CS330 Review Session: MAML
What We’ll Cover Today

1. Review of the meta-learning problem setup
2. Model-Agnostic Meta-Learning (MAML)
3. Useful PyTorch functions

No pytorch code, but will connect the lecture materials to the details of practical implementation.
What We’ll Cover Today

1. Review of the meta-learning problem setup
2. Model-Agnostic Meta-Learning (MAML)
3. Useful PyTorch functions
### Running Example

**5-way, 1-shot image classification (MiniImagenet)**

**Given 1 example of 5 classes:**

<table>
<thead>
<tr>
<th>meta-test</th>
<th>Classify new examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>🐶 🐾 🐺 🦁 🐯</td>
<td>🐯 🐊 🍃</td>
</tr>
</tbody>
</table>
Inner Loop Learning

\[ T_i \]
i-th task

\[ D^{tr}_i \]
Training data

We adapt the model on this data...

\[ D^{ts}_i \]
Test data

We repeat this over many tasks

and test on this data
Task Sampling (5-way 1-shot classification)

To sample one task:

1. Sample 5 classes
2. Training set: sample 1 image from each class
3. Test set: sample N images from each class

(training and test set must not overlap!)
To sample one task:

1. **Sample 5 classes**
2. Training set: sample 1 image from each class
3. Test set: sample N images from each class

(Training and test set must not overlap!)
What We’ll Cover Today

1. Review of the meta-learning problem setup
2. **Model-Agnostic Meta-Learning (MAML)**
3. Useful PyTorch functions
MAML Inner Loop

\[
\min_{\theta} \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, D_i^{tr}), D_i^{ts})
\]

\(T_i\)

Initial network parameters \(\theta\)

Parameters adapted to task \(i\) \(\phi_i\)
MAML Outer Loop

\[
\min_{\theta} \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, D_{i}^{\text{tr}}, D_{i}^{\text{ts}}))
\]

\(T_i\)

\(D_i^{\text{tr}}\)

\(D_i^{\text{ts}}\)

\(\theta\)

Initial network parameters

\(\nabla_{\theta} \mathcal{L}\)

\(\phi_i\)

Parameters adapted to task \(i\)

\(L(\phi_i, D_i^{\text{ts}})\)

Backprop
MAML Meta-Testing

Novel task constructed from unseen classes

\[ \nabla_{\theta} \mathcal{L} \]

\[ \theta \] Meta-learned network parameters

\[ \phi \] Parameters adapted to test task

\[ D^{tr} \] Dataset for training

\[ D^{ts} \] Dataset for testing

\[ L(\phi, D^{ts}) \] Loss function
MAML Summary

Meta-Training

Repeat until convergence:

1. Sample task $T_i = (D_i^{tr}, D_i^{ts})$
2. Optimize $\phi_i \leftarrow \theta - \alpha \nabla_\theta L(\theta, D_i^{tr})$
3. Update $\theta \leftarrow \theta - \beta \nabla_\theta L(\phi_i, D_i^{ts})$

Meta-Testing

1. Given task $T = (D^{tr}, D^{ts})$
2. Optimize $\phi \leftarrow \theta - \alpha \nabla_\theta L(\theta, D^{tr})$
3. Make predictions on $D^{ts}$ using $\phi$

In practice, we parallelize both meta-training and meta-testing with minibatches of tasks.
What We’ll Cover Today

1. Review of the meta-learning problem setup
2. Model-Agnostic Meta-Learning (MAML)
3. Useful PyTorch functions
In the provided code, the provided `_forward()` is stateless: it takes current model parameters as input.
torch.autograd.grad()

\[
\min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_i^{tr}), D_i^{ts})
\]

torch.autograd.grad(outputs, inputs, grad_outputs=None, retain_graph=None, create_graph=False, only_inputs=True, allow_unused=None, is_grads_batched=False, materialize_grads=False) [SOURCE]

Computes and returns the sum of gradients of outputs with respect to the inputs.

If you want to backpropagate through the gradient later:
torch.autograd.grad(outputs, inputs, create_graph=True)

Otherwise:
torch.autograd.grad(outputs, inputs, create_graph=False)
Parameters are a dictionary with (parameter_name, parameter_value) pairs. You should explicitly compute the updated parameter.

$$\min_{\theta} \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}^\text{tr}_i), \mathcal{D}^\text{ts}_i)$$

```python
parameters = {
    k: torch.clone(v)
    for k, v in self._meta_parameters.items()
}
```
What We Covered Today

1. Review of the meta-learning problem setup
2. Model-Agnostic Meta-Learning (MAML)
3. Useful PyTorch functions