Global curvature for second-order optimization of neural networks

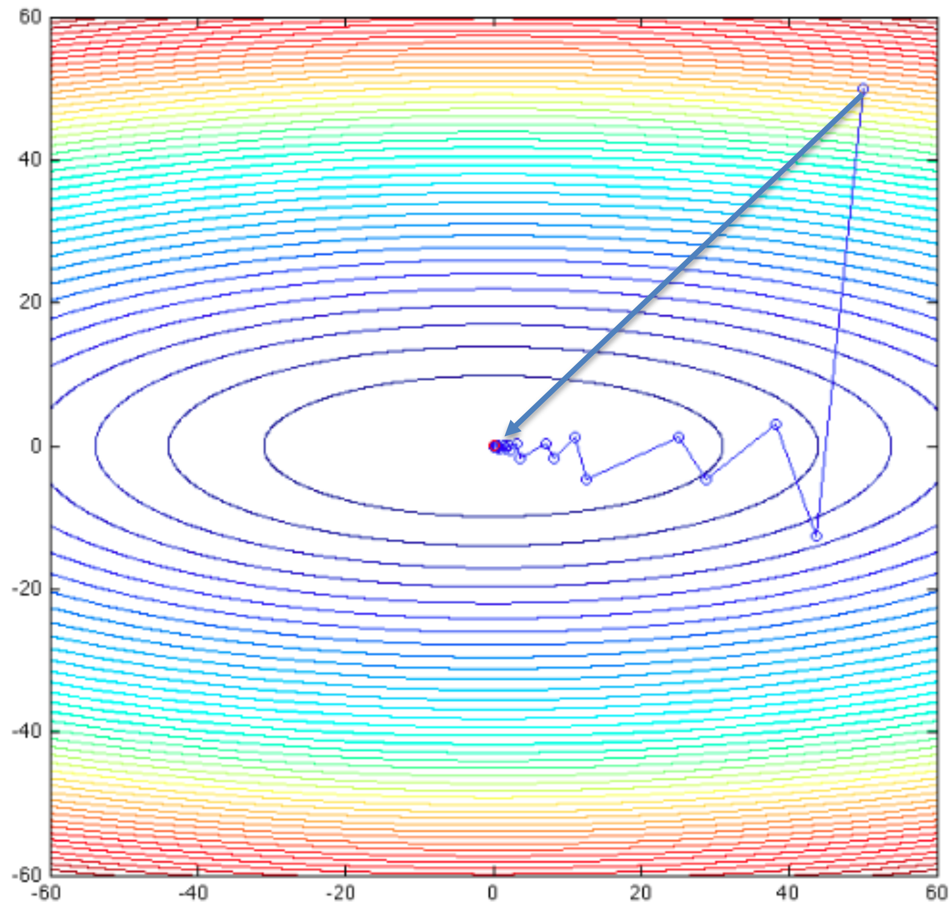
Alberto Bernacchia

MediaTek Research

Second-order optimization

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha M \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

M is the inverse of a curvature matrix

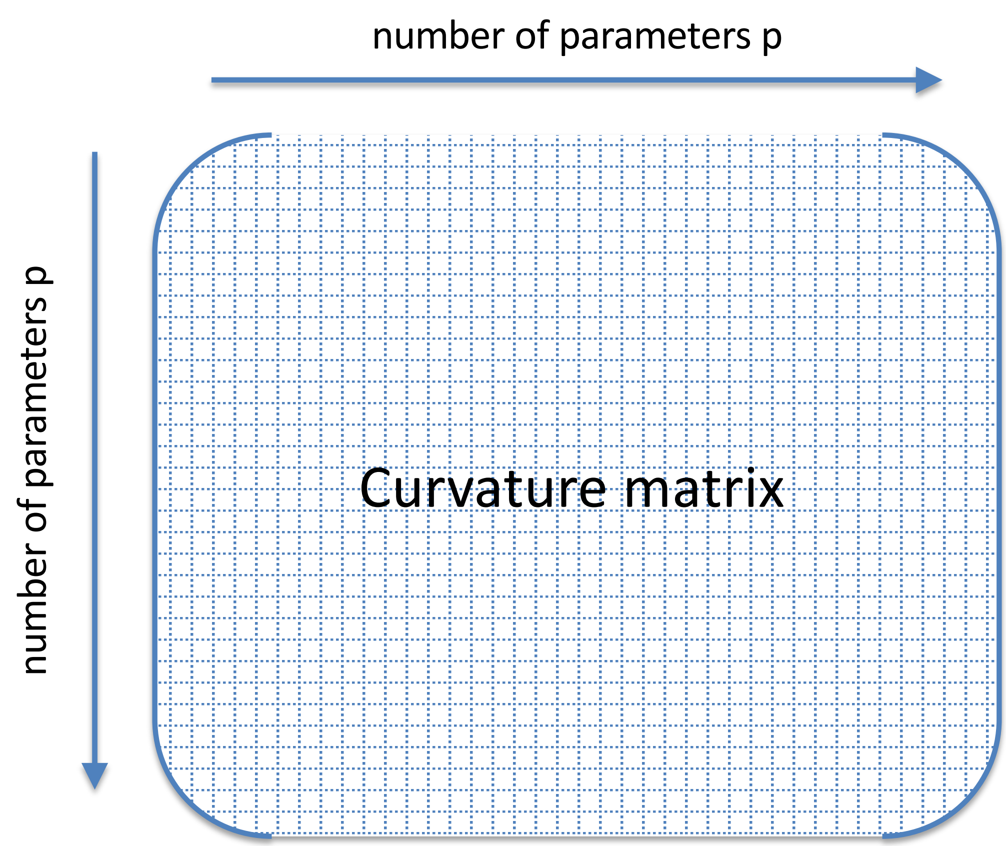


Knowledge of curvature allows using different learning rates for different directions

Second-order optimization

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha M \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

M is the inverse of a curvature matrix



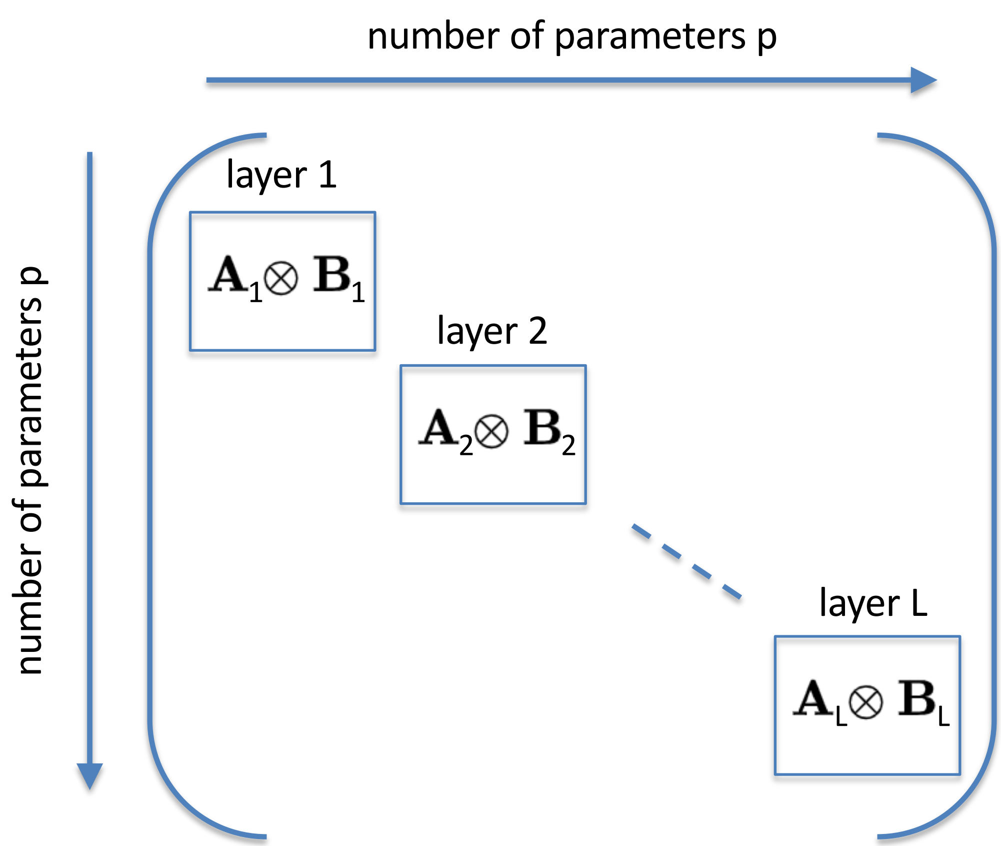
Too large to compute and store for neural networks

Complexity $O(p^2)$

Second-order optimization

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha M \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

M is the inverse of a curvature matrix



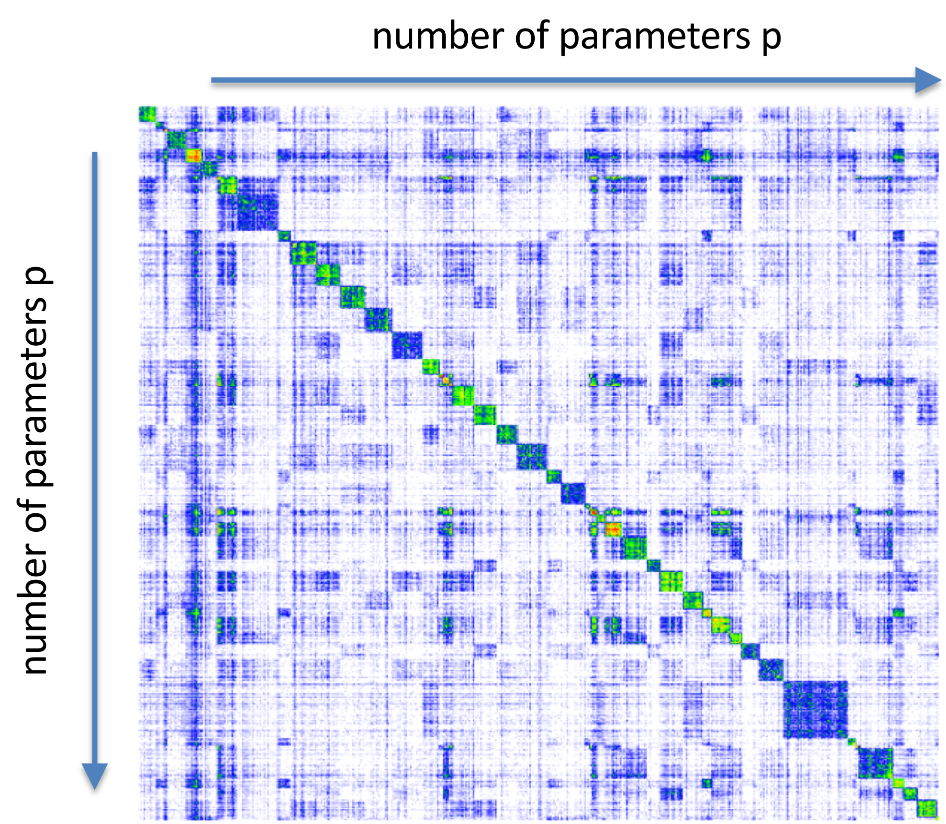
Block-diagonal and Kronecker-factored approximation

Complexity O(p)

Second-order optimization

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha M \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

M is the inverse of a curvature matrix



True curvature is not (block-) diagonal

Complexity $O(p^2)$

Main contribution

Can we estimate and use the full curvature matrix accurately and tractably?

Yes, but we need to look at *non-local* properties of the curvature

Global curvature by ensemble averaging

Global Hessian

$$H_t = \mathop{\mathbb{E}}_{\boldsymbol{\theta}_t} \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t)$$

Curvature matrix is global, **averaged over a model distribution** (ensemble)

It depends on time, the distribution changes during optimization

Gradient covariance

$$\Sigma_t = \mathop{\mathbb{E}}_{\boldsymbol{\theta}_t} \nabla \mathcal{L}(\boldsymbol{\theta}_t) \nabla \mathcal{L}(\boldsymbol{\theta}_t)^T - \boldsymbol{\mu}_t \boldsymbol{\mu}_t^T$$

$$\boldsymbol{\mu}_t = \mathop{\mathbb{E}}_{\boldsymbol{\theta}_t} \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

Global curvature by ensemble averaging

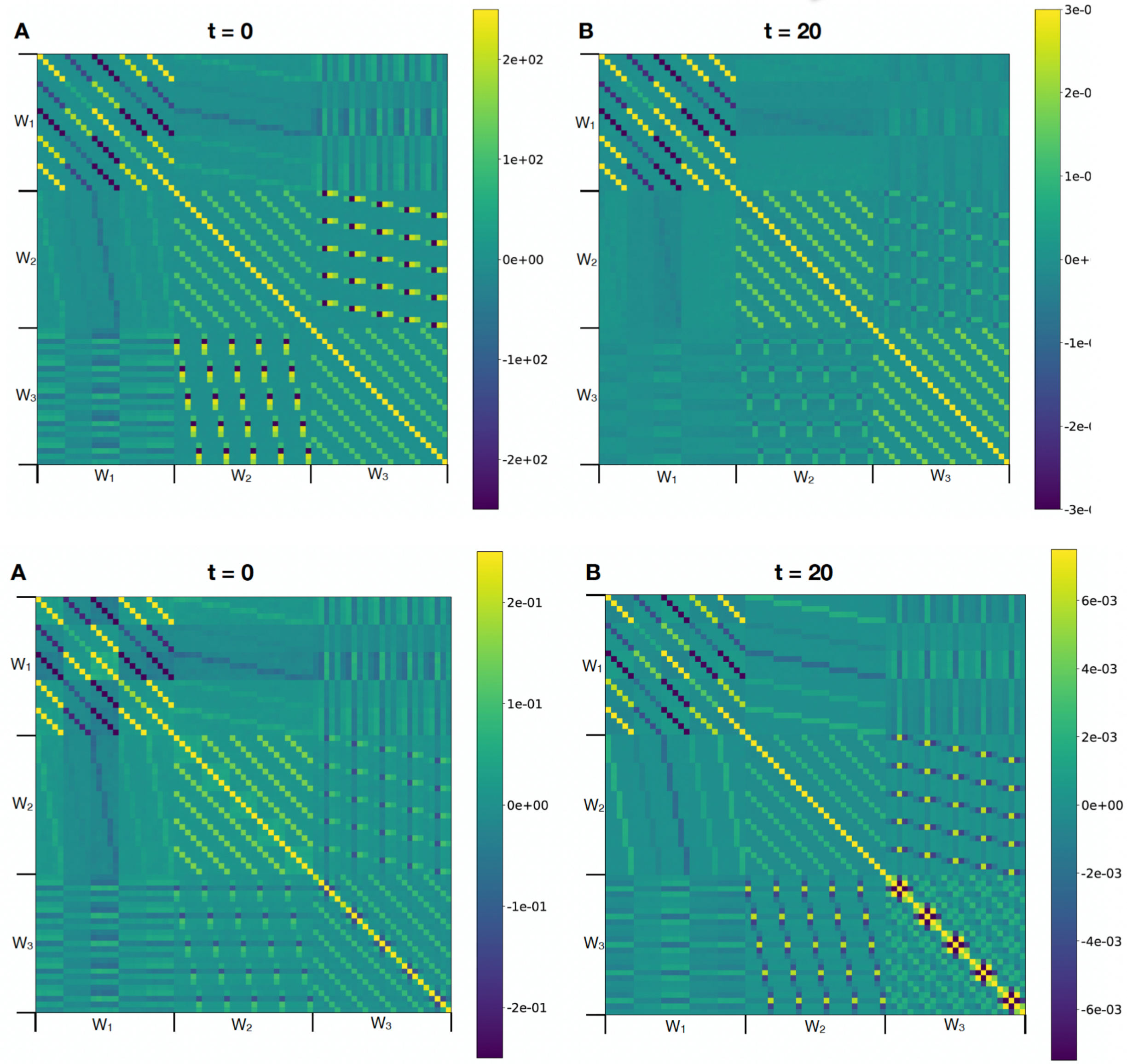
3-layer MLP, ReLU, no bias, 5 neurons per layer = 75 parameters

10,000 models drawn from a Gaussian distribution (init)

gradient descent

Global Hessian

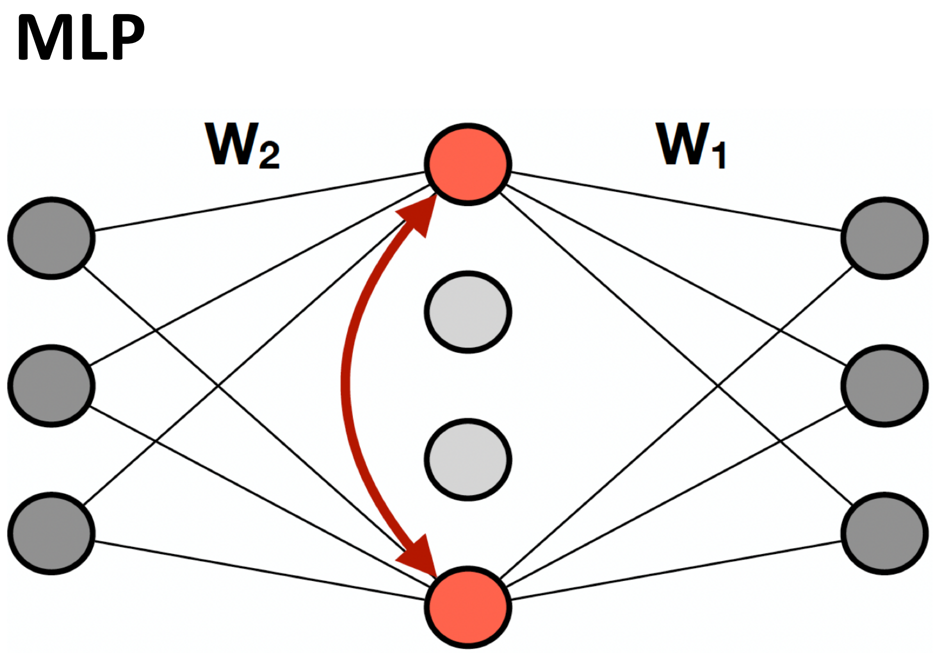
Gradient covariance

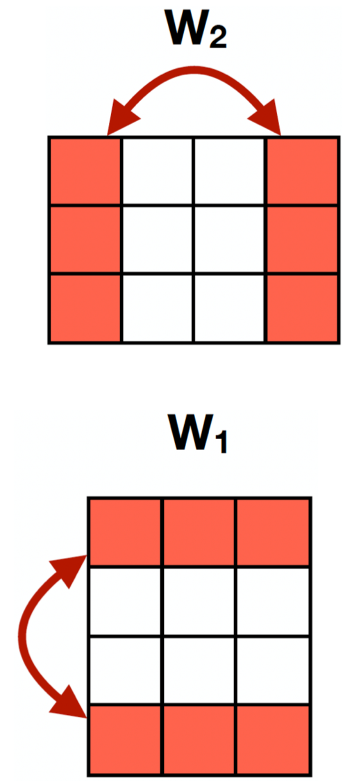


The structure of curvature seems identical, and is **not** block-diagonal

Can we capture this structure with a general theory?

Symmetries of a neural network





$$\mathbf{h}_\ell = W_\ell \, \sigma_\ell(\mathbf{h}_{\ell-1}) + \mathbf{b}_\ell \qquad \ell = 1, \ldots, L$$

$$\mathbf{b}_\ell \longrightarrow V_\ell \mathbf{b}_\ell$$

$$W_\ell \longrightarrow V_\ell W_\ell V_{\ell-1}^T$$

Output of MLP is invariant for any permutation matrices V_ℓ

$$\mathcal{L}(G\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) \quad \forall G \in \mathbb{G}$$

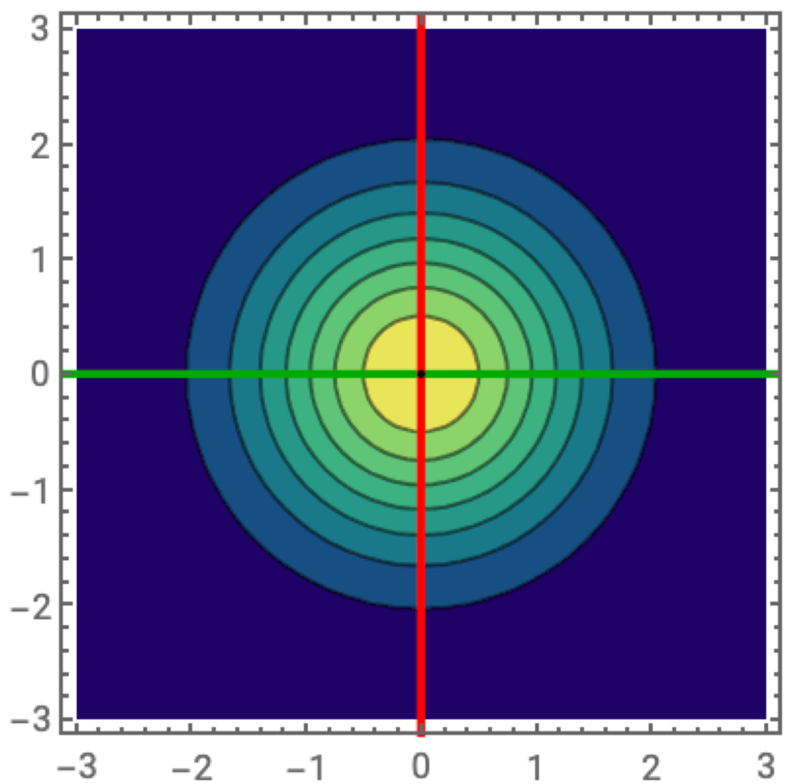
G includes permutations of all layers

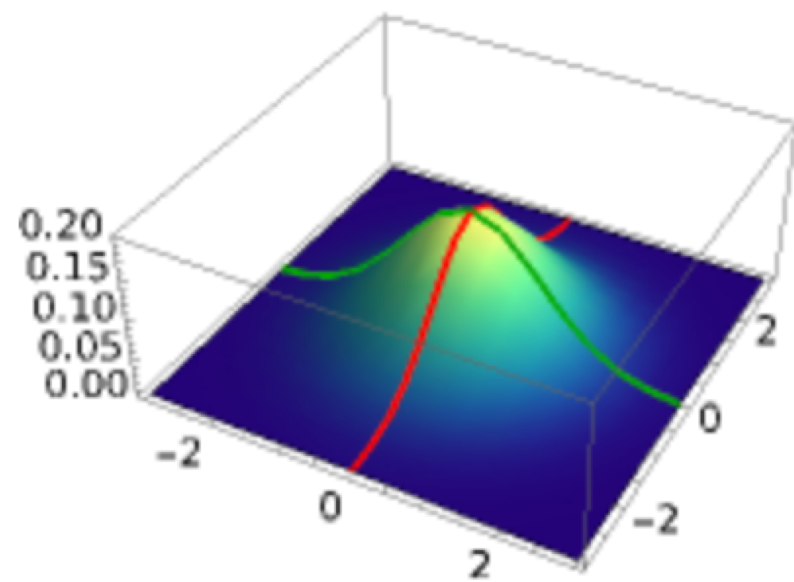
Symmetries of a model distribution

Assumption

$$p_0(G\boldsymbol{\theta}_0) = p_0(\boldsymbol{\theta}_0) \quad \forall G \in \mathbb{G}$$

The distribution of parameters (at time zero) and the loss are invariant for the same group of transformations





Symmetries of a model distribution

Assumption

$$p_0(G\boldsymbol{\theta}_0) = p_0(\boldsymbol{\theta}_0) \quad \forall G \in \mathbb{G}$$

Gradient Descent,
Momentum,
Adam,
...

Theorem 2.2. *Assume that the update rule* $\boldsymbol{\theta}_t = \mathbf{u}_t(\boldsymbol{\theta}_{t-1})$ *is differentiable and its Jacobian is non-singular almost everywhere. Furthermore, it is equivariant under a volume-preserving transformation* G*, namely*

$$\mathbf{u}_t(G\boldsymbol{\theta}) = G\mathbf{u}_t(\boldsymbol{\theta}). \tag{11}$$

Then, if the probability distribution of parameters is invariant at step $t-1$*, then it must be invariant also at step* t

$$p_t(G\boldsymbol{\theta}_t) = p_t(\boldsymbol{\theta}_t) \tag{12}$$

Corollary 2.3. *Assume the update is equivariant at all steps. Given Assumption 2.1 and Theorem 2.2, by induction, the distribution is invariant at all steps,* $p_t(G\boldsymbol{\theta}_t) = p_t(\boldsymbol{\theta}_t), \forall t.$

Symmetry equations

Lemma 2.4. *Assume both the loss function and the probability distribution are invariant for an orthogonal transformation G, namely $\mathcal{L}(G\boldsymbol{\theta}_t) = \mathcal{L}(\boldsymbol{\theta}_t)$ and $p_t(G\boldsymbol{\theta}_t) = p_t(\boldsymbol{\theta}_t)$. Assume that the mean $\boldsymbol{\mu}$ and covariance Σ of the gradient exist and are finite. Then, the mean satisfies the eigenvalue equation*

$$\boldsymbol{\mu}_t = G\boldsymbol{\mu}_t \tag{13}$$

and the covariance matrix is invariant upon the congruent transformation

$$\Sigma_t = G\Sigma_t G^T \tag{14}$$

Solutions for MLP

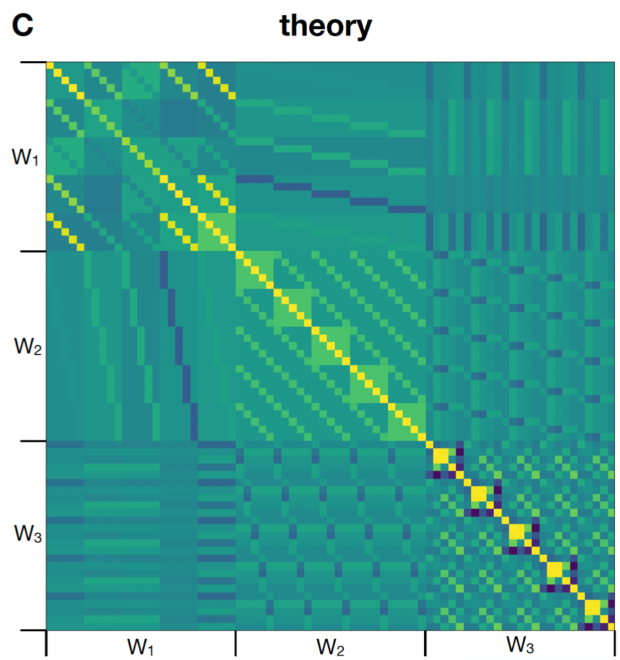
$$\mathbf{h}_\ell = W_\ell \, \sigma_\ell(\mathbf{h}_{\ell-1}) + \mathbf{b}_\ell \qquad \ell = 1, \ldots, L$$

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \tilde{\Sigma}_{11}^T & \ldots & \Sigma_{1L} & \tilde{\Sigma}_{L1}^T \\ \tilde{\Sigma}_{11} & \tilde{\tilde{\Sigma}}_{11} & \ldots & \tilde{\Sigma}_{1L} & \tilde{\tilde{\Sigma}}_{LL} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Sigma_{L1} & \tilde{\Sigma}_{1L}^T & \ldots & \Sigma_{LL} & \tilde{\Sigma}_{LL}^T \\ \tilde{\Sigma}_{L1} & \tilde{\tilde{\Sigma}}_{L1} & \ldots & \tilde{\Sigma}_{LL} & \tilde{\tilde{\Sigma}}_{LL} \end{pmatrix} \quad G = \begin{pmatrix} V_0 \otimes V_1 & & & & \\ & V_1 & & & \\ & & \ddots & & \\ & & & V_{L-1} \otimes V_L & \\ & & & & V_L \end{pmatrix}$$

$$\Sigma_{\ell\ell'} = (V_{\ell-1} \otimes V_\ell) \, \Sigma_{\ell\ell'} \left(V_{\ell'-1}^T \otimes V_{\ell'}^T\right)$$

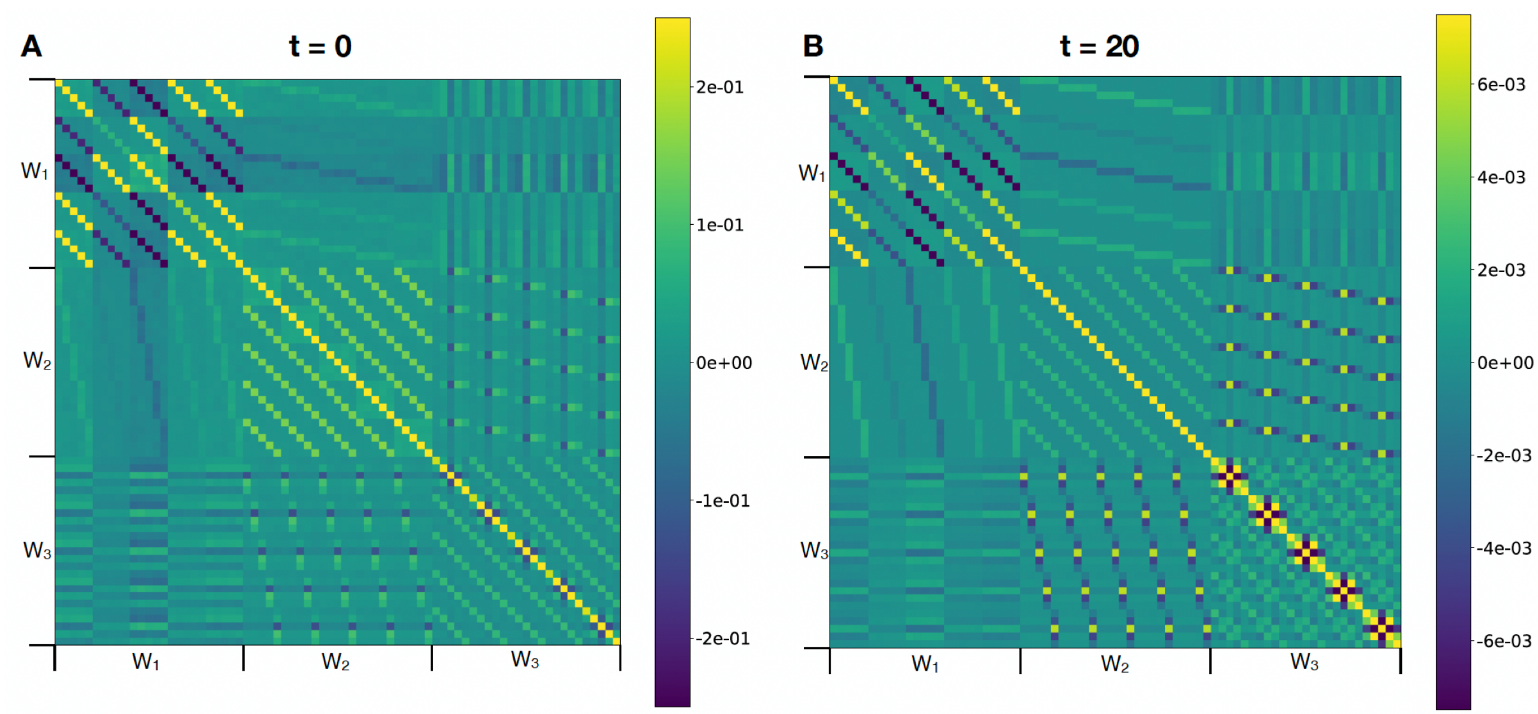
Solutions: linear combination of basis matrices

Solutions for MLP



Theory predicts the structure of the global curvature, (but not the values)

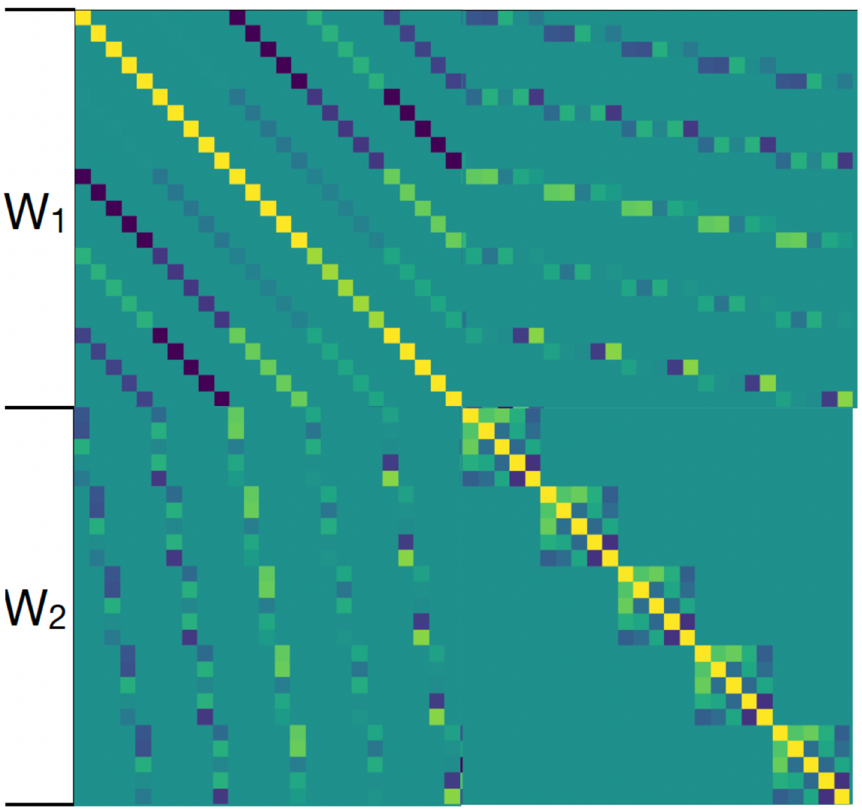
Can we estimate the factors and use them for optimization?



Estimation of factors

2-layer MLP, Tanh, no bias

$$\Sigma = \begin{pmatrix} \Phi_1 \otimes \mathrm{I}_{d_1} & (\Psi_1 \otimes \mathrm{I}_{d_1})\, K \\ K\,(\Psi_1^T \otimes \mathrm{I}_{d_1}) & \mathrm{I}_{d_1} \otimes \Phi_2 \end{pmatrix}$$



Estimate with a <u>single gradient</u>

Σ = g g^T

$$\Phi_1 = \frac{1}{d_1}\left(\frac{\partial \mathcal{L}}{\partial W_1}\right)^T \left(\frac{\partial \mathcal{L}}{\partial W_1}\right)$$

$$\Phi_2 = \frac{1}{d_1}\left(\frac{\partial \mathcal{L}}{\partial W_2}\right) \left(\frac{\partial \mathcal{L}}{\partial W_2}\right)^T$$

$$\Psi_1 = \frac{1}{d_1}\left(\frac{\partial \mathcal{L}}{\partial W_1}\right)^T \left(\frac{\partial \mathcal{L}}{\partial W_2}\right)^T$$

Tractable ✓

Size d x d

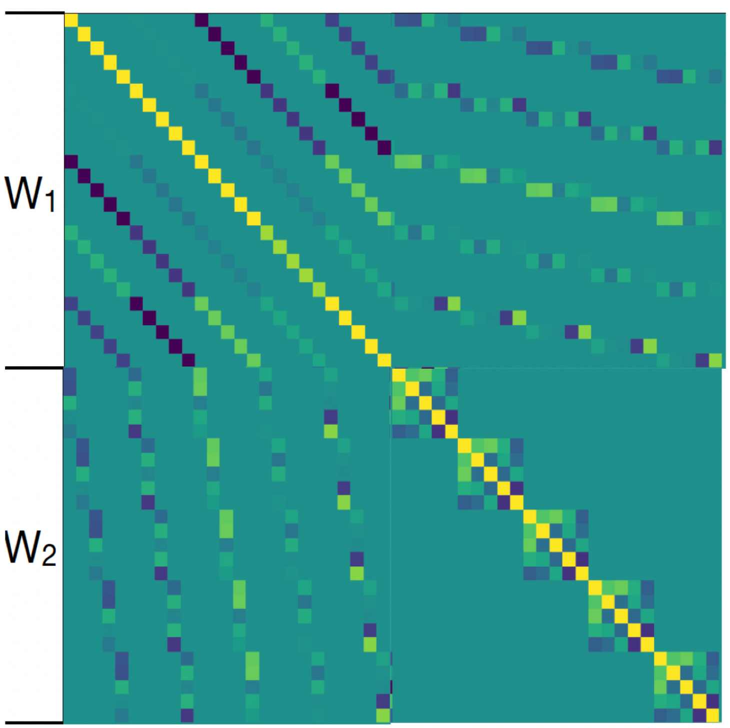
Table 1. Correlation between single-model estimates of the factors and ground truth, varying the width of the hidden layer. Single model estimates improve with the layer width.

Layer widths (d_0, d_1, d_2)	Φ_1	Ψ_1	Φ_2
(100, 10, 100)	0.67±0.05	0.38±0.06	0.30±0.05
(100, 100, 100)	0.90±0.02	0.61±0.04	0.52±0.03
(100, 1000, 100)	0.96±0.01	0.64±0.03	0.54±0.03
(100, 10000, 100)	0.97±0.01	0.65±0.03	0.56±0.03

Efficient matrix computations

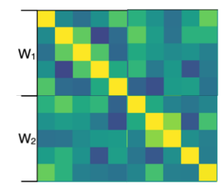
Matrix inverse square root

$$\Sigma^{-\frac{1}{2}} = \begin{pmatrix} \Phi_1 \otimes \mathrm{I}_{d_1} & (\Psi_1 \otimes \mathrm{I}_{d_1})\, K \\ K\,(\Psi_1^T \otimes \mathrm{I}_{d_1}) & \mathrm{I}_{d_1} \otimes \Phi_2 \end{pmatrix}^{-\frac{1}{2}} = \begin{pmatrix} \mathrm{I} & 0 \\ 0 & K \end{pmatrix} \left[\begin{pmatrix} \Phi_1 & \Psi_1 \\ \Psi_1^T & \Phi_2 \end{pmatrix}^{-\frac{1}{2}} \otimes \mathrm{I}_{d_1} \right] \begin{pmatrix} \mathrm{I} & 0 \\ 0 & K \end{pmatrix}$$



Size d^2 x d^2

Intractable ✗



Size d x d

Tractable ✓

For example, a large language model has embedding dimension around $d = 10^3$ or 10^4

Efficient matrix computations

Compute second-order update without ever computing or storing the full matrix

$$\Sigma^{-\frac{1}{2}} = \begin{pmatrix} \Phi_1 \otimes \mathrm{I}_{d_1} & (\Psi_1 \otimes \mathrm{I}_{d_1}) K \\ K(\Psi_1^T \otimes \mathrm{I}_{d_1}) & \mathrm{I}_{d_1} \otimes \Phi_2 \end{pmatrix}^{-\frac{1}{2}} = \begin{pmatrix} \mathrm{I} & 0 \\ 0 & K \end{pmatrix} \left[\begin{pmatrix} \Phi_1 & \Psi_1 \\ \Psi_1^T & \Phi_2 \end{pmatrix}^{-\frac{1}{2}} \otimes \mathrm{I}_{d_1} \right] \begin{pmatrix} \mathrm{I} & 0 \\ 0 & K \end{pmatrix}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \Sigma_t^{-\frac{1}{2}} \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

Vector of gradients

$$\nabla \mathcal{L} = \mathrm{Vec}\left(\frac{\partial \mathcal{L}}{\partial W_1}, \frac{\partial \mathcal{L}}{\partial W_2} \right)$$

Update = matrix-vector product

$$\Sigma^{-\frac{1}{2}} \nabla \mathcal{L} = \mathrm{Vec}\left(\underbrace{\frac{\partial \mathcal{L}}{\partial W_1}} \underbrace{\Phi_1^{\mathrm{isr}}} + \frac{\partial \mathcal{L}}{\partial W_2^T} {\Psi_1^{\mathrm{isr}}}^T, \; {\Psi_1^{\mathrm{isr}}}^T \frac{\partial \mathcal{L}}{\partial W_1^T} + \Phi_2^{\mathrm{isr}} \frac{\partial \mathcal{L}}{\partial W_2} \right)$$

Tractable ✓ Size d x d

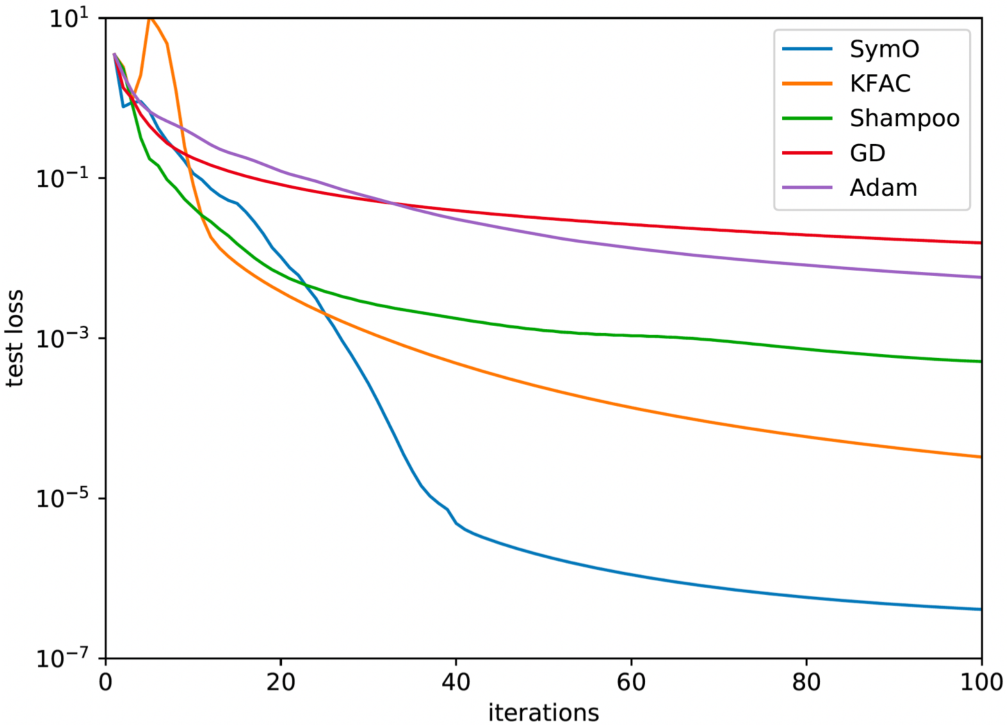
Example optimization on synthetic data

SymO: Symmetric Optimizer

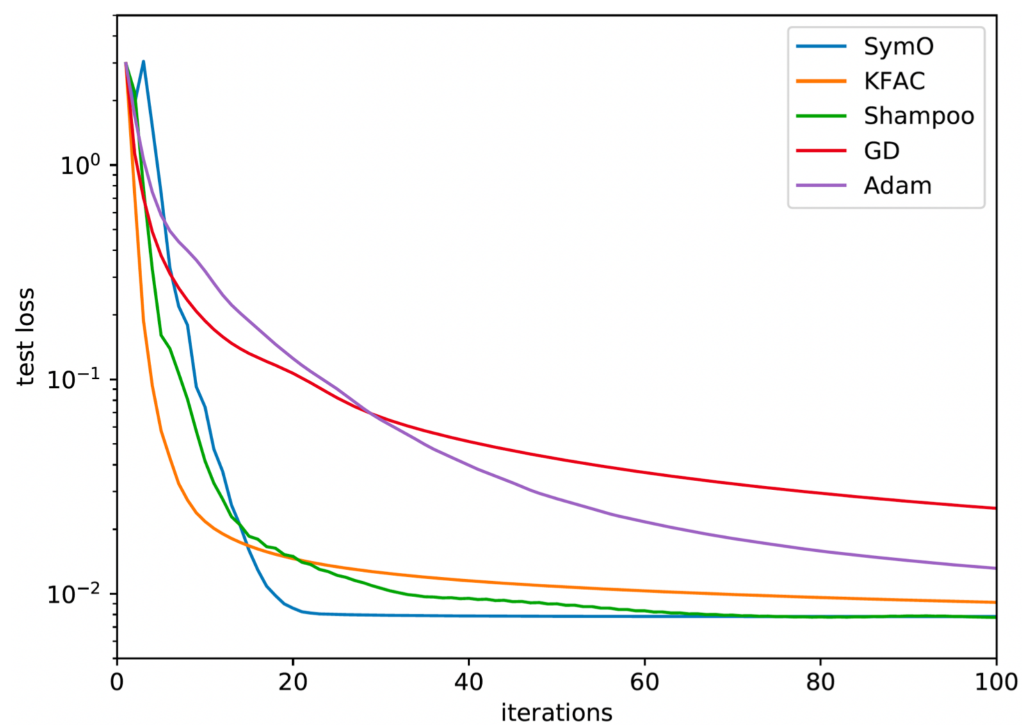
$$(W_1)_{t+1} = (W_1)_t - \alpha \left(\frac{\partial \mathcal{L}}{\partial W_1} \Phi_1^{\mathrm{isr}} + \frac{\partial \mathcal{L}}{\partial W_2^T} {\Psi_1^{\mathrm{isr}}}^T \right)$$

$$(W_2)_{t+1} = (W_2)_t - \alpha \left({\Psi_1^{\mathrm{isr}}}^T \frac{\partial \mathcal{L}}{\partial W_1^T} + \Phi_2^{\mathrm{isr}} \frac{\partial \mathcal{L}}{\partial W_2} \right)$$

Linear activations



Tanh activations

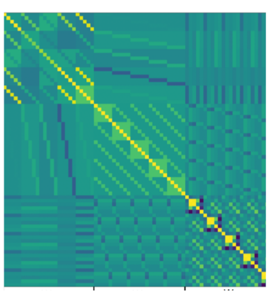


KFAC and Shampoo assume block-diagonal curvature, SymO does not

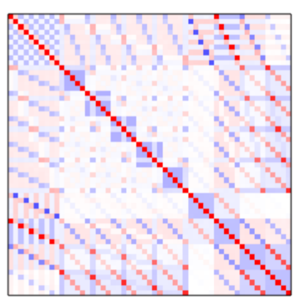
Model-aware optimization

```python
def forward(self, x):
    # Pass through first
    x = self.fc1(x)
    x = self.bn1(x)  # A
    x = F.relu(x)  # App
    x = self.dropout(x)

    # Pass through secon
    x = self.fc2(x)
    x = self.bn2(x)  # A
    x = F.relu(x)  # App
    x = self.dropout(x)

    # Pass through outpu
    x = self.fc3(x)
    return x  # No activ
```

Optimizer 1

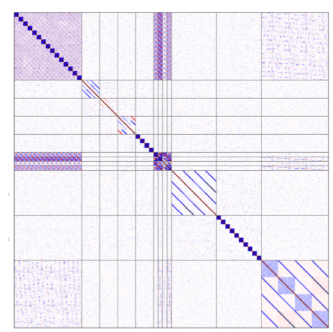


```python
def forward(self, x):
    # First convolutional bloc
    x = self.conv1(x)
    x = self.bn1(x)
    x = F.relu(x)
    x = self.pool(x)  # Apply 

    # Second convolutional blo
    x = self.conv2(x)
    x = self.bn2(x)
    x = F.relu(x)
    x = self.pool(x)  # Apply 

    # Third convolutional bloc
    x = self.conv3(x)
    x = self.bn3(x)
    x = F.relu(x)
    x = self.pool(x)  # Apply 

    # Flatten the output for f
    x = x.view(x.size(0), -1)

    # Fully connected layers
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout(x)
    x = self.fc2(x)

    return x
```



Optimizer 2

```python
def forward(self, x):
    batch_size, seq_len, embed_dim = x.shape

    # Compute Q, K, V matrices
    Q = self.query(x)  # Shape: [batch_size, seq_len, embed
    K = self.key(x)    # Shape: [batch_size, seq_len, embed
    V = self.value(x)  # Shape: [batch_size, seq_len, embed

    # Split for multi-head attention
    Q = Q.view(batch_size, seq_len, self.num_heads, self.he
    K = K.view(batch_size, seq_len, self.num_heads, self.he
    V = V.view(batch_size, seq_len, self.num_heads, self.he

    # Compute attention scores
    attention_scores = torch.matmul(Q, K.transpose(-2, -1))
    attention_weights = F.softmax(attention_scores, dim=-1)

    # Compute attention output
    attention_output = torch.matmul(attention_weights, V)

    # Concatenate multi-head outputs
    attention_output = attention_output.transpose(1, 2).con

    # Final linear projection
    output = self.out(attention_output)  # Shape: [batch_si
    return output
```



Optimizer 3

Thanks