Model Pruning: Weekly Report 1

Patricia Gschoßmann

1. Weekly Progress

The goal for this week was to start implementing task 3[1] with the new pruning approach. Initially, the original VGG network was trained as usual. Next, a new model, consisting of the pretrained network as well as a parallel branch for each convolutional layer, was trained following the new approach. As an initial starting point, each parallel branch aims to reduce the output of its corresponding convolutional layer to 75%. For now, a dummy α-schedule was used, reducing α from 0.9 to 0 in steps of 0.1. The weights of the resulting model were then pruned according to the new approach to obtain the smaller version.

2. Challenges and Possible Solutions

While implementing the parallel branches for the VGG network, I stumbled across the problem of where to place batch normalization and max pooling layers as well as the final flatten-opertation before feeding the convolutional output into the linear layer. For now, each of these operations is performed separately in the parallel running branches. My decision was based on the assumption, that otherwise the weights cannot be multiplied during pruning.

Furthermore, pruning, as it is implemented now, was not able to preserve the original accuracy of 92%. The model maintained an accuracy of 84% until $\alpha = 0.4$. After that, the accuracy dropped below 45%, until it reached only 10% at the end. This could indicate, that α needs to be reduced slower than at the beginning, after dropping below a certain threshold.

Lastly, an unanswered question about multipling matrices with more than two dimensions using `torch.einsum` remains. To illustrate the problem, consider the first convolutional layer in the VGG model: `conv0` is a convolutional layer with the dimensions $(64, 3, 3, 3)$ and should be pruned to output only 48 channels resulting in the dimensions $(48, 3, 3, 3)$. `pconv0` is the first convolutional layer in the corresponding parallel branch with the dimensions $(48, 64, 3, 3)$. To prune the weights of `conv0`, they have to be multiplied with the weights of `pconv0`. I came up with two possible solutions.

- `torch.einsum("abcd, befg -> aefg", pconv0_weights, conv0_weights)`
- `torch.einsum("abcd, befg -> aecd", pconv0_weights, conv0_weights)`

I still have to investigate which of them is correct and why, by checking the documentation. For now, the first one is implemented, since it is used in the tutorial script `conv_pruning.ipynb`.

3. Plan

My goal for the following week is to prune the model while maintaining its original accuracy. I will pursue following attempts:

- Do not prune each convolutional layer at once
- Specify a more suitable alpha schedule

Of course, the second point will require more research and thinking. As a starting point, I want to look into multiple different learning rate scheduler.

Furthermore, I want to adapt my implementation to a more general framework, such that the model architecture is not hard coded in any way.

4. Summary of new scripts

- `abstract_model.py`: Abstract superclass for all models. Dataloaders, train-, val- and teststep are shared across all submodels. The forward method needs to be implemented.
- `vgg.py`: Architecture of the pre-trained VGG16 network. The model has been slightly modified from the original, combining the last three linear layers to only one. Subclass of `abstract_model.py`.
- `pruned_model.py`: Architecture of the model used for the pruning procedure. It consists of a pre-trained network as well as parallel branches for each layer to prune. Subclass of `abstract_model.py`.

[1] Pruning a VGG network for the task of classifying CIFAR10 images

- `smaller_model.py`: Architecture of the reduced model after pruning. Subclass of `abstract_model.py`.
- `train.py`: Script to train a pre-trained model on the CIFAR10 dataset.
- `objective_selection.py`: Script to train a `pruned_model.py` with the new approach using a dummy α-schedule.
- `prune.py`: Script to prune the weights of a trained `pruned_model.py` following the new approach to obtain a `smaller_model.py`.
- `config.py`: Specifies training and network architecture configurations.