Meta Learning

<학습하는 과정을 학습하다.>

Meta learning은 현재 AI에서 가장 유망하고 트랜디한 연구분야로 `AGI(Artificial General Intelligence)`로 나아갈 수 있는 매우 중요한 디딤돌이라고 볼 수 있다.

AGI란 '일반 인공지능', '범용 인공지능'으로 불리는데 이는 AI의 다음단계를 말한다. 기본적으로 AI는 수학 계산의 연장선으로 한정된 환경에서만 가치가 있다. 그 유명한 알파고 마저도 단순히 바둑에서만 데이터를 분석하고 최선의 결과를 제시할 뿐 다른 분야에서는 전혀 사용할 수 없는 소프트웨어다. 반면, AGI는 일반적인 사고와 행동을 할 수 있는 인공지능으로 쉽게 말하면 **'보다 사람에 가까운 인공지능' 이라고** 할 수 있다.

현재 딥 러닝의 실체

AI의 한계를 돌파할 수 있는 주요 돌파구인 메타 러닝에 대해 알아보기 전에 현재 AI, 머신 러닝, 딥 러닝이라고 불리는 것들은 어떻게 작동하는지 알아보자. 이것들을 앞으로는 딥 러닝으로 뭉뚱그려 부르겠다. 딥 러닝을 완성하기 위해서는 데이터와 모델이 필요하다. 수많은 데이터를 가공하여 모델에 넣으면 모델에서 이 데이터를 이용해 결과를 도출한다. 수년 동안 딥 러닝은 훌륭한 모델들의 개발로 빠르게 발전했지만 큰 문제점이 있는데 바로 이 **'수많은 데이터'**이다. 딥 러닝 모델을 학습하기 위해서는 엄청나게 많은 데이터가 필요하다. 또 다른 문제점이 있는데 A라는 task를 수행할 수 있도록 모델을 훈련시켰다고 가정하자. Task A는 가죽 지갑과 천 지갑을 비교하는 task이다. 하지만 비슷한 또 다른 task B는 가죽 장갑과 천 장갑을 비교하는 일인데 기존의 task A를 훈련한 모델로는 이러한 task를 수행할 수 없다.

이것들이 과연 진정한 AI라고 할 수 있을까? 사람은 단 몇 장의 사진만 가지고도 가죽 지갑과 천 지갑을 구분할 수 있고, 이를 통해 가죽 장갑과 천 장갑도 구분해 낼 수 있다. 사람은 몇 번의 학습만으로도 여러 개의 개념들을 학습할 수 있지만, 기존의 딥 러닝 알고리즘은 그러지 못하고 단 하나의 task만 수행한다. 이것이 메타 러닝이 탄생하게 된 **이유**이다.

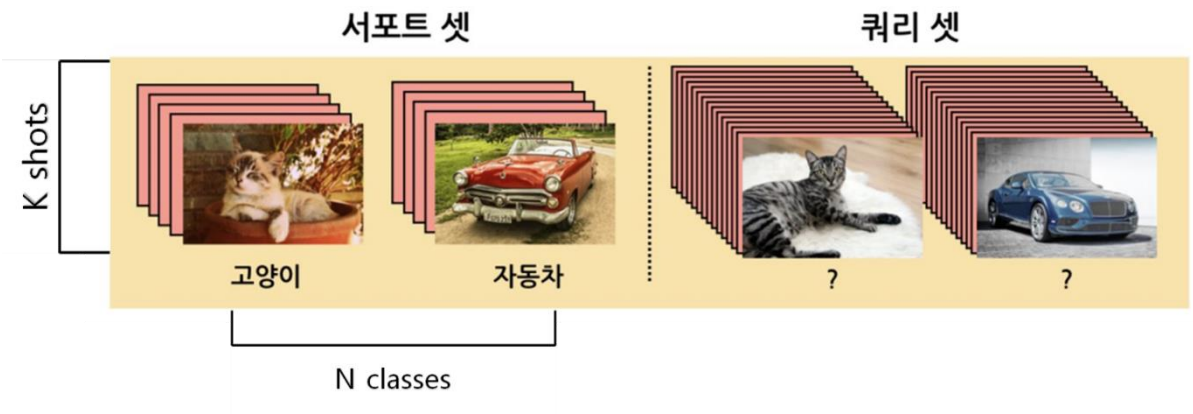
메타 러닝이란?

메타 러닝은 적은 데이터로 모델을 훈련하면 관련 task를 모두 수행할 수 있다. Task A만 훈련시키면 task B는 추가적인 훈련 없이도 바로 구분해낼 수 있는 모델이 탄생하는 것이다. 현재 AI 연구자들은 이러한 메타러닝이 AGI를 달성하는데 큰 도움이 될 것으로 기대하고 있다.

메타 러닝과 퓨 샷, Meta learning and few-shot

적은 데이터로 훈련을 하는 것을 퓨 샷 러닝(`few-shot learning`) 혹은 `n-way k-shot learning`이라고 부른다. 이 때 n은 클래스의 갯수고, k는 각 클래스당 필요한 데이터를 의미한다. 예를 들어 강아지와 고양이를 구분하는 모델을 만들고자 하는데 강아지와 고양이 사진이 각각 1장씩 있으면 2-way 1-shot learning이 되는 것이고 각각 5장씩 있다면 2-way 5-shot learning이 되는 것이다.

메타 러닝에서는 데이터셋(Dataset)을 `서포트셋(support set)`과 `쿼리셋(query set)`으로 나누어 부른다. 둘 다 같은 종류의 데이터셋이지만, 서로 다른 데이터를 가지고 있어야 한다. 우리는 서포트셋을 이용해 train하고 쿼리셋을 이용해 test를 진행한다. 그리고 train을 할 때 한 epoch에서 episode라는 단어를 사용하는데 episode란 n-way k-shot model일 때 n개의 class당 k개의 데이터로 구성된 하나의 데이터셋을 말한다.



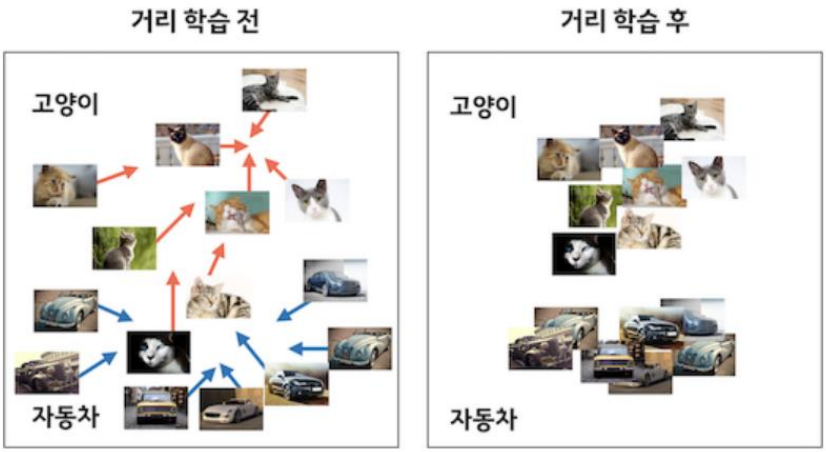
하나의 episode에 n-way k-shot이 구성되는 방법, ©kakaobrain

메타 러닝은 크게 3 가지로 카테고리를 나눌 수 있고, 카테고리는 아래와 같다.

1. Learning the metric space
2. Learning the initializations
3. Learning the optimizer

1. Learning the metric space

두 이미지의 특징을 한 공간에 나타내고 거리를 통해 similarity를 계산하는 방식이다. 같은 이미지면 거리가 가깝게, 다른 이미지면 거리를 멀게 하는 방식으로 이미지를 classification한다. 아래 사진처럼 훈련이 끝나면 비슷한 사진끼리 가깝게 모이는 것을 알 수 있다.



학습하기 전과 후 거리 차이, ©kakaobrain

거리 기반의 학습을 하는 알고리즘은 **siamese networks, prototypical networks, relation networks**등이 있다.

2. learning the initializations

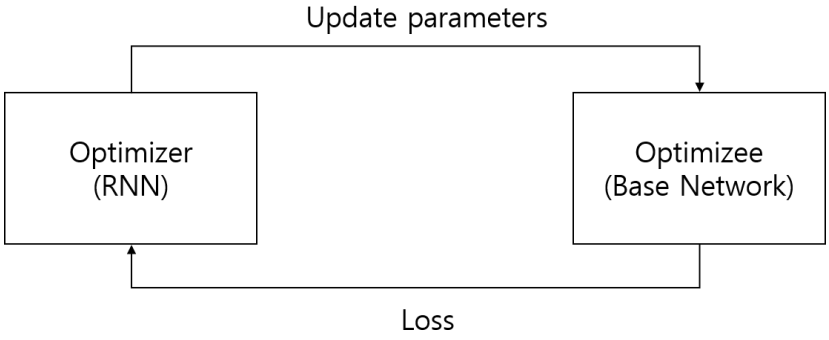
최적의 초기 파라미터 값을 설정하는 것을 배우는 방법이다. 모델을 학습하기 위해서는 모델을 구축하고 weight을 초기화한 다음에 loss를 계산하고 gradient descent를 통해 loss를 최소화한다. 이 과정에서 weight을 초기화 할 때 보통 랜덤하게 값을 넣어 초기화 해주지만 랜덤값을 사용하지 않고 최적화된 initial weight을 찾아서 이 값을 weight의 초기 값으로 세팅해준다면 더 빠르게 학습하고 학습 결과도 더 좋은 결과를 얻을 수 있을 것이다. Optimal initial weight을 찾는 알고리즘은 **MAML, Reptile, Meta-SGD**등이 있다.

3. Learning optimizer

Optimizer를 학습하도록 만든다. 우리는 neural network를 많은 데이터셋을 훈련시키고 gradient descent로 loss를 최소화하는 방향으로 모델을 최적화한다. 하지만, few shot learning은 적은 데이터셋을 사용하기 때문에 이러한 방법이 효과적으로 작동하지 못한다. 따라서 optimizer가 스스로 학습할 수 있도록 한다. Base network와 Meta network 두 가지를 사용해서 Base network는 실제로 맡은 업무를 train하고 meta network는 base network를 optimize하도록 역할을 구성한다.

Learning to learn gradient descent by gradient descent

Learning to learn gradient descent by gradient descent는 메타러닝의 알고리즘을 부르는 말이다. 메타러닝의 목표는 `To learn the learning process'이다. 일반적으로 모델을 학습할 때 gradient descent를 통해 loss를 최적화하는 방향으로 모델을 학습한다. 결과적으로 보통 모델을 optimize 하는 것은 gradient descent를 이용해서 하는데 gradient descent를 사용하는 대신 자동적으로 이러한 process를 optimization 하도록 설계한다. 어떻게 이게 가능할까? Traditional optimizer인 gradient descent대신 그 자리에 RNN(Recurrent Neural Network)를 넣어준다. 그리고 RNN을 optimization하기 위해 gradient descent를 사용한다. 이해하기 조금 어려울 수 있지만, model을 RNN을 통해 optimize 하고 RNN은 gradient descent를 통해 optimize한다. 즉, 우리는 RNN을 통해 gradient descent하는 방법을 학습하고 RNN은 gradient descent를 통해 optimized된다. 이것이 메타 러닝이 learning to learn gradient descent by gradient descent라고 불리는 이유이다.



Learning to learn gradient descent by gradient descent 모형도

RNN을 optimizer로, base network를 optimizee라고 부르고, 모델 f는 파라미터 θ로 parameterized 된다고 가정한다. 목적은 최적화된 파라미터 θ를 찾는 것이다. 따라서 optimizer(RNN)은 최적화된 파라미터θ를 찾고 이것을 optimizee에게 넘겨준다. 그러면 optimizee(base network)는 이 파라미터θ를 통해 loss를 계산하고 이 loss를 다시 optimizer(RNN)으로 넘겨준다. RNN은 loss를 gradient descent를 통해 최적화하고 이 값을 다시 optimizee(base network)로 넘겨주는 방식이다.

Optimizee(base network)는 θ로 parameterized되어 있고 optimizer(RNN)은 Φ로 parameterized되어 있다고 할 때 optimizer의 loss function은 아래와 같다.

$$L(\Phi) = E_f[f(\Theta\left(\mathrm{f}, \Phi\right))]$$

그리고 loss를 최적화하는 방법은 gradient descent를 통해 최적화된 Φ를 찾는다. 따라서 RNN을 function m으로 나타내어 보면

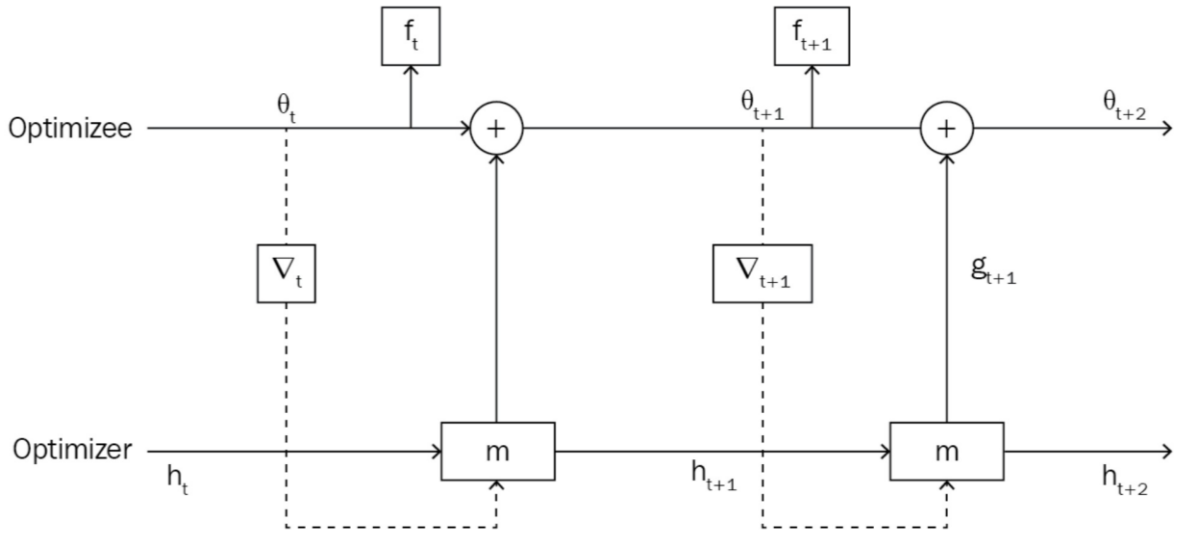
$$(g_t, h_{t+1}) = m(\nabla_t, h_t, \Phi)$$

라고 할 수 있다.

1. $\nabla_t$is the gradient of our model (optimizee)f, that is, $\nabla_t = \nabla_t f(\theta_t)$
2. $h_t$is the hidden state of the RNN
3. Φis the parameter for the RNN
4. Output $(g_t, h_{t+1})$is the update and next state of RNN respectively

출처: packt, hands on Meta learning with python

결국 optimizee의 파라미터 값을 $\theta_{t+1} = \theta_t + g_t$로 업데이트 한다.



Learning to learn gradient descent by gradient descent 흐름도, ***©packt***

이러한 방식으로 gradient descent를 이용한 gradient descent 최적화 방식을 학습할 수 있다.

Reference

Packt, Hands-On meta learning with python