Optimization as a model for few shot learning. Re-implementation for ECE 50024 Final Project

Aaryan Garg, ECE Undergrad

1. Introduction

1.1. The problem

Deep neural networks are usually trained on large sets of data, but when a new type of input is given in training, it requires a lot of examples to learn and cannot generalize given just a few examples. Few-shot learning is the branch that tries to develop models which are capable of making meaningful learnings when given only a few examples.

1.1.1. Importance of solving the Few Shot learning problem

This problem is extremely important to solve because in many practical use cases, there is class imbalance of data. Some classes which have very few examples and need to be learned by generalizing features in only a few shots.

The idea that attracted me to this paper was that, this approach presents a way to make models learn how to learn. For instance, if a model learns few shot classification of images, then it can learn to classify images, given only a few examples, even if it hasn't been shown any picture previously from that class.

1.2. My objective

I implemented the model and algorithm as described in the paper and trained it to perform 5-shot image classification on the same dataset (mini Image Net).

2. Related Work

2.1. Goal

The problem chosen in this paper is to perform few-shot learning for the image classification task. The selected approach is the use of a model-agnostic meta learner. This is essentially having a primary neural network perform its task, but having a separate neural network which takes the parameter values, gradients, and losses of the first one and generates the next set of parameter values for the primary network. This paper proposes the use of an 2-layer LTSM based neural network as a "meta-learner" and using a convolutional neural network as the primary "learner".

To demonstrate the effectiveness of this idea, the authors made the learner learn 5-class classification of images. The model was then trained to learn five-shot and one-shot learning for this. Then the accuracy was compared to the baseline models selected.

2.2. Neural networks and their architecture

In this project there are two neural networks: Learner, and Meta-Learner.

2.2.1. Learner network

The learner neural network consists of 4 convolutional layers. Each convolutional layer is followed by a batch normalization, then ReLu activation, and finally a max-pool layer that reduces height and widths to half. the result of this is then flattened and fed to a linear layer with same outputs at the number of classes. See the figure for a pictorial representation of the learner network.

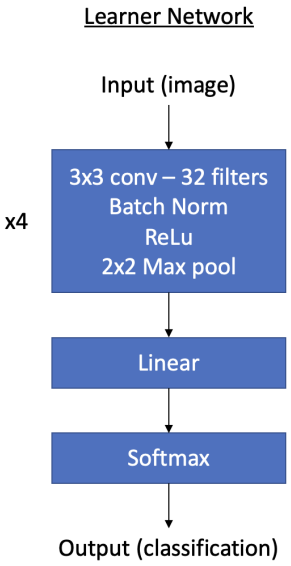

Figure 1. Learner Network Block Diagram

Note: According to the authors' description, the last layer is followed by softmax activation, and the loss used is negative log loss. However, since my implementation is done using the PyTorch library, I did not add the softmax activation at the end and used the cross-entropy loss instead. These two methods are mathematically the same. This is also quite standard practice when implementing a multi-class classification task. The cross entropy loss equation is shown here:

$$J(\theta) = -\sum_{c=1}^{C} log \frac{e^{x_c}}{\sum_{i=1}^{C} e^{x_i}} y_c$$

Where,

C is the number of classes

y_c are the target classes

x_c are the predicted classes

2.2.2. Meta-Learner

The meta-learner is an LSTM based neural network. The first layer is just a regular LSTM, and the second layer is a modified version of LSTM. Its architecture is described in the figure below.

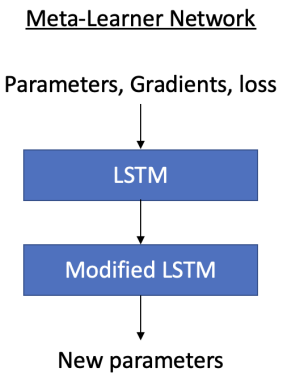

Figure 2. Meta-Learner Network Block Diagram

The modified version of LSTM has the input gate (i_t) and the forget gate (f_t), with their values calculated as shown in the equations below.

$$i_t = \sigma\left(W_I[\nabla_{\theta_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}] + b_I\right)$$

$$f_t = \sigma\left(W_F[\nabla_{\theta_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}] + b_F\right)$$

These are then used to update the cell state (c_t) at each time step t according to the following equation.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_{t-1}$$

Also, when passing the losses and gradients to the LSTM, they can sometimes be of very different magnitudes. So, a preprocessing is applied and they are scaled in a non-linear fashion as suggested by Andrychowicz et al. according to the following equation:

$$x_{preprocessed} = \begin{cases} \left(\frac{log(|x|)}{10}, sgn(x)\right), & \text{if } |x| \geq e^{-10} \text{ odd} \\ \left(-1, e^{10}x\right), & \text{otherwise} \end{cases}$$

2.3. Training process

The training process is divided into episodes. In each episode, for k-shot N-class classification, there are $k \times N$ images to train the learner and n-eval images per class to see the performance of the learner for training the meta-learner. This means that the training loop has two main steps:

2.3.1. Updating Learner parameters

- Forward propagate on the Learner using the training part of the episode dataset. Then, compute losses and gradients for it.
- Preprocess the loss and gradients (of loss wrt. learner's parameters) and feed them to the meta-learner LSTM. Use the meta-learner output to update the learner's parameters.
- Repeat this process for the number of epochs selected. The chosen value is 8 epochs.

2.3.2. Training the Meta-Learner

- Forward propagate on the Learner using the evaluation part of the episode dataset. Then, compute loss for it.
- Now, compute gradient (of loss wrt. meta-learner's parameters).
- Use this gradient with the Adam optimizer to train the meta-learner.

For better visualization of the training process of the networks, the flow is described in the figure below. θ represents the parameters of the Learner network. Each training episode, the gradient and losses of the Learner are fed into the meta-learner LSTM. This flow of parameters is actually not back-propagated over and hence are represented by a dashed arrow.

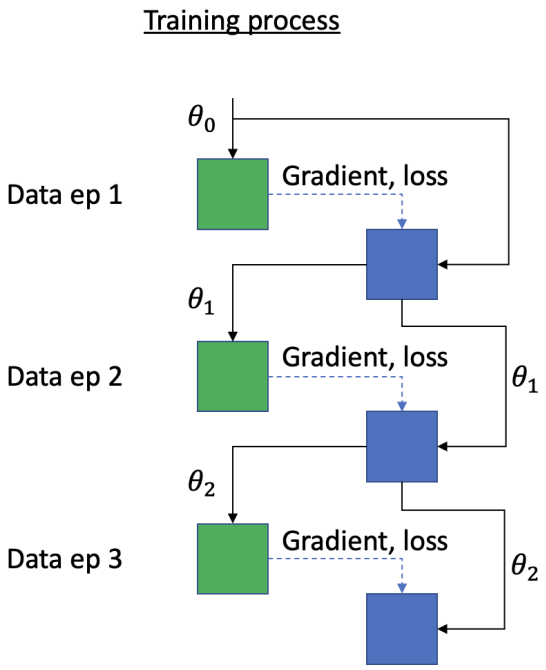

Figure 3. Training process

2.4. Dataset and Hyperparameters used

2.4.1. Dataset

The authors used the mini ImageNet dataset. This is a subset of the ImageNet dataset with 100 classes, each containing 600 RGB images of size 84x84. It is split into train, validation, and test sets containing 64, 16, and 20 classes respectively.

The link to download the miniImageNet dataset is: miniImageNet

2.4.2. Hyperparameters

The following table shows the hyperparameters used as presented in the original paper, and I used the same values in my implementation.

Table 1. Hyperparameters and specifications used.

Parameter	Value
k-shot(s)	5 or 1
n-eval	15
N-class	5
learning rate	0.001
Number of conv filters	32
Gradient clipping	0.25
Initial forget gate	5 (high)
Initial input gate	-5 (low)

3. Method

3.1. Story of my implementation

The author's implementation is found at https://github.com/twitter-research/meta-learning-lstm. This link was referenced in the original paper by the authors so I started playing with this as it is reliable. Initially, I had a high-level understanding of LSTMs but needed to read more to gain a deep enough understanding for implementing them in code. Also, the original author's implementation has a lot of code in lua. I learned lua at a functional level to understand some implementation details.

Then I implemented the model's forward propagation in python, using the PyTorch library. I chose this because I had experience with the library from ECE 570. To write the training loop, I followed the flowchart in the paper and the author's code.

My implementation/code can be found in this repository: https://github.com/ryngrg/MetaLearner-Few-shotLearning.

3.2. Structure of my code

All of my code is implemented in one Jupyter notebook since i was working on it on Google Colab. Primarily, I relied on cells of the .ipynb notebook to organize different aspects of the code.

At a high level, this is what the cells contain:

- Imports
- Classes for neural networks, inheriting from torch.nn.module
  - Learner network
  - Modified LSTM layer
  - Meta Learner
- Setting parameters like number of classes, shots, learning rate etc.
- Preparation of dataset, by defining classes that inherit from torch.utils.data.dataset
- Setting up the data loaders
- Training function for learner
- Testing function for meta-learner
- Initializing the neural networks
- Training loop
- Plot generation

3.3. My process

I chose the same hyper-parameters as the authors. I chose 5-shot learning for 5-class classification. The training part of the episode dataset has 5 images per class, and the evaluation part used for training the meta-learner has 15 images per class. Using the same learning rates and batch norm parameters as the author, I trained on the mini Image-Net datatset.

4. Experiment

4.1. Results

After training the model for 1000 episodes, the loss did not seem to go down as rapidly as it was earlier. Also, the accuracy on the training episodes was getting somewhat stagnant. The testing dataset was used after every 100 episodes of training. Accuracy and loss wowas computed on those images as well.

The figure below shows the loss across the 1000 training episodes. As you can observe, the variance of the training loss begins to increase and the mean value does not decrease as rapidly as did in the first 200 episodes.

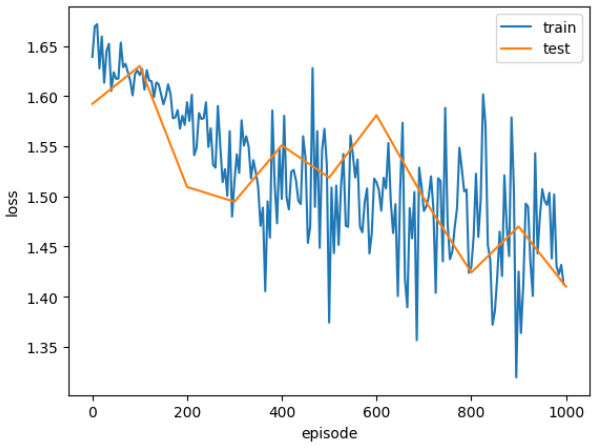

Figure 4. Loss

The following figure shows the accuracy on the training set of the episodes in blue and once every 100 episodes, the accuracy on the test set is also computed. The test accuracy is shown in orange. It peaked at 1000 episodes. After that the accuracy did not improve much, so I restricted the plot to 1000 episodes only. In the end the accuracy on the test set was 51.33%.

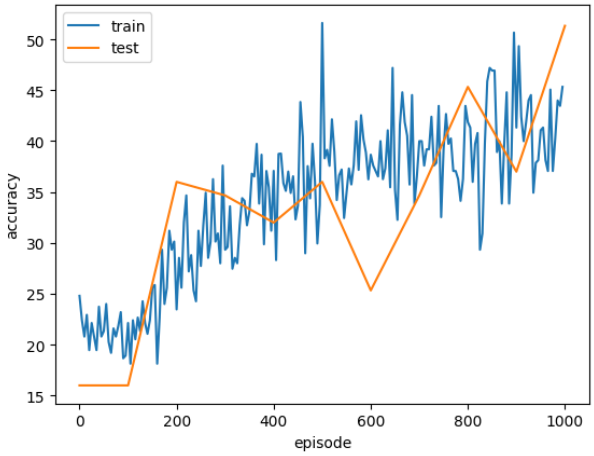

Figure 5. Accuracy

51.33% accuracy in classification after 5-shot learning.

4.2. Comparison with baseline

The following table shows a comparison of my implementation's performance as compared to the baseline models. For the baseline comparison, I referred to the paper for accuracy values since they are trained on the same dataset.

4.2.1. Justification and analysis

The table below summarizes the results.

Table 2. 5-shot learning performance on mini-ImageNet

Model	Accuracy
Baseline Finetune	49.79%
Baseline Nearest neighbor	51.04%
Matching network	51.09%
My implementation	51.33%
Matching network FCE	55.31%
Author's implementation	60.60%

The first comparison is with the baseline finetune model. This is just a coarser version of the model before fine-tuning the hyperparameters. My model has better accuracy than this.

Next, the baseline nearest neighbor, and the mathing networks models. My implementation achieves comparable results to these models.

The author's implementation gets around 60% accuracy, which is 9% high than what I got. I believe this is because of the high number of training episodes they could have trained it for.

5. Conclusion

5.1. Takeaways

Through this project on implementing the paper, "Optimization as a model for Few-Shot Learning", I learnt a lot of things about practical implementation of machine learning models in PyTorch. Going through a research paper, and utilizing all the minute details outlined throughout it helped me develop a renewed appreciation for how much consideration goes into presenting a novel idea.

My 5-shot implementation could achieve better classification accuracy than the benchmark models used by the authors. However, it could not beat the author's main implementation. Overall, I would conclude the experiment was a success because it could exceed the accuracy of the baseline models at least.

5.2. Further ideas

To achieve equivalent performance to the authors, I feel more episodes and computational resources would be the main factors, along with fine tuning the hyper-parameters further.

6. Acknowledgements

The following people have had a big impact on this paper.

- Sachin Ravi and Hugo Larochelle:

  The entire idea of this implementation, including the hyperparameters and architecture intricacies were derived from the paper, "Optimization as a model for Few-Shot Learning". This paper is authored by Sachin Ravi and Hugo Larochelle, who at the time worked at Twitter Research.

- Vinyals et al.:

  The dataset used, mini-ImageNet, was proposed by Vinyals et al. in the paper "Matching Networks for One Shot Learning".

- The staff and course designers of ECE 50024 - Machine Learning:

  Thank you so much for introducing me to the paper on few-shot learning and providing me with the opportunity to present my re-implementation and analysis of it.

References

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJY0-Kcll.