RESEARCH

REHASHED

Praveen Kumar
Gayathri
Bharadwaja MeherRushi

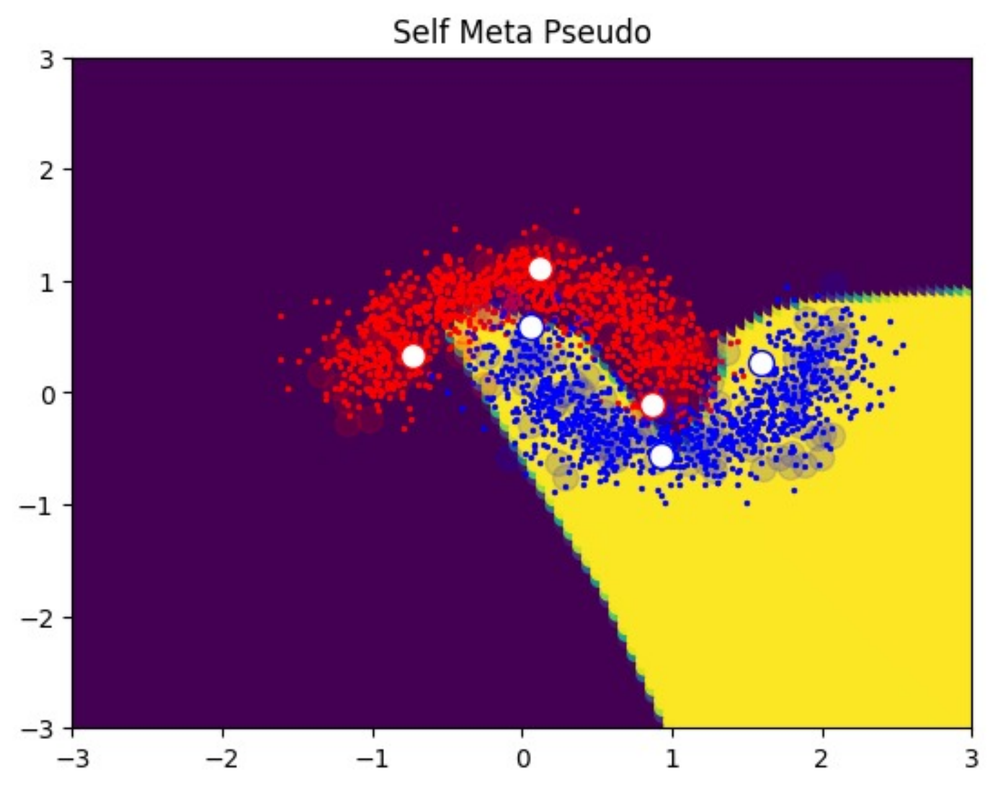
SELF META PSEUDO LABEL

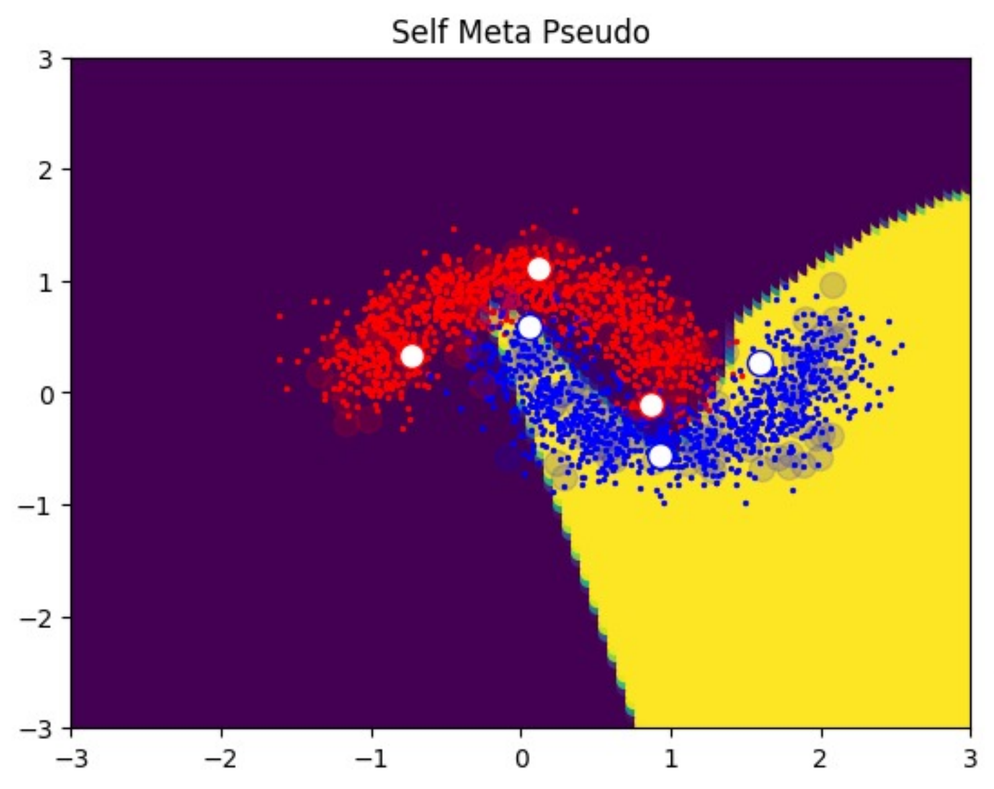
https://arxiv.org/pdf/2212.13420.pdf

We present Self Meta Pseudo Labels, a novel semi-supervised learning method similar to Meta Pseudo Labels [1] but without the teacher model. We introduce a novel way to use a single model for both generating pseudo labels and classification, allowing us to store only one model in memory instead of two. Our method attains similar performance to the Meta Pseudo Labels method while drastically reducing memory usage.

TWO MOON DATASET

Google Colaboratory
google.com





UNDERSTANDING THE CHANGE TERM (MATHEMATICALLY AND LOGICALLY)

```
# Calculate the change in the loss
change = loss_new - loss1.detach()
```

$$dot\_product = \eta\nabla_{\theta'}\mathrm{CE}(\mathrm{y}_l,\mathrm{S}(\mathrm{x}_l,\theta'))\cdot\nabla_{\theta}\mathrm{CE}(\mathrm{y}_u,\mathrm{S}(\mathrm{x}_u,\theta)) \text{ where } \theta' = \theta - \eta\nabla_{\theta}CE(y_u,S(x_u,\theta))$$

$$let\ d\theta = \eta\nabla_{\theta}CE(y_u,S(x_u,\theta))$$

$$so\ \theta = \theta' + \eta\nabla_{\theta}CE(y_u,S(x_u,\theta)) = \theta' + d\theta$$

$$\theta \text{ is the old value, } \theta' \text{ is updated from it and it is the new}$$

$$loss\_1\_old - loss\_1\_new$$

$$= E(y_l,S(x_l,\theta)) - CE(y_l,S(x_l,\theta'))$$

$$= CE(y_l,S(x_l,\theta' + d\theta)) - CE(y_l,S(x_l,\theta'))$$

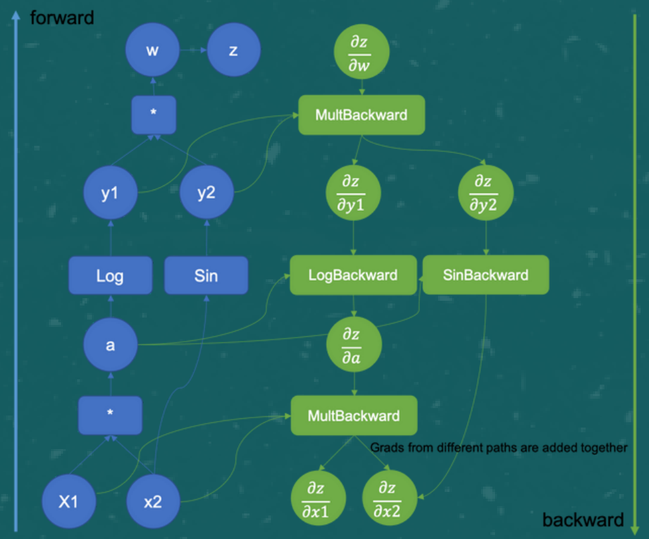
$$= \nabla_{\theta'}CE(y_l,S(x_l,\theta')) \cdot d\theta$$

$$= \eta\nabla_{\theta'}CE(y_l,S(x_l,\theta'))\cdot\nabla_{\theta}CE(y_u,S(x_u,\theta))$$

OUR IMPLEMENTATION JOURNEY

FIRST ISSUE : LOADING THE DATASET

NEXT ISSUE : BACK PROP ERROR - UNDERSTANDING AUTOGRAD COMPUTATIONAL GRAPHS



.detach()

.item()

```
ce_new = loss_ce_y.item()
ce_old = loss_old.item()
change = ce_new - ce_old
```

```
loss_mpl = ((change)* (nn.CrossEntropyLoss()(new_unlabeled_logits,new_pseudo_labels)))
```

```
loss_2 = loss_uda + loss_mpl
(loss_2).backward()
```

NEXT ISSUE : NAN VALUES

nan values for loss. we thought our loss is very less. But nope.
Then we thought it was a gradient clipping error and then we manually debugged the code and we realised that it was a log term error.

```
BETA_K = beta_zero * torch.min(i,((epoch+1)/a))
loss_partial = BETA_K * torch.sum(-(torch.softmax(new_unlabeled_logits,dim=1)) * torch.log(torch.softmax(new_unlabeled_logits_augmented,dim=1), dim=0).mean()
loss_uda = loss_ce_y + loss_partial
```

We tried gradient clipping and log1p to solve the issues

```
torch.nn.utils.clip_grad_norm_(model.parameters(), clipping_value)
```

TORCH.LOG1P

torch.log1p(*input*, *, *out=None*) → Tensor

Returns a new tensor with the natural logarithm of (1 + `input`).

$$y_i = \log_e(x_i + 1)$$

• NOTE

This function is more accurate than `torch.log()` for small values of `input`

Parameters

NEXT ISSUE : DATALOADER ORDER ISSUE

Gayathri : Ok...I just had a thought...so when we are doing this batch wise training, we dint do shuffling coz we wanted like the image n its augmented image to be together...so in each batch we hv images of the same class ryt...so ig the model might just learn the order in which images n not their features...correct me if im wrong 😅

Solution:

Random seed

```
tensor([8, 1, 8, 0, 5, 4, 3, 4, 8, 1, 0, 4, 1, 1, 5, 5, 0, 8, 6, 0, 3, 4, 6, 6,
        0, 8, 3, 7, 5, 8, 4, 0, 7, 1, 6, 3, 4, 8, 3, 9, 1, 6, 7, 4, 0, 8, 7, 5,
        7, 2, 8, 3, 2, 5, 4, 0, 1, 3, 8, 1, 6, 6, 9, 5, 6, 9, 7, 7, 2, 0, 1, 4,
        6, 3, 2, 2, 7, 1, 2, 1, 0, 0, 8, 4, 3, 5, 0, 9, 8, 6, 0, 5, 7, 2, 5, 4,
        1, 1, 8, 9, 0, 1, 6, 0, 4, 5, 5, 3, 9, 4, 7, 1, 4, 3, 9, 4, 2, 5, 3, 4,
        2, 7, 1, 6, 7, 6, 9, 5])
tensor([8, 1, 8, 0, 5, 4, 3, 4, 8, 1, 0, 4, 1, 1, 5, 5, 0, 8, 6, 0, 3, 4, 6, 6,
        0, 8, 3, 7, 5, 8, 4, 0, 7, 1, 6, 3, 4, 8, 3, 9, 1, 6, 7, 4, 0, 8, 7, 5,
        7, 2, 8, 3, 2, 5, 4, 0, 1, 3, 8, 1, 6, 6, 9, 5, 6, 9, 7, 7, 2, 0, 1, 4,
        6, 3, 2, 2, 7, 1, 2, 1, 0, 0, 8, 4, 3, 5, 0, 9, 8, 6, 0, 5, 7, 2, 5, 4,
        1, 1, 8, 9, 0, 1, 6, 0, 4, 5, 5, 3, 9, 4, 7, 1, 4, 3, 9, 4, 2, 5, 3, 4,
        2, 7, 1, 6, 7, 6, 9, 5])
```

Praveen :

```
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0... readmore
```

Praveen : this is each batch and labels which we train

NEXT ISSUE : LOG1P IS STILL THE PROBLEM

ok, I figured something out in the training loop as well.

I think the loss is increasing madly negative because of log1p. What log1p does is that it calculates log(1+x) instead of log(x). but is the issue because, out function is - something * log(softmax(x)), and since softmax is between 0 and 1, we want the log to be -ve so that the -something time negative is positive. but using log1p adds 1 to the value and then log becomes +ve the overall becomes -ve.

So I clamped the value and things seem to be better now 10:54 AM

i used .clmap between 1e-15 and 1e15 on the log term 10:54 AM

and loss old is decreasing 10:55 AM

NEXT ISSUE : MASKING

Praveen Kumar nitk cse

Praveen Kumar nitk cse

Photo

we should have done masking for uda loss 1:36 PM

that is the mistake 1:36 PM

that is why loss is not reducing 1:36 PM

Praveen Kumar nitk cse

β_k is a warm-up variable to control the magnitude. It gradually increases until the total number of steps reaches a constant a. $\mathcal{L}_{UDA}$ is masked and only predictions with high confidence are used. $\mathcal{L}_s$ is the cross-entropy loss for the labeled examples. For the semi-supervised loss:

1:36 PM

💀

we should have seen some mpl implementations before this smpl 1:38 PM

Praveen Kumar nitk cse

```
eshold', default=
```

masking threshold which they had is 0.95😂

1:40 PM

Praveen Kumar nitk cse

Praveen Kumar nitk cse

masking threshold which they had is 0.95😂

```
', de
```

ours is 0, no masking 1:41 PM

Yea now it all makes sense, the reason why ssl takes like 8k epochs is basically this, cuz of this threshold, model will learn very slowly🤦 1:46 PM

❤️😮 2

NEXT ISSUE : RANDOM DYNAMIC SEED

```
epochs = EPOCHS
beta_zero = BETA_ZERO
num_epochs = epochs
torch.manual_seed(43)
labeled_batches = iter(labeled_train_loader)
torch.manual_seed(43)
unlabeled_batches = iter(unlabeled_train_loader)
torch.manual_seed(43)
unlabeled_batches_augmented = iter(unlabeled_train_loader_augmented)
torch.manual_seed(43)
test_batches = iter(test_loader)

for epoch_num in range(10):
    acc = 0
    model.train()
    loss_per_epoch_1 = 0
    loss_per_epoch_2 = 0
    for epoch in range(len(unlabeled_train_loader)):
        try:
            labeled_images, labels = next(labeled_batches)
        except StopIteration:
            torch.manual_seed(43)
            labeled_batches = iter(labeled_train_loader)
            labeled_images, labels = next(labeled_batches)

        try:
            unlabeled_images, uy = next(unlabeled_batches)
        except StopIteration:
            torch.manual_seed(43)
            unlabeled_batches = iter(unlabeled_train_loader)
            unlabeled_images, uy = next(unlabeled_batches)

        try:
            unlabeled_images_augmented, uya = next(unlabeled_batches_augmented)
        except StopIteration:
            torch.manual_seed(43)
            unlabeled_batches_augmented = iter(unlabeled_train_loader_augmented)
            unlabeled_images_augmented, uya = next(unlabeled_batches_augmented)
```

Initially , we used a static Random seed.

```
unlabeled_batches = iter(unlabeled_train_loader)
torch.manual_seed(2)
unlabeled_batches_augmented = iter(unlabeled_train_loader_augmented)
torch.manual_seed(2)
test_batches = iter(test_loader)
curr_loss = 10101
r_seed = 2
for epoch_num in range(25):
    acc = 0
    print("\n\n\n\n\nEPOCH NUMBER",epoch_num+1,"\n\n\n\n")
    model.train()
    loss_per_epoch_1 = 0
    loss_per_epoch_2 = 0
    for epoch in range(len(unlabeled_train_loader)):
        try:
            labeled_images, labels = next(labeled_batches)
        except StopIteration:
            r_seed+=1
            torch.manual_seed(r_seed)
            labeled_batches = iter(labeled_train_loader)
            labeled_images, labels = next(labeled_batches)

        try:
            unlabeled_images, uy = next(unlabeled_batches)
        except StopIteration:
            torch.manual_seed(r_seed)
            unlabeled_batches = iter(unlabeled_train_loader)
            unlabeled_images, uy = next(unlabeled_batches)

        try:
            unlabeled_images_augmented, uya = next(unlabeled_batches_augmented)
        except StopIteration:
            torch.manual_seed(r_seed)
            unlabeled_batches_augmented = iter(unlabeled_train_loader_augmented)
            unlabeled_images_augmented, uya = next(unlabeled_batches_augmented)


        with torch.no_grad():
                    pseudo_labels = torch.argmax(torch.softmax(model(unlabeled_images.to(
```

Finally, we used Dynamic Random Seed

RESULTS

SEMI SUPERVISED

```
1.1414213180541992 0.5173166990280151 0.5638799071311951 0.2881859540939331 0.852
1.0736099481582642 0.5944867730140686 0.6262534856796265 0.028398998081684113 0.6
1.0170241594314575 0.48753514885902405 0.4892403185367584 0.021584734320640564 0
1.0171313285827637 0.45978158712387085 0.45083490014076233 0.026474185287952423 0
1.1154637336730957 0.8316525220870972 1.051405668258667 0.02493058145046234 1.076
Accuracy per epoch : 77.77887658227849
Loss_1:191.14717623591423
Loss_2:0.0
```

Epoch number 72
Accuracy : 77.7 (on train)
Accuracy : 75.09 (on test)
Least old loss : 0.35
Avg old loss : 0.50

SUPERVISED

```
loss_old: tensor(0.1400, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.2075, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.2154, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.3288, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.2172, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.1119, device='cuda:0', grad_fn=<NllLos
loss_old: tensor(0.1039, device='cuda:0', grad_fn=<NllLos
Accuracy per epoch : 73.41772151898734
Loss_1:0.1983805252239108
```

Epoch Number : (many ~50)
Accuracy on Test : 73.41
Least old loss : 0.1119
Avg old loss : 0.20


THANK YOU!