Deep Learning — Assignment 7

Seventh assignment for the 2023 Deep Learning course (NWI-IMC070) of the Radboud University.

Names:

Group:

Introduction

For this assignment we are doing things a bit different.

- Your task is to reproduce the paper The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks; Jonathan Frankle, Michael Carbin.
- Try to follow the experimental settings in that paper, unless there is a reason to deviate.
- If the paper is not clear on some details, make a reasonable choice yourself, and motivate that choice.
- You will have 3 weeks to work on this assignment.
- Be aware that this assignment will take more time than previous ones. It is ok if you do not completely finish it.
- We will *not* be providing you with much code. You will have to implement many things yourself.
- You may freely use code from earlier weeks, and from the d2l books. Please add a comment to reference the original source.
- You may *not* use implementations of the paper you find online.

Tips and hints

- It is allowed and recommended to use more than just this notebook. Make separate python files for a library of functions, and for training and analyses.
- If you like working with jupyter notebooks: make a separate notebook for trying things out, and keep this one clean.
- Use checkpoint files before and during training.
- In the notebook only display and discuss these results.
- You may add new cells to this notebook as needed.
- While the task is to reproduce parts of a paper, the big picture is more important than the exact details.

- It is allowed to discuss the assignment with other groups, but try not to spoil too much.
- If you get stuck, contact the teachers via discord.

Required software

If you need to import any additional libraries, add them below.

In [ ]:
```
%config InlineBackend.figure_formats = ['png']
%matplotlib inline
%load_ext autoreload
%autoreload 2
import torch
import torch.nn
import torch.nn.utils.prune
import torchvision
import matplotlib.pyplot as plt
from d2l import torch as d2l
from dl_assignment_7_common import *  # Your functions should go here if you

device = d2l.try_gpu()
```

7.1 The paper (2 points)

(a) Read sections 1, 2, and 5 of the paper The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks; by Jonathan Frankle, Michael Carbin

We will refer to this as the "LTH paper" from now on, or just "the paper". To answer some later questions you will also need to look at other sections, and search through the appendices.

(b) In your own words, briefly explain the key message of the paper. (2 points)

Note: "briefly" means: a few sentences at most.

TODO: your answer here.

7.2 Models and datasets (9 points)

(a) What neural network architectures are used in the paper? (1 point)

TODO: your answer here.

To keep things simple, we will start with a simple architecture, corresponding to what the paper calls `Lenet`.

(b) Define a function that constructs a `Lenet` network using PyTorch. **(2 points)**

Copy these definitions to `dl_assignment_7_common.py`.

Hint: see Figure 2.

Note: the LTH paper is not entirely clear about this, but the convolution layers use `padding='same'`.

In [ ]:
```python
def create_lenet():
    # TODO: your code here
    pass
```

(c) Define a function that can construct a network given the architecture name.

To keep the code as generic as possible, we can make function

Move the function below to `dl_assignment_7_common.py`, and don't forget to remove it here.

In [ ]:
```python
def create_network(arch, **kwargs):
    # TODO: Change this function for the architectures you want to support
    if arch == 'arch1':
        return create_network_arch1(**kwargs)
    elif arch == 'arch2':
        return create_network_arch2(**kwargs)
    else:
        raise Exception(f"Unknown architecture: {arch}")
```

We will do all our experiments with two datasets: MNIST and FashionMNIST

(d) Are these datasets also used in the papers? **(1 point)**

TODO: your answer here.

(e) Define a function that loads a dataset given the dataset name. **(3 points)**

Hint: Standard datasets such as MNIST and CIFAR10 are available in the torchvision library.

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

(f) Most of these datasets come with a predefined train/test split. Is this used in the LTH paper? If so, update the dataset loader to return a pair `(trainset, testset)`. **(1 point)**

TODO: your answer here.

(g) Does the LTH paper use a validation set? If so, update the dataset loader to return `(train_dataset, validation_dataset, test_dataset)`. **(1 point)**

Hint: `random_split`, and/or see assignment 2.

TODO: your answer here.

7.3 Training (12 points)

(a) What optimization algorithm is used in the paper? What values are used for the hyperparameters? **(1.5 points)**

If you are unable to find the values used for some of the hyperparameters, use reasonable default values.

TODO: your answer here.

(b) Implement an evaluation function, that evaluates a model on a validation or test set (passed as an argument). **(2 points)**

The function should return loss and accuracy.

Hints: the book defines a function for this that you may use (see assignment 3).

(c) Implement a training loop. **(4 points)**

Make sure that the network parameters are saved to a file before and during training.

Because you will be doing many experiments, it would be a shame to have to re-run them when you reload the notebook. A better solution is to save model checkpoints. See the tutorial on saving and loading model parameters for how to implement this in PyTorch.

(d) Change the training function so that it saves the model at the start and at the end of training. **(1 point)**

Hint: Saving a model requires a filename. Because you will be running many experiments, come up with a descriptive naming convention and/or directory structure. Example: `path = f"checkpoints/model-{arch}-{dataset}-{run}-{phase_of_the_moon}-{iteration}.pth"`.

Hint 2: it is easier to save the whole model, see the bottom of the tutorial.

(e) Train a simple network on a simple dataset. **(1 point)**

You may want to create a new python script (`simple_training.py`), and just load the trained network here instead.

(f) Does the training converge? How well does your network perform? (1 point)

TODO: your answer here.

(g) Re-train the same network, with the *same* initial weights. Are the results *exactly* the same? (2 points)

TODO: your answer here.

(h) The LTH paper uses a variant of 'early-stopping'. How is this done? Implement it in your training loop. (1 point)

Hint: A simple way to keep track of the best model is to create a model checkpoint in a file `"checkpoints/model-...-best.pth"` .

Hint 2: It is okay to compute the validation scores less often, this can speed up training.

TODO: your answer here.

7.4 Pruning (13 points)

Next up, you should implement pruning. Starting with the one-shot pruning method.

Hint: Pruning is implemented already in PyTorch, in the module torch.utils.prune. The pruning method used in the LTH paper corresponds is called l1_unstructured in PyTorch.

(a) The PyTorch pruning function accepts an amount to prune. Is that the amount of weights to set to 0 or the amount to keep nonzero? Is the paper using the same? (1 point)

TODO: your answer here.

(b) Should all parameters be pruned or only weights? Is the pruning rate the same for all types of layers? (1 point)

TODO: your answer here.

(c) Define a function to prune a network as used in the LTH paper. It should take an amount to prune as an argument. (3 points)

Hint: for a `Sequential` layer, you can access the layers as `net.children()`. For a layer, you can use `isinstance(layer, torch.nn.Linear)` to check if it is a linear layer.

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`.

(d) Check your pruning function on a very simple neural network. Print the layer weights before and after pruning to make sure you understand what is going on. (1 point)

Hint: the PyTorch pruning functions do not change the trainable parameters, rather they set `module.weight` to `weight_orig * weight_mask`.

(e) Define a function that applies the pruning mask from a pruned network to another network of the same architecture. (2 points)

This function should only do pruning (so some weights become 0), other weights should remain the same.

Hint: the pruning functions already generate and store pruning masks. You should be able to extract these from a pruned network.

Hint 2: `custom_from_mask`

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

(f) Check your mask copy function on a very simple neural network. Check that only the pruning mask is copied. (1 point)

(g) Define a function that randomly prunes a network. (1 point)

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

(h) Check the above function. (1 point)

(i) Define a function that performs the experiment described in Section 1 of the LTH paper on a given dataset and with a given architecture. (2 points)

Save all needed results to a file, such as test loss and accuracy. This will make your job easier later on.

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

7.5 Confirming the Lottery Ticket Hypothesis (10 points)

(a) Perform the experiments needed to reproduce the red lines in Figure 1 from the LTH paper. (6 points)

- It is ok to ignore the error bars for now, and focus on doing one series of experiments.
- You may also reduce the number of points in the plot to keep the computation time managable.
- You do not have to match the visual style of the figure.

Hint: create a python script (`experiment1-{dataset}-{method}.py`) that does all the training as needed. Then load the checkpoint files and do your analysis here. You may also want to define more helper functions.

Hint 2: look at previous assignments for how to plot. If you do want to include error bars, see also documentation for `plt.errorbar` .

TODO: Training implemented in `TODO.py`

In [ ]:
```
# TODO: your code here
```

(b) Do your results match the paper? Discuss similarities and differences. (2 points)

TODO: your answer here.

(c) What can you conclude from this experiment? (2 points)

TODO: your answer here.

7.6 Experiments from Section 2 (12 points)

(a) What is the difference between the experiment in Figure 1 and Figure 3 of the paper. (1 point)

Hint: are there differences in the method, the architecture, or the dataset?

TODO: your answer here.

(b) Implement the iterative pruning method from the paper. (3 points)

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

(c) Perform the experiments needed to reproduce Figure 4a from the paper. (4 points)

Hint: see previous section

TODO: Training implemented in `TODO.py`

In [ ]:
```
# TODO: your code here
```

(d) Do your results match the paper? Discuss similarities and differences. (2 points)

TODO: your answer here.

(e) What can you conclude from this experiment? (2 points)

TODO: your answer here.

7.7 Experiments from Section 4 (7 points)

Section 3 and 4 deal with convolutional neural networks. We are going to skip the networks in section 3, and move on to Figure 8.

(a) Section 4 of the paper describes a slightly different pruning method. Implement that method. (2 points)

Hint: look at `torch.nn.utils.prune.global_unstructured` and at the examples on that page.

If you get stuck on this step, you can continue with the same pruning methods as before.

TODO: Function implemented as: `TODO` in `dl_assignment_7_common.py`

(b) Implement a function that constructs the network architecture used in Figure 8. (1 point)

Extend the `create_network` function defined earlier.

Hint: VGG16 and Resnet18 are predefined in torchvision.

(c) Perform the experiments needed to reproduce Figure 8 from the LTH paper. (2 points)

- Again: you do not need to include error bars.
- You may limit yourself to one of the figures.

TODO: Training implemented in `TODO.py`

In [ ]:
```
# TODO: your code here
```

(d) Do your results match the paper? Discuss similarities and differences. **(2 points)**

TODO: your answer here.

The end

Well done! Please double check the instructions at the top before you submit your results.

This assignment has 65 points.

Version 97b3d19 / 2023-10-19