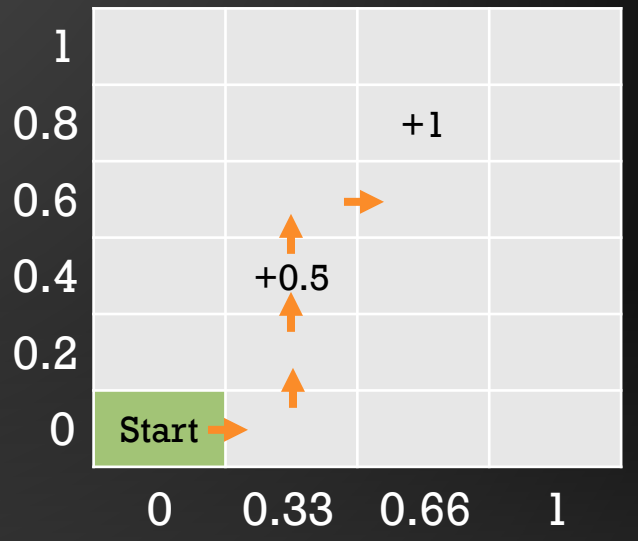
s	a	r	s'
(0,0)	(0,1,0,0)	0	(0.33,0)
(0.33,0)	(1,0,0,0)	0	(0.33,0.2)
(0.33, 0.2)	(1,0,0,0)	0.5	(0.33,0.4)
(0.33,0.4)	(1,0,0,0)	0	(0.33,0.6)



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

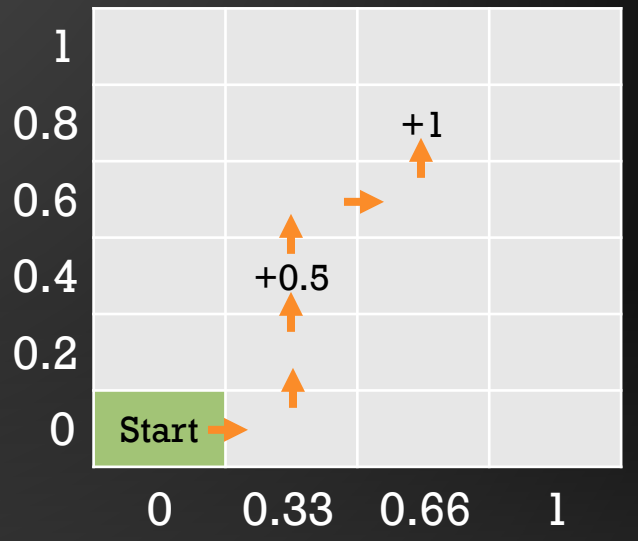
s	a	r	s'
(0,0)	(0,1,0,0)	0	(0.33,0)
(0.33,0)	(1,0,0,0)	0	(0.33,0.2)
(0.33, 0.2)	(1,0,0,0)	0.5	(0.33,0.4)
(0.33,0.4)	(1,0,0,0)	0	(0.33,0.6)
(0.33,0.6)	(0,1,0,0)	0	(0.66,0.6)



BUILDING A BUFFER

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

s	a	r	s'
(0,0)	(0,1,0,0)	0	(0.33,0)
(0.33,0)	(1,0,0,0)	0	(0.33,0.2)
(0.33, 0.2)	(1,0,0,0)	0.5	(0.33,0.4)
(0.33,0.4)	(1,0,0,0)	0	(0.33,0.6)
(0.33,0.6)	(0,1,0,0)	0	(0.66,0.6)
(0.66,0.6)	(1,0,0,0)	1	(0.66,0.8)



Continue adding in future episodes, sample for training

EXPLOITING PAST KNOWLEDGE

Loss function:

Online

$$\mathcal{L} = [r + \gamma \max_{a'} Q(s', a'; \theta^t) - Q(s, a; \theta^p)]^2$$

Considering experience buffer:

$$\text{buffer}_t = (s_t, a_t, r_t, s_{t+1})$$

Batch

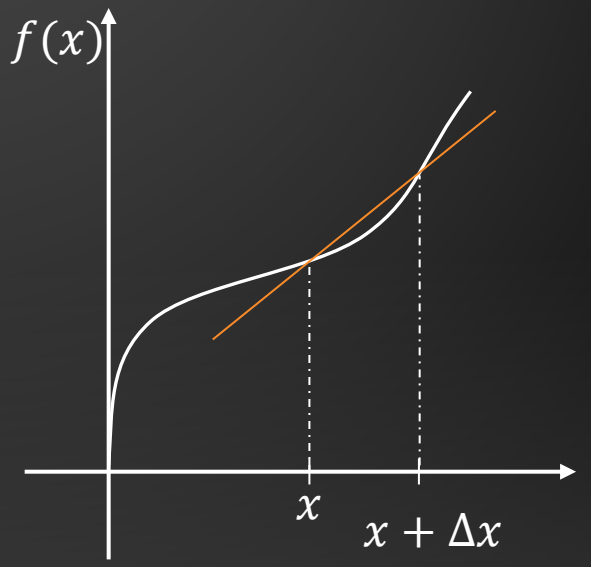
$$\mathcal{L} = \sum_{t=1}^{N} [r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^t) - Q(s_t, a_t; \theta^p)]^2$$

Lots of gradient calculation!

HOW TO CALCULATE OUR DERIVATIVES

Symbolic Differentiation

Numerical Differentiation



AUTOMATIC DIFFERENTIATION

Automatic Differentiation

+ Pass local information numerically (avoid expression swell)
+ Resulting calculation exact
+ Synergises with backpropagation

BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

Intermediate variables

```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```

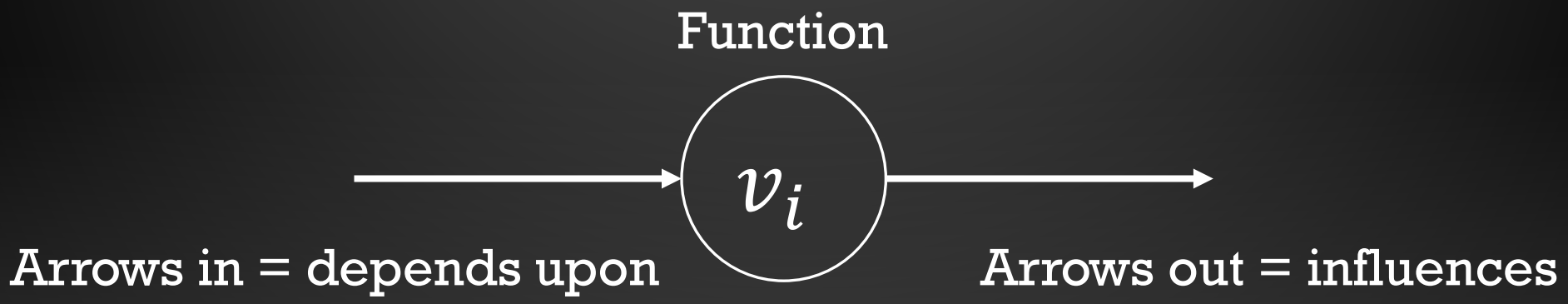
Primitive operations

BUILDING COMPUTATIONAL GRAPHS

Computational graphs built from:

- Intermediate variables
- Primitive operations

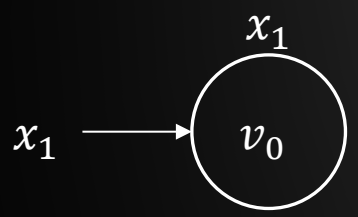
One operation per variable!



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```

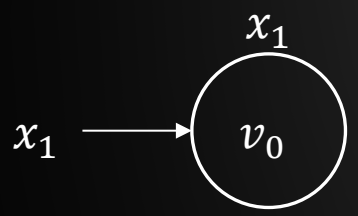


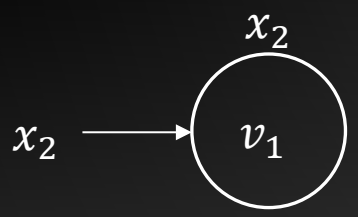
x_2

BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```

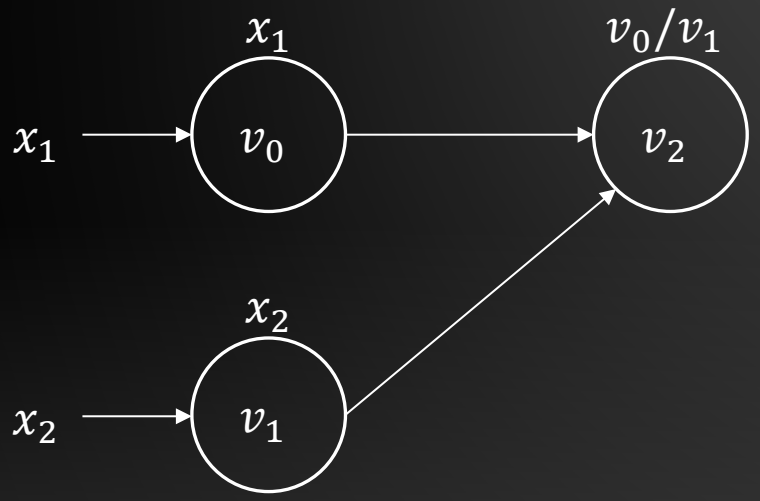




BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

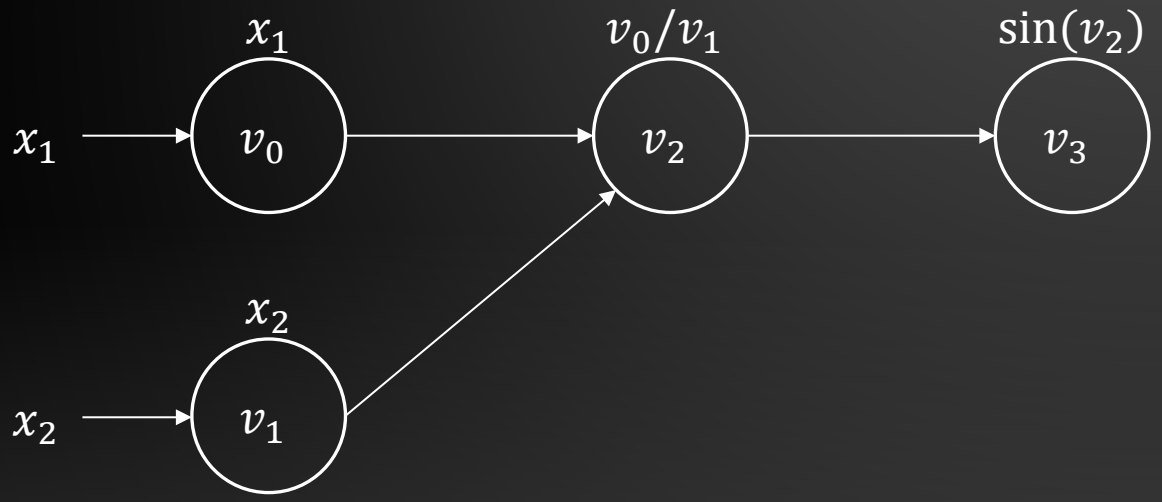
```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

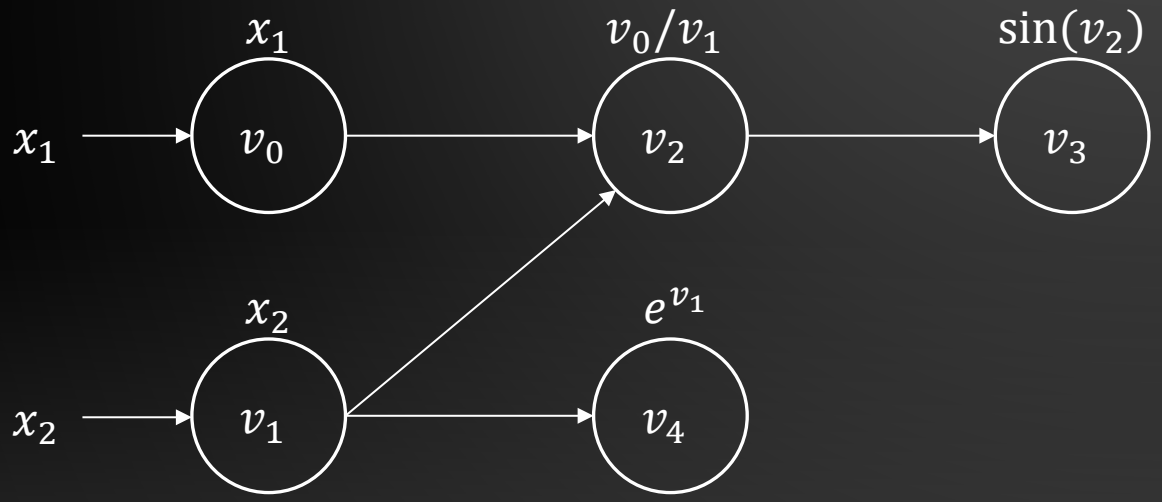
```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

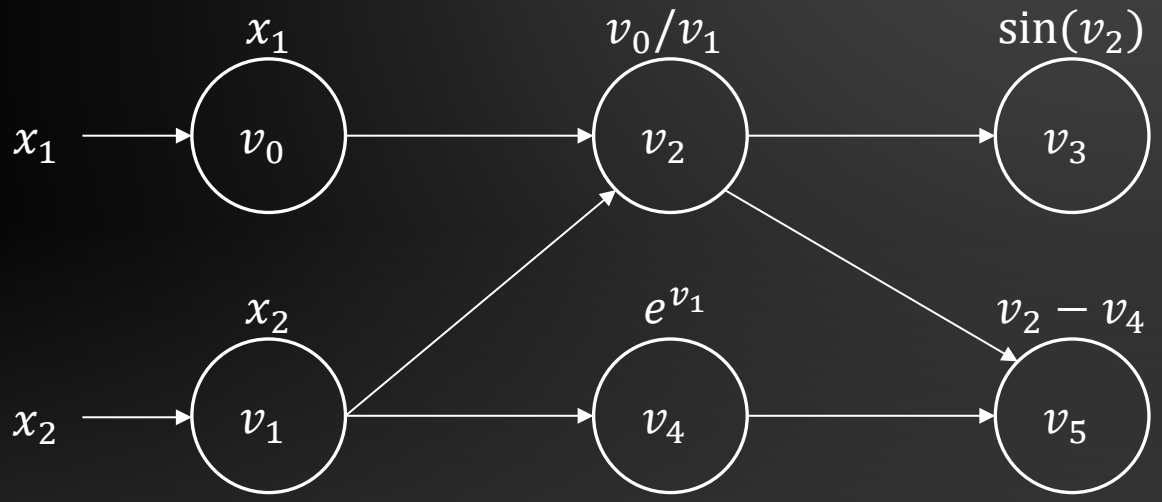
```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

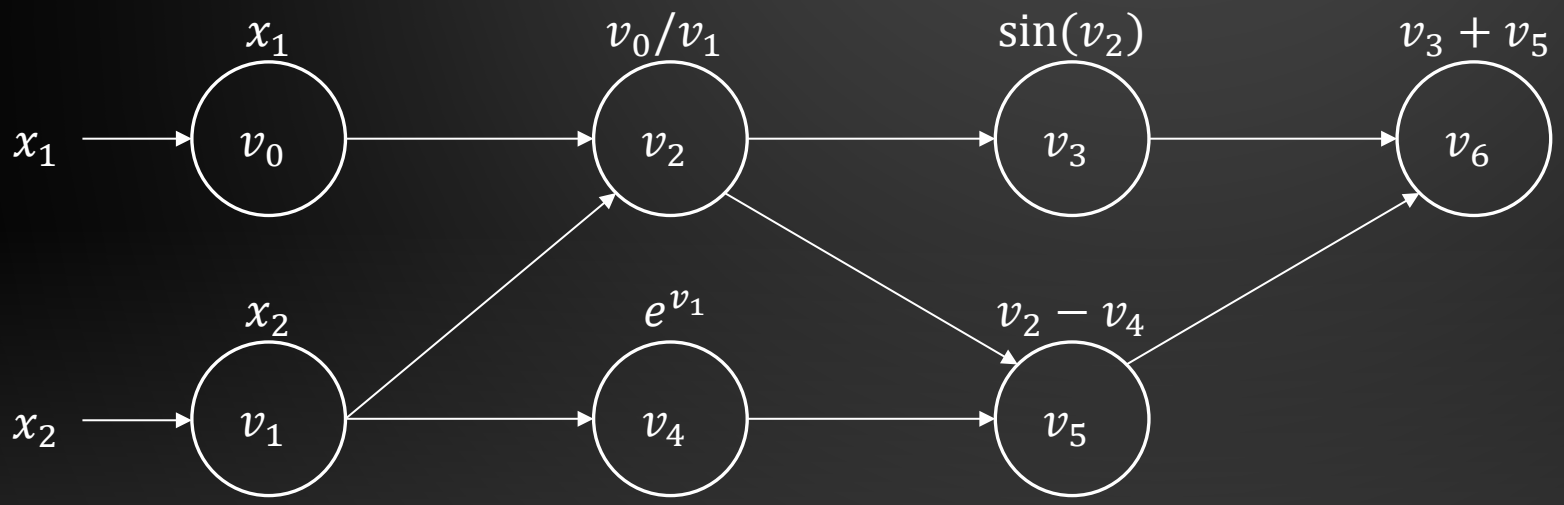
```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

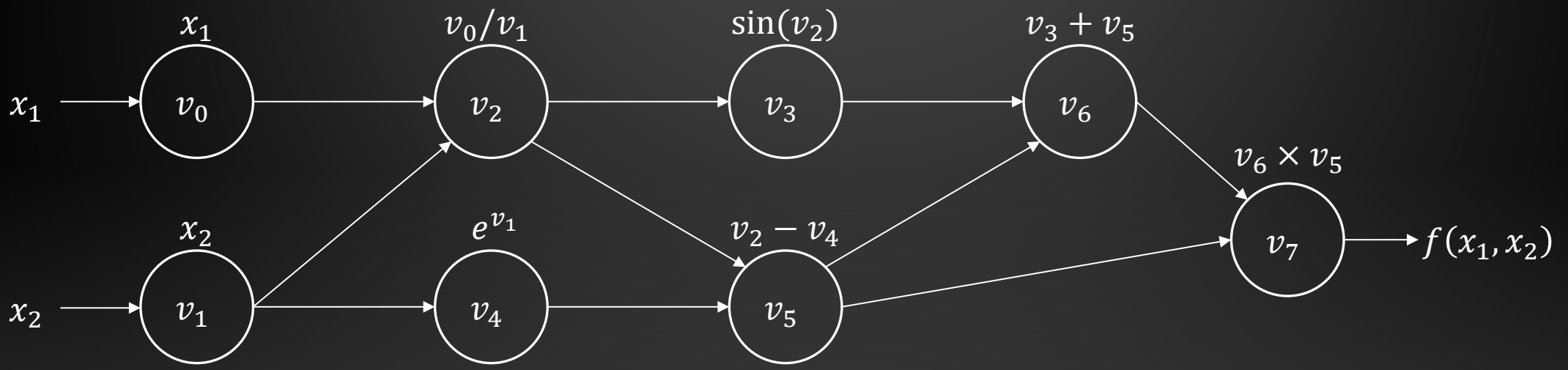
```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



BUILDING COMPUTATIONAL GRAPHS

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2} \right] \times \left[\frac{x_1}{x_2} - e^{x_2} \right]$$

```
def f(x1, x2):
    a = (x1/x2)
    b = np.sin(a)
    c = np.exp(x2)
    return (b + a - c) * (a - c)
```



We have broken down a complex function to a series of simple operations with known derivatives

FORWARD MODE AUTO-DIFF

Forward mode: Calculate x and $\dot{x}$ and pass both through graph.

Perform $\dot{x}$ operations according to gradient rules

$$\frac{\partial(\frac{u}{v})}{\partial x} = \frac{v\frac{\partial u}{\partial x} - u\frac{\partial v}{\partial x}}{v^2}$$

$$\frac{\partial(uv)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$

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

FORWARD MODE AUTO-DIFF

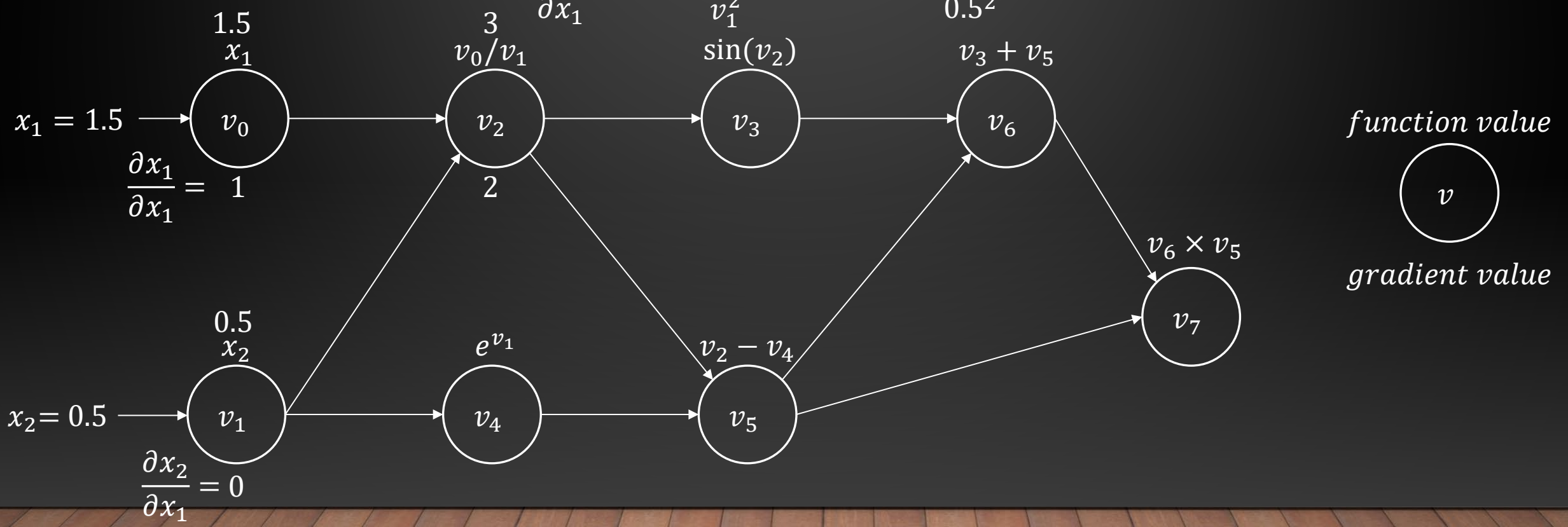
$$\frac{\partial(\frac{u}{v})}{\partial x} = \frac{v\frac{\partial u}{\partial x} - u\frac{\partial v}{\partial x}}{v^2}$$

$$\frac{\partial(uv)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$

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

Considering $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial v_2}{\partial x_1} = \frac{v_1\frac{\partial v_0}{\partial x_1} - v_0\frac{\partial v_1}{\partial x_1}}{v_1^2} = \frac{0.5 * 1 - 1.5 * 0}{0.5^2} = 2$$



FORWARD MODE AUTO-DIFF

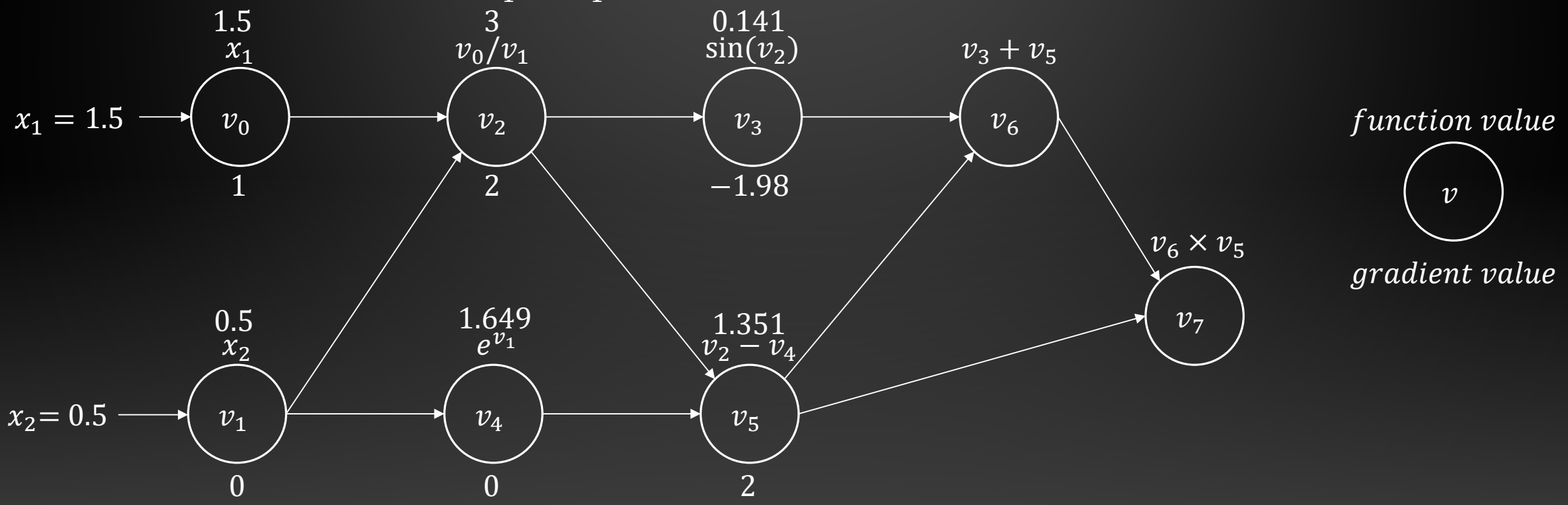
$$\frac{\partial(\frac{u}{v})}{\partial x} = \frac{v\frac{\partial u}{\partial x} - u\frac{\partial v}{\partial x}}{v^2}$$

$$\frac{\partial(uv)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$

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

Considering $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_2}{\partial x_1} * \cos(v_2) = 2 * \cos(3) = -1.98$$



function value

v

gradient value

$$\frac{\partial v_4}{\partial x_1} = 0$$

$$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} = 2$$

FORWARD MODE AUTO-DIFF

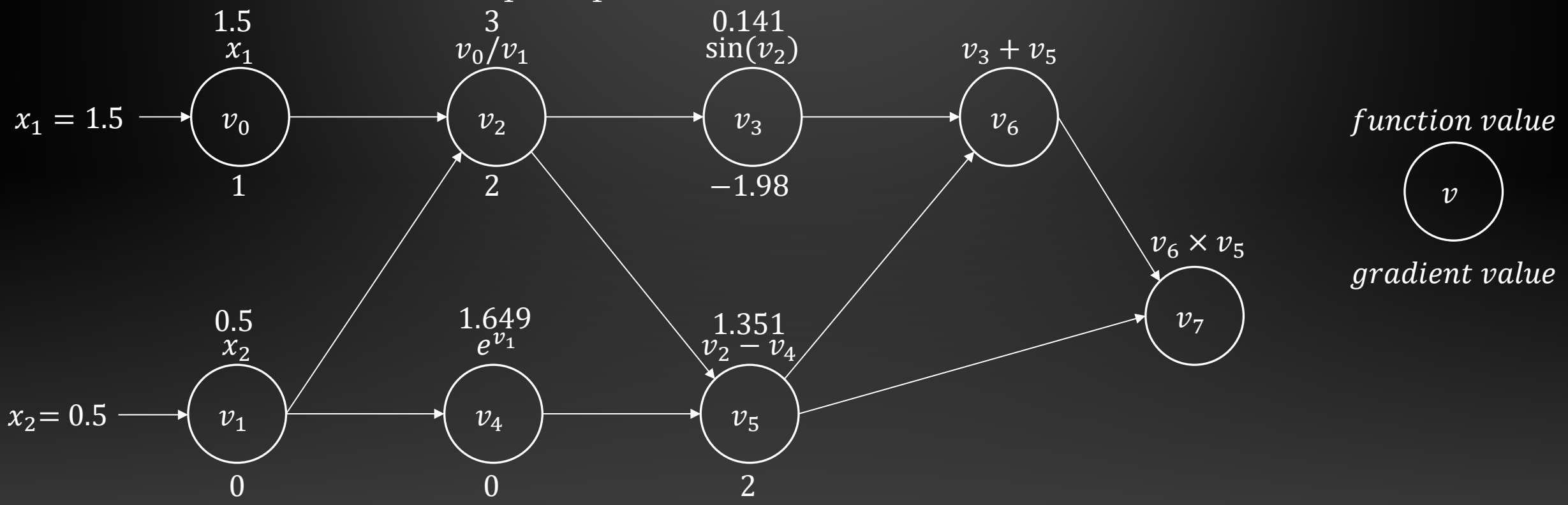
$$\frac{\partial(\frac{u}{v})}{\partial x} = \frac{v\frac{\partial u}{\partial x} - u\frac{\partial v}{\partial x}}{v^2}$$

$$\frac{\partial(uv)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$

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

Considering $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_2}{\partial x_1} * \cos(v_2) = 2 * \cos(3) = -1.98$$



function value

v

gradient value

$$\frac{\partial v_4}{\partial x_1} = 0$$

$$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} = 2$$

FORWARD MODE AUTO-DIFF

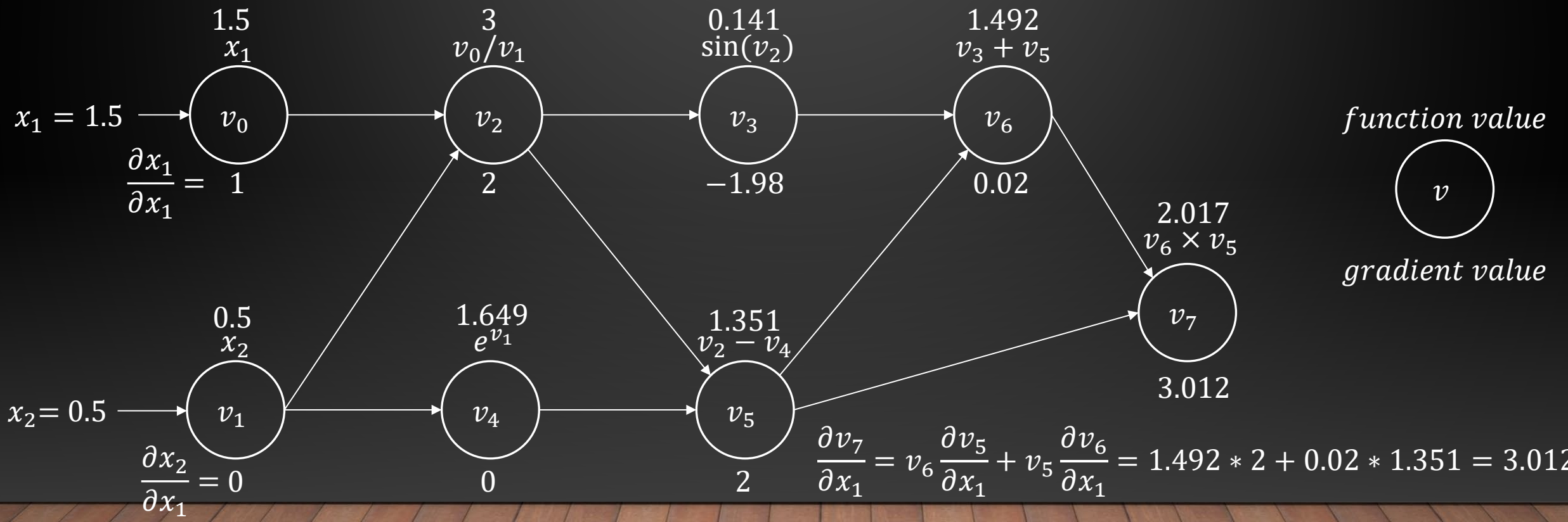
$$\frac{\partial(\frac{u}{v})}{\partial x} = \frac{v\frac{\partial u}{\partial x} - u\frac{\partial v}{\partial x}}{v^2}$$

$$\frac{\partial(uv)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$

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

Considering $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_3}{\partial x_1} + \frac{\partial v_5}{\partial x_1} = 0.02$$



$$\frac{\partial v_7}{\partial x_1} = v_6\frac{\partial v_5}{\partial x_1} + v_5\frac{\partial v_6}{\partial x_1} = 1.492 * 2 + 0.02 * 1.351 = 3.012$$

FORWARD MODE AUTO-DIFF

Consider one input at a time

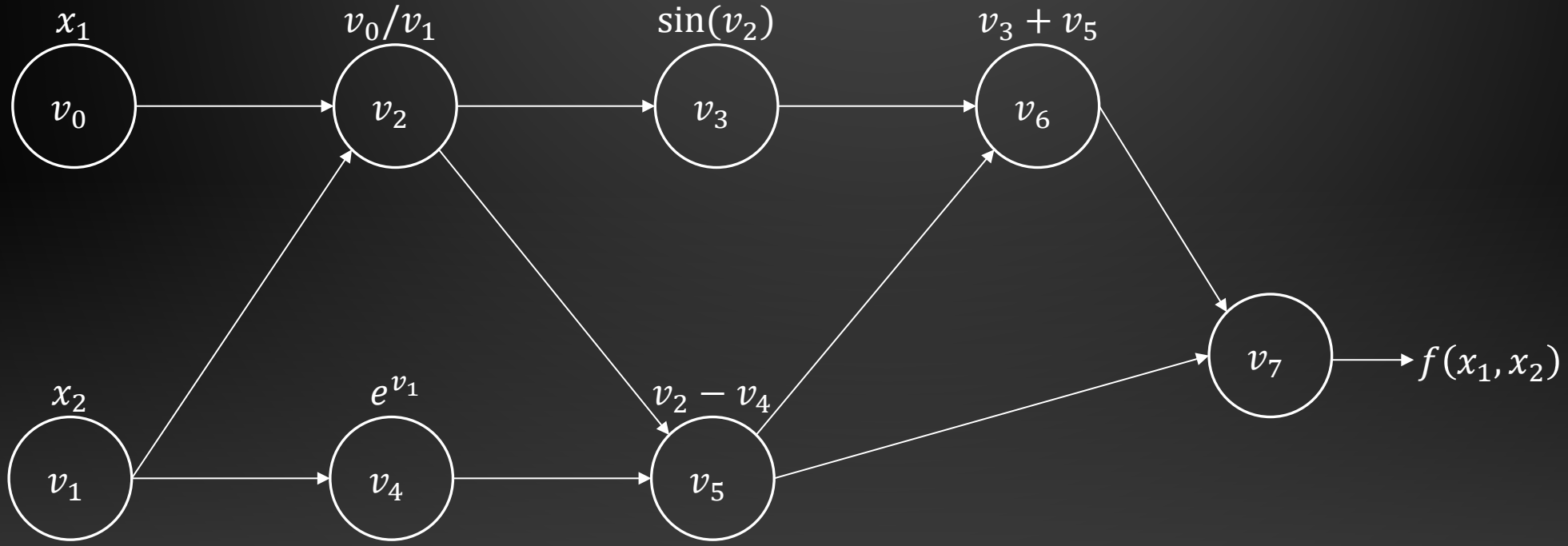
Considering many outputs from one input simple

Useful when number of N outputs > N inputs

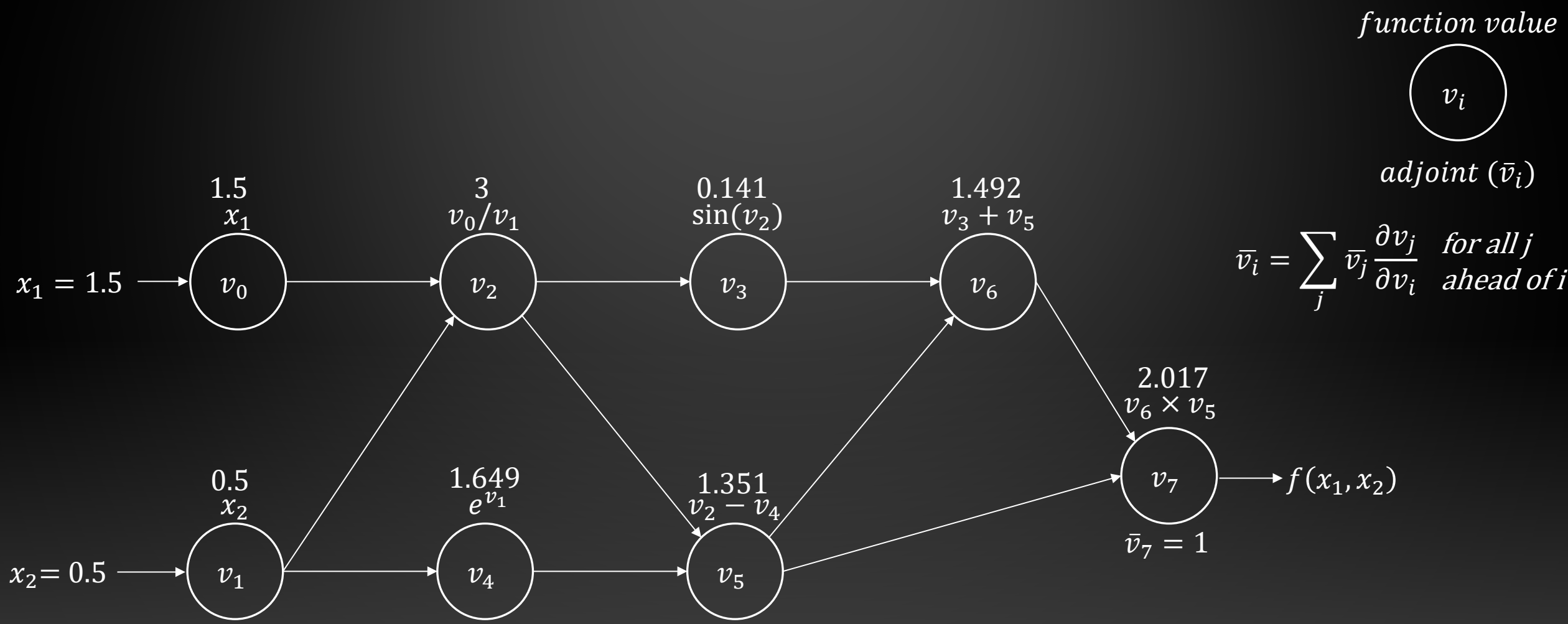
REVERSE MODE AUTO-DIFF

Reverse mode: pass x normally, and store dependencies (computational graph) in memory. Propagate derivatives backwards from the output. (Very similar to backpropagation!)

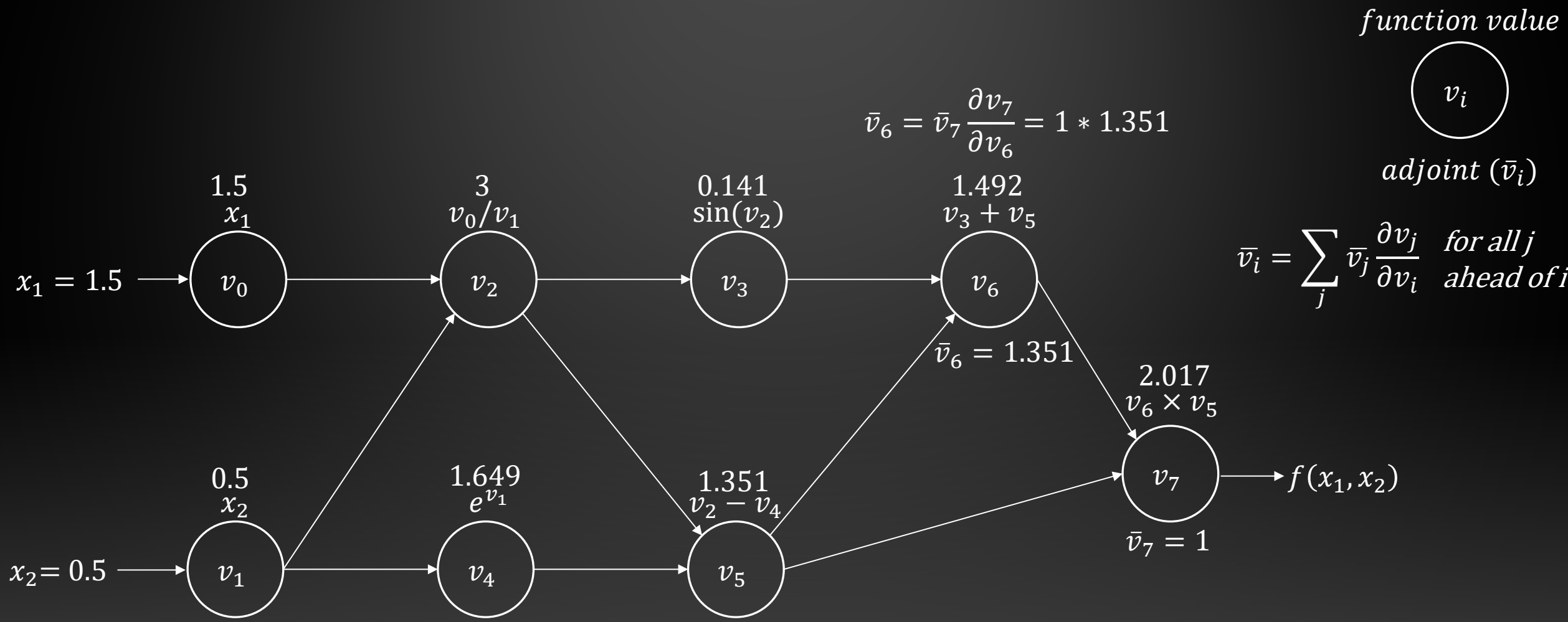
At each point in the network, we consider how all pathways ahead depend upon each intermediate variable:



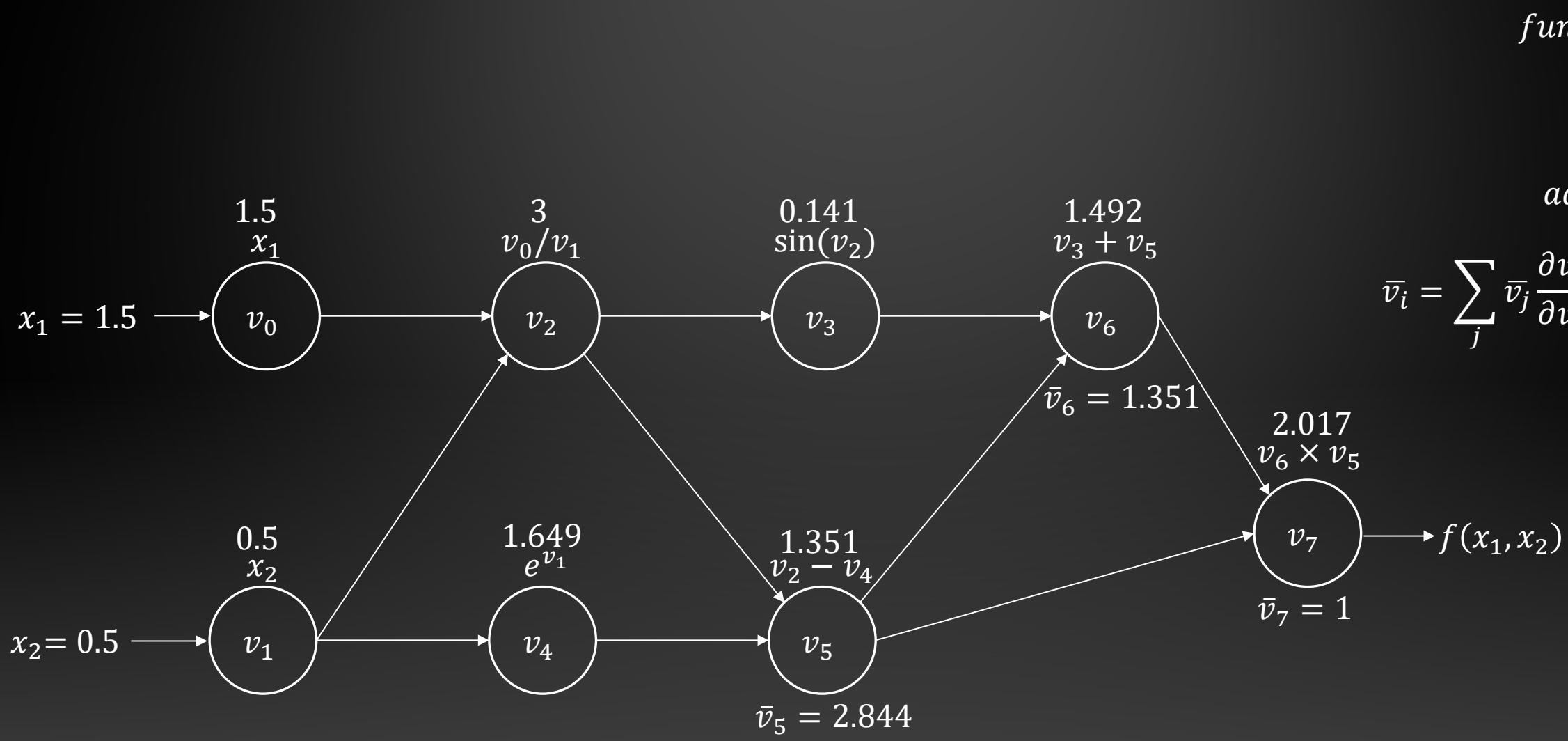
REVERSE MODE AUTO-DIFF



REVERSE MODE AUTO-DIFF

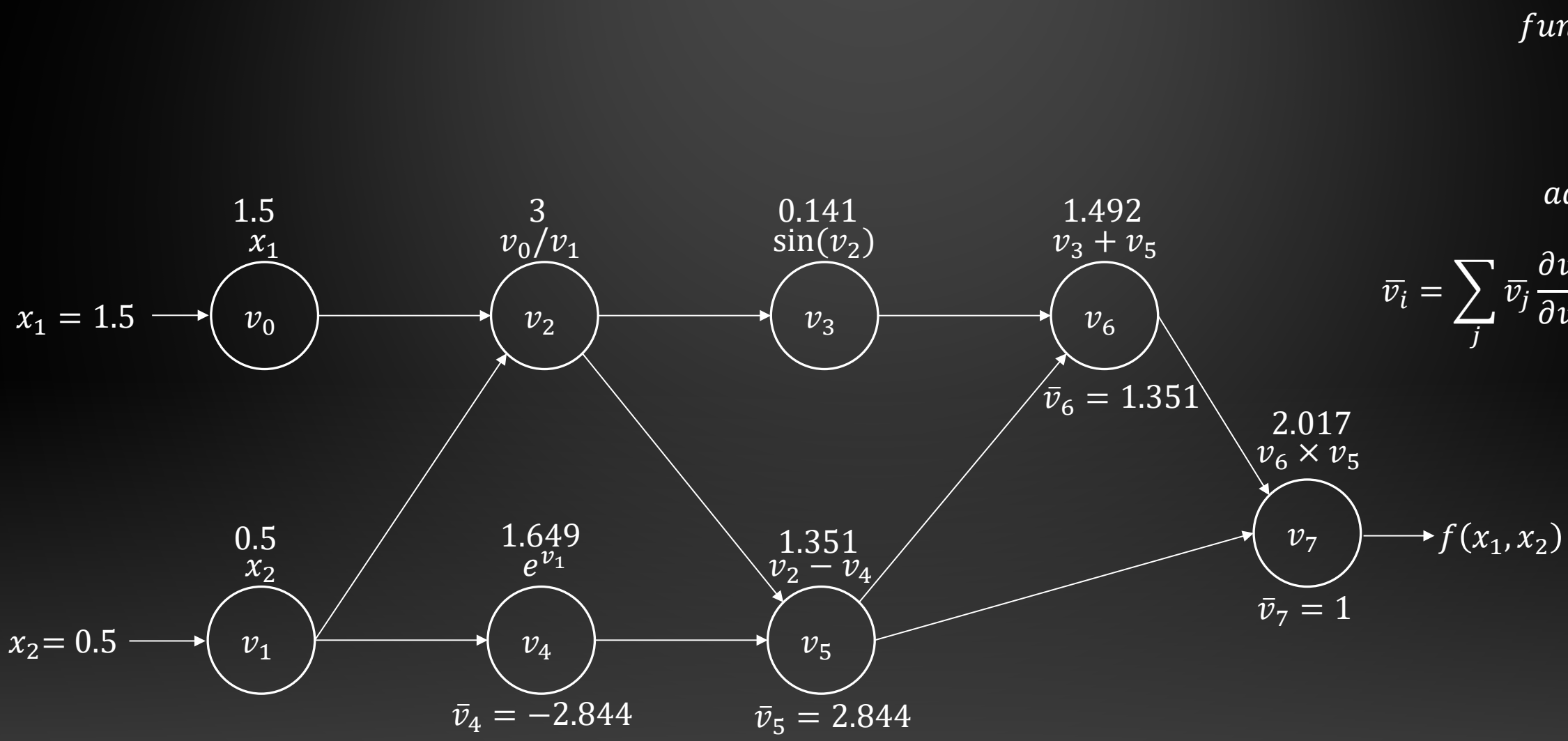


REVERSE MODE AUTO-DIFF



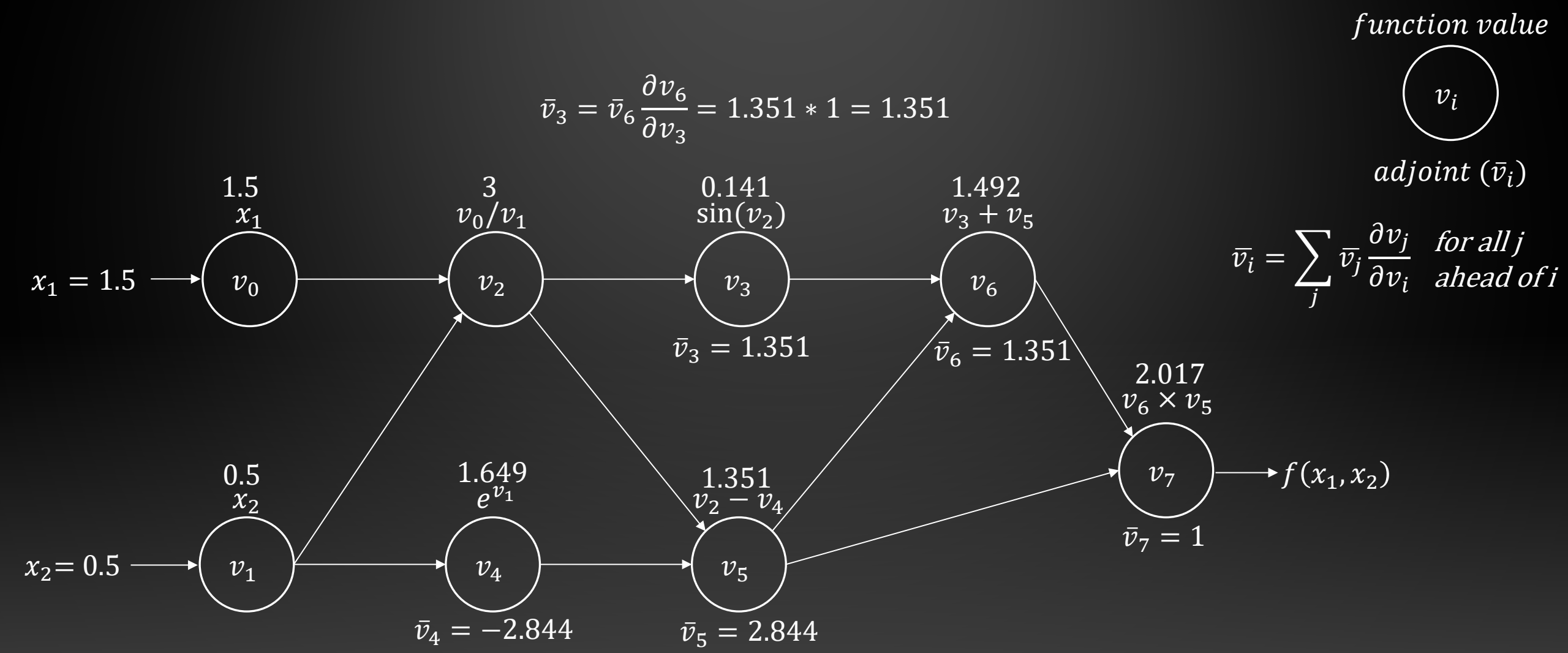
$$\bar{v}_5 = \bar{v}_7 \frac{\partial v_7}{\partial v_5} + \bar{v}_6 \frac{\partial v_6}{\partial v_5} = 1 * 1.492 + 1.351 * 1 = 2.844$$

REVERSE MODE AUTO-DIFF

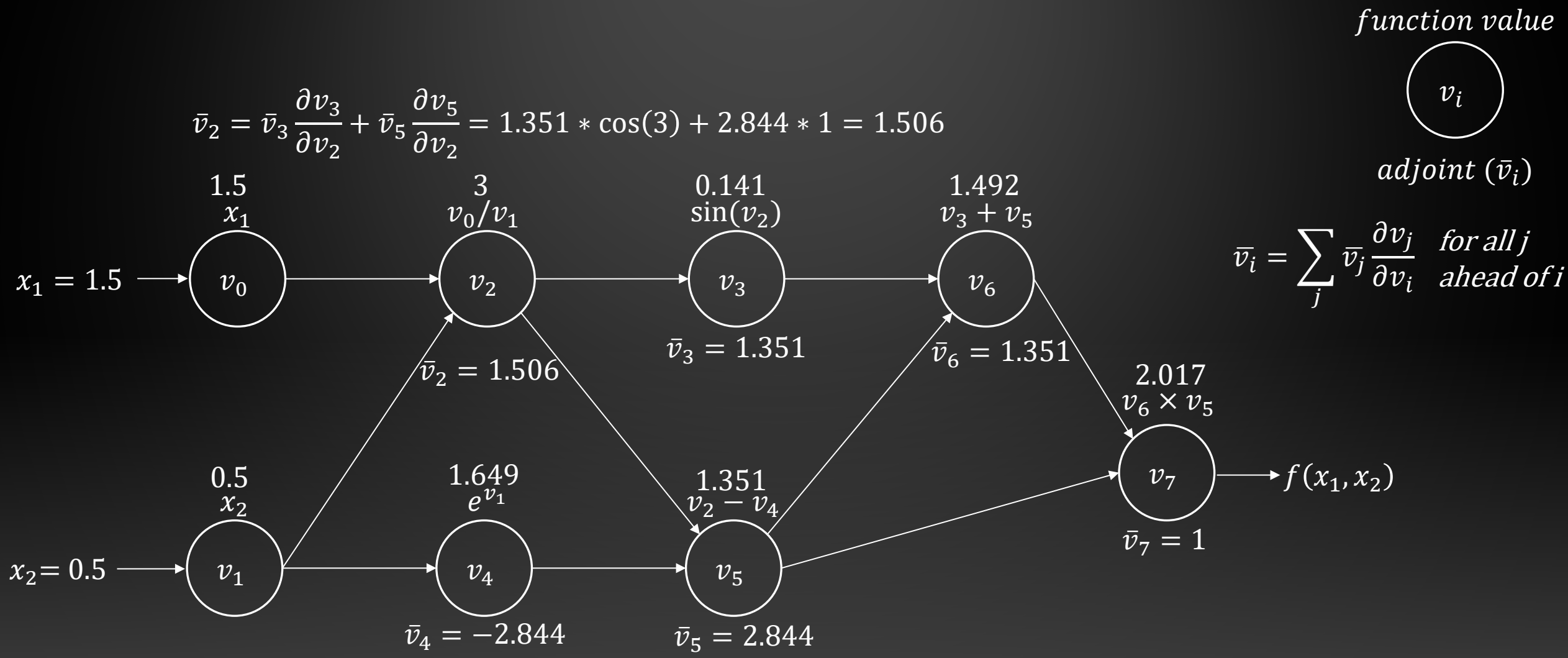


$$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4} = 2.844 * -1$$

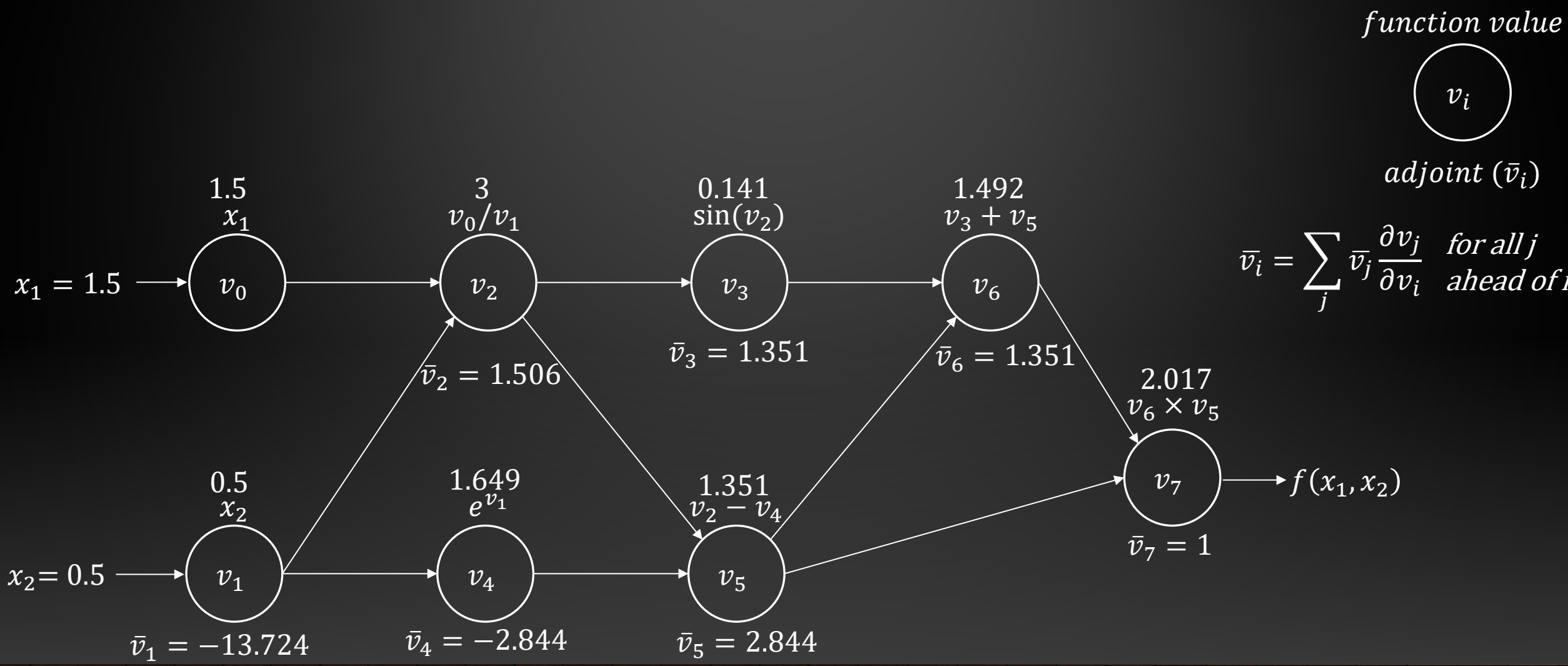
REVERSE MODE AUTO-DIFF



REVERSE MODE AUTO-DIFF

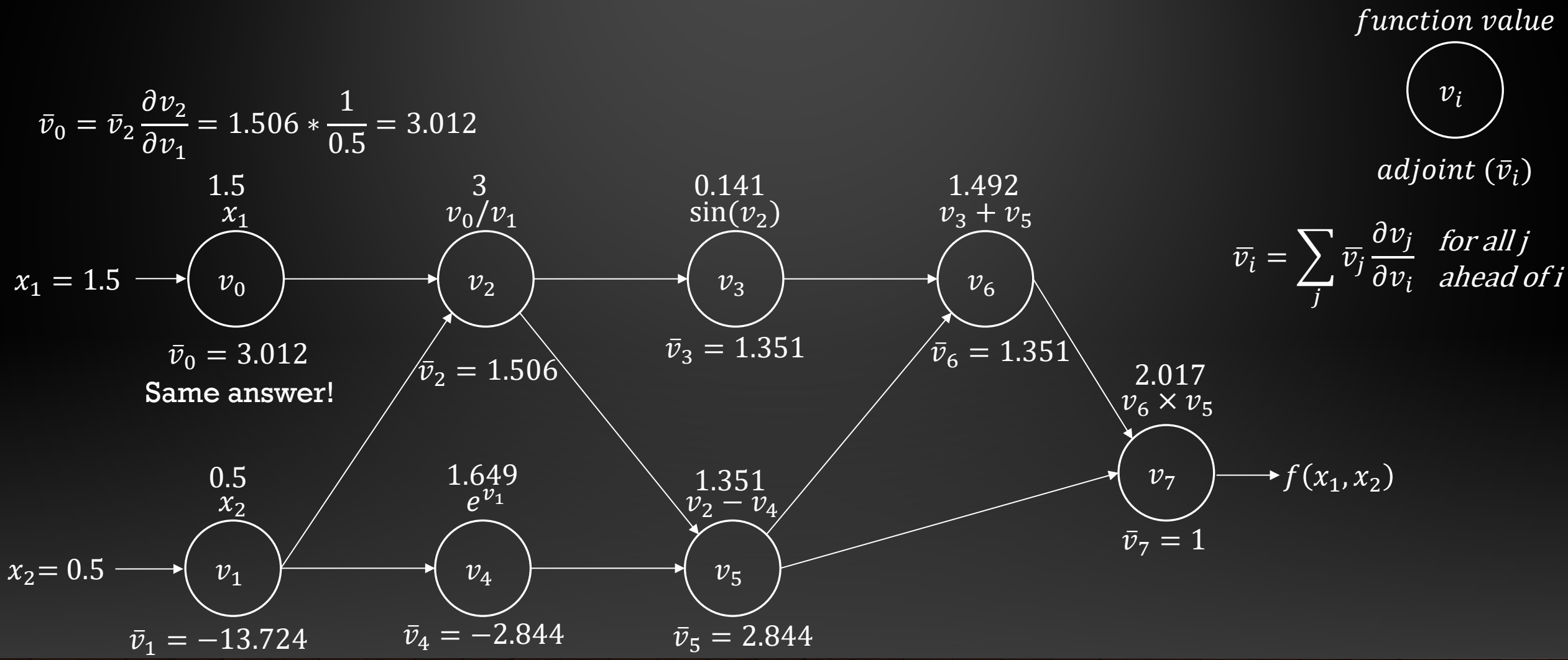


REVERSE MODE AUTO-DIFF



$$\bar{v}_1 = \bar{v}_2 \frac{\partial v_2}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = 1.506 * -\frac{1.5}{0.5^2} - 2.844 * e^{0.5} = -13.724$$

REVERSE MODE AUTO-DIFF



REVERSE MODE AUTO-DIFF

Easily find partials wrt. all parameters for each output

Works well for ML/RL where N parameters >> N outputs

Can be memory intensive as all intermediate states must be stored

Computationally relatively cheap (maximum 6x the cost of forward pass)

SUMMARY

Neural Networks effective for approximating Q values

However, adjustments must be made for improved stability

Automatic differentiation is a useful tool for finding partial derivatives of programs

Reverse mode auto-diff more suited to ML/RL applications