Light LEMONADE

20203758 박민영

1. Definition of the Problem

현재 알려진 뉴럴 네트워크들은 대부분 전문가가 직접 디자인하는 경우가 일반적이다. 하지만 좋은 성능을 보이는 뉴럴 네트워크를 찾기 위해서 수많은 시행착오를 거쳐야하기 때문에 시간이 많이 소요되고, 경우의 수가 많아 찾기가 어렵다는 단점이 있다. 그래서 최근에는 뉴럴 네트워크의 구조를 자동으로 찾는 연구들이 진행되어 사람이 디자인한 것보다 더 좋은 성능을 보이는 경우도 있다. 뉴럴 네트워크를 자동으로 찾는 분야인 Neural Architecture Search(NAS)에서 single objective의 경우에는 validation loss를 줄이는 목표만 가지고 있다. 이런 방식으로 성능이 좋은 모델을 찾을 수는 있지만, 모델을 찾는데 너무 오랜 시간이 걸려서 GPU 하나로 수천일이 소요된다. 그래서 성능과, computing source의 절약, 이 두가지 목표로 모델을 찾는 multi-objective optimization 연구가 최근 들어 이루어지고 있다. 여기서도 각각의 object에 weighted sum을 하여 loss를 구하는 경우에는 실험 전에 각 object에 대한 가중치를 미리 정해줘야한다는 문제점이 있다. 따라서 이번 프로젝트에서는 앞선 한계점들을 극복하여 성능을 최대화하면서, 파라미터의 개수는 최소화하는 NAS를 수행해보고자 한다. 본 프로젝트는 Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution〔1〕 논문에 소개된 Larmarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign(LEMONADE) 알고리즘을 작은 스케일로 재구현하는 것을 목표로 한다. 따라서 프로젝트의 제목을 Light LEMONADE라고 선정하였다.

2. Methodology

2.1 Dataset

실험에 사용할 데이터셋은 Fashion-MNIST로써 총 7만장의 28*28 grayscale 이미지로 구성되며 상의, 하의, 신발, 가방 종류의 10가지 클래스의 라벨을 포함한다〔2〕. 데이터의 구성은 아래 표와 같다.

Set	Train	Validation	Test	Total
Examples	55000	5000	10000	70000

Table1. Dataset의 구성

2.2 Network Operator

LEMONADE 알고리즘은 진화 알고리즘이기 때문에 operator를 정의해주어야한다. 논문의 내용을 참고하여 정의해보면, $\mathcal{N}(x)$ 는 뉴럴 네트워크가 존재하는 space일 때, $N \in \mathcal{N}(x)$은 각 원소로서 하나의 뉴럴 네트워크를 의미한다. $x \subset \mathbb{R}^n$는 뉴럴 네트워크의 input 이미지이다. 이 때 네트워크 operator T는 $\omega \in \mathbb{R}^k$의 파라미터를 갖는 뉴럴 네트워크 N^ω 를 $\tilde{\omega} \in \mathbb{R}^j$ 의 파라미터를 갖는 뉴럴 네트워크 $(TN)^{\tilde{\omega}} \in \mathcal{N}(x)$ 로 변환한다. 쉽게 말해 뉴럴 네트워크의 구조와 weight를 바꿔주는 operator로써 총 4가지를 설정하였다.

1. 임의의 ReLU layer 다음에 Convolution(Conv)-BatchNormalization(BatchNorm)-ReLU 블록을 삽입한다.
2. 임의의 Conv layer 다음에 연속하여 Conv layer를 추가한다.
3. 임의의 ReLU activation layer output 2개를 골라 skip connection을 추가한다.
4. 임의로 선택한 레이어 한 개를 제거한다.

1~3번은 child 네트워크 구조가 parent에 비해 더 커지는 operator이다. 이는 더 크고, 복잡한 구조의 모델이 학습할 수 있는 용량이 커짐에 따라서 성능이 높아질 수 있다는 가정에 의하여 설정하였다. 4번은 Multi-objective 중 parameter 개수를 최소화하는 조건을 만족시킬 수 있도록, child 네트워크 구조가 parent에 비해 작아지는 operator를 설정하였다. 임의의 layer를 제거해도 네트워크가 연속적으로 잘 동작할 수 있도록 모든 layer에서 input과 output 크기를 'same'으로 설정하였다.

Parent network가 operator를 통해 child network가 될 때 미리 학습된 weight도 함께 전달하도록 설정하였다. 이 방법으로 expensive object인 validation loss를 계산하는데 시간을 절약할 수 있다.

2.3 Light LEMONADE

Input
Conv2D(filter, 3 * 3, padding='same')
BatchNormalization
ReLU
Flatten
Dense(10)
Softmax

Table 2. 초기 뉴럴 네트워크의 구조

이제 앞에서 설명한 network operator를 가지고 Light LEMONADE 알고리즘을 설명하고자 한다.

Algorithm 1 Light LEMONADE

```
input : P0, f, ngen, nac
P ← P0
for i ← 1, … , ngen do
   Nc ← GenerateChildren(P)
   Compute children distribution pchild(Eq.1)
   Nac ← AcceptSubSet(Nc, pchild, nac)
   Evaluate fexp for N ∈ Nac
   P ← ParetoFront(P∪Nac, f)
end for
return P
```

Population P는 parent networks를 포함한다. 초기 Population P_0에 포함될 초기 뉴럴 네트워크 구조는 다음과 같다.

여기서 Conv2D에 사용될 filter는 $filter \in \{4,8,16,32,64\}$ 5가지로 설정해주었고, 따라서 초기 Population은 5개이다. 여기에 operator를 적용하여 Conv layer가 추가되더라도 하나의 모델에서는 모든 Conv layer가 같은 filter 개수를 갖도록 설정해두었다.

그러면 generation(n_{gen})동안 다음 과정을 반복한다. Population P에 대하여 위에서 설명한 network operator로 child network를 생성한다. 초기 population과 operator 종류가 많지 않으므로 한 parent 당 모든 operator를 적용하여 4개의 child가 생성되도록 하였다. 생성된 모든 child를 $\mathbf{N}_c$에 저장하고 cheap objective에 대해 분포를 계산한다. 여기서

cheap objective란 object 중 상대적으로 계산을 빠르게 할 수 있는 것을 의미하며, 이 프로젝트에서는 parameter 개수가 이에 해당한다. 즉, child network의 parameter 개수에

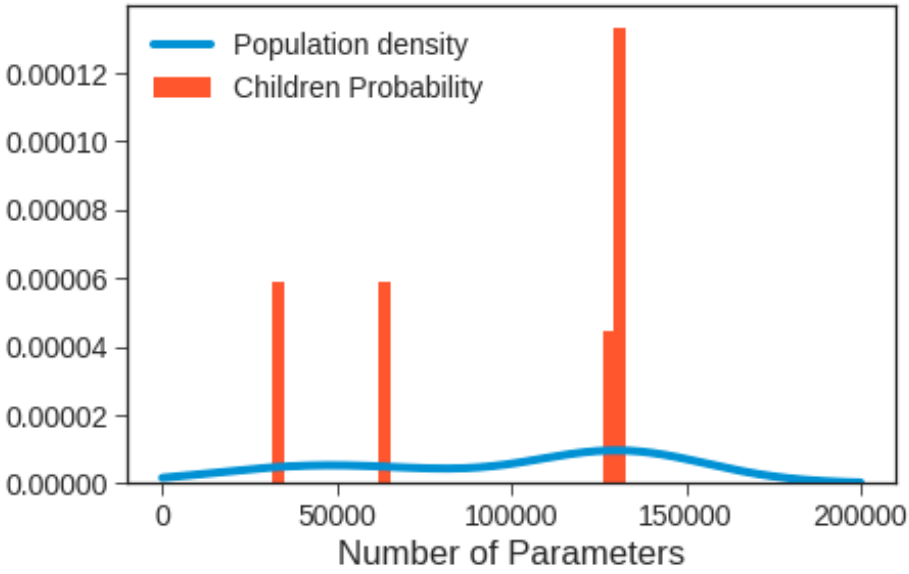

Figure 1. child network의 분포와 gaussian kernel density 추정 결과

대한 distribution을 계산한다. 실험에서는 gaussian kernel density estimator를 사용해서 child network의 parameter 개수의 밀도를 추정했다.

그리고 추정한 밀도의 역수에 비례하는 확률로 child network를 n_{ac}만큼 샘플링한다. 즉, 각 child를 샘플링하는 확률은 아래 Equation 1과 같다.

$$P_{child} = \frac{c}{p_{KDE}(f_{cheap}(N))}, c = (\sum_{N \in N_c} \frac{1}{f_{cheap}(N)})^{-}1 \quad (1)$$

child network의 parameter 개수의 밀도가 높은 부분에서는 낮은 확률로 child를 샘플링하며, 밀도가 낮은 부분에서는 높은 확률로 샘플링하게 되는데, 이는 pareto front를 추정할 때 pareto front 사이의 gap을 채우기 위한 것이다.

선택된 child network를 $\mathbf{N}_{ac}$에 저장하고 이 모델들에 대해서만 expensive objective인 validation loss를 계산한다. 그리고 parent network가 포함된 population과 $\mathbf{N}_{ac}$를 합하여 그 중에서 multi-objective $\mathbf{f}$ 에 대해 pareto front를 계산한다. 실험에서 $\mathbf{f}$는 parameter의 개수와 validation loss이다. 그렇게 선정된 pareto front를 다음 세대 population으로 설정하면 한 세대가 끝나게 된다. 이 과정을 n_{gen} 만큼 반복했을 때, 최종적으로 남은 P가 알고리즘이 추정한 pareto front가 된다.

2.4 Environment

실험에 사용한 프로그래밍 언어는 Python이고, Deep learning library는 Tensorflow Keras를 사용했으며, 클라우드 GPU를 사용할 수 있는 Google Colab 환경에서 실험을 진행하였다.

3. Evaluation

Light LEMONADE 알고리즘으로 Neural Architecture Search를 하는 동안 Pareto front를 추정한 과정을 나타내는 그래프이다.

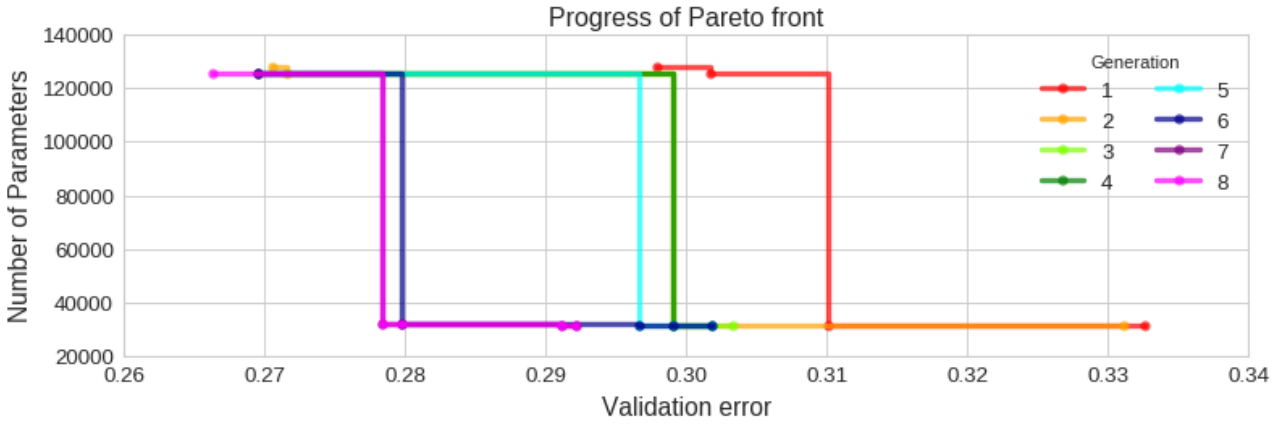

Figure 2. Progress of Pareto front 1

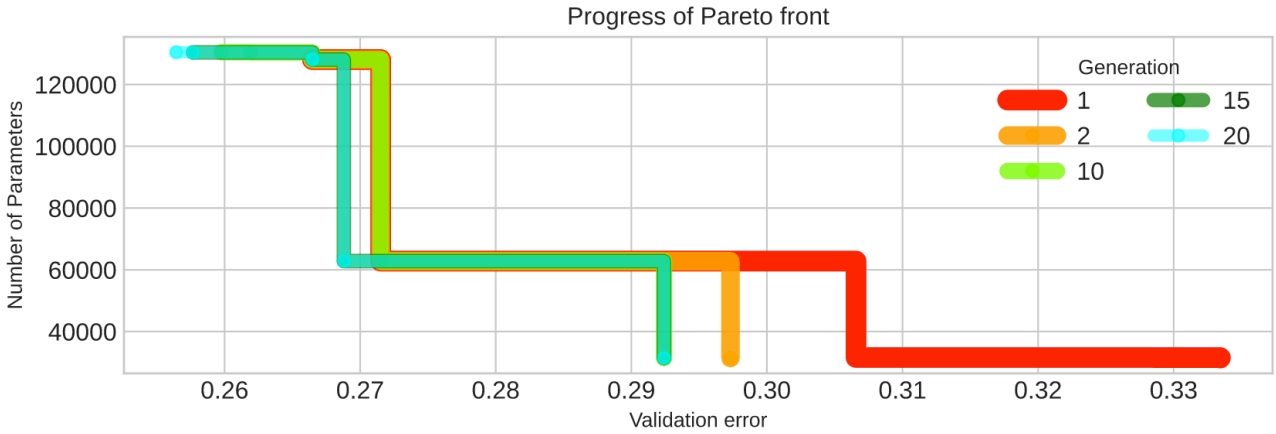

Figure 3. Progress of Pareto front 2

실험을 여러번 진행하였으며(Fig.2, Fig.3), 항상 세대를 거듭할수록 더 optimal한 pareto front를 찾아내는 것을 그래프를 통해 확인할 수 있었다. 하지만 앞선 실험들에서 sampling 하는 child 개수인 n_{ac}=50, expensive object인 validation loss를 계산할 때 필요한 train epoch을 50으로 설정하면 1 generation에서 pareto front를 추정하는데 1시간 이상 소요되었다. 하지만 실험을 진행한 환경이 Colab이었고, 짧은 시간 간격으로 연

산을 마쳐야했기 때문에 충분한 실험을 하는데 어려움이 있었다. 그래서 마지막 실험에서는 짧은 시간에 optimal한 pareto front를 추정하고자 n_{gen} = 20, n_{ac} = 20, train epoch = 5로 설정하였다. 그 결과는 아래 Figure 4와 같다.

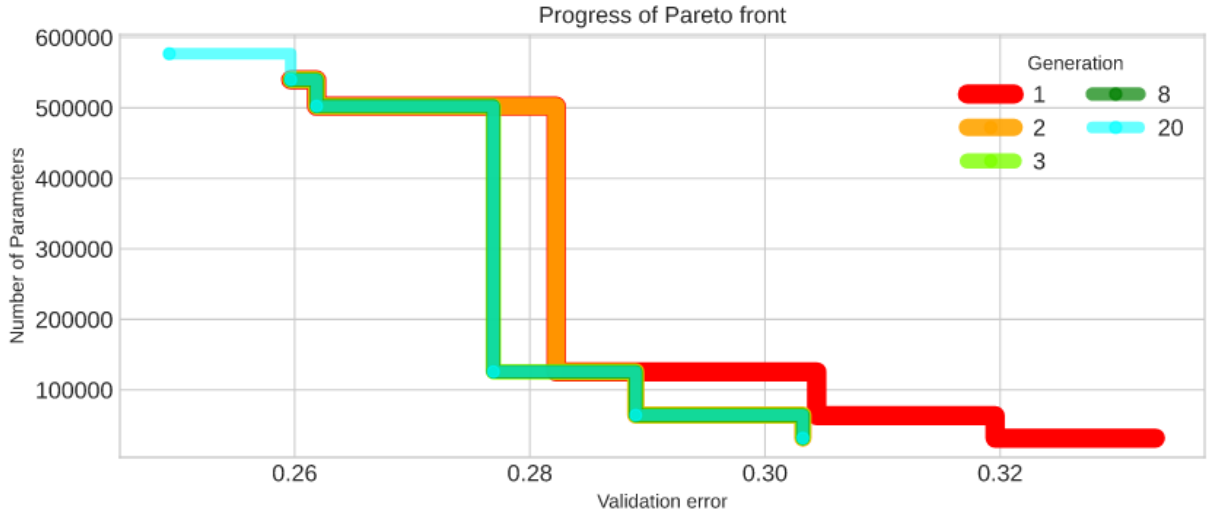

Figure 4. Progress of Pareto front 3

마지막 실험에서는 20세대동안 400개의 모델을 비교하여 pareto front를 추정하는데 2시간 이내로 소요되었으며, 오히려 이렇게 1 generation에 걸리는 시간을 짧게 했을 때 validation error 측면에서는 전체 실험 중에서 가장 optimal한 모델을 찾아낼 수 있었다. Appendix에 마지막 실험에서 20번째 generation에 찾아낸 pareto optimal한 네트워크의 구조들을 첨부하였으며, 아주 간단한 네트워크 구조부터 조금 더 복잡한 네트워크 구조까지 찾아낼 수 있었다.

4. Conclusion

Light LEMONADE 알고리즘으로 Neural Architecture Search를 하여 pareto front를 추정하는 실험을 하였다. 작은 스케일로 만들어진 실험이었기 때문에 실제 State of the Art 뉴럴 네트워크 모델들과 비교해볼 수 없었다는 점이 아쉬웠다. 진화 알고리즘으로 NAS를 할 때 매 세대를 거듭할수록 더 pareto optimal한 모델을 비교적 빠른 시간에 찾아낼 수 있었다. Computing source가 충분한 환경에서 실험을 한다면 SOTA보다 뛰어난 모델들을 찾아낼 수 있을 것이라는 가능성을 실험을 통해 확인할 수 있었다. 그렇게 되면, 데이터셋이나 네트워크 구조에 대한 전문적인 지식이 부족한 사람도 알고리즘을 통해 적합한 모델을 찾아내고, pareto front 중에서 원하는 모델을 선택하여 사용할 수 있을 것이다.

5. GitHub Repository

https://github.com/MY-Park/Light-LEMONADE

6. References

〔1〕 Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution, Thomas Elsken et al. (ICLR 2019)

〔2〕 Fashion-MNIST dataset (https://github.com/zalandoresearch/fashion-mnist)

A. Appendix

Fig.4 에 해당하는 실험에서 generation 20에 pareto front를 구성하는 모델들이다. Light LEMONDAE 알고리즘이 작은 모델부터, 조금 더 복잡한 모델까지 찾아낸 것을 확인할 수 있다.

```
Model: "functional_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_8 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_11 (Activation)   (None, 28, 28, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                501770
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 539,850
Trainable params: 539,594
Non-trainable params: 256
```

Figure 5. discovered model 1

```
_________________________________________________________________
Model: "functional_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
activation_8 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                501770
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 502,410
Trainable params: 502,410
Non-trainable params: 0
```

Figure 6. discovered model 2

```
Model: "functional_69"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 8)         80
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 28, 28, 8)         584
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 28, 28, 8)         584
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 8)         32
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 8)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                62730
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 64,010
Trainable params: 63,994
Non-trainable params: 16
```

Figure 7. discovered model 3

```
Model: "functional_45"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 4)         40
_________________________________________________________________
activation (Activation)      (None, 28, 28, 4)         0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 10)                31370
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 31,410
Trainable params: 31,410
Non-trainable params: 0
```

Figure 8. Smallest discovered model

```
Model: "functional_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
activation_4 (Activation)    (None, 28, 28, 16)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                125450
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 125,610
Trainable params: 125,610
Non-trainable params: 0
```

Figure 9. discovered model 4

```
Model: "functional_369"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_8 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
conv2d_118 (Conv2D)          (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_11 (Activation)   (None, 28, 28, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                501770
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 576,778
Trainable params: 576,522
Non-trainable params: 256
_________________________________________________________________
```

Figure 10. Largest discovered model