Summary of Self-ensemble Methods

1. Introduction

Self-ensemble methods are used for semi-supervised learning (SSL). These methods can also be called as **perturbation-based methods** or **consistency regularization methods**.

The main idea of self-ensemble methods is based on the **low-density separation assumption** in SSL, which means that the decision boundary should lie in the low-density regions of data points.

Self-ensemble methods enforce the low-density separation assumption by encouraging invariant prediction $f(u) = f(u + \delta)$ for perturbations $u + \delta$ of unlabeled points u. Usually there is a pair of teacher-student networks in self-ensemble methods. Predictions of the student network should be consistent with that of the teacher network after perturbations. Different methods have different ways to define the teacher network and enforce the consistency.

The intuition behind consistency regularization methods is straightforward. If the decision boundary lies in a high-density region, the perturbed data points $u + \delta$ will be very likely to cross over the boundary. As a result, the consistency between u and $u + \delta$ will be violated, and the corresponding punishment for inconsistency will be high. Therefore, the decision boundary will be pushed to low-density regions by consistency regularization.

2. Methods

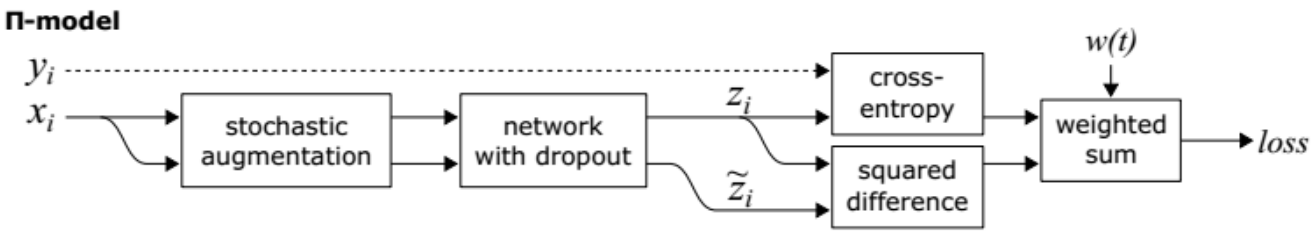
2.1 Π model & Temporal Ensembling

【**Paper**】Temporal Ensembling for Semi-supervised Learning

【**Source**】ICLR 2017

This paper proposed two ways to implement self-ensembling, called Π model and Temporal ensembling.

1) Π model



For each training input x_i, the network of Π model is evaluated twice with stochastic augmentation and dropout. The consistency is computed between two stochastic predictions z_i and $\tilde{z}_i$ given the same input x_i, in the form of mean squared difference.

Π model can also be seen as two network pathways with shared parameters and different perturbations.

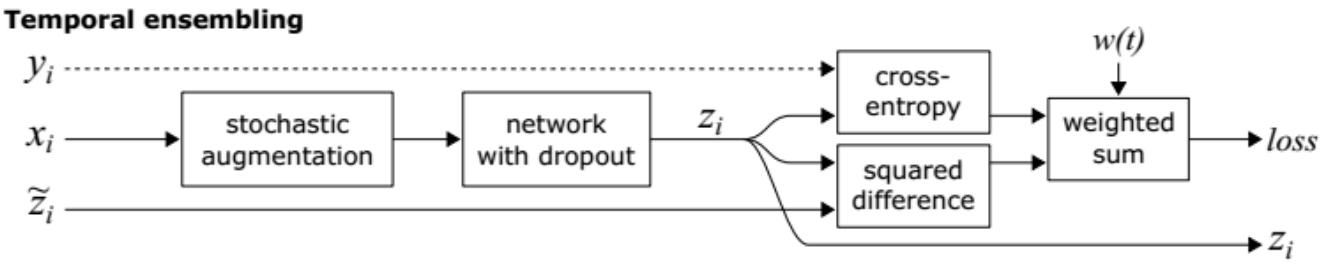
The authors use a time-dependent weighting function $w(t)$ for the consistency loss. The weighting function $w(t)$ starts from zero and rams up along a Gaussian curve.

Algorithm 1 Π-model pseudocode.

Require: x_i = training stimuli
Require: L = set of training input indices with known labels
Require: y_i = labels for labeled inputs $i \in L$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters θ
Require: $g(x)$ = stochastic input augmentation function
for t in $[1, num\_epochs]$ **do**
  for each minibatch B **do**
    $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$ ▷ evaluate network outputs for augmented inputs
    $\tilde{z}_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$ ▷ again, with different dropout and augmentation
    $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$ ▷ supervised loss component
      $+ w(t) \frac{1}{C|B|} \sum_{i \in B} ||z_i - \tilde{z}_i||^2$ ▷ unsupervised loss component
    update θ using, e.g., ADAM ▷ update network parameters
  end for
end for
return θ

2) Temporal Ensembling

Temporal ensembing is an adapted version of Π model. However, the network of temporal ensembling is evaluated only once per input per epoch, and the target vectors $\tilde{z}$ for consistency loss are based on prior network predictions, instead of a second evaluation of the network.



For each input sample, temporal ensembling maintains an exponential moving average $Z_i \leftarrow \alpha Z_i + (1 - \alpha) z_i$ of the previous predictions Z_i and current predictions z_i. Then Z is biased by dividing by factor $(1 - \alpha^t)$ to obtain the target vectors $\tilde{z}_i$.

Temporal ensembling forces the network to predict consistently with the ensembles of its previous predictions, which means that the network should take a small step each time.

Algorithm 2 Temporal ensembling pseudocode. Note that the updates of Z and $\tilde{z}$ could equally well be done inside the minibatch loop; in this pseudocode they occur between epochs for clarity.

Require: x_i = training stimuli
Require: L = set of training input indices with known labels
Require: y_i = labels for labeled inputs $i \in L$
Require: α = ensembling momentum, $0 \leq \alpha < 1$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters θ
Require: $g(x)$ = stochastic input augmentation function

$Z \leftarrow \mathbf{0}_{[N \times C]}$ ▷ initialize ensemble predictions
$\tilde{z} \leftarrow \mathbf{0}_{[N \times C]}$ ▷ initialize target vectors
for t in $[1, num\_epochs]$ **do**
  for each minibatch B **do**
    $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}, t))$ ▷ evaluate network outputs for augmented inputs
    $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$ ▷ supervised loss component
      $+ w(t) \frac{1}{C|B|} \sum_{i \in B} ||z_i - \tilde{z}_i||^2$ ▷ unsupervised loss component
    update θ using, e.g., ADAM ▷ update network parameters
  end for
  $Z \leftarrow \alpha Z + (1 - \alpha) z$ ▷ accumulate ensemble predictions
  $\tilde{z} \leftarrow Z / (1 - \alpha^t)$ ▷ construct target vectors by bias correction
end for
return θ

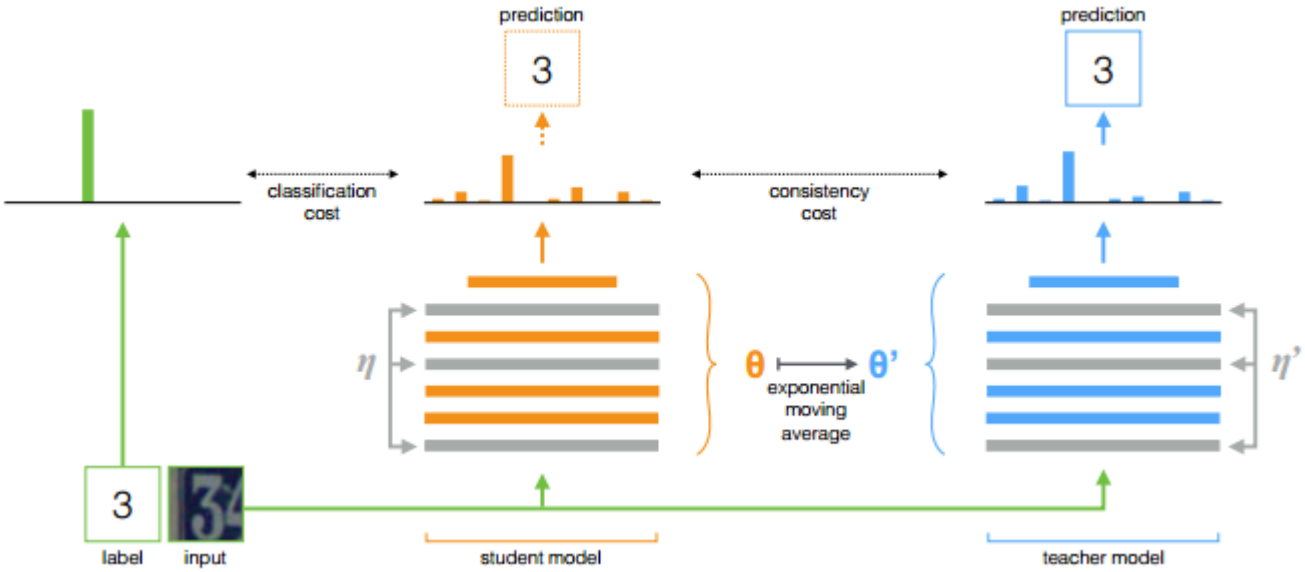
Compared with Π model, the training of temporal ensembling is faster because the network is evaluated only once. In addition, the training targets for consistency loss is less noisy because of the moving average. However, in order to maintain the moving averages for each sample, an auxiliary storage is needed. Moreover, a new hyperparameter α is also introduced by temporal ensembling.

2.2 Mean Teachers

【**Paper**】Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results

【**Source**】NIPS 2017

Mean teachers (MT) is an improvement of temporal ensembling. Temporal ensembling updates the consistence targets for each sample only once per epoch. As a result, the learned information is incorporated into the training process at a low pace. Moreover, temporal ensembling is not scalable to large datasets because of the auxiliary storage for the moving averages of each sample. To overcome the limitations of temporal ensembling, MT averages model weights instead of label predictions.



The teacher model weights are updated as an exponential moving average of the student weights.

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t$$

By this way, the targets provided by the teacher model can be more accurate. Note that the inputs to student and teacher network are added with different noises.

Both model outputs can be used for prediction, but at the end of the training the teacher prediction is more likely to be correct because the teacher parameters are obtained by moving average of the student parameters.

2.3 VAT

【**Paper**】Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning

【**Source**】TPAMI 2018

VAT adds adversarial perturbations to the inputs instead of random noises. In other words, VAT takes the "worst cases" of perturbations into account. The consistence is computed between the network outputs of the original samples and the corresponding adversarial samples.

Adversarial training for supervised learning finds adversarial perturbations based on the divergence of ground-truth label distributions and the output distributions of adversarial samples.

$$\begin{aligned} L_{\text{adv}}\left(x_l, \theta\right) &:= D\left[q\left(y|x_l\right), p\left(y|x_l + r_{\text{adv}}, \theta\right)\right] \\ \text{where } r_{\text{adv}} &:= \underset{r;\|r\|\leq\epsilon}{\arg\max} D\left[q\left(y|x_l\right), p\left(y|x_l + r, \theta\right)\right] \end{aligned}$$

In order to adapt semi-supervised scenarios, VAT finds adversarial perturbations based on the divergence between the output distributions of the original samples and adversarial samples.

$$\begin{aligned} \text{LDS}(x_*, \theta) &:= D\left[p\left(y|x_*, \hat{\theta}\right), p\left(y|x_* + r_{\text{vadv}}, \theta\right)\right] \\ r_{\text{vadv}} &:= \arg\max_{r;\|r\|_2\leq\epsilon} D\left[p\left(y|x_*, \hat{\theta}\right), p\left(y|x_* + r\right)\right] \end{aligned}$$

If we view the output distributions of original samples as "virtual" labels, then VAT can be seen as computing adversarial directions based on the current virtual labels.

The consistency regularization term is the average of $\mathrm{LDS}(x_*, \theta)$ over all data points:

$$\mathcal{R}_{\mathrm{vadv}}\left(\mathcal{D}_l, \mathcal{D}_{ul}, \theta\right) := \frac{1}{N_l + N_{ul}} \sum_{x_* \in \mathcal{D}_l, \mathcal{D}_{ul}} \mathrm{LDS}(x_*, \theta)$$

Note that the consistency loss is computed in the form of **KL divergence**.

In order to find the adversarial perturbations r_{vadv}, VAT needs to compute gradients with respect to the inputs, which is computational inefficient. Therefore, the authors propose a fast approximation method for computing the adversarial perturbations. More details can be found in the paper.

Algorithm 1 Mini-batch SGD for $\nabla_\theta \mathcal{R}_{\mathrm{vadv}}(\theta)|_{\theta=\hat{\theta}}$, with a one-time power iteration method.

1) Choose M samples of $x^{(i)}(i = 1, \ldots, M)$ from dataset $\mathcal{D}$ at random.
2) Generate a random unit vector $d^{(i)} \in R^I$ using an iid Gaussian distribution.
3) Calculate r_{vadv} via taking the gradient of D with respect to r on $r = \xi d^{(i)}$ on each input data point $x^{(i)}$:

$$g^{(i)} \leftarrow \nabla_r D\left[p(y|x^{(i)}, \hat{\theta}), p(y|x^{(i)} + r, \hat{\theta})\right]\Big|_{r=\xi d^{(i)}},$$

$$r_{\mathrm{vadv}}^{(i)} \leftarrow g^{(i)} / \|g^{(i)}\|_2$$

4) **Return**

$$\nabla_\theta \left(\frac{1}{M} \sum_{i=1}^{M} D\left[p(y|x^{(i)}, \hat{\theta}), p(y|x^{(i)} + r_{\mathrm{vadv}}^{(i)}, \theta)\right]\right)\Big|_{\theta=\hat{\theta}}$$

The authors also adopt a conditional entropy regularization term of $p(y|x, \theta)$ as an additional cost:

$$\begin{aligned}\mathcal{R}_{\mathrm{cent}} &= \mathcal{H}(Y|X) \\ &= -\frac{1}{N_l + N_{ul}} \sum_{x \in \mathcal{D}_l, \mathcal{D}_{ul}} \sum_y p(y|x, \theta) \log p(y|x, \theta)\end{aligned}$$

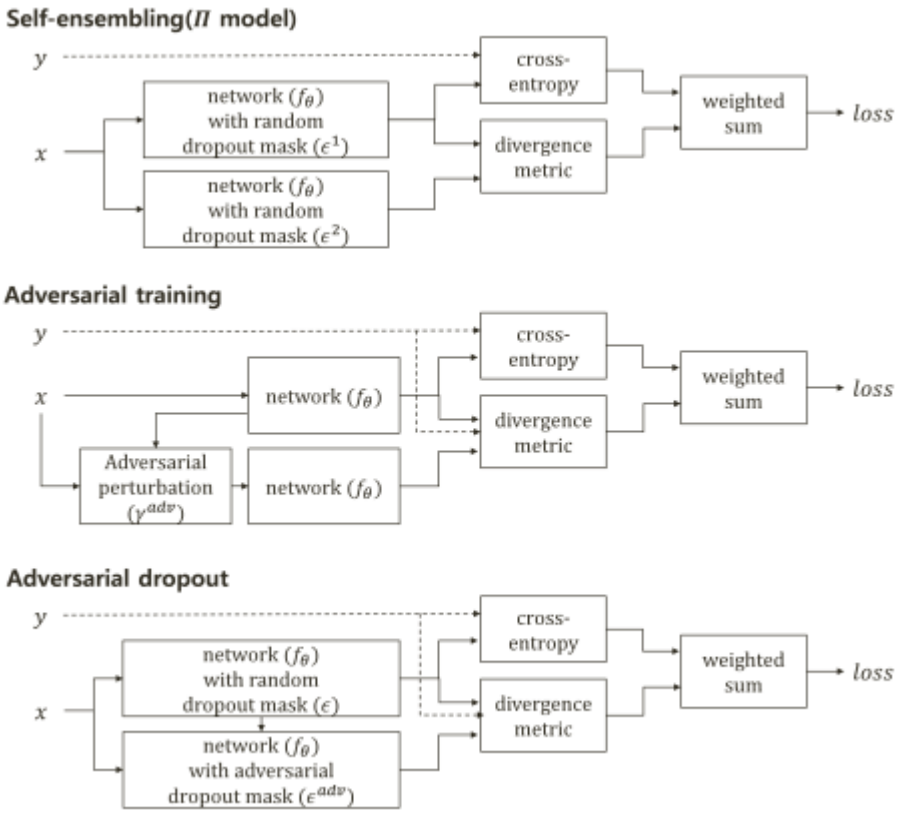
The conditional entropy minimization has an effect of exaggerating the prediction of the model $p(y|x, \theta)$ on each data point. VAT with this conditional entropy minimization is called "VAT+EntMin".

2.4 VAdD

【**Paper**】Adversarial Dropout for Supervised and Semi-Supervised Learning

【**Source**】AAAI 2018

The idea of adversarial dropout is very similar to VAT, but instead of adding perturbations to the input data points, AdD applies the adversarial training concept to the sampling of neural network structures through dropout.



AdD can also be interpreted as a special case of Π model, where the second dropout mask is optimized adversarially to the model's label (or virtual label) assignment, instead of another random one.

In semi-supervised scenarios, given a random sampled dropout mask, virtual AdD (VAdD) will find the adversarial mask as bellow:

$$\mathcal{L}_{VAdD}\left(\mathbf{x}, y, \epsilon^s; \boldsymbol{\theta}, \delta\right) := D\left[f_\theta\left(\mathbf{x}, \boldsymbol{\epsilon}^s\right), f_{\boldsymbol{\theta}}\left(\mathbf{x}, \boldsymbol{\epsilon}^{adv}\right)\right]$$
$$\text{where} \quad \epsilon^{adv} := \operatorname{argmax}_{\epsilon; \|\epsilon^s - \epsilon\|_2 \le \delta H} D\left[f_\theta\left(\mathbf{x}, \epsilon^s\right), f_\theta(\mathbf{x}, \epsilon)\right]$$

Intuitively, AdD can be interpreted as deactivating the most contributing neurons and giving the "dead" neurons learning opportunities.

The authors find that adversarial dropout will induce a sparser structure of neural network than normal dropout.

VAdD can also be incorporated with VAT, which means both the input permutations and network permutations are obtained by adversarial training.

Similar to VAT, the authors also propose a fast approximation method for VAdD. Details can be found in the paper.

Algorithm 1: Finding Adversarial Dropout Condition

Input : ϵ^s is current sampled dropout mask
Input : δ is a hyper-parameter for the boundary
Input : $\mathbf{J}$ is the Jacobian vector
Input : H is the layer dimension.
Output: ϵ_{adv}

begin
  $\boldsymbol{z} \longleftarrow |\mathbf{J}|$ // absolute values of the Jacobian
  $\boldsymbol{i} \longleftarrow$ Arg Sort $\boldsymbol{z}$ as $z_{i_1} \leq ... \leq z_{i_H}$
  $\boldsymbol{\epsilon}^{adv} \longleftarrow \boldsymbol{\epsilon}^s$
  $d \longleftarrow 1$
  while $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}^{adv}\|_2 \leq \delta H$ *and* $d \leq H$ **do**
    if $\epsilon^{adv}_{i_d} = 0$ *and* $\mathbf{J}_{i_d} > 0$ **then**
      $\epsilon^{adv}_{i_d} \longleftarrow 1$
    else if $\epsilon^{adv}_{i_d} = 1$ *and* $\mathbf{J}_{i_d} < 0$ **then**
      $\epsilon^{adv}_{i_d} \longleftarrow 0$
    end
    $d \longleftarrow d + 1$
  end
end

2.5 SNTG

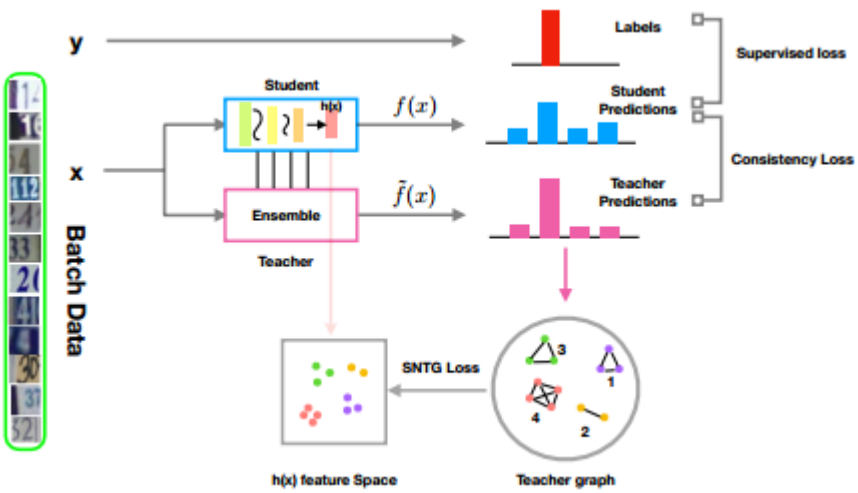
【**Paper**】Smooth Neighbors on Teacher Graphs for Semi-supervised Learning

【**Source**】CVPR 2018

This paper introduces an incremental component called SNTG to previous self-ensembling methods.

Previous methods only consider the perturbations around single data points, while ignoring the connections between data points, which can provide information about unlabeled data structure such as clusters or manifolds.

Motivated by this, SNTG introduces to **learn a teacher graph based on the targets generated by the teacher** (hard targets after *argmax*).



The teacher graph is built by connecting data points that belong to the same predicted class $\tilde{y}$:

$$W_{ij} = \begin{cases} 1 & \text{if } \tilde{y}_i = \tilde{y}_j \\ 0 & \text{if } \tilde{y}_i \neq \tilde{y}_j \end{cases}$$

Then a **graph constraint is added to the features of student network** to ensure that neighbor points have consistent features.

$$\ell_G = \begin{cases} \|h(x_i) - h(x_j)\|^2 & \text{if } W_{ij} = 1 \\ \max(0, m - \|h(x_i) - h(x_j)\|)^2 & \text{if } W_{ij} = 0 \end{cases}$$

Compared to previous methods, SNTG explores more information in the teacher network and thus improves the target quality to guide the student network to learn better.

In order to accelerate training, the authors propose a doubly stochastic sampling approximation method. This method samples data points in a mini-batch to build a sub-graph instead of using all the data points in that mini-batch.

Algorithm 1 Mini-batch training of SNTG for SSL

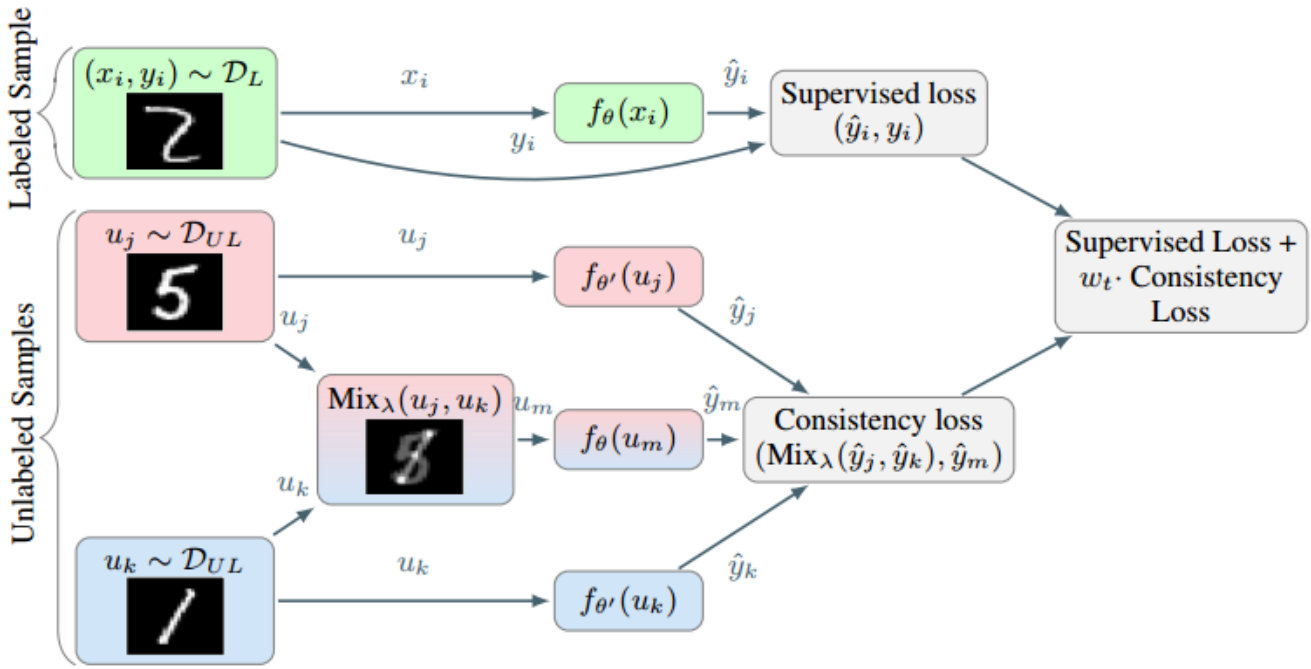
Require: x_i = training inputs, y_i for labeled inputs in $\mathcal{L}$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = neural network with parameters θ
for t in [1, numepochs] **do**
  for each minibatch B **do**
    $f_i \leftarrow f_\theta(x_{i\in B})$ evaluate network outputs
    $\tilde{f}_i \leftarrow \tilde{f}(x_{i\in B})$ given by the teacher model
    for (x_i, x_j) in a minibatch pairs S from B **do**
      Compute W_{ij} according to Eq. (6)
    end for
    loss $\leftarrow -\frac{1}{|B|}\sum_{i\in(B\cap\mathcal{L})} \log[f_i]_{y_i}$
    $+w(t)\Big[\lambda_1 \frac{1}{|B|}\sum_{i\in B} d(\tilde{f}_i, f_i)$
      $+\lambda_2 \frac{1}{|S|}\sum_{i,j\in S} \ell_G(h(x_i), h(x_j), W_{ij})\Big]$
    update θ using optimizers, *e.g.*, Adam [23]
  end for
end for
return θ

2.6 ICT

【**Paper**】Interpolation Consistency Training for Semi-Supervised Learning

【**Source**】arxiv 2019

ICT incorporates the idea of Mixup into self-ensemble methods. ICT encourages consistent predictions $f(\alpha u_1 + (1-\alpha)u_2) = \alpha f(u_1) + (1-\alpha) f(u_2)$ at interpolations $\alpha u_1 + (1-\alpha)u_2$ of unlabeled points u_1 and u_2.



ICT also uses a mean teacher $f_{\theta'}$, where θ' is a moving average of θ.

Mixup operation can push the decision boundary far away from the class boundaries by enforcing the prediction model to change linearly in between samples.

Algorithm 1 The Interpolation Consistency Training (ICT) Algorithm

Require: $f_\theta(x)$: neural network with trainable parameters θ

Require: $f_{\theta'}(x)$ mean teacher with θ' equal to moving average of θ

Require: $\mathcal{D}_L(x, y)$: collection of the labeled samples

Require: $\mathcal{D}_{UL}(x)$: collection of the unlabeled samples

Require: α: rate of moving average

Require: $w(t)$: ramp function for increasing the importance of consistency regularization

Require: T: total number of iterations

Require: Q: random distribution on [0,1]

Require: $\text{Mix}_\lambda(a, b) = \lambda a + (1 - \lambda)b$.

for $t = 1, \ldots, T$ **do**

Sample $\{(x_i, y_i)\}_{i=1}^B \sim \mathcal{D}_L(x, y)$ ▷ Sample labeled minibatch

$L_S = \text{CrossEntropy}(\{(f_\theta(x_i), y_i)\}_{i=1}^B)$ ▷ Supervised loss (cross-entropy)

Sample $\{u_j\}_{j=1}^U, \{u_k\}_{k=1}^U \sim \mathcal{D}_{UL}(x)$ ▷ Sample two unlabeled examples

$\{\hat{y}_j\}_{j=1}^U = \{f_{\theta'}(u_j)\}_{j=1}^U, \{\hat{y}_k\}_{k=1}^U = \{f_{\theta'}(u_k)\}_{k=1}^U$ ▷ Compute fake labels

Sample $\lambda \sim Q$ ▷ sample an interpolation coefficient

$(u_m = \text{Mix}_\lambda(u_j, u_k), \hat{y}_m = \text{Mix}_\lambda(\hat{y}_j, \hat{y}_k))$ ▷ Compute interpolation

$L_{US} = \text{ConsistencyLoss}(\{(f_\theta(u_m), \hat{y}_m)\}_{m=1}^U)$ ▷ e.g., mean squared error

$L = L_S + w(t) \cdot L_{US}$ ▷ Total Loss

$g_\theta \leftarrow \nabla_\theta L$ ▷ Compute Gradients

$\theta' = \alpha\theta' + (1 - \alpha)\theta$ ▷ Update moving average of parameters

$\theta \leftarrow \text{Step}(\theta, g_\theta)$ ▷ e.g. SGD, Adam

end for

return θ

Experimental results:

Table 1: Error rates (%) on CIFAR-10 using CNN-13 architecture. We ran three trials for ICT.

Model	1000 labeled 50000 unlabeled	2000 labeled 50000 unlabeled	4000 labeled 50000 unlabeled
Supervised	39.95 ± 0.75	31.16 ± 0.66	21.75 ± 0.46
Supervised (Mixup)	36.48 ± 0.15	26.24 ± 0.46	19.67 ± 0.16
Supervised (Manifold Mixup)	34.58 ± 0.37	25.12 ± 0.52	18.59 ± 0.18
Π model (Laine & Aila, 2016)	31.65 ± 1.20	17.57 ± 0.44	12.36 ± 0.31
TempEns (Laine & Aila, 2016)	23.31 ± 1.01	15.64 ± 0.39	12.16 ± 0.24
MT (Tarvainen & Valpola, 2017)	21.55 ± 1.48	15.73 ± 0.31	12.31 ± 0.28
VAT (Miyato et al., 2018)	–	–	$11.36 \pm$ NA
VAT+Ent (Miyato et al., 2018)	–	–	$10.55 \pm$ NA
VAdD (Park et al., 2018)	–	–	11.32 ± 0.11
SNTG (Luo et al., 2018)	18.41 ± 0.52	13.64 ± 0.32	10.93 ± 0.14
MT+ Fast SWA (Athiwaratkun et al., 2019)	$15.58 \pm$ NA	$11.02 \pm$ NA	$9.05 \pm$ NA
ICT	$\mathbf{15.48 \pm 0.78}$	$\mathbf{9.26 \pm 0.09}$	$\mathbf{7.29 \pm 0.02}$

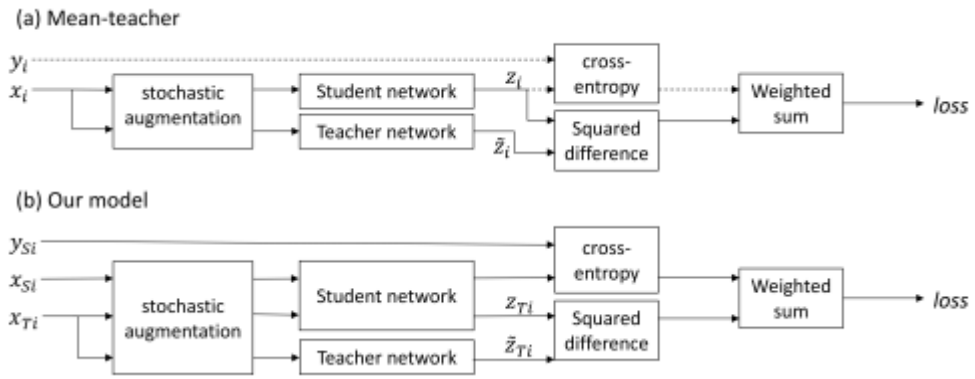
3. Application

3.1 MT for DA

【**Paper**】Self-ensembling for visual domain adaptation

【**Source**】ICLR 2018

The authors modify the mean teacher model for domain adaptation scenarios.



The supervised classification loss is computed with source samples. The unsupervised consistency loss is computed with target samples.

The authors process source and target mini-batches separately per iteration so that batch normalization uses different statistics for each domain during training.

To achieve the best performance, the authors use various tricks, including confidence thresholding, data augmentation and class balance loss. More details can be found in the paper.