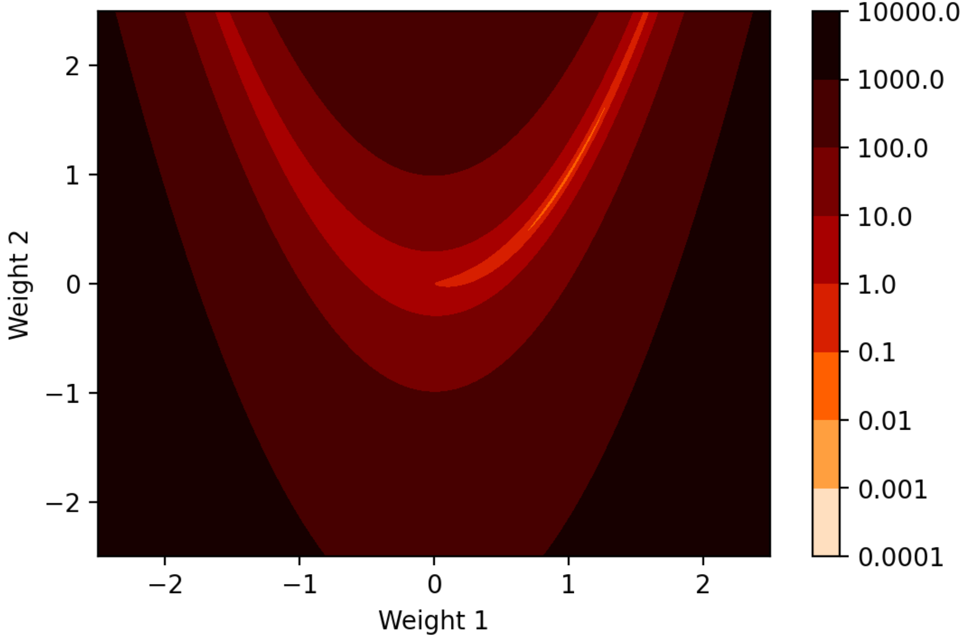
Fisher-Legendre (FishLeg) Optimization of Deep Neural Networks

Jezabel R Garcia, Federica Freddi, Stathi Fotiadis, Maolin Li, Sattar Vakili, Alberto Bernacchia*, Guillaume Hennequin*

Motivation

- Use the curvature

(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}$

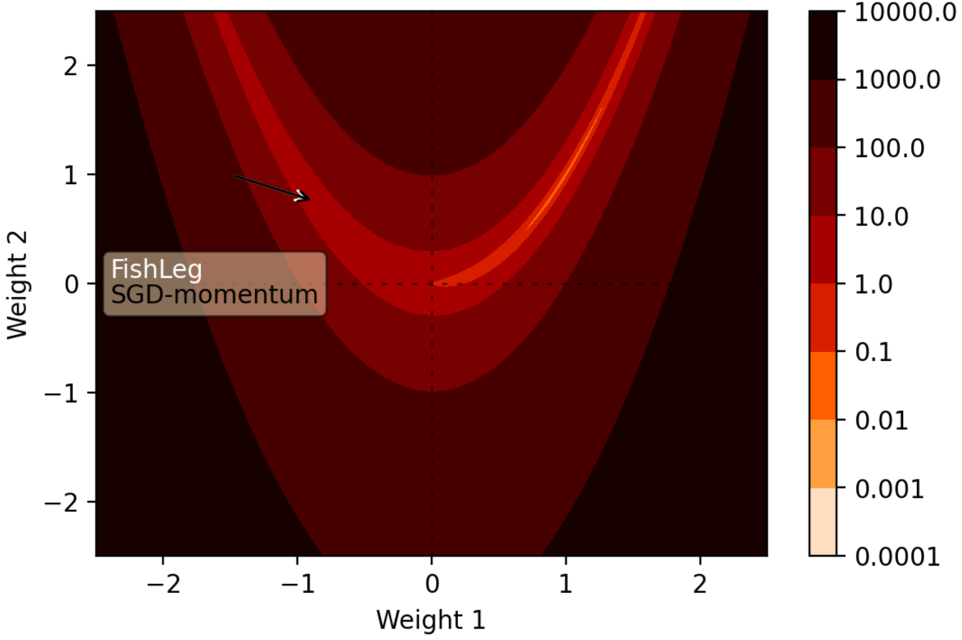


MEDIATEK research

Motivation

- Use the curvature

(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^{\top}\mathbf{s} + \frac{1}{2}\mathbf{s}^{\top}H(\mathbf{w})\mathbf{s}$

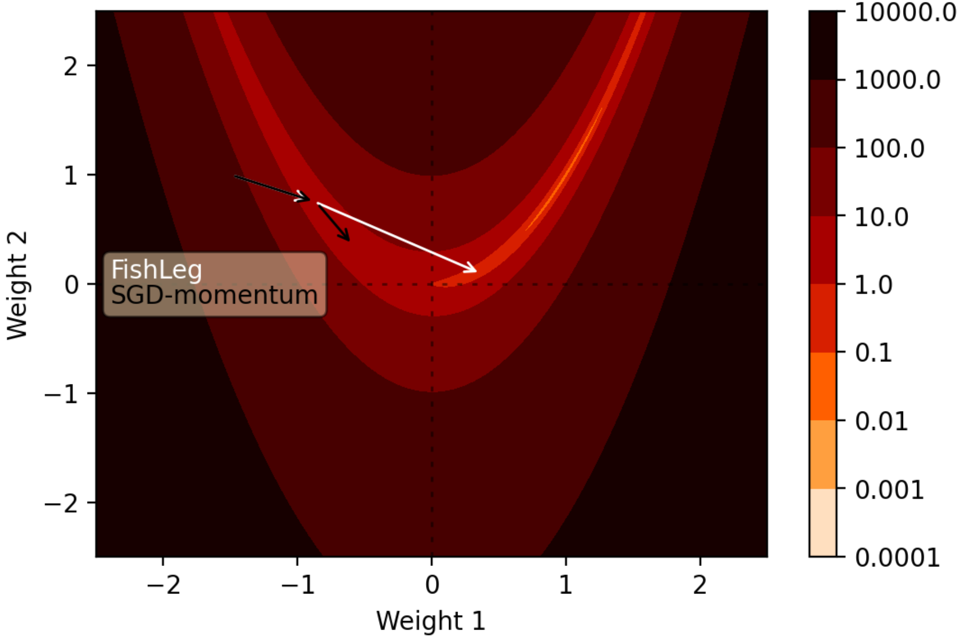


MEDIATEK research

Motivation

- Use the curvature

(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}$

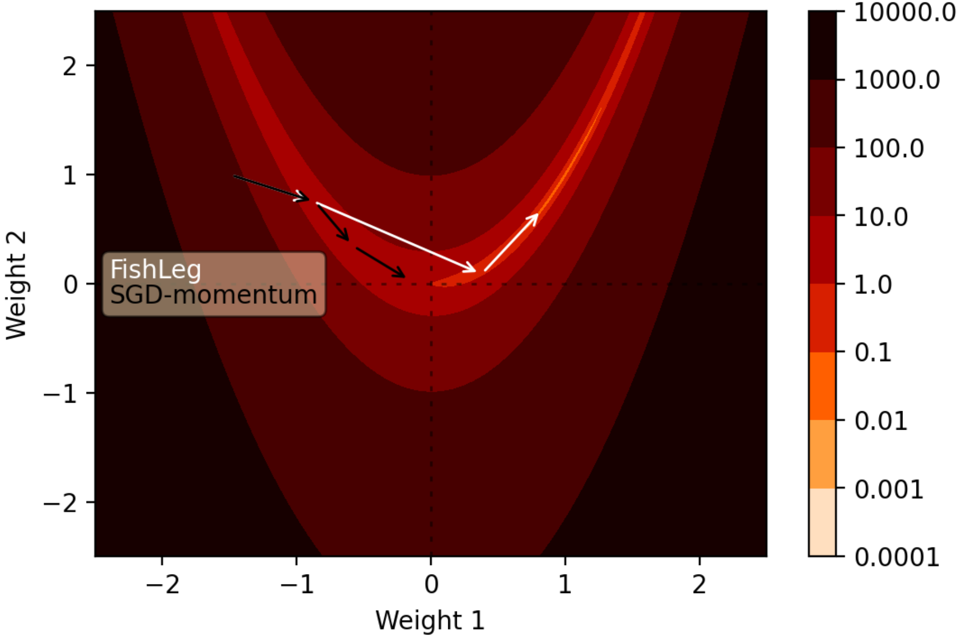


MEDIATEK research

Motivation

- ☐ **Use the curvature**

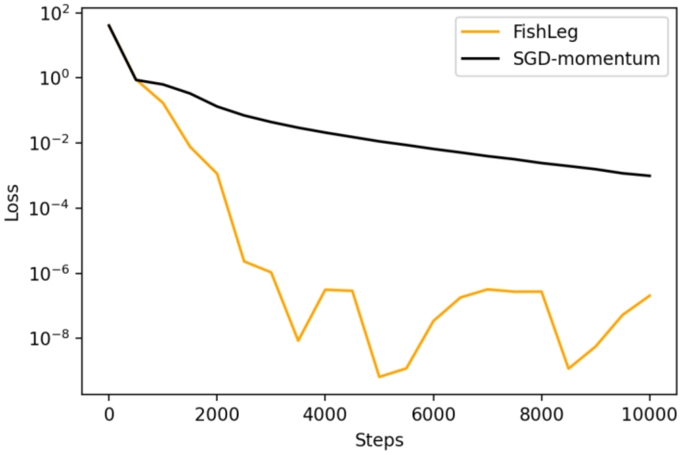
(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}$

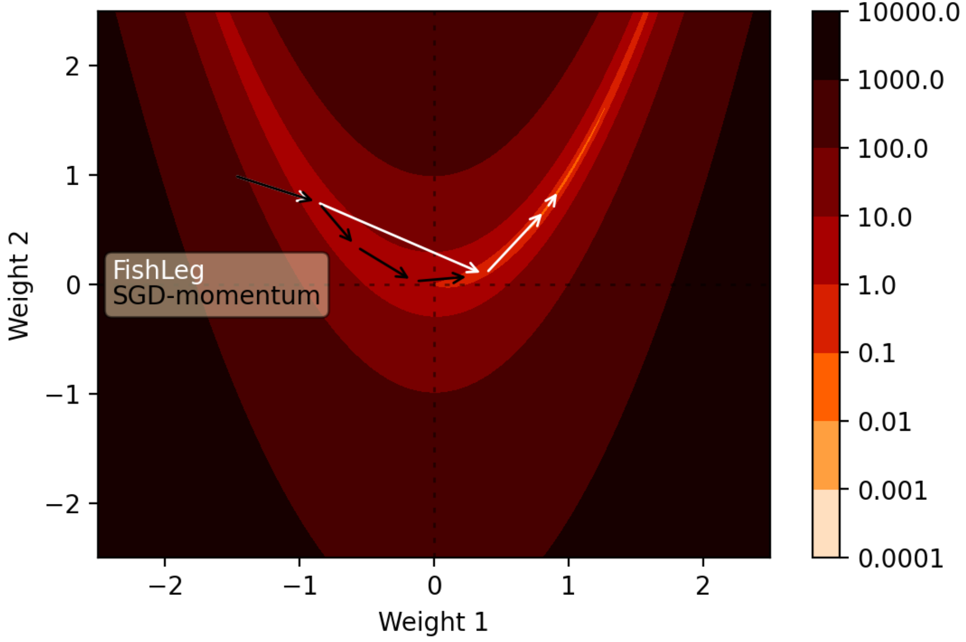


Motivation

- Use the curvature

(1) $$\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}$$



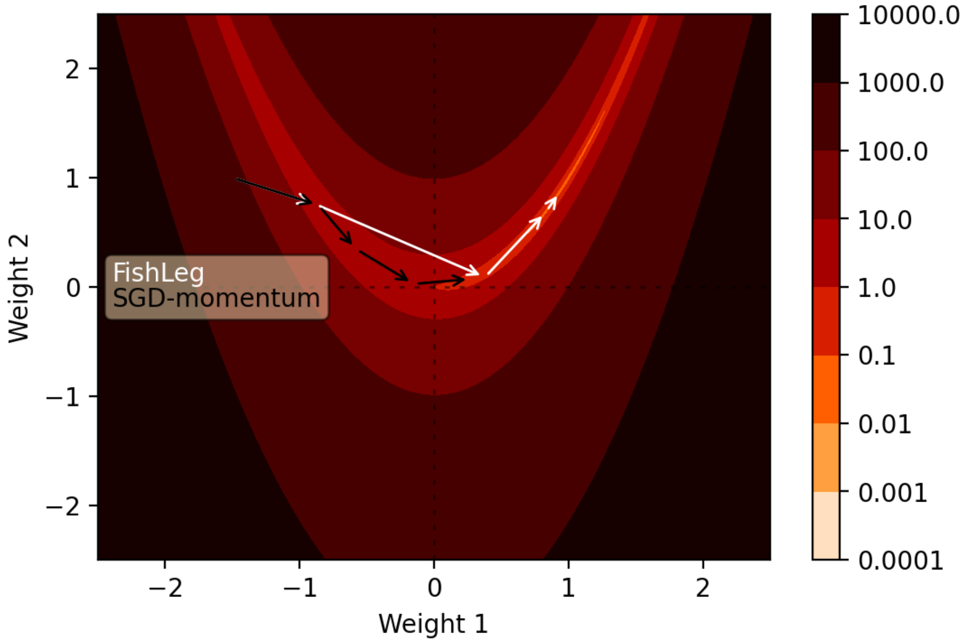


Motivation

- **Use the curvature**

(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}$

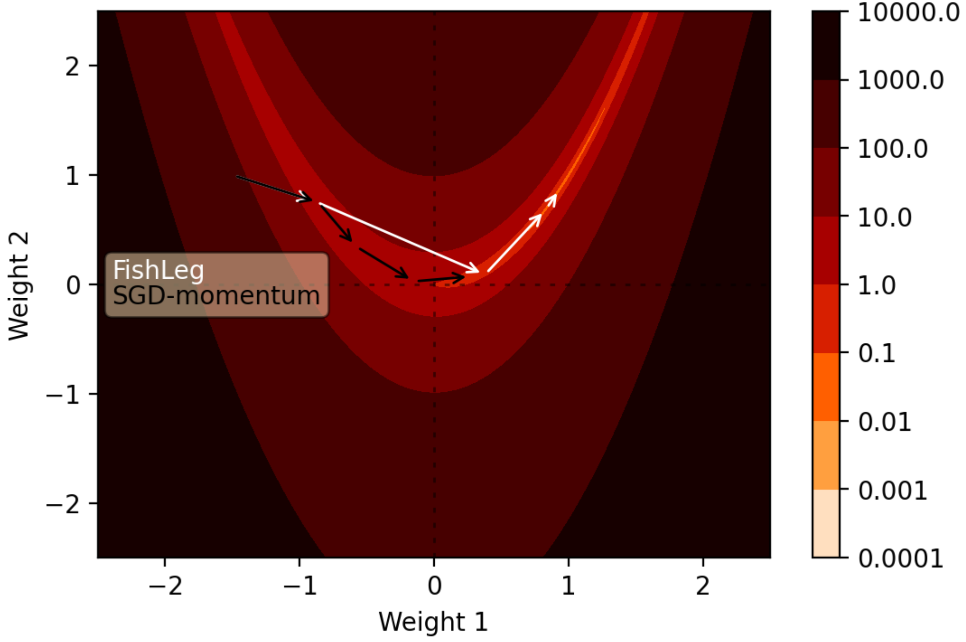
- **Problems in deep learning**
  - ML models have many parameters
  - ML datasets are large



Motivation

- **Use the curvature**

(1) $\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^{\top}\mathbf{s} + \frac{1}{2}\mathbf{s}^{\top}H(\mathbf{w})\mathbf{s}$



- **Problems in deep learning**
  - ML models have many parameters
  - ML datasets are large

FishLeg: Learn the curvature using natural gradients and the Legendre transform

Outline

- Background: Natural gradient, Fisher information matrix
- Method: Fisher-Legendre optimization (FishLeg)
- Experimental results
- PyTorch FishLeg library

Background: Natural Gradient and Fisher Information

Probabilistic model:

Negative Log-Likelihood (NLL): $$l(\theta, D) = -\log p(D|\theta) \quad (2)$$

Fisher Information Matrix (FIM): $$I(\theta) = \mathbb{E}_{D\sim p(D|\theta)} \nabla_\theta\, l(\theta, D)\, \nabla_\theta\, l(\theta, D)^T \quad (3)$$

Background: Natural Gradient and Fisher Information

Probabilistic model:

Negative Log-Likelihood (NLL): $l(\theta, D) = -\log p(D|\theta)$ (2)

Fisher Information Matrix (FIM): $I(\theta) = \mathbb{E}_{D\sim p(D|\theta)} \nabla_\theta\, l(\theta, D)\, \nabla_\theta\, l(\theta, D)^T$ (3)

Maximum likelihood:

$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$ with $L(\theta) = \mathbb{E}_{D\sim p^*} l(\theta, D)$ (4) (5)

Background: Natural Gradient and Fisher Information

Probabilistic model:

Negative Log-Likelihood (NLL): $l(\theta, D) = -\log p(D|\theta)$ (2)

Fisher Information Matrix (FIM): $I(\theta) = \mathbb{E}_{D\sim p(D|\theta)} \nabla_\theta\, l(\theta, D)\, \nabla_\theta\, l(\theta, D)^T$ (3)

Maximum likelihood:

$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$ with $L(\theta) = \mathbb{E}_{D\sim p^*} l(\theta, D)$ (4) (5)

Natural Gradient Descent:

$\theta_{t+1} = \theta_t - \eta I(\theta)^{-1} g(\theta)$ with $g(\theta) = \nabla_\theta\, L(\theta)$ (6) (7)

Method: Inverse Fisher and the LF Conjugate

Cross entropy between $\mathrm{p(D|\theta)}$ and $\mathrm{p(D|\theta+\delta)}$:

(8) $$\mathcal{H}(\theta,\delta)=\mathbb{E}_{D\sim p(D|\theta)}l(\theta+\delta,D)$$

and given the function

(9) $$\tilde{\delta}(\theta,u)=\underset{\delta}{\operatorname{argmin}}\,\mathcal{H}(\theta,\delta)-u^T\delta$$

We prove that,

(10) $$I(\theta)^{-1}=\nabla_u\tilde{\delta}(\theta,0)$$

and following, for the natural gradient step:

(11) $$\theta_{t+1}=\theta_t-\eta\nabla_u\tilde{\delta}(\theta_t,0)g(\theta_t)$$

Method: Inverse Fisher and the LF Conjugate

Cross entropy between $\mathrm{p}(\mathrm{D}|\theta)$ and $\mathrm{p}(\mathrm{D}|\theta + \delta)$:

where $l(\theta, D)$ is the negative log-likelihood

(8) $$\mathcal{H}(\theta, \delta) = \mathbb{E}_{D\sim p(D|\theta)} l(\theta + \delta, D)$$

and given the function

(9) $$\tilde{\delta}(\theta, u) = \operatorname*{argmin}_{\delta} \mathcal{H}(\theta, \delta) - u^T \delta$$

expression to minimise for the Legendre-Fenchel conjugate of the cross-entropy

We prove that,

(10) $$I(\theta)^{-1} = \nabla_u \tilde{\delta}(\theta, 0)$$

and following, for the natural gradient step:

(11) $$\theta_{t+1} = \theta_t - \eta \nabla_u \tilde{\delta}(\theta_t, 0) g(\theta_t)$$

Recall natural gradient step

$$\theta_{t+1} = \theta_t - \eta I(\theta)^{-1} g(\theta_t)$$

MEDIATEK research

Method: Inverse Fisher and the LF Conjugate

Cross entropy between $p(D|\theta)$ and $p(D|\theta + \delta)$:

where $l(\theta, D)$ is the negative log-likelihood

(8) $$\mathcal{H}(\theta, \delta) = \mathbb{E}_{D\sim p(D|\theta)} l(\theta + \delta, D)$$

and given the function

(9) $$\boldsymbol{\tilde{\delta}(\theta, u) = \underset{\delta}{\operatorname{argmin}}\, \mathcal{H}(\theta, \delta) - u^T \delta}$$

expression to minimise for the Legendre-Fenchel conjugate of the cross-entropy

We prove that,

(10) $$\boldsymbol{I(\theta)^{-1} = \nabla_u \tilde{\delta}(\theta, 0)}$$

and following, for the natural gradient step:

(11) $$\theta_{t+1} = \theta_t - \eta \nabla_u \tilde{\delta}(\theta_t, 0) g(\theta_t)$$

Recall natural gradient step

$$\theta_{t+1} = \theta_t - \eta I(\theta)^{-1} g(\theta_t)$$

Method: Learning the approximation

We want to learn an approximation $\bar{\delta}\,(\theta, u, \lambda)$ of the true $\tilde{\delta}(\theta, u)$.

(12) $$\bar{\boldsymbol{\delta}}\,(\boldsymbol{\theta}, \boldsymbol{u}, \boldsymbol{\lambda}) = \boldsymbol{\mathcal{Q}}(\boldsymbol{\lambda})\boldsymbol{u}$$

where $\mathcal{Q}(\lambda)$ therefore estimates the inverse FIM.

In order to learn λ, we perform gradient descent using Adam on the auxiliary loss

(13) $$\mathcal{A}(\theta, u, \lambda) = \mathcal{H}(\theta, \mathcal{Q}(\lambda)u) - u^T \mathcal{Q}(\lambda)u$$

Method: Learning the approximation

We want to learn an approximation $\bar{\delta}(\theta, u, \lambda)$ of the true $\tilde{\delta}(\theta, u)$.

(12) $$\bar{\boldsymbol{\delta}}(\boldsymbol{\theta}, \boldsymbol{u}, \boldsymbol{\lambda}) = \boldsymbol{Q}(\boldsymbol{\lambda})\boldsymbol{u}$$

where $Q(\lambda)$ therefore estimates the inverse FIM.

Because
$Q(\lambda) = \nabla_u \bar{\delta}(\theta, 0, \lambda) \cong I(\theta)^{-1}$

In order to learn λ, we perform gradient descent using Adam on the auxiliary loss

(13) $$\mathcal{A}(\theta, u, \lambda) = \mathcal{H}(\theta, Q(\lambda)u) - u^T Q(\lambda)u$$

Method: Learning the approximation

We want to learn an approximation $\bar{\delta}\,(\theta, u, \lambda)$ of the true $\tilde{\delta}(\theta, u)$.

(12) $$\boldsymbol{\bar{\delta}\,(\theta, u, \lambda) = \mathcal{Q}(\lambda) u}$$

where $\mathcal{Q}(\lambda)$ therefore estimates the inverse FIM.

Because
$\mathcal{Q}(\lambda) = \nabla_u \bar{\delta}(\theta, 0, \lambda) \cong I(\theta)^{-1}$

In order to learn λ, we perform gradient descent using Adam on the auxiliary loss

(13) $$\mathcal{A}(\theta, u, \lambda) = \mathcal{H}(\theta, \mathcal{Q}(\lambda) u) - u^T \mathcal{Q}(\lambda) u$$

From the LF conjugate
$\tilde{\delta}(\theta, u) = \underset{\delta}{\operatorname{argmin}}\, \mathcal{H}(\theta, \delta) - u^T \delta$

MEDIATEK research

Method: FishLeg Algorithm

In summary, we alternate between

Outer loop

(14) $$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \boldsymbol{\alpha}\,\mathbf{AdamUpdate}(\nabla_\lambda \boldsymbol{\mathcal{A}}(\boldsymbol{\theta}_t, \boldsymbol{\epsilon g}(\boldsymbol{\theta}_t), \boldsymbol{\lambda}_t))$$

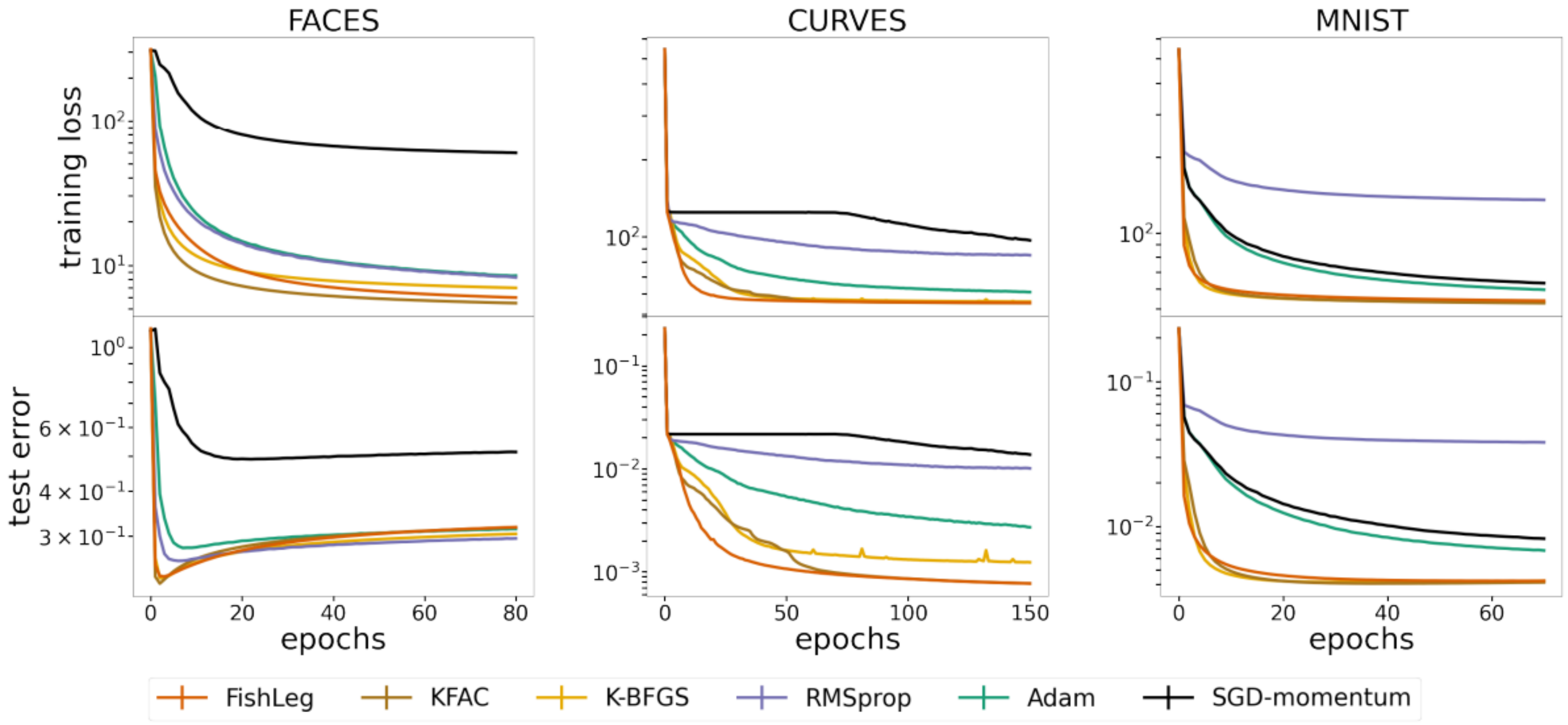
(15) $$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta} \boldsymbol{Q}(\boldsymbol{\lambda}_{t+1}) \boldsymbol{g}(\boldsymbol{\theta}_t)$$

Inner loop

For the matrix $Q(\lambda)$, we use a Kronecker-factored block-diagonal structure that follows the structure of the layers.

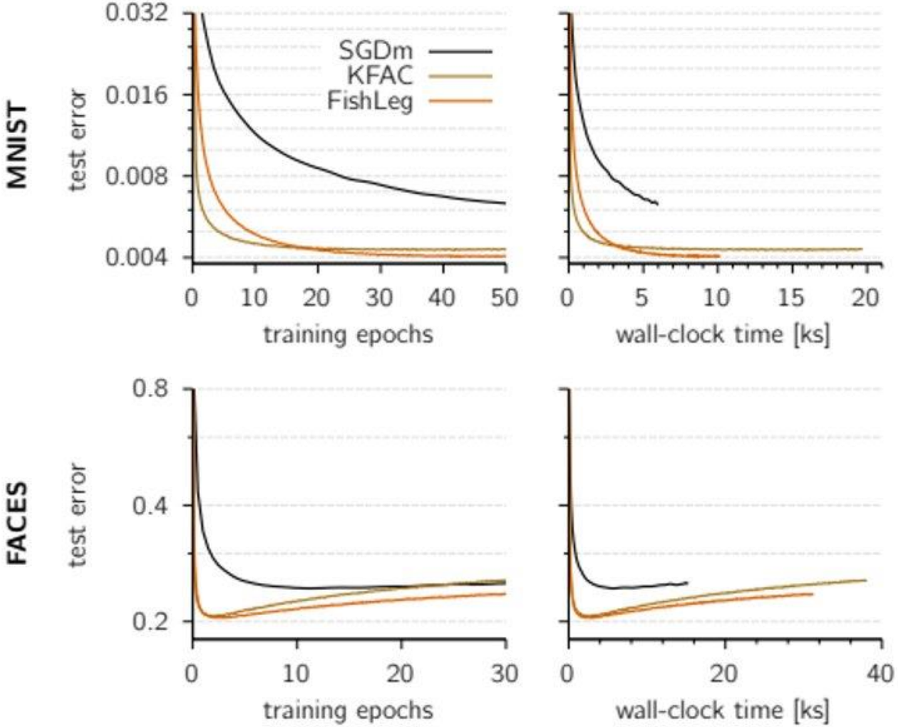
MEDIATEK research

Results



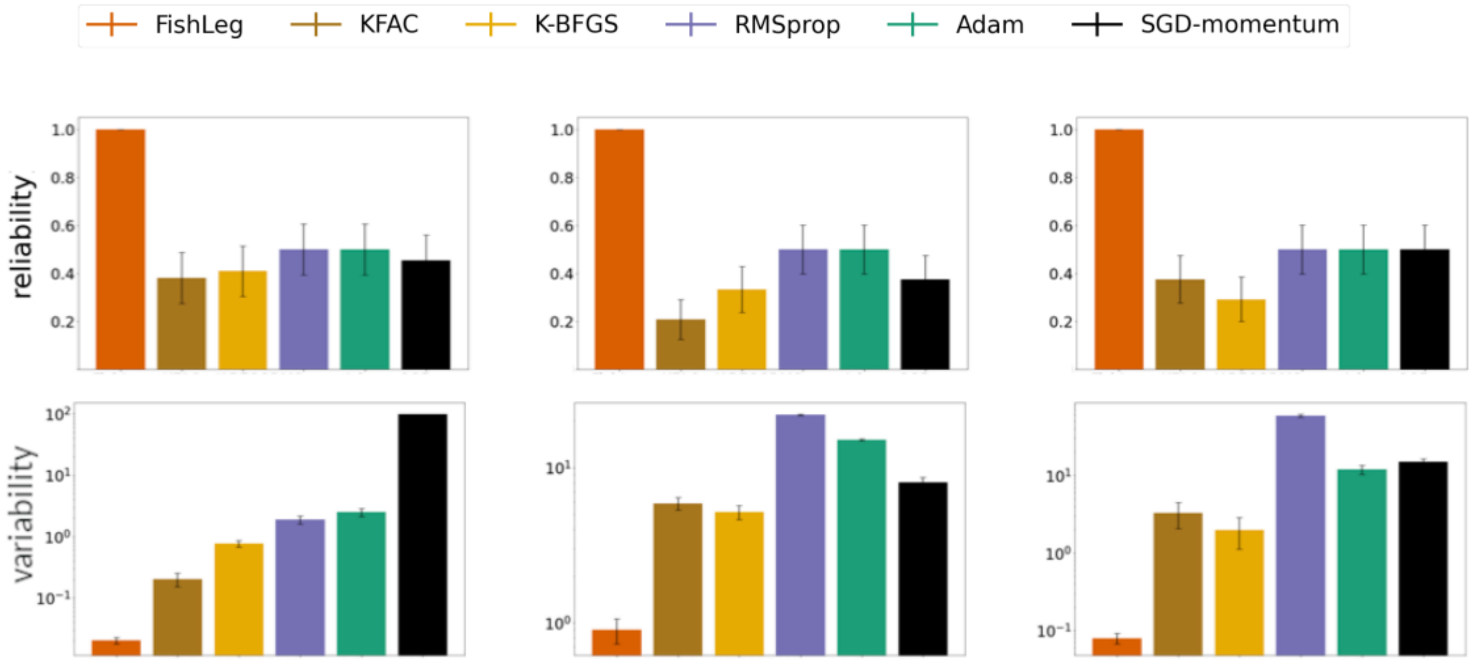
MEDIATEK research

Results



MNIST
test error
0.032
0.016
0.008
0.004
SGDm
KFAC
FishLeg
0
10
20
30
40
50
training epochs
0
5
10
15
20
wall-clock time [ks]
FACES
test error
0.8
0.4
0.2
0
10
20
30
training epochs
0
20
40
wall-clock time [ks]

Results



MEDIATEK research

FishLeg PyTorch Library

New repository

mtkresearch / **FishLeg** Public

PyTorch

Submission repository

someauthors / **fishleg** Public

Jax and OCaml

README.md

FishLeg

Repository containing a PyTorch implementation of the Fisher-Legendre Second Order Optimization method.

DOCUMENTATION | TECHNICAL SUPPORT | RELEASE V1.0 | LICENSE APACHE-2.0

Library Example – How to?

Same model definition for both optimizers

Example for training ResNet18 model:

```
net = ResNet18()
```

FishLeg

```
optimizer = FishLeg(net, draw, nll, likelihood, auxloader)

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = optimizer.model.forward(inputs)
    loss = likelihood.nll(targets, outputs)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Adam

```
optimizer = optim.AdamW(net.parameters())

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, targets)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Library Example – How to?

Same model definition for both optimizers

Example for training ResNet18 model:

```
net = ResNet18()
```

FishLeg

```
optimizer = FishLeg(net, draw, nll, likelihood, auxloader)

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = optimizer.model.forward(inputs)
    loss = likelihood.nll(targets, outputs)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Adam

```
optimizer = optim.AdamW(net.parameters())

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, targets)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Library Example – How to?

Example for training a ResNet model:

```
net = ResNet18()
```

FishLeg

Not dependent on the model architecture

```
from FishLeg import CategoricalLikelihood
# GaussianLikelihood, BernoulliLikelihood, SoftMaxLikelihood
likelihood = CategoricalLikelihood()

def nll(model, data_x, data_y):
    pred_y = model.forward(data_x)
    return likelihood.nll(data_y, pred_y)

def draw(model, data_x):
    pred_y = model.forward(data_x)
    return likelihood.draw(pred_y)

auxloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=2)
```

```
optimizer = FishLeg(net, draw, nll, likelihood, auxloader)

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = optimizer.model.forward(inputs)
    loss = likelihood.nll(targets, outputs)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Adam

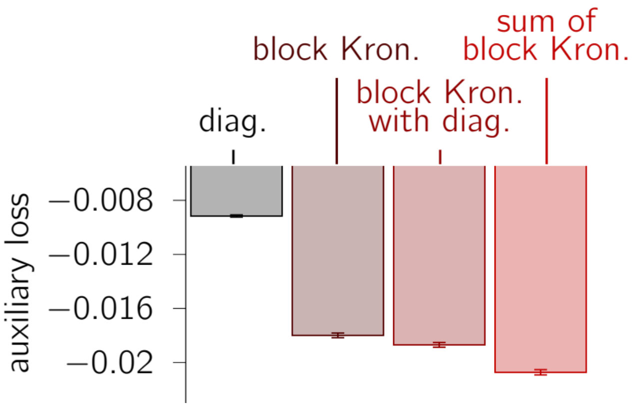
```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.AdamW(net.parameters())

for batch_idx, (inputs, targets) in enumerate(trainloader):
    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, targets)
    loss.backward() # grads are stored in the model
    optimizer.step()
```

Flexibility of the method

Very flexible in the choice of approximation for the Inverse Fisher.



Block Kron.

$$Q_\ell = (R_\ell R_\ell^T \otimes L_\ell L_\ell^T)$$

Block Kron. with diag.

$$Q_\ell = A_\ell (R_\ell \otimes L_\ell) B_\ell^2 (R_\ell^T \otimes L_\ell^T) A_\ell$$

Sum of block Kron.

$$Q_\ell = (R_\ell^{(1)} {R_\ell^{(1)}}^T) \otimes (L_\ell^{(1)} {L_\ell^{(1)}}^T) + (R_\ell^{(2)} {R_\ell^{(2)}}^T) \otimes (L_\ell^{(2)} {L_\ell^{(2)}}^T)$$

Contributing to the Library

It is possible to ***easily write new custom layer approximations*** compatible with the library.

The new layer classes need to inherit from the FishModule abstract class that requires the implementation of the ***Qv method***.

Abstract Module: FishModule

```python
class FishModule(nn.Module):
    """Base class for all neural network modules in FishLeg to

    #. Initialize auxiliary parameters, :math:`\lambda` and its forms, :math:`Q(\lambda)`.
    #. Specify quick calculation of :math:`Q(\lambda)v` products.
```

⋮

```python
    @abstractmethod
    def Qv(self, aux: Dict, v: Tuple[Tensor, ...]) -> Tuple[Tensor, ...]:
        Args:
            aux: (Dict, required): auxiliary parameters,
                    :math:`\lambda`, a dictionary with keys, the name
                    of the auxiliary parameters, and values, the auxiliary parameters
                    of the module. These auxiliaray parameters will form :math:`Q(\lambda)`.
            v: (Tuple[Tensor, ...], required): Values of the original parameters,
                    in an order that align with `self.order`, to multiply with
                    :math:`Q(\lambda)`.
        Returns:
            Tuple[Tensor, ...]: The calculated :math:`Q(\lambda)v` products,
                    in same order with `self.order`.

        """
        raise NotImplementedError(f'Module is missing the required "Qv" function')
```

MEDIATEK research

Contributing to the Library

Example Implementation: FishLinear

```
class FishLinear(nn.Linear, FishModule):
```

⋮

```
    def Qv(self, v: Tuple[Tensor, Tensor]) -> Tuple[Tensor, Tensor]:
        """For fully-connected layers, the default structure of :math:`Q` as a
        block-diagonal matrix is,

        .. math::
                    Q_l = (R_lR_l^T \otimes L_lL_l^T)

        where :math:`l` denotes the l-th layer. The matrix :math:`R_l` has size
        :math:`(N_{l-1} + 1) \\times (N_{l-1} + 1)` while the matrix :math:`L_l` has
        size :math:`N_l \\times N_l`. The auxiliarary parameters :math:`\lambda`
        are represented by the matrices :math:`L_l, R_l`.

        """
        L = self.fishleg_aux["L"]
        R = self.fishleg_aux["R"]
        u = torch.cat([v[0], v[1][:, None]], dim=-1)
        z = torch.linalg.multi_dot((R.T, R, u, L, L.T))
        return (z[:, :-1], z[:, -1])
```

FishLinear

This specific implementation of the Qv method implements the Block Kronecker approximation.

Block Kron.

$$Q_\ell = (R_\ell R_\ell^T \otimes L_\ell L_\ell^T)$$

Contact email:

jezabel.garcia@mtkresearch.com
federica.freddi@mtkresearch.com

Thank you for your attention

Questions and Discussion

Poster Session: MH1-2-3-4 #49

Acknowledgements

Thank you to the new members of the team that are currently contributing to the continuation of FishLeg:
Jamie McGowan, Rui Xia, Chan Hsu, Ritwik Niyogi, Yilei Liang