TorchJD: Jacobian Descent in PyTorch

Valérian Rey — valerian.rey@gmail.com
Pierre Quinton — pierre.quinton@epfl.ch

PyTorch CONFERENCE EUROPE 2026

GitHub
github.com/SimplexLab/TorchJD

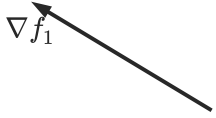
TorchJD
torchjd.org

arXiv
arxiv.org/pdf/2406.16232

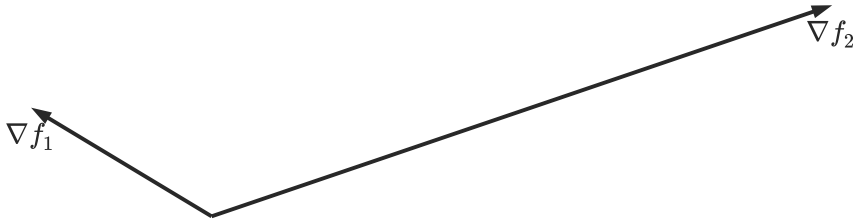
Motivation

Jacobian descent

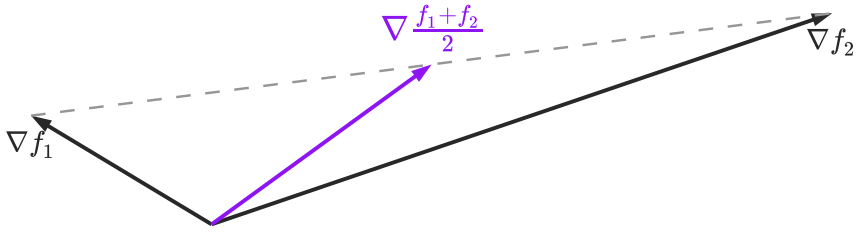
Jacobian descent



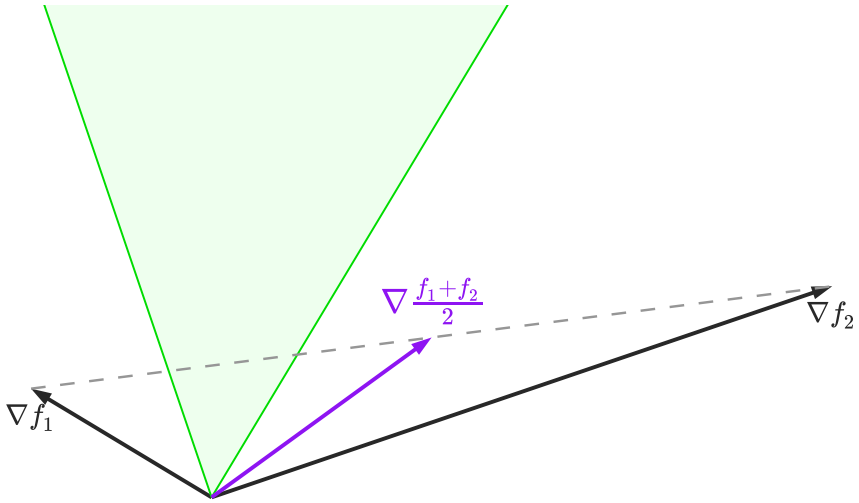
Jacobian descent



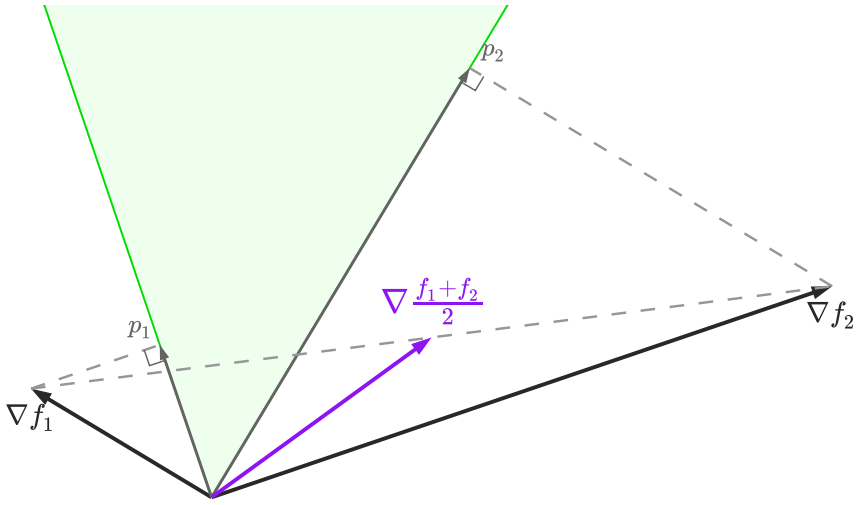
Jacobian descent



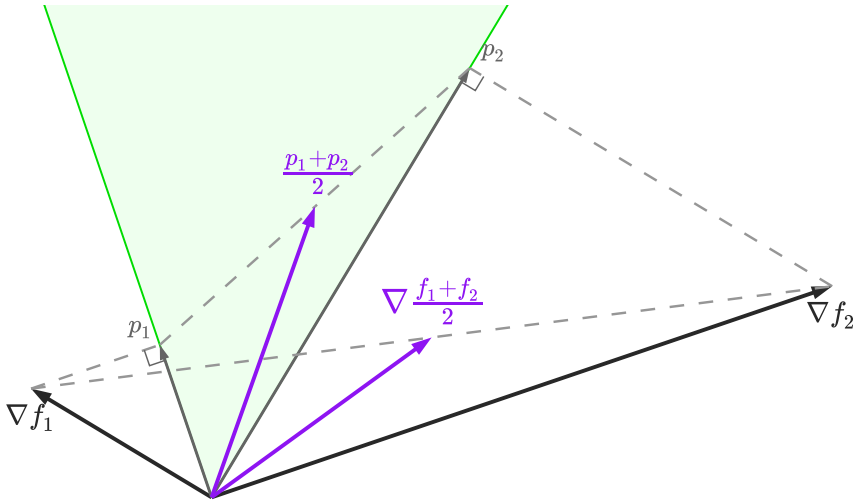
Jacobian descent



Jacobian descent



Jacobian descent



TorchJD

TorchJD: core functions

Library	Function	Description
`torch`	`autograd.grad`	compute gradients
`torch`	`autograd.backward`	compute gradients and accumulate in `.grad`

TorchJD: core functions

Library	Function	Description
`torch`	`autograd.grad`	compute gradients
`torch`	`autograd.backward`	compute gradients and accumulate in `.grad`
`torchjd`	`autojac.jac`	compute jacobians

TorchJD: core functions

Library	Function	Description
`torch`	`autograd.grad`	compute gradients
`torch`	`autograd.backward`	compute gradients and accumulate in `.grad`
`torchjd`	`autojac.jac`	compute jacobians
`torchjd`	`autojac.backward`	compute jacobians and accumulate in `.jac`

TorchJD: core functions

Library	Function	Description
`torch`	`autograd.grad`	compute gradients
`torch`	`autograd.backward`	compute gradients and accumulate in `.grad`
`torchjd`	`autojac.jac`	compute jacobians
`torchjd`	`autojac.backward`	compute jacobians and accumulate in `.jac`
`torchjd`	`autojac.jac_to_grad`	aggregate `.jac` fields into `.grad`

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad
```

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad

batch_size = 16
loss_fn = MSELoss( reduction="none" )  # do not average over the batch
```

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad

batch_size = 16
loss_fn = MSELoss( reduction="none" )  # do not average over the batch

for input, target in dataloader:
  output = model(input)  # shape: [16]
  losses = loss_fn(output, target)  # shape: [16]
```

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad

batch_size = 16
loss_fn = MSELoss( reduction="none" )  # do not average over the batch

for input, target in dataloader:
  output = model(input)  # shape: [16]
  losses = loss_fn(output, target)  # shape: [16]
  autojac.backward(losses)
  # parameters now have a .jac field
```

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad

batch_size = 16
loss_fn = MSELoss(reduction="none")  # do not average over the batch

for input, target in dataloader:
  output = model(input)  # shape: [16]
  losses = loss_fn(output, target)  # shape: [16]
  autojac.backward(losses)
  # parameters now have a .jac field
  autojac.jac_to_grad(model.parameters(), UPGrad())
  # parameters now have a .grad field
```

TorchJD: basic usage

```
from torchjd import autojac
from torchjd.aggregation import UPGrad

batch_size = 16
loss_fn = MSELoss(reduction="none")  # do not average over the batch

for input, target in dataloader:
  output = model(input)  # shape: [16]
  losses = loss_fn(output, target)  # shape: [16]
  autojac.backward(losses)
  # parameters now have a .jac field
  autojac.jac_to_grad(model.parameters(), UPGrad())
  # parameters now have a .grad field
  optimizer.step()
  optimizer.zero_grad()
```

Discussion