Discovering Weight Initializers with Meta Learning

Dmitry Baranchuk and Artem Babenko

Goal


- In practice, finding effective model-specific initializers is an arduous task that requires human intuition and theoretical insights;
- Many sophisticated architectures and training protocols appear regularly.

Contributions

- Propose DIMAML – an automated approach that learns initial parameter distributions for a given architecture by directly optimizing its training performance;
- Evaluate DIMAML on several architectures and demonstrate that the learned initialization strategies match or perform better than existing alternatives;
- Examine universality of the learned initializers by measuring their ability to generalize to unseen tasks, including different data distributions and training protocols;
- The code is available at https://github.com/yandex-research/learnable-init.

General Approach

DIMAML parameterizes weight initializers and exploits the ideas from Model-Agnostic Meta-Learning [1] to learn them by backpropagating through the training loop:

1. Define \( T_{train} \) tasks to learn initializers and different \( T_{test} \) tasks for evaluation;
2. Define initial weight distributions \( p_{\psi}(\theta_{init}) \) with trainable parameters \( \psi \);
3. Sample i.i.d. initial weights \( \theta_{init} \sim p_{\psi}(\theta_{init}) \);
4. Train the model for a problem-specific objective using gradient descent and calculate validation loss on intermediate training steps;
5. Backpropagate through the entire training procedure and update the meta-parameters \( \psi \).

Method

Initial Parameterizations

We consider two parameterizations for initial distributions \( p_{\phi}(\theta_{init}) \):

- **Normal Initializers**
  \[ p_{\phi}(\theta_{init}) - \text{a normal distribution with parameters } \psi = (\mu, \sigma) \]
  - Sample initial weights for layer \( l \): \( \theta_{l} = \mu_{l} + \sigma_{l} \cdot z \sim N(0, 1) \).

- **PLIF Initializers**
  \[ p_{\phi}(\theta_{init}) \text{ is modeled as } Q_{\psi}, \text{ which is modeled as } PLIF = \text{a piecewise linear increasing function} \]
  - Inverse transform sampling – sample initial weights for layer \( l \): \( \theta_{l} = \psi^{-1}(z) = Q_{\psi}(z) \cdot z - U(0, 1) \).

Training Algorithm

1. Define the initial distribution for the \( l \)-th weight tensor: \( p_{\phi}(\theta_{l}) \);
2. Sample \( \theta_{l} \sim p_{\phi}(\theta_{l}) \) for each weight tensor in the model;
3. Train the model with the gradient descent algorithm (e.g., SGD or Adam) for \( N \) training steps;
4. Compute the average validation loss measured on intermediate training steps:
   \[ L_{val} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} L_{step}(x_{val}, y_{val}) \]
5. Compute \( \partial L_{val} / \partial \psi \) by backpropagating through the training loop;
6. Update meta-parameters \( \psi \) by meta-optimizer.

Memory Efficient MAML

- Limitation: large memory footprint when applying MAML to commonly used neural network architectures and datasets.
- Solution: Gradient checkpointing [4] stores only 1 m optimizer states in device memory, recomputing intermediate steps on the fly. Thus, it allows us to fit about 10-100x more optimizer steps in the same GPU memory.

Experiments

Autoencoders

<table>
<thead>
<tr>
<th>( T_{train} )</th>
<th>Tiny ImageNet</th>
<th>Tiny ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Kaiming</td>
<td>0.471</td>
<td>0.383</td>
</tr>
<tr>
<td>DeltaOrthogonal</td>
<td>0.496</td>
<td>0.392</td>
</tr>
<tr>
<td>MetaInit [4]</td>
<td>0.552</td>
<td>0.437</td>
</tr>
<tr>
<td>DIMAML-Normal</td>
<td>0.386</td>
<td>0.352</td>
</tr>
<tr>
<td>DIMAML-PLIF</td>
<td>0.586</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Comparison of autoencoder models with different initializers in terms of mean squared error. (*) corresponds to convergence.

Language models

<table>
<thead>
<tr>
<th>( T_{train} )</th>
<th>WikiTreebank</th>
<th>WikiTree2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Kaiming</td>
<td>1.983</td>
<td>1.843</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>2.017</td>
<td>1.849</td>
</tr>
<tr>
<td>DIMAML-Normal</td>
<td>1.870</td>
<td>1.812</td>
</tr>
<tr>
<td>DIMAML-PLIF</td>
<td>1.870</td>
<td>1.810</td>
</tr>
</tbody>
</table>

Performance of the character-level language model in bits-per-character. DIMAML initial distributions speedup the training and converges to better optima for the same number of epochs. (*) corresponds to convergence.

References