

# Automatic Differentiation

## Seminar

Optimization for ML. Faculty of Computer Science. HSE University

## Forward mode

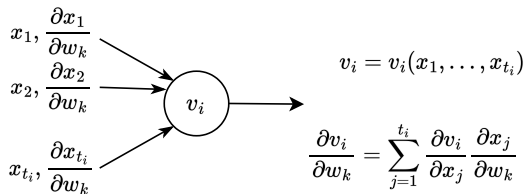


Figure 1: Illustration of forward chain rule to calculate the derivative of the function  $v_i$  with respect to  $w_k$ .

- Uses the forward chain rule
- Has complexity  $d \times \mathcal{O}(T)$  operations

## Reverse mode



Figure 2: Illustration of reverse chain rule to calculate the derivative of the function  $L$  with respect to the node  $v_i$ .

- Uses the backward chain rule
- Stores the information from the forward pass
- Has complexity  $\mathcal{O}(T)$  operations

## Toy example

### Example

$$f(x_1, x_2) = x_1 * x_2 + \sin x_1$$

Let's calculate the derivatives  $\frac{\partial f}{\partial x_i}$  using forward and reverse modes.

## Toy example

### Example

$$f(x_1, x_2) = x_1 * x_2 + \sin x_1$$

Let's calculate the derivatives  $\frac{\partial f}{\partial x_i}$  using forward and reverse modes.



Figure 3: Illustration of computation graph of  $f(x_1, x_2)$ .

# Automatic Differentiation with JAX

## Example №1

$$f(X) = \text{tr}(AX^{-1}B)$$

$$\nabla f = -X^{-T}A^TB^TX^{-T}$$

# Automatic Differentiation with JAX

## Example №1

$$f(X) = \text{tr}(AX^{-1}B)$$

$$\nabla f = -X^{-T}A^TB^TX^{-T}$$

## Example №2

$$g(x) = 1/3 \cdot \|x\|_2^3$$

$$\nabla^2 g = \|x\|_2^{-1}xx^T + \|x\|_2I_n$$

# Automatic Differentiation with JAX

## Example №1

$$f(X) = \text{tr}(AX^{-1}B)$$

$$\nabla f = -X^{-T}A^TB^TX^{-T}$$

## Example №2

$$g(x) = 1/3 \cdot \|x\|_2^3$$

$$\nabla^2 g = \|x\|_2^{-1}xx^T + \|x\|_2 I_n$$

Let's calculate the gradients and Hessians of  $f$  and  $g$  in python 🐍



# Problem 1

## Question

Which of the AD modes would you choose (forward/ reverse) for the following computational graph of primitive arithmetic operations?



Figure 4: Which mode would you choose for calculating gradients there?

## Problem 2

Suppose, we have an invertible matrix  $A$  and a vector  $b$ , the vector  $x$  is the solution of the linear system  $Ax = b$ , namely one can write down an analytical solution  $x = A^{-1}b$ .

### Question

Find the derivatives  $\frac{\partial L}{\partial A}$ ,  $\frac{\partial L}{\partial b}$ .



Figure 5:  $x$  could be found as a solution of linear system

## Problem 3

Suppose, we have the rectangular matrix  $W \in \mathbb{R}^{m \times n}$ , which has a singular value decomposition:

$$W = U\Sigma V^T, \quad U^T U = I, \quad V^T V = I, \\ \Sigma = \text{diag}(\sigma_1, \dots, \sigma_{\min(m,n)})$$

The regularizer  $R(W) = \text{tr}(\Sigma)$  in any loss function encourages low rank solutions.

Question

Find the derivative  $\frac{\partial R}{\partial W}$ .



Figure 6: Computation graph for singular regularizer

# Computation experiment with JAX

Let's make sure numerically that we have correctly calculated the derivatives in problems 2-3 🧠

# Feedforward Architecture



Figure 7: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The activations marked with an  $f$ . The gradient of the loss with respect to the activations and parameters marked with  $b$ .

# Feedforward Architecture



Figure 7: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The activations marked with an  $f$ . The gradient of the loss with respect to the activations and parameters marked with  $b$ .

## ! Important

The results obtained for the  $f$  nodes are needed to compute the  $b$  nodes.

## Vanilla backpropagation



Figure 8: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

## Vanilla backpropagation



Figure 8: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- All activations  $f$  are kept in memory after the forward pass.



## Vanilla backpropagation



Figure 8: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- All activations  $f$  are kept in memory after the forward pass.
- Optimal in terms of computation: it only computes each node once.

## Vanilla backpropagation



Figure 8: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- All activations  $f$  are kept in memory after the forward pass.
- Optimal in terms of computation: it only computes each node once.
- High memory usage. The memory usage grows linearly with the number of layers in the neural network.

## Memory poor backpropagation

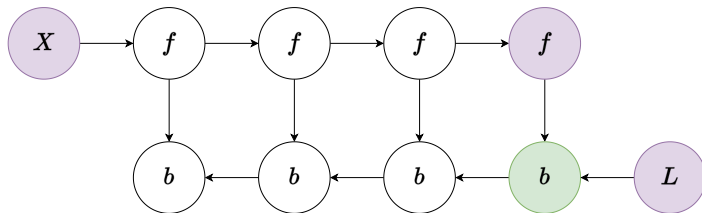


Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

## Memory poor backpropagation



Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Each activation  $f$  is recalculated as needed.

## Memory poor backpropagation

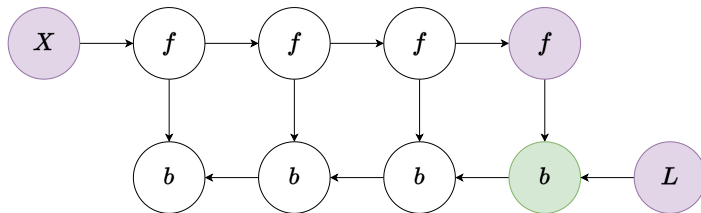


Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Each activation  $f$  is recalculated as needed.
- Optimal in terms of memory: there is no need to store all activations in memory.

## Memory poor backpropagation



Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Each activation  $f$  is recalculated as needed.
- Optimal in terms of memory: there is no need to store all activations in memory.
- Computationally inefficient. The number of node evaluations scales with  $n^2$ , whereas it vanilla backprop scaled as  $n$ : each of the  $n$  nodes is recomputed on the order of  $n$  times.

## Checkpointed backpropagation

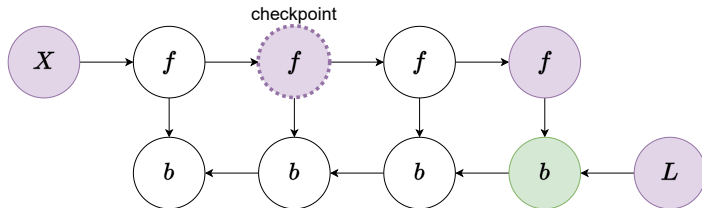


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

## Checkpointed backpropagation

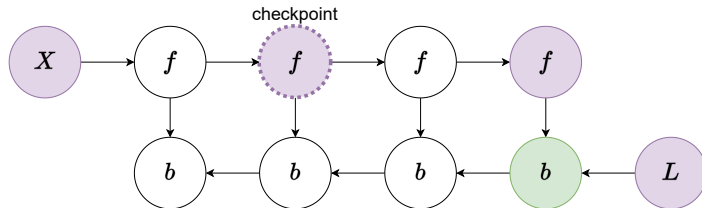


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Trade-off between the **vanilla** and **memory poor** approaches. The strategy is to mark a subset of the neural net activations as checkpoint nodes, that will be stored in memory.



# Checkpointed backpropagation



Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Trade-off between the **vanilla** and **memory poor** approaches. The strategy is to mark a subset of the neural net activations as checkpoint nodes, that will be stored in memory.
- Faster recalculation of activations  $f$ . We only need to recompute the nodes between a  $b$  node and the last checkpoint preceding it when computing that  $b$  node during backprop.

# Checkpointed backpropagation

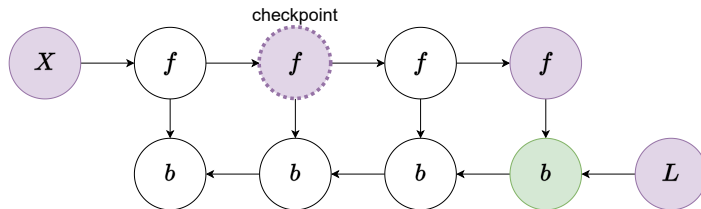


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with  $n$  layers. The purple color indicates nodes that are stored in memory.

- Trade-off between the **vanilla** and **memory poor** approaches. The strategy is to mark a subset of the neural net activations as checkpoint nodes, that will be stored in memory.
- Faster recalculation of activations  $f$ . We only need to recompute the nodes between a  $b$  node and the last checkpoint preceding it when computing that  $b$  node during backprop.
- Memory consumption depends on the number of checkpoints. More effective than **vanilla** approach.

# Gradient checkpointing visualization

The animated visualization of the above approaches 

An example of using a gradient checkpointing 