CS 326 Low Resource Deep Learning
Practical assignment #2: Few-Shot Learning

September 19, 2021

1 Intro

In few-shot learning, our goal is to build a model which would be able to solve a task with a very limited amount of supervision. For image classification, this means that we want to have a model which can learn to categorize new classes with few examples per category. Most often, this is done by some sort of pretraining on a larger dataset to induce general knowledge into the model which in turn should help to capture novel concepts faster.

In this assignment, we will consider K-shot image classification for different values of K and see how different methods perform in terms of quantitive performance, ease of implementation, training speed, etc. To make things fair, we will use good ol' LeNet (with some slight modifications) for all the methods. The methods we are going to compare are the following:

1. Unpretrained baseline. The simplest baseline ever: do nothing, just train your model on these K examples you had been given to.

2. Pretrained baseline. Similar to the above one, but now we pretrain the model on some bigger dataset and then fine-tune it on few target examples.

3. ProtoNet Snell et al. [2017]. Train an image embedder model and classify images based on the closest centroid.

4. MAML Finn et al. [2017]. A "learning-to-learn" approach: we pretrain our model in such a way that it has a good init for learning other tasks.

This assignment targets people who have good hands-on experience in deep learning and Pytorch. The approximate time to solve the assignment is $\sim$ 2 full days. A GPU card would be very, very useful.

2 The assignment

We are going solve a 5-class K-shot classification of Omniglot characters Lake et al. [2015]. Omniglot dataset contains 50 alphabets that are split into 30 train and 20 test alphabets that give 964 train and 659 test different symbols in total. Train and test symbols do not overlap, i.e. the test set is entirely different from the training one.

To keep things simple, we are going to mix the alphabets between each other inside a corresponding split. We'll randomly select symbols from the source alphabets to construct the current classification task and we'll evaluate in a similar manner on the test split.

We are going to do K-shot classification for $n = 1, 2, 3, 5, 10, 15$ and see how different methods compare between each other for different values of K.

You are given a pytorch code template to build upon, that you can modify the way you like or even ignore completely and do everything on your own with any frameworks/libraries you like. However, using the code template may make things easier for you.

2.1 Unpretrained baseline (0 pts)

Unpretrained baseline is already implemented for you so you have a warmer start. You will just need to launch it for different K and check the results. You can run it by calling:

```
CUDA_VISIBLE_DEVICES=0 python run_experiment.py -m unpretrained_baseline
```

2.2 Pretrained baseline (5 pts)

To run the pretrained baseline you should pick decent hyperparameter values, familiarize yourself with the code template and modify things in such a way that LeNet is getting pretrained.

Your task is:

- Implement the pretrained baseline.
- Answer the following questions:
    1. Does it work better than the unpretrained baseline? If not, why?

2.3 Prototypical Networks (25 pts)

ProtoNet has been covered in the lecture, but here we will briefly expose our setup. We are given a neural network $f_\theta : x \mapsto v$ that embeds and image x into feature vector $v = f_\theta(x)$. We compute prototypes for each class c as:

$$p_c = \frac{1}{N_c} \sum_{i=1}^{N_c} f_\theta(x_c^{(i)}) \tag{1}$$

where all $x_c^{(i)}$ belong to the same class c and N_c is the number of such instances in a given batch. For simplicity, we'll use the same number of examplars for each class in a given mini-batch. Ideally, we would like to compute prototypes based on the all training examples of a given class (not only those that are in the current batch), but it would be too costly.

We compute logits by measuring the distance between p_c and $v^{(i)}$. In the original paper Snell et al. [2017], authors used L_2 distance, but cosine should be also fine in our case.

Your task is:

- Implement prototypical networks. You can use either euclidian distance to compute the logits (like in the original paper) or cosine similarity.
- Answer the following questions:

- What are the advantages of the model? What are the disadvantages? How is it better/worse compared to other methods in terms of the quantitative performance? In terms of the implementation simplicity? In terms of training speed? In terms of inference speed? In terms of memory consumption?
- How well does it perform, when we *do not train the model at all*? Why?

2.4 MAML (70 pts)

MAML has been covered at the lecture, but you make like to read this article to refresh your knowledge about MAML.

Your task is:

- Implement MAML. You can choose either first-order or second-order approximation.
- Answer the following questions:
  1. What are the advantages/disadvantages of MAML? How does it compare to other methods in terms of performance/training speed/simplicity/ease of implementation?
  2. Why MAML is unstable? How can we alleviate this?
  3. If you implemented the model in pytorch, explain why we couldn't use a traditional *nn.Module* for MAML implementation.
- (+10 bonus points) Implement both first-order and second-order MAML. Compare the difference between them in terms of performance and training stability.

Hints:

- Google things *a lot*. Check open-source implementations.
- Keep in mind that MAML takes time to converge. For many setups, it might take up to several hours. You can play with open-source implementations to get the feeling.
- Keep in mind that MAML is unstable. What tricks do you know to stabilize training? (it's high time to reminisce your GAN and RL training experience)
- Make sure that your MAML works "without MAML", i.e. when we have 0 inner loop steps, when we have zero inner learning rate, etc.

3 Submission format

Your submission should be a zip named *Firstname_Lastname.zip* archive containing two things:

1. A small report (*up to 2 pages*) describing the experiments you ran, hyperparameters you used, the results you obtained and your answers to the questions. *The report must be in pdf format*. We prefer latex, but you won't be penalized for writing it in Microsoft Word or something else.
2. Your source code. The source code must be reproducible: when we'll run it we must obtain the same results that you report. You won't be penalized if your code will be "dirty", but please don't go too hard on us.

The core part of your report should be a plot with the comparison of 4 methods: simple baseline, pretrained baseline, ProtoNet and MAML for different values of K. So, x-axis is the number of shots (i.e. value of K) and y-axis is the average test accuracy. This plot should contain 4 lines, where each line is a specific method. You will get +2, +2, +2 and +4 bonus points if you'll add the corresponding confidence intervals across several random seeds to the plot for unpretrained baseline, pretrained baseline, ProtoNet and MAML respectively.

4 Rules

1. This is an individual assignment, you are not allowed to share your solution with others. In case of cheating, both you and the person you shared your code with will receive 0 points for the assignment.

2. You can modify the provided template any way you like. You can ignore it completely and do everything on your own with any frameworks/libraries you like. You can partially use the provided code and partially use some open-source implementation. In the latter case, you must specify all the sources that you have used.

3. You are allowed to use any information or code (including other people's implementations) you can find in the wild, *but you must specify the sources.*[1] This assignment is designed in such a way that you will still have to do a lot on your own.

4. You are not allowed to openly release your solution. In case it is found in public domain, we assume that you have explicitly shared it with someone so rule 1 is being applied.

References

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/finn17a.html.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. ISSN 0036-8075. doi: 10.1126/science.aab3050. URL https://science.sciencemag.org/content/350/6266/1332.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017. URL http://arxiv.org/abs/1703.05175.

[1] We require this to be able to understand when solutions are similar because they use the same open-source code or because the students shared the code with each other.