Gradient computations made easy with JAX

Thea Sukianto[1]

[1]Department of Statistics and Data Science, Carnegie Mellon University

StatBytes Seminar
20 November 2024

Carnegie Mellon University
Statistics & Data Science

Autodiff: What is it, why you should do it, and how to do it in one line of code

Thea Sukianto[1]

[1]Department of Statistics and Data Science, Carnegie Mellon University

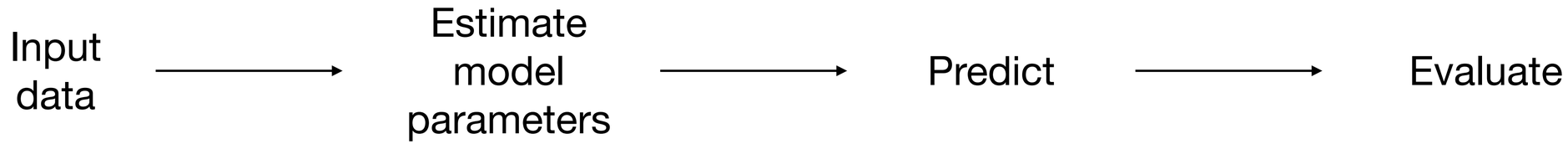
StatBytes Seminar
20 November 2024

Carnegie Mellon University
Statistics & Data Science

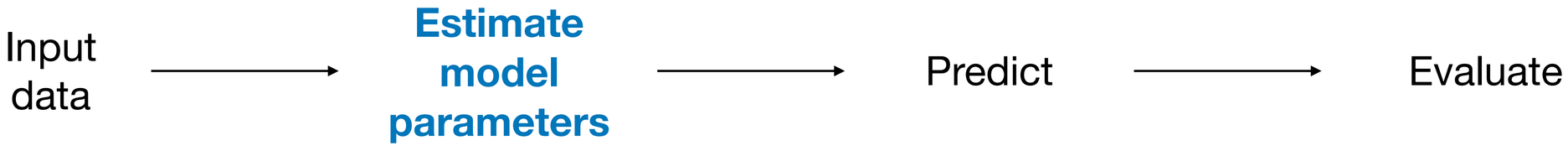
1. What is **automatic differentiation**?

2. Introduction to **JAX**

The basic prediction problem workflow



The basic prediction problem workflow



Often involves minimizing some loss function using **gradient-based optimization**

A simple example: Logistic regression

You have a labeled dataset $\{x_i, y_i\}_{i=1}^N$ where $y_i \in \{-1, 1\}, x_i \in \mathbb{R}^d$

You want to fit a linear model $f(x_i; \theta) = \langle \theta, x_i \rangle$ that will give the best predictions (on the log-odds scale)

How do we define “best prediction”?

A simple example: Logistic regression

You have a labeled dataset $\{x_i, y_i\}_{i=1}^N$ where $y_i \in \{-1, 1\}, x_i \in \mathbb{R}^d$

You want to fit a linear model $f(x_i; \theta) = \langle \theta, x_i \rangle$ that will give the best predictions (on the log-odds scale)

How do we define “best prediction”?

We want to find the parameter vector θ^* that minimizes a **loss function**

$$L(y_i, f(x_i; \theta)) = \frac{1}{N} \sum_{i=1}^{N} \log \left(1 + e^{-y_i \langle \theta, x_i \rangle} \right)$$

A simple example: Logistic regression

You have a labeled dataset $\{x_i, y_i\}_{i=1}^N$ where $y_i \in \{-1,1\}, x_i \in \mathbb{R}^d$

You want to fit a linear model $f(x_i;\theta) = \langle \theta, x_i \rangle$ that will give the best predictions (on the log-odds scale)

How do we define “best prediction”?

We want to find the parameter vector θ^* that minimizes a **loss function**

$$L(y_i, f(x_i;\theta)) = \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i\langle\theta, x_i\rangle}\right)$$

We can find θ^* iteratively using an optimization algorithm:

$$\theta_t = \theta_{t-1} - \eta \nabla L(\theta_{t-1})$$

(Gradient descent)

A simple example: Logistic regression

We can use the chain rule to find the partial derivatives w.r.t. the parameters:

$$L(y_i, x_i; \theta) = \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \langle \theta, x_i \rangle}\right)$$

$$f_1 = \log(f_2)$$
$$f_2 = 1 + e^{f_3}$$
$$f_3 = -y_i \langle \theta, x_i \rangle$$

$$\nabla L = [\frac{\delta L}{\delta \theta_1}, \ldots, \frac{\delta L}{\delta \theta_d}]$$

$$\frac{\delta L}{\delta \theta_j} = \frac{1}{N} \sum_{i=1}^{N} \frac{\delta f_1}{\delta f_2} \frac{\delta f_2}{\delta f_3} \frac{\delta f_3}{\delta \theta_j}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{-y_i x_{ij} e^{-y_i \langle \theta, x_i \rangle}}{1 + e^{-y_i \langle \theta, x_i \rangle}}$$

A simple example: Logistic regression

```python
# gradient
def grad_f(X, y, w):
  Xw = np.dot(X,w)
  a = (-1*y*np.exp(-1*y*Xw))/(1+np.exp(-1*y*Xw))
  return np.mean(X*a[:,None], axis = 0)
```

(stolen from my 10-725 homework)

Here we can derive the gradient and code it up in a few lines, but what if…

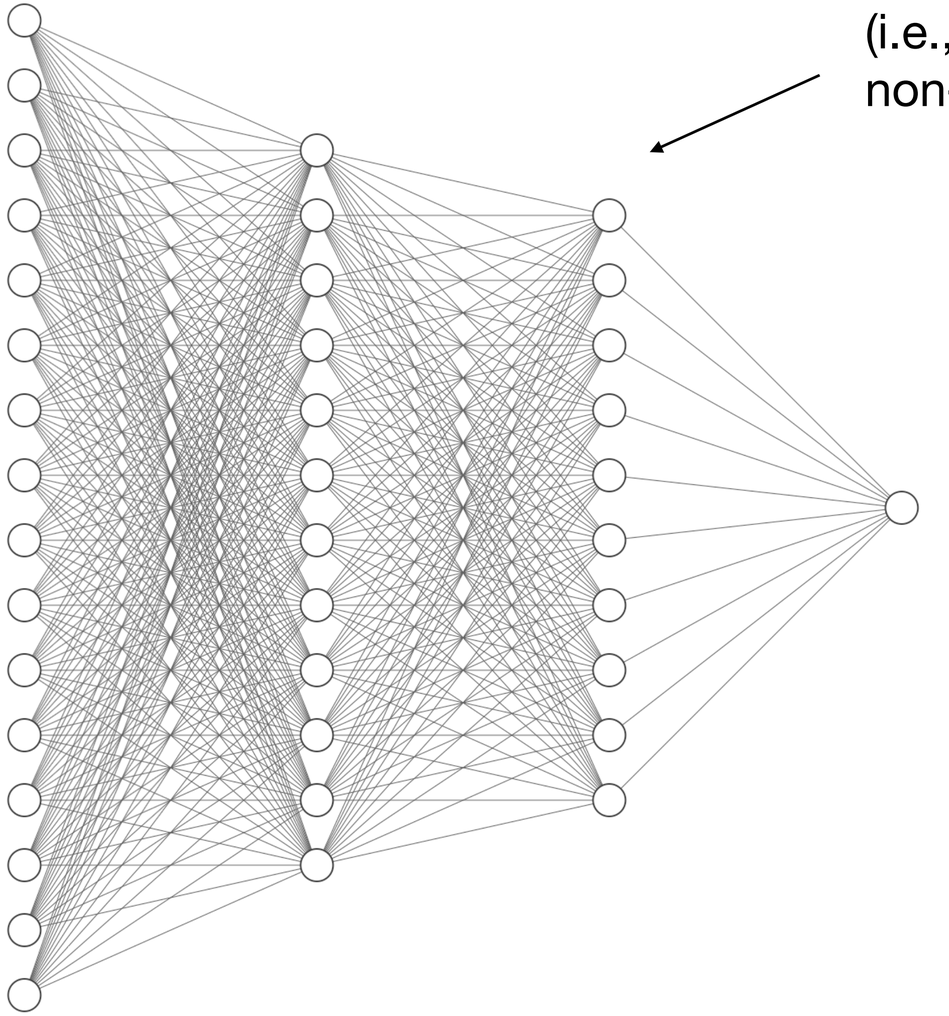
$$\nabla L = [\frac{\delta L}{\delta \theta_1}, \ldots, \frac{\delta L}{\delta \theta_d}]$$

$$\frac{\delta L}{\delta \theta_j} = \frac{1}{N} \sum_{i=1}^{N} \frac{\delta f_1}{\delta f_2} \frac{\delta f_2}{\delta f_3} \frac{\delta f_3}{\delta \theta_j}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{-y_i x_{ij} e^{-y_i \langle \theta, x_i \rangle}}{1 + e^{-y_i \langle \theta, x_i \rangle}}$$

A not-so-simple example: a deep neural network

$$f(x_i; \theta) =$$



(i.e., a very large,
non-linear function)

$$\nabla L(y_i, f(x_i; \theta))?$$

A not-so-simple example: a deep neural network

$f($

$\nabla L(y_i, f(x_i; \theta))???$

Automatic differentiation

In practice, many gradients are too infeasible to derive/code by hand, esp. for modern machine learning architectures (hundreds+ of parameters, non-linearity, etc…)

A tool you should consider is **automatic differentiation:**

“Derivative evaluations, not expressions” (Baydin et al. 2018)

So, a closed-form expression for the gradient is not required

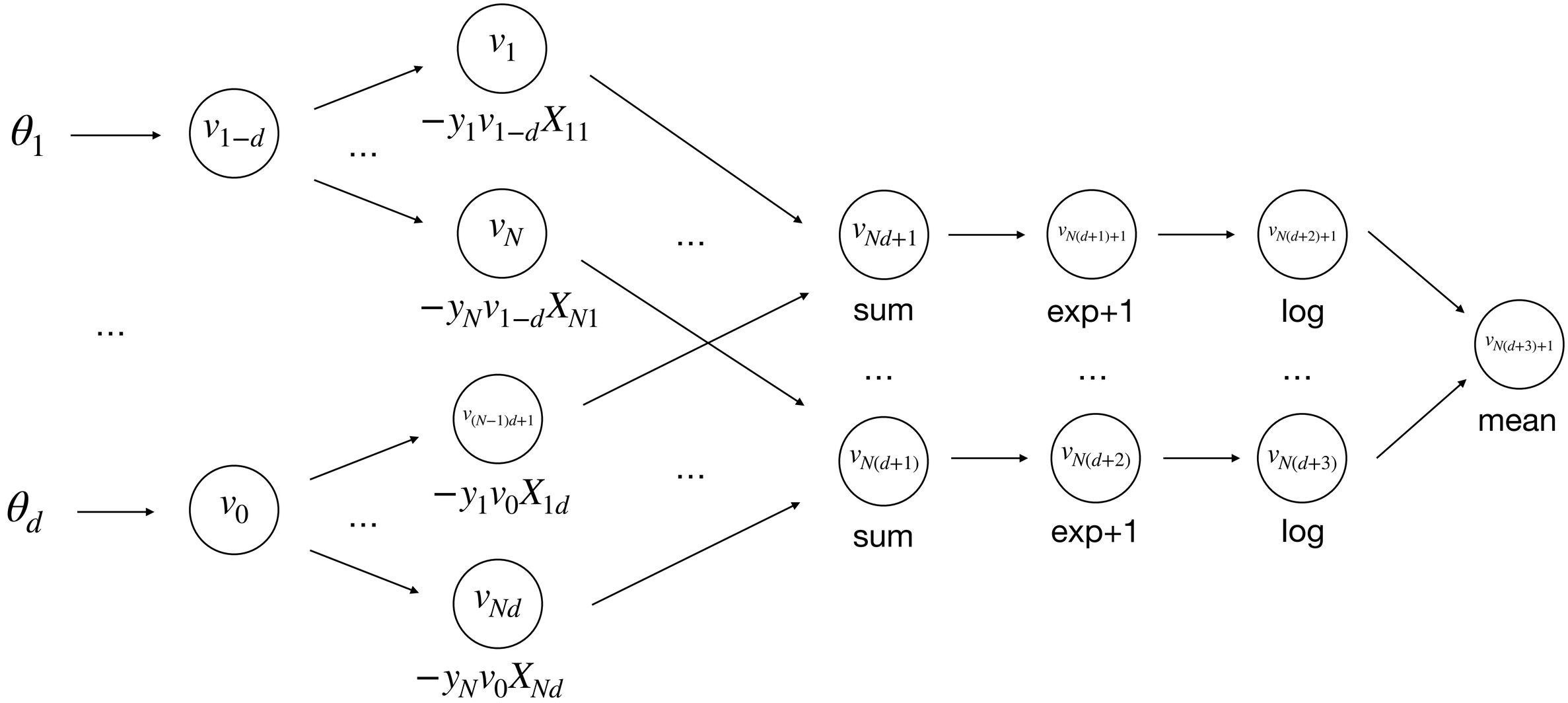
Evaluation trace

Automatic differentiation is based on the idea that you can draw an **evaluation trace** for most functions:

$$L(y_i, f(x_i; \theta)) = \frac{1}{N} \sum_{i=1}^{N} \log \left(1 + e^{-y_i \langle \theta, x_i \rangle} \right)$$ (e.g. our logistic loss)

Evaluation trace

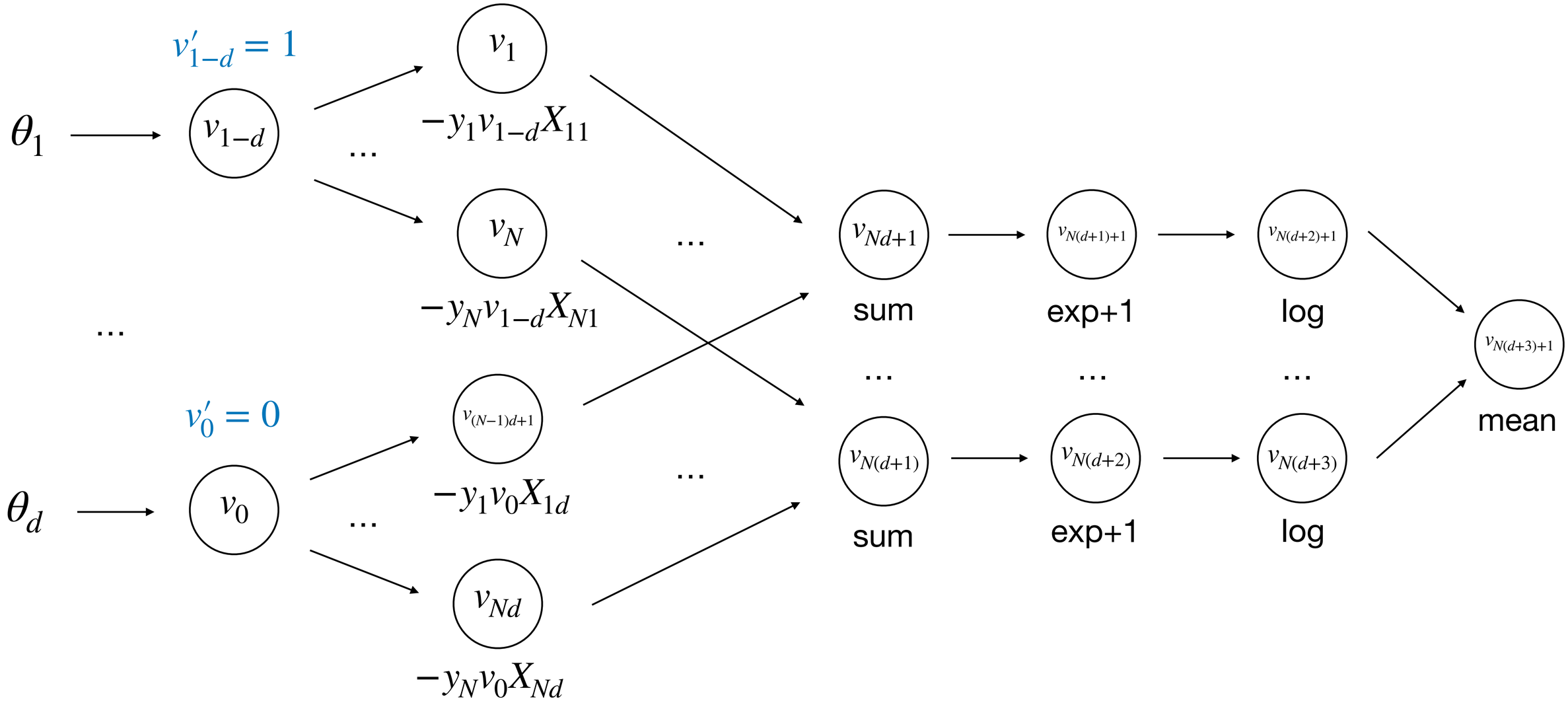
$$L(y_i, f(x_i;\theta)) = \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i\langle\theta, x_i\rangle}\right)$$



Forward mode

Set $\frac{\partial v_{1-d}}{\partial \theta_1} = 1$ with all other partial derivatives in the first column = 0

$\frac{\delta L}{\delta \theta_1}$?

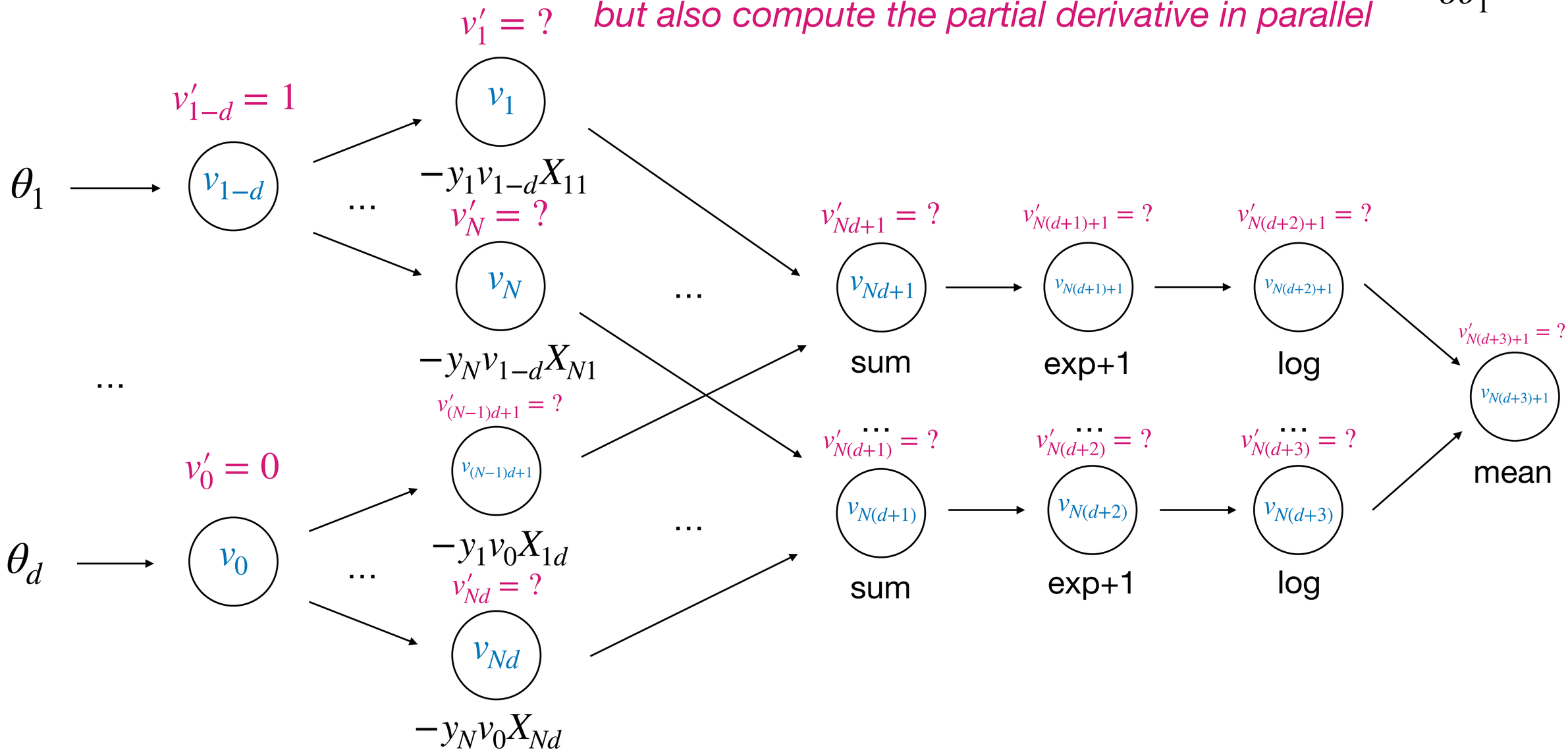


Forward mode

Compute v_k as you pass through each node,

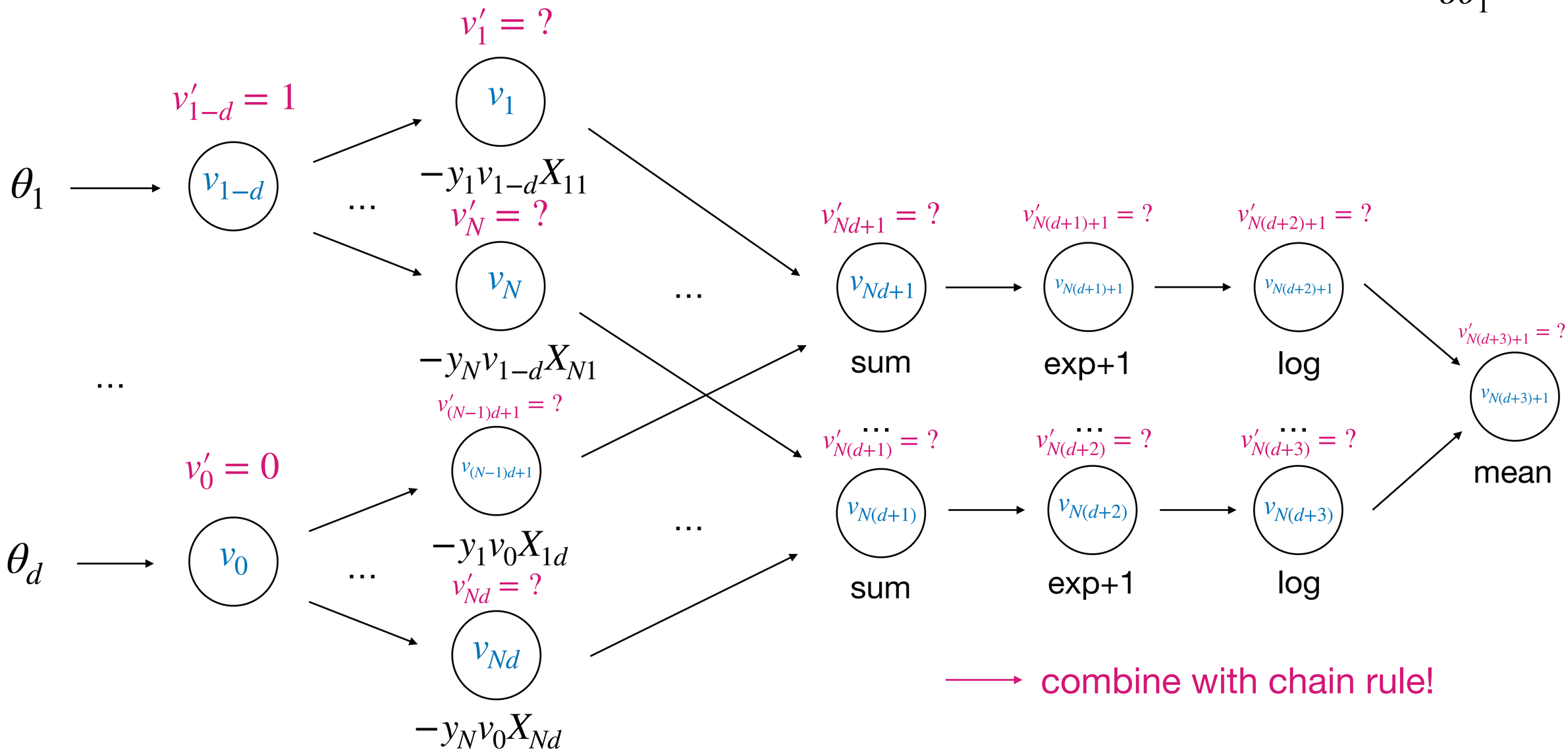
but also compute the partial derivative in parallel

$\frac{\delta L}{\delta \theta_1}$?



Forward mode

$$\frac{\delta L}{\delta \theta_1} ?$$



Forward mode: some considerations

Notice that to compute the full gradient, we need to make as many passes as there are parameters to estimate

This is especially problematic for neural networks since the number of parameters is often very large

(conversely, forward mode is great when the output dimension is very large but you only have a few parameters)

Forward mode: some considerations

Notice that to compute the full gradient, we need to make as many passes as there are parameters to estimate

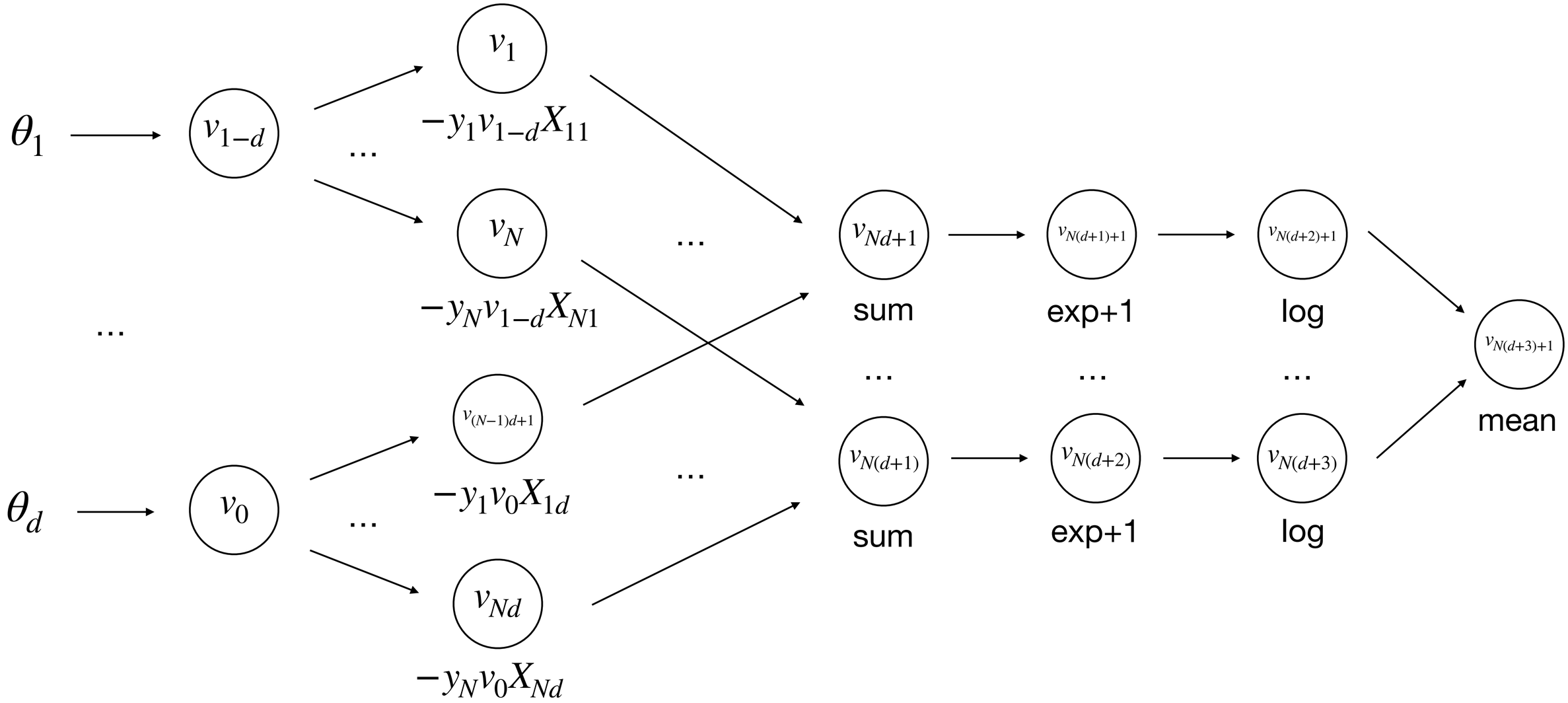
This is especially problematic for neural networks since the number of parameters is often very large

(conversely, forward mode is great when the output dimension is very large but you only have a few parameters)

This is why backpropagation (i.e. a special case of **reverse mode autodiff**) is more popular for NNs

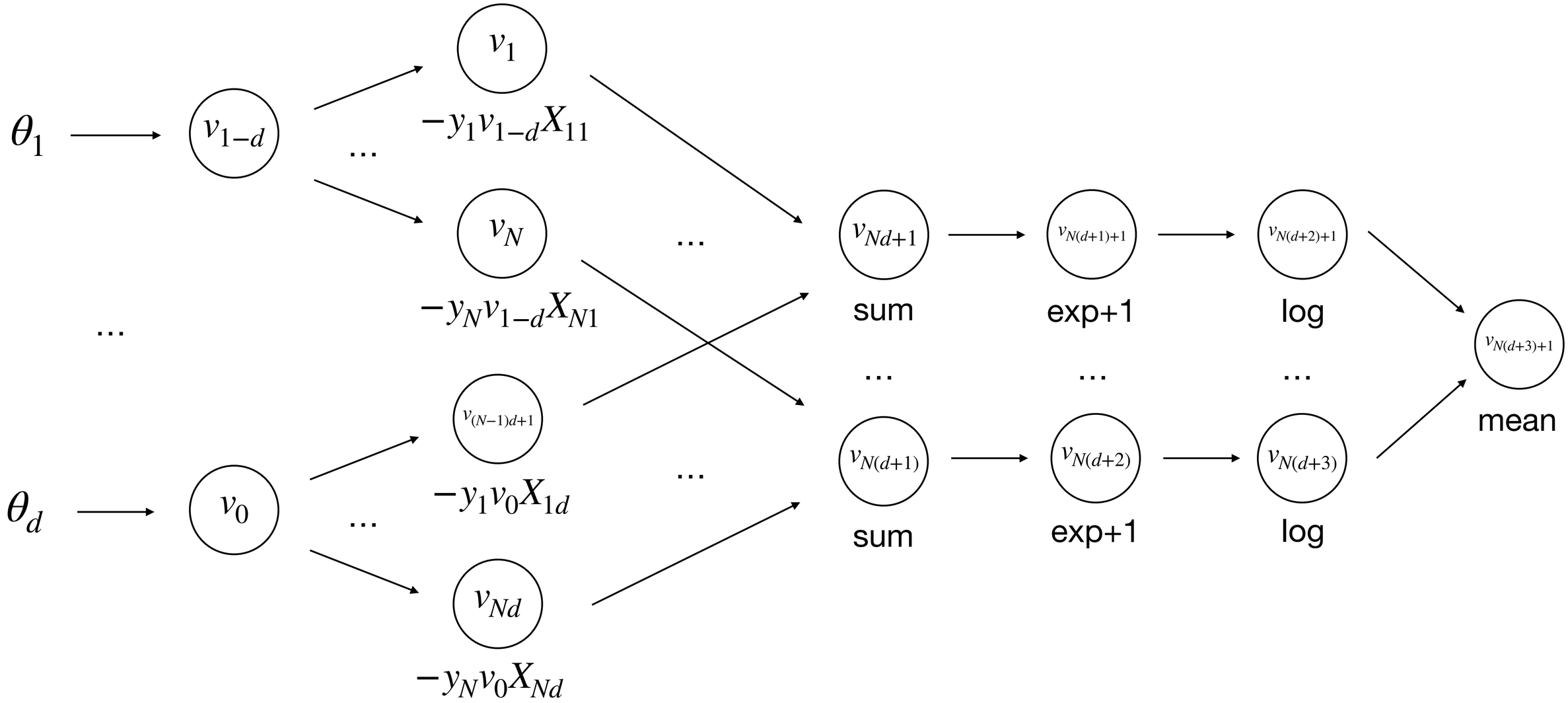
Reverse mode

$$L(y_i, f(x_i; \theta)) = \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \langle \theta, x_i \rangle}\right)$$



Reverse mode

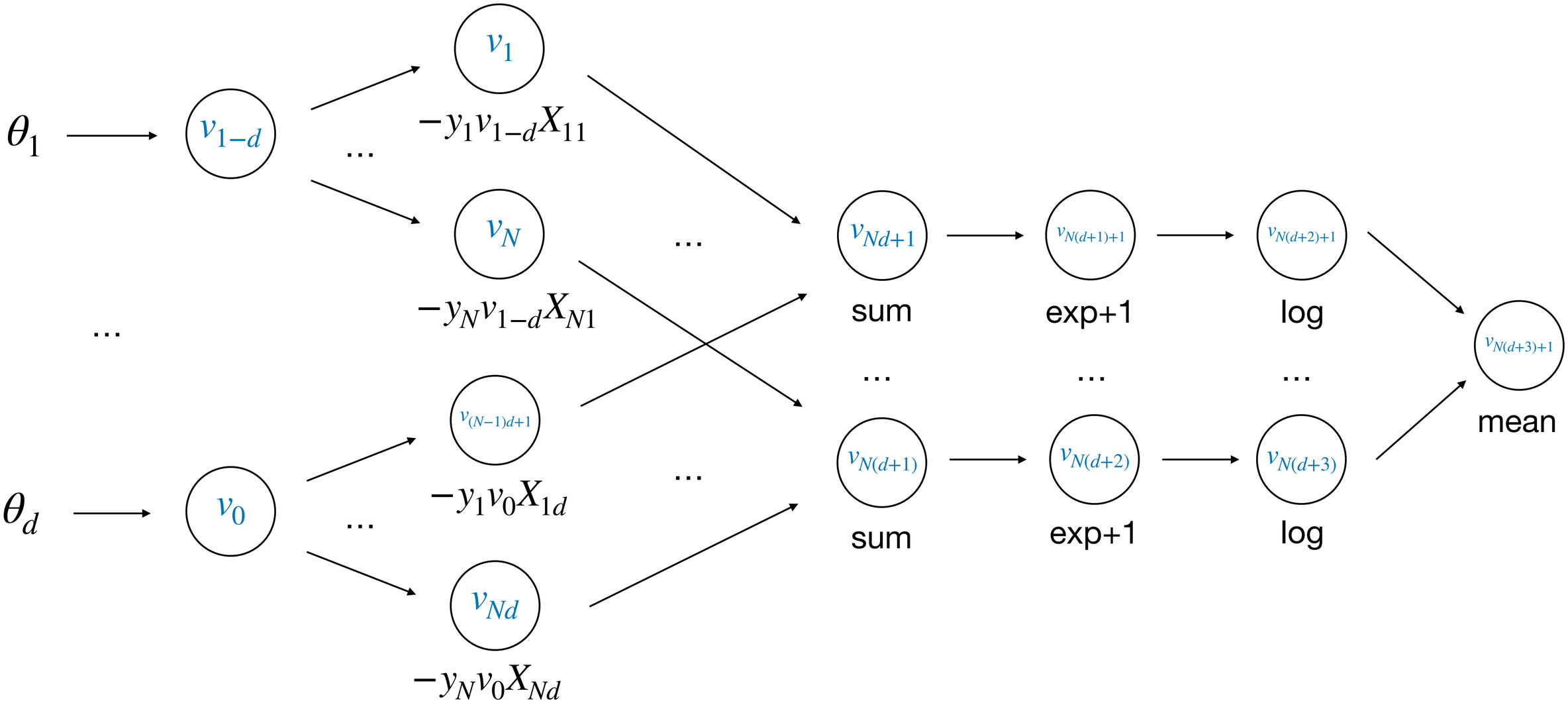
∇L?



Reverse mode

Compute v_k as you pass through each node

∇L?

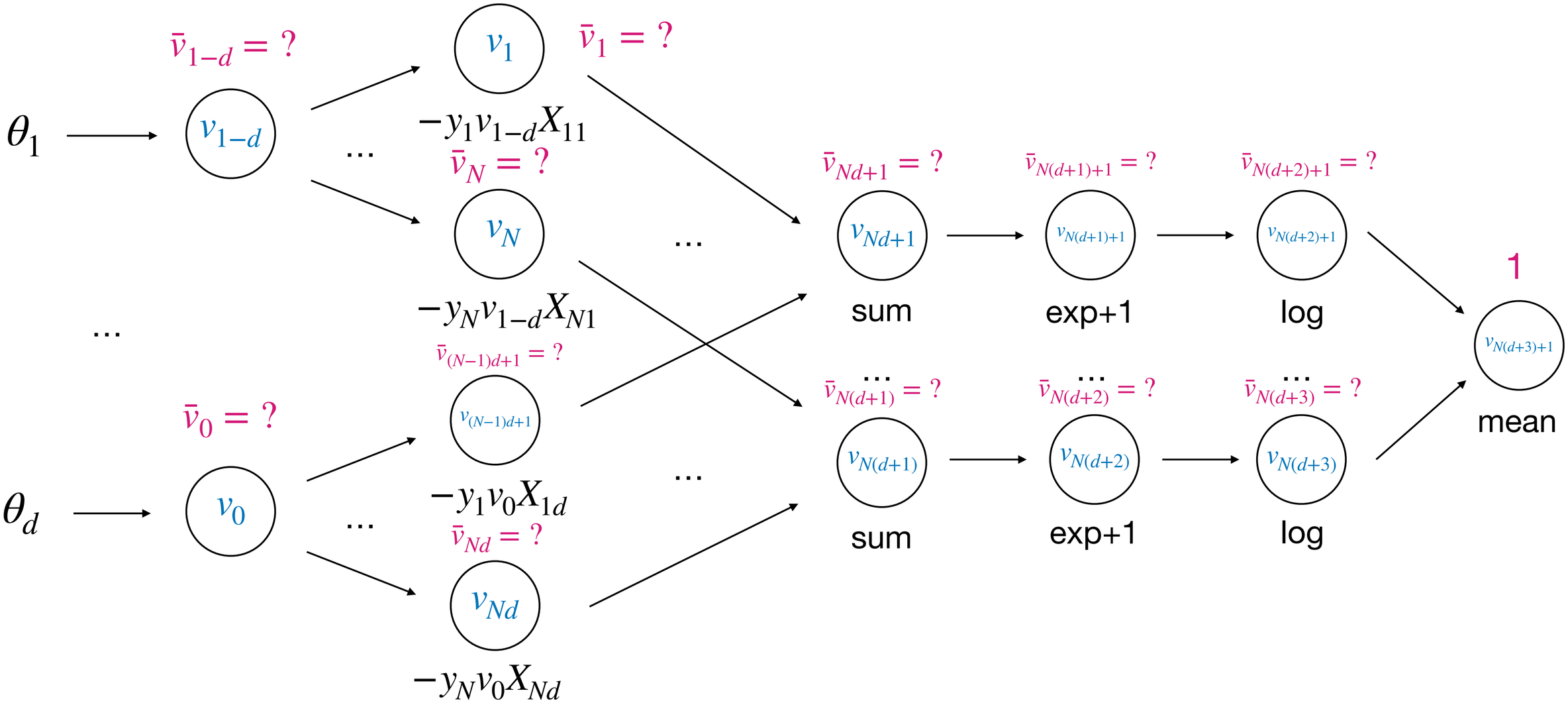


Reverse mode

Compute v_k as you pass through each node

Compute the adjoints $\bar{v}_k = \frac{\partial L}{\partial v_k}$ *in reverse*

∇L?

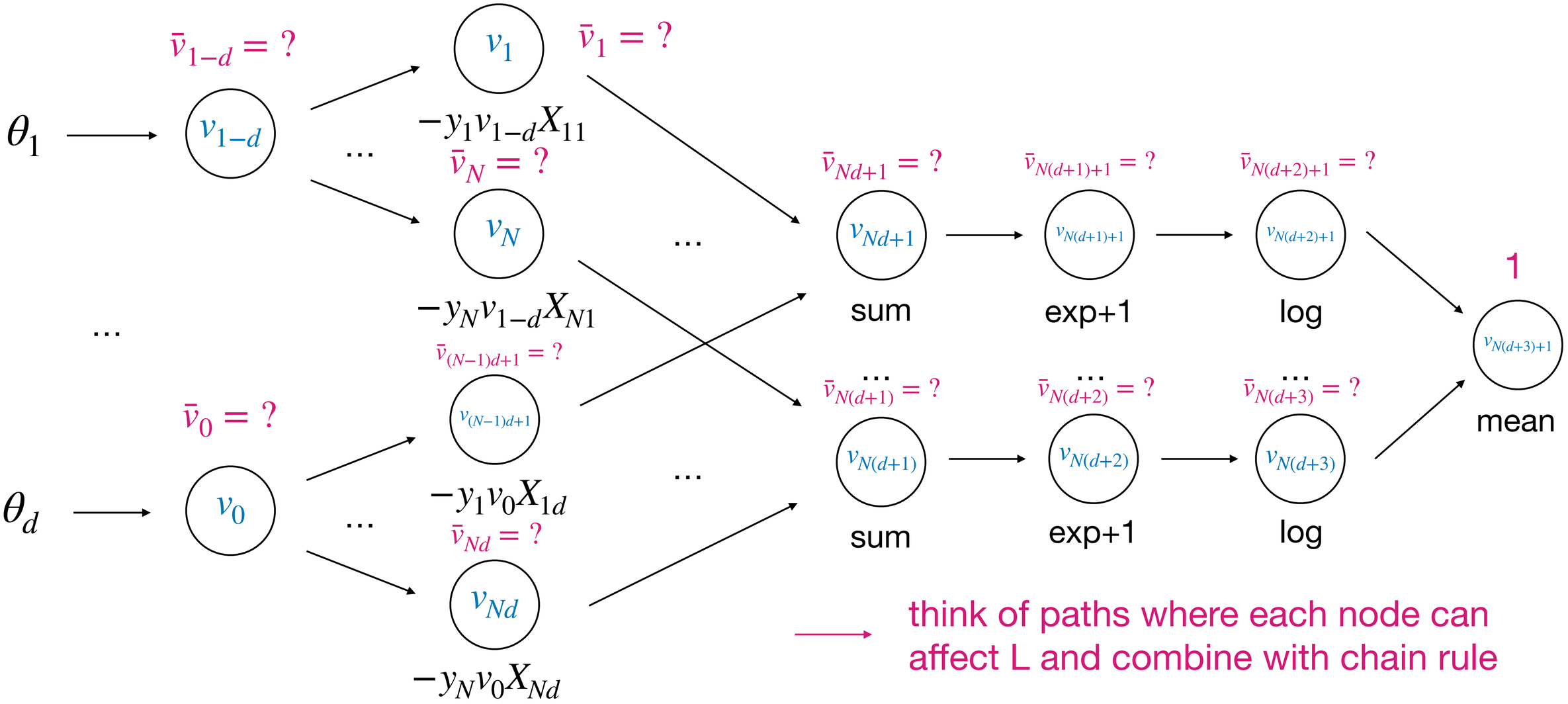


Reverse mode

Compute v_k as you pass through each node

Compute the adjoints $\bar{v}_k = \frac{\partial L}{\partial v_k}$ *in reverse*

∇L?



think of paths where each node can affect L and combine with chain rule

Reverse mode: some considerations

Notice that to compute the full gradient this time, we only needed to make one forward pass and one reverse pass

This is especially advantageous when we have a large number of parameters but only one output (the typical loss function minimization scenario)

Conversely, reverse mode is less efficient when there are few parameters compared to the dimension of the output

Differentiation methods: pros and cons

	Strengths	Weaknesses
Manual	Exact	Can be time-consuming More prone to errors
Numerical	Easy to implement	Prone to round-off/truncation errors
Symbolic	Exact Faster than manual	 $\nabla L(y_i, f(x_i; \theta))???$
Automatic	Works on complicated functions (with no closed form), including “code logic”	Can be memory-intensive Can be tricky to implement

(Baydin et al. 2018)

Differentiation methods: pros and cons

	Strengths	Weaknesses
Manual	Exact	Can be time-consuming More prone to errors
Numerical	Easy to implement	Prone to round-off/truncation errors
Symbolic	Exact Faster than manual	Requires a closed form Prone to “expression swell”
Automatic	Works on complicated functions (with no closed form), including “code logic”	Can be memory-intensive Can be tricky to implement

(Baydin et al. 2018)

Differentiation methods: pros and cons

	Strengths	Weaknesses
Manual	Exact	Can be time-consuming More prone to errors
Numerical	Easy to implement	Prone to round-off/truncation errors
Symbolic	Exact Faster than manual	Requires a closed form Prone to “expression swell”
Automatic	Works on complicated functions (with no closed form), including “code logic”	Can be memory-intensive ~~Can be tricky to implement~~

(Baydin et al. 2018)

Python libraries are your best friend!

Further reading

Journal of Machine Learning Research 18 (2018) 1-43 Submitted 8/17; Published 4/18

Automatic Differentiation in Machine Learning: a Survey

Atılım Güneş Baydin GUNES@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom

Barak A. Pearlmutter BARAK@PEARLMUTTER.NET
Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland

Alexey Andreyevich Radul AXCH@MIT.EDU
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States

Jeffrey Mark Siskind QOBI@PURDUE.EDU
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, United States

Editor: Léon Bottou

https://doi.org/10.48550/arXiv.1502.05767

Has worked examples for both forward and reverse mode + a timing study comparing the differentiation methods

1. What is **automatic differentiation**?

2. Introduction to **JAX**

The JAX Python library

https://jax.readthedocs.io/en/latest/index.html

A relatively recent Python library for high-performance array computing

In active development by Google researchers (v0.3.4)

Provides an easy-to-use yet customizable automatic differentiation framework

There is also an ecosystem of related libraries that use JAX to e.g. train neural networks

I already use PyTorch, why should I consider JAX?

Intuitive syntax: You can work with arrays the same way as NumPy if you’re used to it

Speed: JAX is optimized for GPU computing with just-in-time compilation (JIT)

Easy autodiff: No need to use requires_grad=True and f.backward(), just call grad()

I already use PyTorch, why should I consider JAX?

Intuitive syntax: You can work with arrays the same way as NumPy if you're used to it

Speed: JAX is optimized for GPU computing with just-in-time compilation (JIT)

Easy autodiff: No need to use requires_grad=True and f.backward(), just call grad()

However, PyTorch is a more established library with a more developed software ecosystem and may offer greater flexibility when defining network architectures (choose the library that is right for your project)