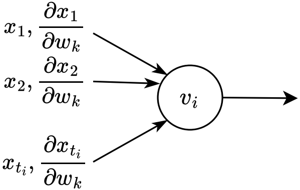
Automatic Differentiation.

Seminar

Optimization for ML. Faculty of Computer Science. HSE University

Forward mode



$$v_i = v_i(x_1, \ldots, x_{t_i})$$

$$\frac{\partial v_i}{\partial w_k} = \sum_{j=1}^{t_i} \frac{\partial v_i}{\partial x_j} \frac{\partial x_j}{\partial w_k}$$

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

Forward mode

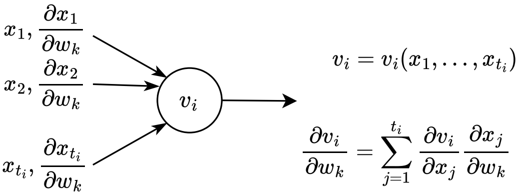


Figure 1: Illustration of forward chain rule to calculate the derivative of the function v_i with respect to w_k.

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

Reverse mode

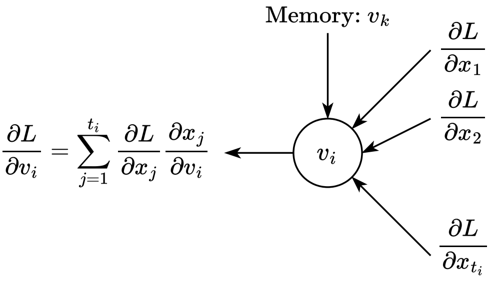


Figure 2: Illustration of reverse chain rule to calculate the derivative of the function L with respect to the node v_i.

- Uses the backward chain rule

Reverse mode

$$\frac{\partial L}{\partial v_i} = \sum_{j=1}^{t_i} \frac{\partial L}{\partial x_j} \frac{\partial x_j}{\partial v_i}$$

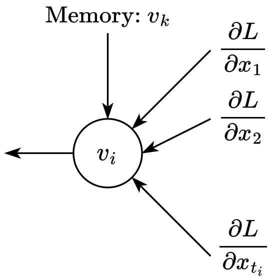


Figure 2: Illustration of reverse chain rule to calculate the derivative of the function L with respect to the node v_i.

- Uses the backward chain rule
- Stores the information from the forward pass

Reverse mode

$$\frac{\partial L}{\partial v_i} = \sum_{j=1}^{t_i} \frac{\partial L}{\partial x_j} \frac{\partial x_j}{\partial v_i}$$

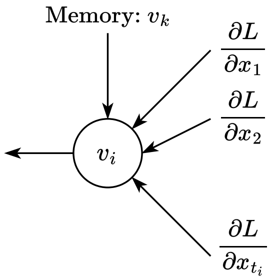


Figure 2: Illustration of reverse chain rule to calculate the derivative of the function L with respect to the node v_i.

- 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 $\dfrac{\partial f}{\partial x_i}$ 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 $\dfrac{\partial f}{\partial x_i}$ using forward and reverse modes.

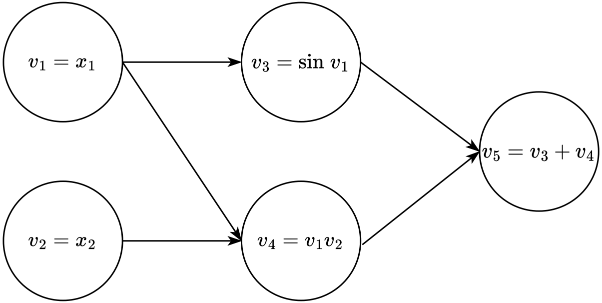


Figure 3: Illustration of computation graph of $f(x_1, x_2)$.

Automatic Differentiation with JAX

Example №1

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

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

Automatic Differentiation with JAX

Example №1

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

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

Example №2

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

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

Automatic Differentiation with JAX

Example №1

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

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

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?

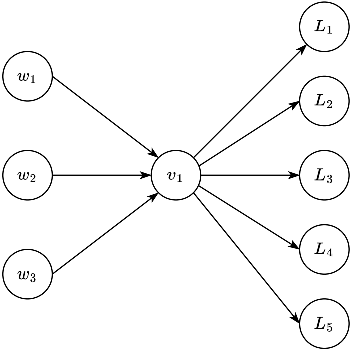


Figure 4: Which mode would you choose for calculating gradients there?

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 $\dfrac{\partial L}{\partial A}, \dfrac{\partial L}{\partial b}$.

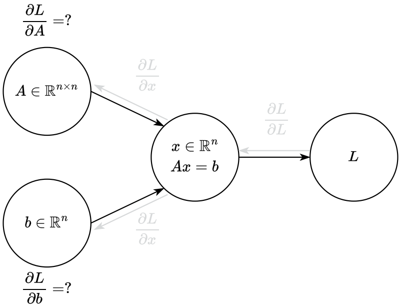


Figure 5: x could be found as a solution of linear system

Gradient propagation through the linear least squares

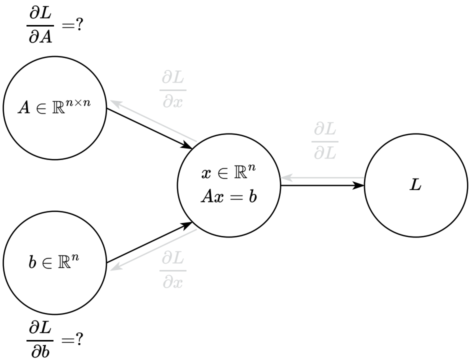


Figure 6: x could be found as a solution of linear system

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 $x = A^{-1}b$, in this example we will show, that computing all derivatives $\frac{\partial L}{\partial A}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial x}$, i.e. the backward pass, costs approximately the same as the forward pass.

Gradient propagation through the linear least squares

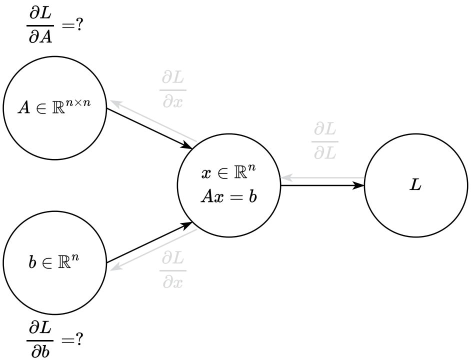


Figure 6: x could be found as a solution of linear system

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 $x = A^{-1}b$, in this example we will show, that computing all derivatives $\frac{\partial L}{\partial A}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial x}$, i.e. the backward pass, costs approximately the same as the forward pass.

It is known, that the differential of the function does not depend on the parametrization:

$$dL = \left\langle \frac{\partial L}{\partial x}, dx \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Gradient propagation through the linear least squares

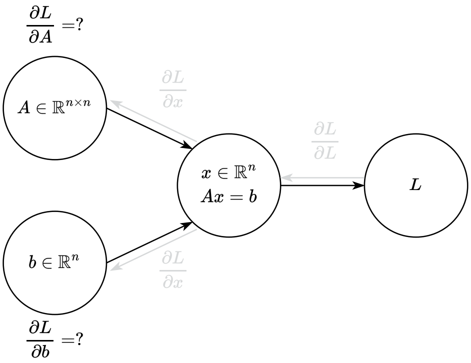


Figure 6: x could be found as a solution of linear system

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 $x = A^{-1}b$, in this example we will show, that computing all derivatives $\frac{\partial L}{\partial A}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial x}$, i.e. the backward pass, costs approximately the same as the forward pass.

It is known, that the differential of the function does not depend on the parametrization:

$$dL = \left\langle \frac{\partial L}{\partial x}, dx \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Given the linear system, we have:

$$Ax = b$$
$$dAx + Adx = db \to dx = A^{-1}(db - dAx)$$

Gradient propagation through the linear least squares

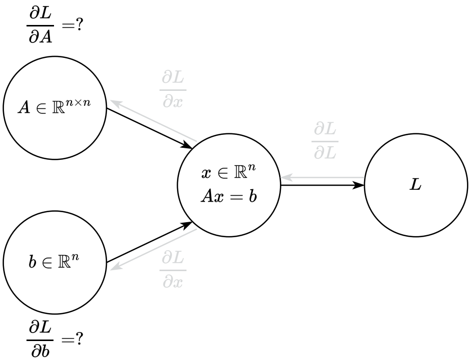


Figure 7: x could be found as a solution of linear system

The straightforward substitution gives us:

$$\left\langle \frac{\partial L}{\partial x}, A^{-1}(db - dAx) \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Gradient propagation through the linear least squares

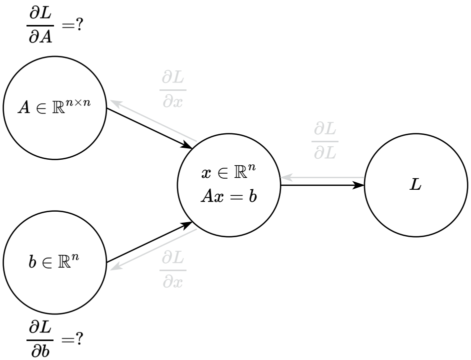


Figure 7: x could be found as a solution of linear system

The straightforward substitution gives us:

$$\left\langle \frac{\partial L}{\partial x}, A^{-1}(db - dAx) \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

$$\left\langle -A^{-T}\frac{\partial L}{\partial x}x^T, dA \right\rangle + \left\langle A^{-T}\frac{\partial L}{\partial x}, db \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Gradient propagation through the linear least squares

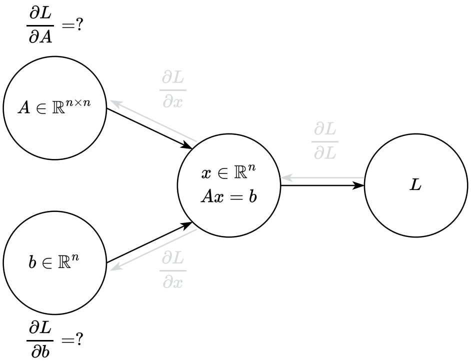


Figure 7: x could be found as a solution of linear system

The straightforward substitution gives us:

$$\left\langle \frac{\partial L}{\partial x}, A^{-1}(db - dAx) \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

$$\left\langle -A^{-T} \frac{\partial L}{\partial x} x^T, dA \right\rangle + \left\langle A^{-T} \frac{\partial L}{\partial x}, db \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Therefore:

$$\frac{\partial L}{\partial A} = -A^{-T} \frac{\partial L}{\partial x} x^T \quad \frac{\partial L}{\partial b} = A^{-T} \frac{\partial L}{\partial x}$$

Gradient propagation through the linear least squares

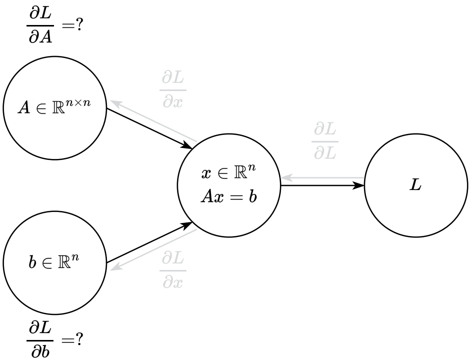


Figure 7: x could be found as a solution of linear system

The straightforward substitution gives us:

$$\left\langle \frac{\partial L}{\partial x}, A^{-1}(db - dAx) \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

$$\left\langle -A^{-T}\frac{\partial L}{\partial x}x^T, dA \right\rangle + \left\langle A^{-T}\frac{\partial L}{\partial x}, db \right\rangle = \left\langle \frac{\partial L}{\partial A}, dA \right\rangle + \left\langle \frac{\partial L}{\partial b}, db \right\rangle$$

Therefore:

$$\frac{\partial L}{\partial A} = -A^{-T}\frac{\partial L}{\partial x}x^T \quad \frac{\partial L}{\partial b} = A^{-T}\frac{\partial L}{\partial x}$$

It is interesting, that the most computationally intensive part here is the matrix inverse, which is the same as for the forward pass. Sometimes it is even possible to store the result itself, which makes the backward pass even cheaper.

Problem 3

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

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

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

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

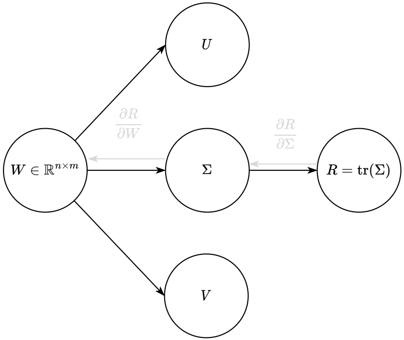
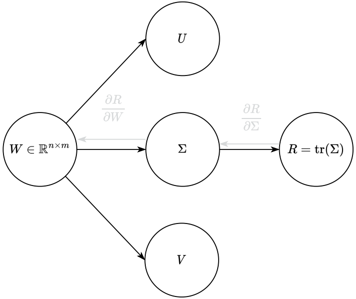


Figure 8: Computation graph for singular regularizer

Gradient propagation through the SVD



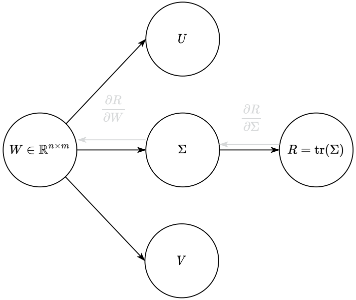
Suppose, we have the rectangular matrix $W \in \mathbb{R}^{m \times n}$, which has a singular value decomposition:

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

1. Similarly to the previous example:

$$\begin{aligned}
W &= U\Sigma V^T \\
dW &= dU\Sigma V^T + Ud\Sigma V^T + U\Sigma dV^T \\
U^TdWV &= U^TdU\Sigma V^TV + U^TUd\Sigma V^TV + U^TU\Sigma dV^TV \\
U^TdWV &= U^TdU\Sigma + d\Sigma + \Sigma dV^TV
\end{aligned}$$

Gradient propagation through the SVD



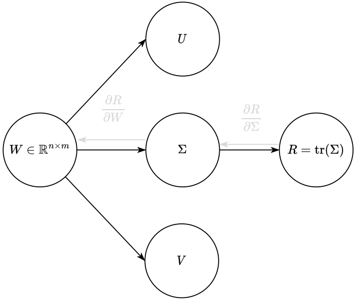
2. Note, that $U^T U = I \to dU^T U + U^T dU = 0$. But also $dU^T U = (U^T dU)^T$, which actually involves, that the matrix $U^T dU$ is antisymmetric:

$$
(U^T dU)^T + U^T dU = 0 \quad \to \quad \text{diag}(U^T dU) = (0, \dots, 0)
$$

The same logic could be applied to the matrix V and

$$
\text{diag}(dV^T V) = (0, \dots, 0)
$$

Gradient propagation through the SVD



2. Note, that $U^T U = I \to dU^T U + U^T dU = 0$. But also $dU^T U = (U^T dU)^T$, which actually involves, that the matrix $U^T dU$ is antisymmetric:

$$(U^T dU)^T + U^T dU = 0 \quad \to \quad \text{diag}(U^T dU) = (0, \ldots, 0)$$

The same logic could be applied to the matrix V and

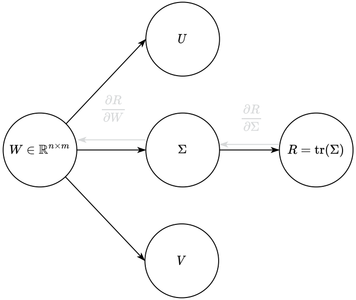
$$\text{diag}(dV^T V) = (0, \ldots, 0)$$

3. At the same time, the matrix $d\Sigma$ is diagonal, which means (look at the 1.) that

$$\text{diag}(U^T dW V) = d\Sigma$$

Here on both sides, we have diagonal matrices.

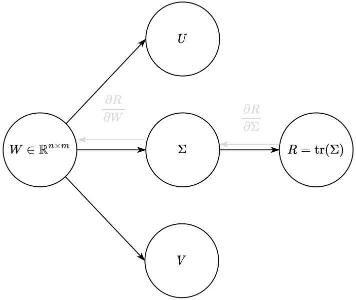
Gradient propagation through the SVD



4. Now, we can decompose the differential of the loss function as a function of Σ - such problems arise in ML problems, where we need to restrict the matrix rank:

$$
\begin{aligned}
dL &= \left\langle \frac{\partial L}{\partial \Sigma}, d\Sigma \right\rangle \\
&= \left\langle \frac{\partial L}{\partial \Sigma}, \text{diag}(U^T dW V) \right\rangle \\
&= \text{tr}\left(\frac{\partial L}{\partial \Sigma}^T \text{diag}(U^T dW V) \right)
\end{aligned}
$$

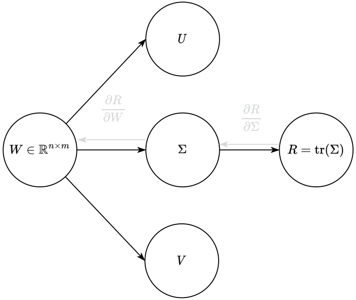
Gradient propagation through the SVD



5. As soon as we have diagonal matrices inside the product, the trace of the diagonal part of the matrix will be equal to the trace of the whole matrix:

$$
\begin{aligned}
dL &= \text{tr}\left(\frac{\partial L}{\partial \Sigma}^T \text{diag}(U^T dW V) \right) \\
&= \text{tr}\left(\frac{\partial L}{\partial \Sigma}^T U^T dW V \right) \\
&= \left\langle \frac{\partial L}{\partial \Sigma}, U^T dW V \right\rangle \\
&= \left\langle U \frac{\partial L}{\partial \Sigma} V^T, dW \right\rangle
\end{aligned}
$$

Gradient propagation through the SVD



6. Finally, using another parametrization of the differential

$$\left\langle U \frac{\partial L}{\partial \Sigma} V^T, dW \right\rangle = \left\langle \frac{\partial L}{\partial W}, dW \right\rangle$$

$$\frac{\partial L}{\partial W} = U \frac{\partial L}{\partial \Sigma} V^T,$$

This nice result allows us to connect the gradients $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial \Sigma}$.

Computation experiment with JAX

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

Feedforward Architecture

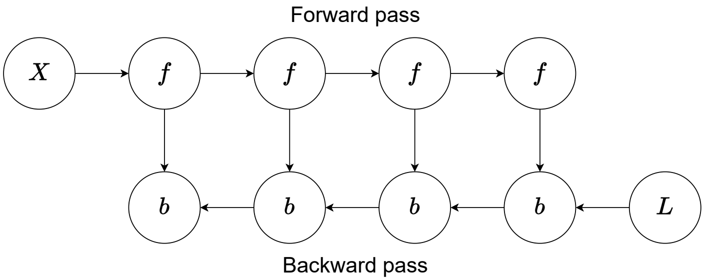


Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The activations marked with an f. The gradient of the loss with respect to the activations and parameters marked with b.

Feedforward Architecture

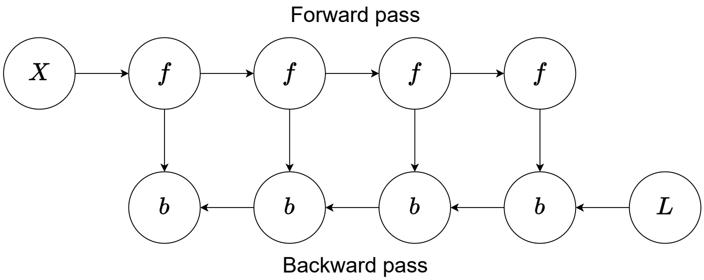


Figure 9: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The activations marked with an f. The gradient of the loss with respect to the activations and parameters marked with b.

! Important

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

Vanilla backpropagation

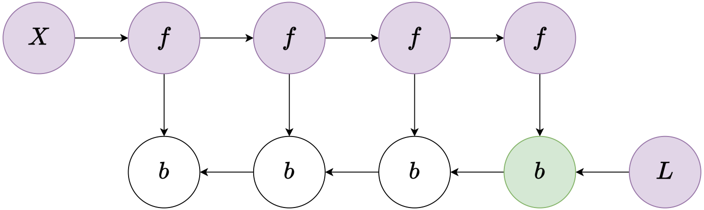


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

Vanilla backpropagation

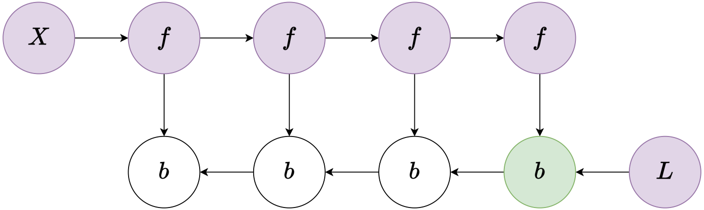


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

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

Vanilla backpropagation

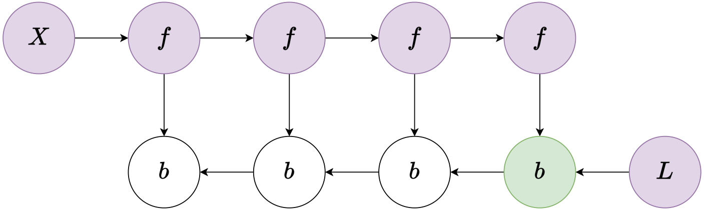


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

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

Vanilla backpropagation

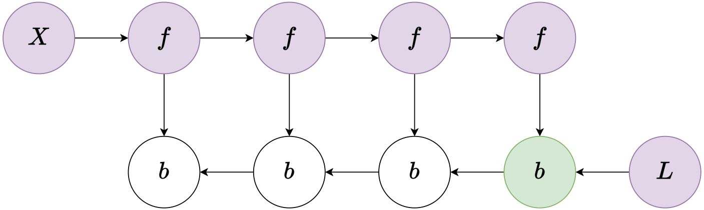


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

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

- Optimal in terms of computation: it only computes each node once.

Vanilla backpropagation

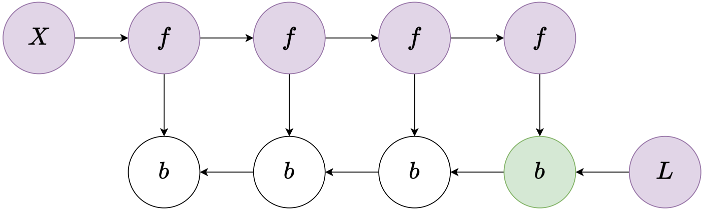


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

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

- Optimal in terms of computation: it only computes each node once.

Vanilla backpropagation

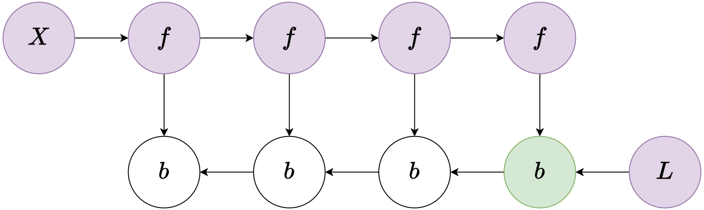


Figure 10: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

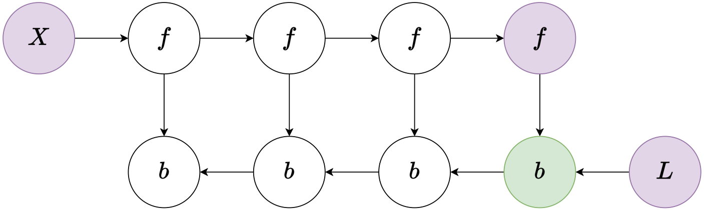


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

Memory poor backpropagation

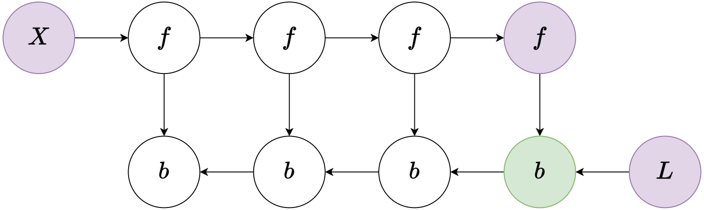


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- Each activation f is recalculated as needed.

Memory poor backpropagation

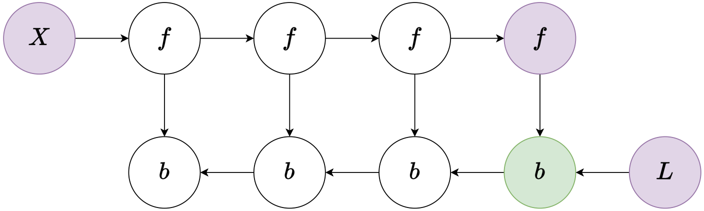


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- Each activation f is recalculated as needed.

Memory poor backpropagation

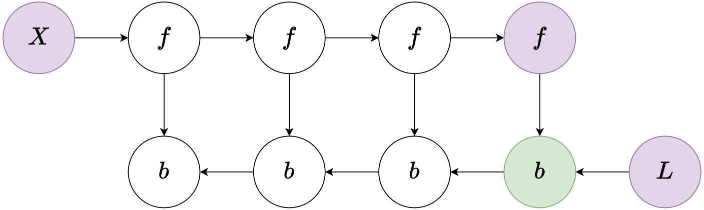


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

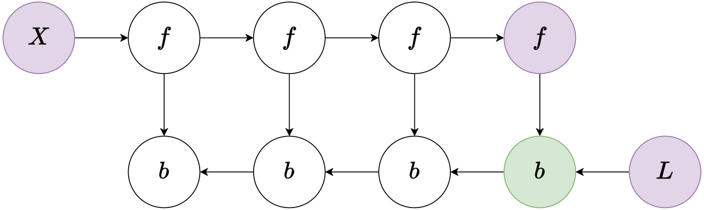


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

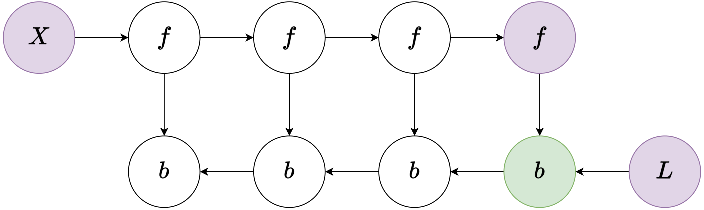


Figure 11: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

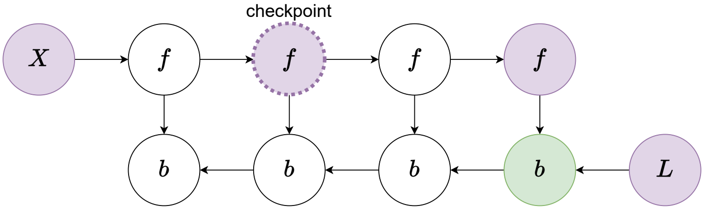


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

Checkpointed backpropagation

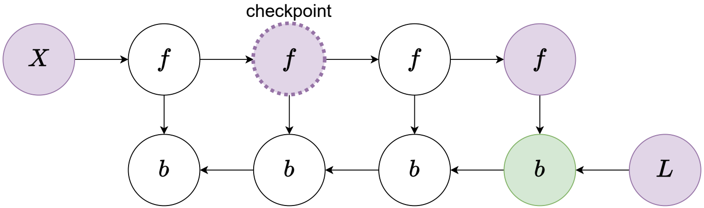


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

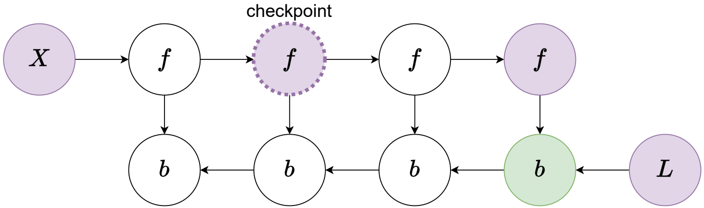


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

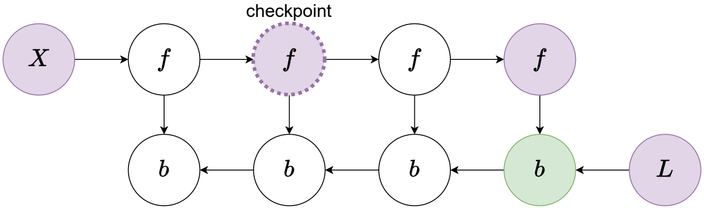


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

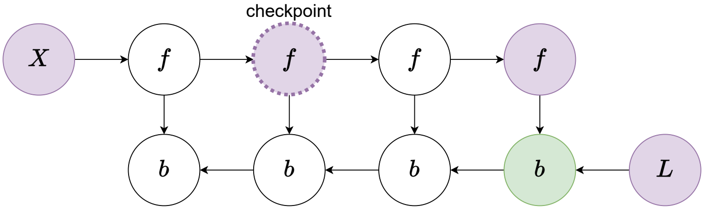


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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

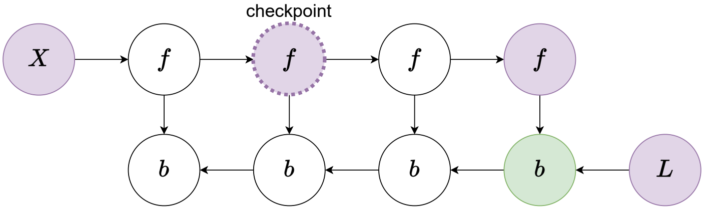


Figure 12: Computation graph for obtaining gradients for a simple feed-forward neural network with n layers. The purple color indicates nodes that are stored in memory.

- 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 then **vanilla** approach.

Gradient checkpointing visualization

The animated visualization of the above approaches

An example of using a gradient checkpointing

Hutchinson Trace Estimation [1]

This example illustrates the estimation the Hessian trace of a neural network using Hutchinson's method, which is an algorithm to obtain such an estimate from matrix-vector products:

Let $X \in \mathbb{R}^{d \times d}$ and $v \in \mathbb{R}^d$ be a random vector such that $\mathbb{E}[vv^T] = I$. Then,

$$\mathrm{Tr}(X) = \mathbb{E}[v^T X v] = \frac{1}{V} \sum_{i=1}^{V} v_i^T X v_i.$$

An example of using Hutchinson Trace Estimation

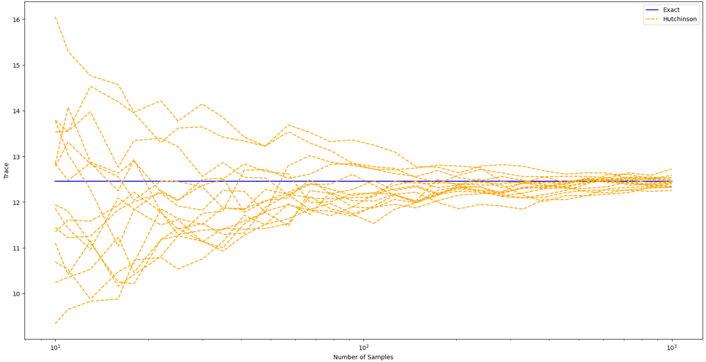


Figure 13: Multiple runs of the Hutchinson trace estimate, initialized at different random seeds.

[1] A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines - M.F. Hutchinson, 1990