Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Network

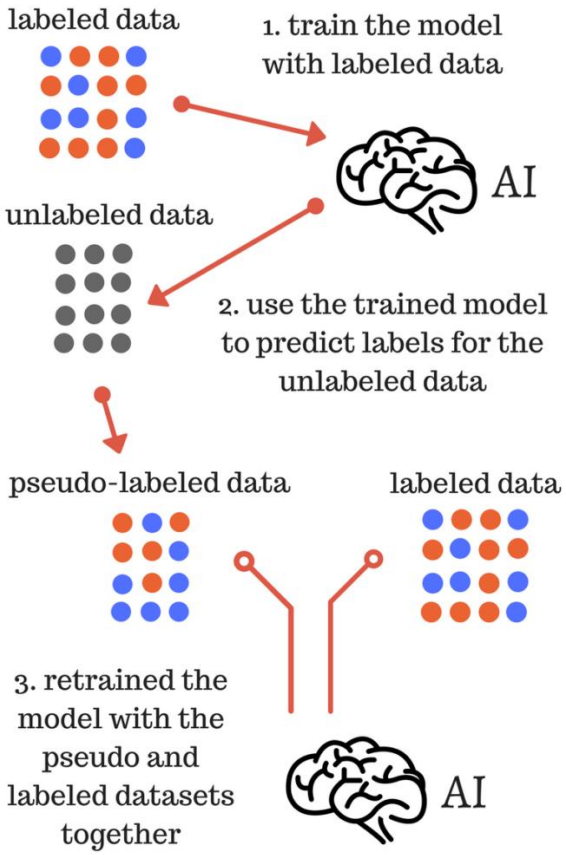
2020020534 김정원

일반적으로 Semi-supervised Learning (SSL)은 labeled data 와 unlabeled data 모두 사용하여 학습하는 것을 의미한다. 그리고 SSL을 사용하는 이유는 만들고자 하는 Model에 쓸 training data가 절대적으로 부족하거나 Large dataset이 될수록 새로 생성되는 data에 대한 Human annotation이 힘들 때 사용한다. 또한 현재 주어진 Labeled data로는 모델의 성능 향상에 한계가 존재할 때 Unlabeled data를 활용해 추가적인 성능 향상의 목적으로 사용하기도 한다.

본 논문에서는 Pseudo-Labeling에 대한 개념을 소개하고, 이를 학습하는 과정을 설명한다. 또한 Pseudo-Label 데이터를 사용했을 때 Semi-supervised Learning의 성능이 어떻게 향상될 수 있는지 Low-Density Separation between Classed 와 Entropy Regularization을 기반으로 설명하고 있다. 마지막으로는 MNIST-data를 활용한 Deep Neural Networks에 Pseudo-Labeling 을 적용한 실험 결과를 보이고 있다.

1. **Pseudo-Labeling**

Pseudo-Labeling의 개념은 Unlabeled data에 대해서 소수의 Labeled data로 학습한 모델을 기반으로 대략적인 Label을 주는 방법을 뜻한다. 아래 그림을 통해 설명을 하면 Pseudo-Labeling의 학습 과정은 다음과 같다. 먼저 소수의 Labeled Data를 통해 target-value를 예측할 수 있는 Model를 생성 & 학습 시킨다. 학습된 모델을 사용해 다수의 Unlabeled Data의 target-value를 예측하고 그 결과를 Label로 사용하는 Pseudo-labeled data를 생성한다. 마지막으로는 Pseudo-labeled data 와 Labeled data를 모두 사용하여 model을 retrain 한다.



2. **Pseudo Labeled data**

Pseudo Label은 아래와 같은 식으로, 각각의 Sample에 대해 예측된 확률이 가장 높은 것으로 정한다.

$$y_i' = \begin{cases} 1 & \text{if } i = \text{argmax}_{i'} f_{i'}(x) \\ 0 & \text{otherwise} \end{cases}$$

예측된 확률이 가장 높은 것을 Label로 선정한다고 했을 때, 제대로 학습을 마치지 못한 Model로 해당 작업을 수행하였을 경우에는 성능을 저해하는 데이터를 만들 뿐입니다. 따라서 Pseudo Label은 학습을 Labeled data를 이용해 일정 수준까지 마친 뒤의 Fine-tuning Phase에서 진행한다.

3. **Loss Function for Pseudo Labeling**

본 논문에서 Pseudo Label로 학습을 할 때 아래와 같은 Loss function을 사용한다.

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y_i'^m, f_i'^m)$$

n: Labeled data의 Batch size의 개수

f_i^m: m번 째 Labeled data sample의 output

y_i^m: m번 째 Labeled data sample의 y값

n': Unlabeled data의 Batch size의 개수

$f_i'^m$: m번 째 Unlabeled data sample의 output

$y_i'^m$: m번 째 Labeled data sample에 해당하는 pseudo – label

α(t): 두 목적함수를 Balancing하는 coefficient, t는 현재 Epoch

위 Loss Function에 해당하는 값들의 의미는 다음과 같다. 그리고 적절한 수치의 α(t)가 네트워크 성능에 있어서 매우 중요한 요소이다. α(t)가 너무 높으면 Labeled data의 training을 방해할 것이고, 반대로 너무 작으면 Unlabeled data를 활용하는 이점을 살릴 수가 없다. 따라서 아래와 같이 점진적으로 그 비율을 늘려주는 방법으로 α(t)를 조절하여 Local minimum에 빠지는 문제를 피하면서 Process의 최적화를 진행한다.

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1 \le t < T_2 \\ \alpha_f & T_2 \le t \end{cases}$$

4. Why could Pseudo-Label work?

본 논문에서는 Pseudo label이 왜 잘 동작하는지 2가지 관점에서 설명을 한다.

4.1 Low-Density Separation between classes

Cluster Assumption (Chapelle et al., 2005)에 의하면 Model의 전반적인 성능을 높이기 위해서는 Model의 Decision boundary는 Low-density regions에 위치해야 한다고 말하고 있다. 즉 Decision boundary를 결정할 때 그 경계를 구분하는 지점의 데이터가 몰려있는 밀도가 낮으면 낮을수록 더 미세한 차이점도 구별한다고 생각하기에 전체적인 성능을 높일 수 있다고 말하고 있다. Pseudo-Label도 Low-Density Separation의 효과를 가져오는 방법이라고 말할 수 있다.

4.2 Entropy Regularization

이는 Bayesian에서 말하는 Maximum posteriori estimation (MAP)의 관점에서 Unlabeled data의 장점을 얻는 수단이다. 이 방식은 Unlabeled-data가 갖는 class별 확률에 대한 Entropy를 최소화 시킴으로써 위에서 언급한 Class들 간의 Low-Density separation을 추구하게 된다.

$$H(y|x') = -\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} P(y_i^m = 1|x'^m)\log P(y_i^m = 1|x'^m)$$

$$C(\theta, \lambda) = \sum_{m=1}^{n} \log P(y^m|x^m;\theta) - \lambda H(y|x';\theta)$$

위 식을 보게 되면, $\sum_{m=1}^{n} logP(y^m|x^m;\theta)$을 첫 번째 항, λH(y|x′:θ)을 두 번째 항이라고 할 때, 첫 번째 항인 labeled data의 log-likelihood를 최대화 시키면서, 두 번째 항인 unlabeled data의 entropy를 최소화 시키기 때문에 좀 더 좋은 성능을 얻을 수 있다고 말하고 있다. 두 번째 항에서 최소화 시킨다는 Entropy는 Class간의 경계가 Overlap 되는 정도를 뜻하는데, Class Overlap이 작아질수록, data들의 밀집된 부분이 더 낮은 decision boundary를 갖게 된다.

5. 실험

본 논문의 실험을 Reproduction하기 위해 **MNIST data**를 이용해 실험을 진행하였다.

```
import torch.nn.functional as F
import torch
import torch.nn as nn


torch.cuda.is_available()
is_cuda = torch.cuda.is_available()

if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")

else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")


class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            self.conv1 = nn.Conv2d(1, 20, kernel_size=5)
            self.conv2 = nn.Conv2d(20, 40, kernel_size=5)
            self.conv2_drop = nn.Dropout2d()
            self.fc1 = nn.Linear(640, 150)
            self.fc2 = nn.Linear(150, 10)
            self.log_softmax = nn.LogSoftmax(dim = 1)

        def forward(self, x):
            x = x.view(-1,1,28,28)
            x = F.relu(F.max_pool2d(self.conv1(x), 2))
            x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
            x = x.view(-1, 640)
            x = F.relu(self.fc1(x))
            x = F.dropout(x, training=self.training)
            x = F.relu(self.fc2(x))
            x = self.log_softmax(x)
            return x

net = Net().to(device)
```

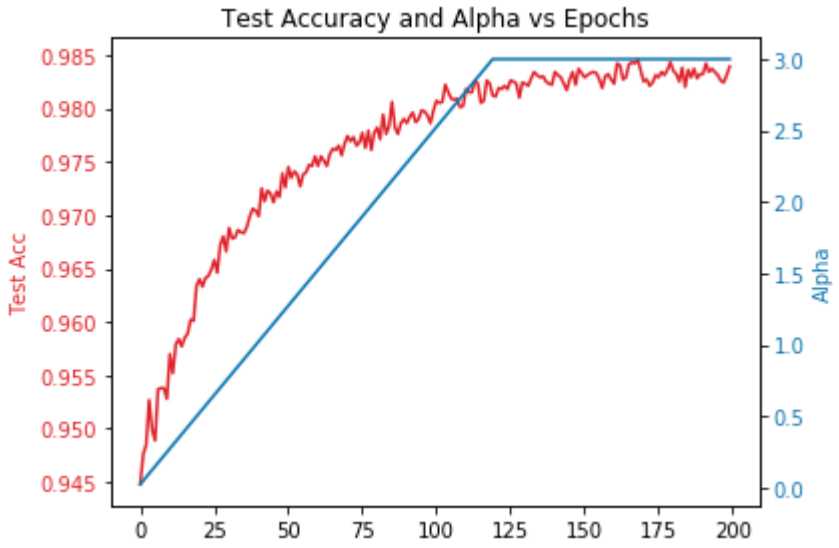
본 논문에서는 간단한 3 fully-connected layer network을 사용하였지만, 본 실험에서는 2 Conv Layer 와 2 Fully connected layer network를 사용하였고 Dropout을 추가하였다.

학습을 위해 1000개의 Labeled image와 19,000개의 unlabeled image를 사용하였고 test data로는 5,000개의 image를 사용하였다.

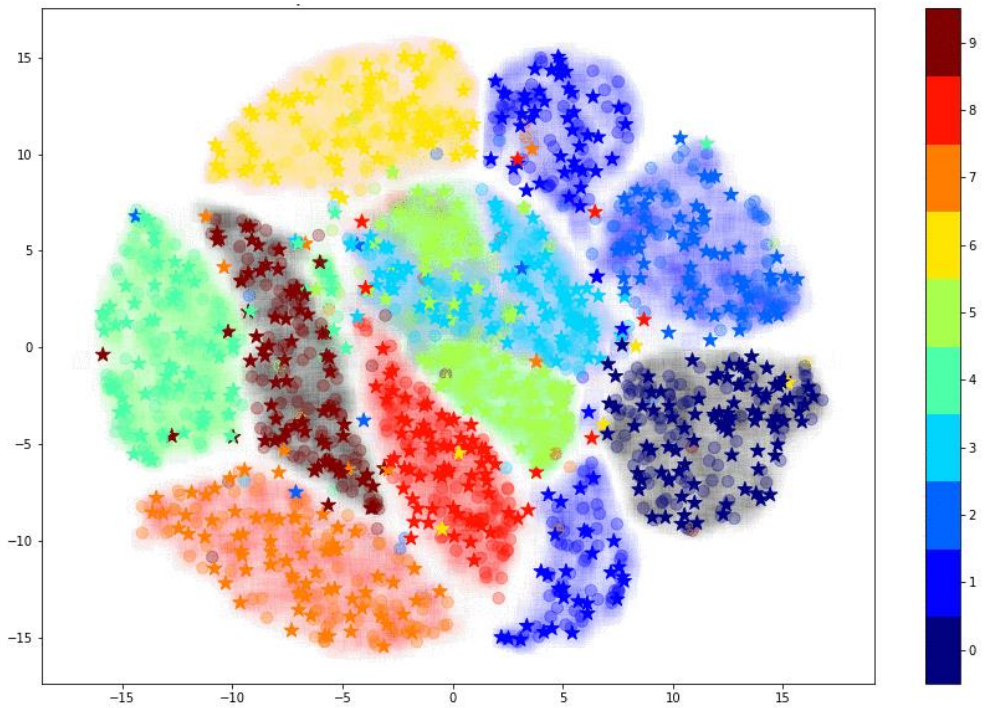
실험의 절차는 다음과 같다. 첫 100 epochs에서는 labeled data만을 사용해 학습을 진행한다. 다음 100 epochs 이상에서는 unlabeled data (알파 가중치 사용) 에 대해서 학습을 진행한다. 총 170 Epochs을 수행한다.

```
T1 = 100
T2 = 700
af = 3

def alpha_weight(step):
    if step < T1:
        return 0.0
    elif step > T2:
        return af
    else:
         return ((step-T1) / (T2-T1))*af


def semisup_train(model, train_loader, unlabeled_loader, test_loader):

    optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)
    EPOCHS = 100

    step = 100

    model.train()
    for epoch in tqdm_notebook(range(EPOCHS)):
        for batch_idx, x_unlabeled in enumerate(unlabeled_loader):

            x_unlabeled = x_unlabeled.cuda()
            model.eval()
            output_unlabeled = model(x_unlabeled)
            _, pseudo_labeled = torch.max(output_unlabeled, 1)
            model.train()


            output = model(x_unlabeled)
            unlabeled_loss = alpha_weight(step) * F.nll_loss(output, pseudo_labeled)


            optimizer.zero_grad()
            unlabeled_loss.backward()
            optimizer.step()


            if batch_idx % 50 == 0:

                for batch_idx, (X_batch, y_batch) in enumerate(train_loader):
                    X_batch = X_batch.cuda()
                    y_batch = y_batch.cuda()
                    output = model(X_batch)
                    labeled_loss = F.nll_loss(output, y_batch)

                    optimizer.zero_grad()
                    labeled_loss.backward()
                    optimizer.step()

                step += 1
```

그리고 Unlabeled data의 50 batch 마다 labeled data에 대해서 1 epoch 학습을 진행한다.

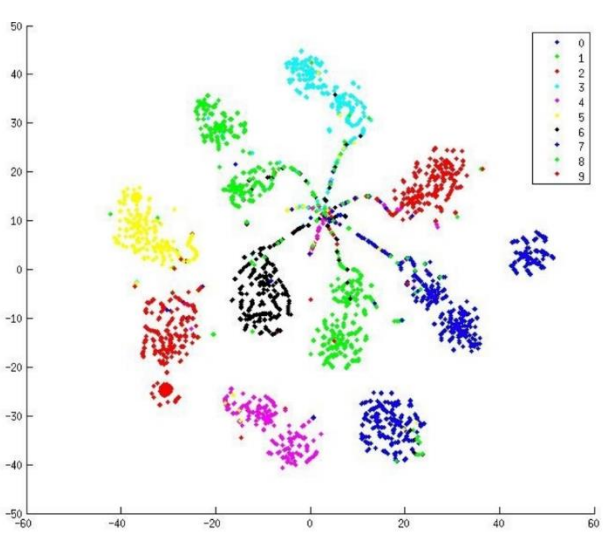


위 그래프를 보게 되면 Alpha가 증가하게 되면 test accuracy 또한 천천히 증가하는 것을 볼 수 있다.

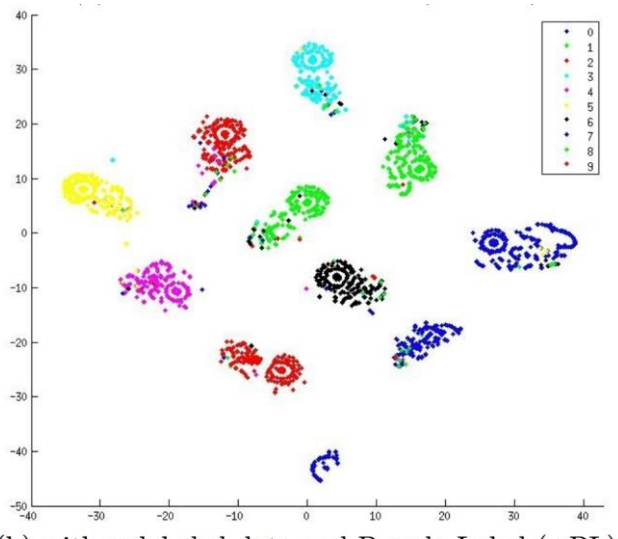


위 그림은 학습결과를 t-SNE을 통해 시각화한 결과이다. Pseudo-Label

을 사용했을 때 분류가 잘 되어 같은 Class를 갖는 객체간 군집을 잘 이루고 있는 것을 볼 수 있다.



(a) without unlabeled data (dropNN)



(b) with unlabeled data and Pseudo-Label (+PL)

위 그림은 논문에서 보여주고 있는 (a) Unlabeled data를 사용하지 않은 결과 (b) unlabeled data 와 Pseudo-Label을 사용한 결과의 t-SNE 결과이다. 위 그림을 보면 (a)에 비해서 (b)가 분류를 더 잘하고 있는 것을 볼 수 있다. 이번 논문 재현에서는 MNIST 데이터를 간단한 CNN을 사용하여 Pseudo-label을 사용한 결과를 재현하였다. 아쉽게 Pseudo-label을 사용하지 않은 실험에 대해서는 진행하지 못하여 결과 비교를 하지 못하였지만, Pseudo-label을 사용했을 때 좋은 성능을 보이고 있는 것을 볼 수 있었다. 추후 다양한 실험을 진행해보면 재미있는 결과를 얻을 수 있을 것 같다.