A Survey on Deep Semi-supervised Learning

Xiangli Y ang, Zixing Song, Irwin King, Fellow, IEEE, Zenglin Xu, Senior Member, IEEE

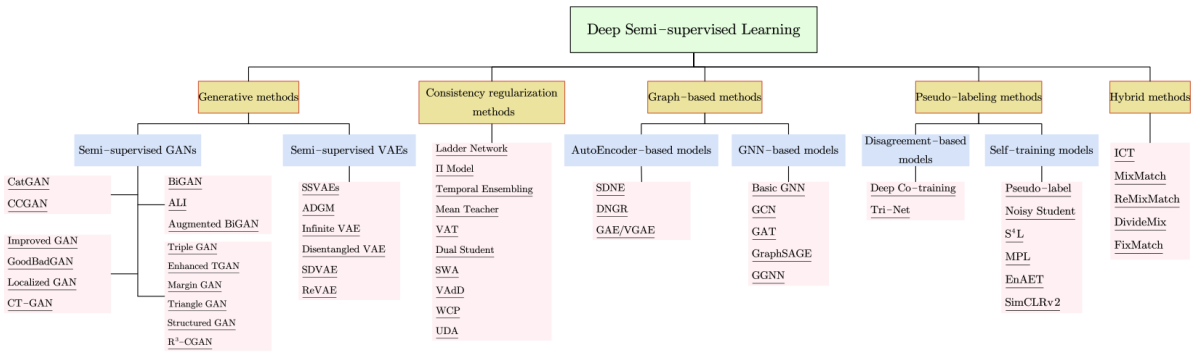
Introduction

Deep learning applications: pattern recognition, data mining, statistical learning, computer vision, natural language processing

SSL has been a hot research topic: Labeled samples are often difficult, expensive, or time-consuming to obtain, while the unlabeled data is usually abundant and can be easily or inexpensively obtained.

SSL: A learning paradigm associated with constructing models that use both labeled and unlabeled data, provide a way to explore the latent patterns from unlabeled examples.

SSL classification: classification, clustering, regression



The taxonomy of major deep semi-supervised learning methods based on loss function and model design

Background

SSL aims to solve the following optimization problem

$$\min_{\theta} \underbrace{\sum_{x\in X_L, y\in Y_L} \mathcal{L}_s(x,y,\theta)}_{\text{supervised loss}} + \alpha \underbrace{\sum_{x\in X_U} \mathcal{L}_u(x,\theta)}_{\text{unsupervised loss}} + \beta \underbrace{\sum_{x\in X} \mathcal{R}(x,\theta)}_{\text{regularization}},$$

Different choices of the unsupervised loss functions and regularization terms lead to different semisupervised models.

SSL can be classified into two settings. Transductive learning generalize over the unlabeled samples, while inductive learning supposes that the learned semisupervised classifier will be still applicable to new unseen data.

Assumptions for SSL

- **Self-training assumption.** When the hypothesis is satisfied, those high-confidence predictions are considered to be ground-truth.
- **Co-training assumption.** instance x has two conditionally independent views, and each view is sufficient for a classification task.
- **Generative model assumption.** Generally, it is assumed that data are generated from a mixture of distributions.
- **Cluster assumption.** If two points x_1 and x_2 are in the same cluster,they should belong to the same category.
- **Low-density separation.** The decision boundary should be in a low-density region, not through a high-density area.
- **Manifold assumption.** If two points x_1 and x_2 are located in a local neighborhood in the low-dimensional manifold, they have similar class labels. This assumption reflects the local smoothness of the decision boundary.

Related Learning Paradigms

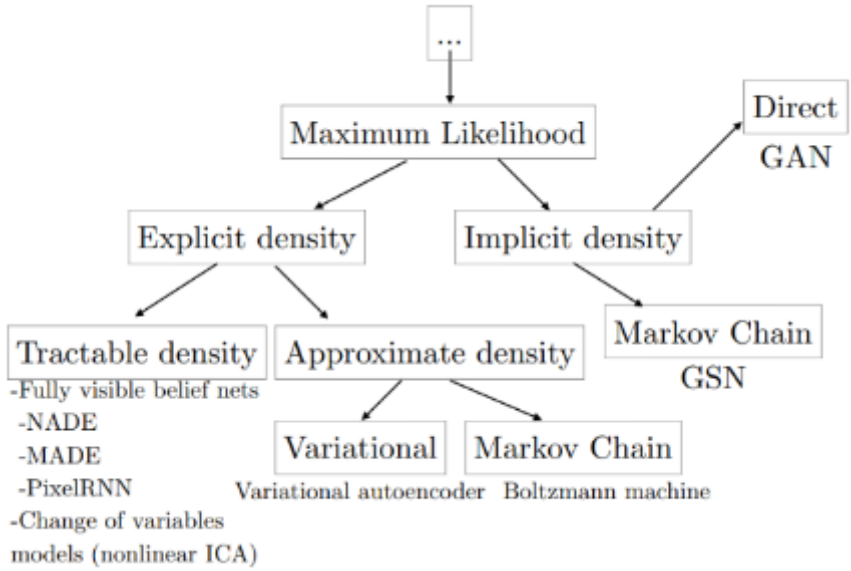
- **Transfer learning.** Transfer learning aims to apply knowledge from one or more source domains to a target domain in order to improve performance on the target task.
- **Weakly-supervised learning.** Weakly-supervised learning relaxes the data dependence that requires groundtruth labels to be given for a large amount of training data set in strong supervision.
  There are three types of weakly supervised data: Incomplete supervised data means only a subset of training data is labeled. Inexact supervised data suggests that the labels of training examples are coarse-grained. Inaccurate supervised data means that the given labels are not always groundtruth.
- **Positive and unlabeled learning.** PU learning is a variant of positive and negative binary classification, where the training data consists of positive samples and unlabeled samples.
- **Meta-learning.** Meta-learning also known as “learning to learn”, aims to learn new skills or adapt to new tasks rapidly with previous knowledge and a few training examples.
- **Self-supervised learning.** It can leverage input data as supervision and use the learned feature representations for many downstream tasks.

Generative mothods

Generative methods can learn the implicit features of data to better model data distributions. They model the real data distribution from the training dataset and then generate new data with

this distribution.

Discriminative model:$p(y|x)$
Generative model:$p(x, y)$



Semi-supervised GANs

A typical GAN belongs to unsupervised learning, it consists of **a generator G and a discriminator D**. The Generator generates fake samples of data and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples.As we can see, D and G play the following two-player minimax game with the value function V (G, D):

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

$p_{data}(x)$ = distribution of real data, joint probability distribution of each pixel of the image
$p(z)$ = distribution of generator
x = sample from $p_{data}(x)$, image
z = sample from $p(z)$, noise
D(x) = Discriminator network
G(z) = Generator network

How to use GANs for SSL
(a) re-using the features from the discriminator
(b) using GAN-generated samples to regularize a classifier
(c) learning an inference model
(d) using samples produced by a GAN as additional training data

A simple SSL approach is to combine supervised and unsupervised loss during training.

CATEGORICAL GENERATIVE ADVERSARIAL NETWORKS——CatGAN

Discriminator perspective. The requirements to the discriminator are that it should
(i) be certain of class assignment for samples from X

(ii) be uncertain of assignment for generated samples
(iii) use all classes equally

The most direct measure is the Shannon entropy *H*, which is defined as the expected value of the information carried by a sample from a given distribution.

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\Big[H\big[p(y\mid\mathbf{x},D)\big]\Big] = \frac{1}{N}\sum_{i=1}^{N} H\big[p(y\mid\mathbf{x}^i,D)\big]$$
$$= \frac{1}{N}\sum_{i=1}^{N} -\sum_{k=1}^{K} p(y=k\mid\mathbf{x}^i,D)\log p(y=k\mid\mathbf{x}^i,D).$$

$$\mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y\mid D(\mathbf{z}),D)\big]\Big] \approx \frac{1}{M}\sum_{i=1}^{M} H\big[p(y\mid G(\mathbf{z}^i),D)\big],\ \text{with}\ \mathbf{z}^i\sim P(\mathbf{z})$$

$$H_{\mathcal{X}}\Big[p(y\mid D)\Big] = H\Big[\frac{1}{N}\sum_{i=1}^{N} p(y\mid\mathbf{x}^i,D)\Big]$$

Generator perspective. The requirements to the generator are that it should
(i) generate samples with highly certain class assignments
(ii) equally distribute samples across all K classes

$$H_G\Big[p(y\mid D)\Big] \approx H\Big[\frac{1}{M}\sum_{i=1}^{M} p(y\mid G(\mathbf{z}^i),D)\Big],\ \text{with}\ \mathbf{z}^i\sim P(\mathbf{z}).$$

we can define the CatGAN objective for the discriminator and for the generator

$$\mathcal{L}_D = \max_D\ H_{\mathcal{X}}\Big[p(y\mid D)\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\Big[H\big[p(y\mid\mathbf{x},D)\big]\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y\mid G(\mathbf{z}),D)\big]\Big],$$
$$\mathcal{L}_G = \min_G -H_G\Big[p\big(y\mid D\big)\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y\mid G(\mathbf{z}),D)\big]\Big],$$

Now consider adapting the formulation to the **semi-supervised setting**.
Let $X^L = (x^1,y^1),(x^L,y^L)$ be a set of L labeled examples, with label vectors $y^i \in R^K$ in one-hot encoding.These additional examples can be incorporated into the objectives by calculating a cross-entropy term between the predicted conditional distribution $p(y|x,D)$ and the true label distribution of examples from X^L.

$$CE\big[\mathbf{y},p(y\mid\mathbf{x},D)\big] = -\sum_{i=1}^{K} y_i\log p(y=y_i\mid\mathbf{x},D)$$

The semi-supervised CatGAN problem is then given through the following two objectives

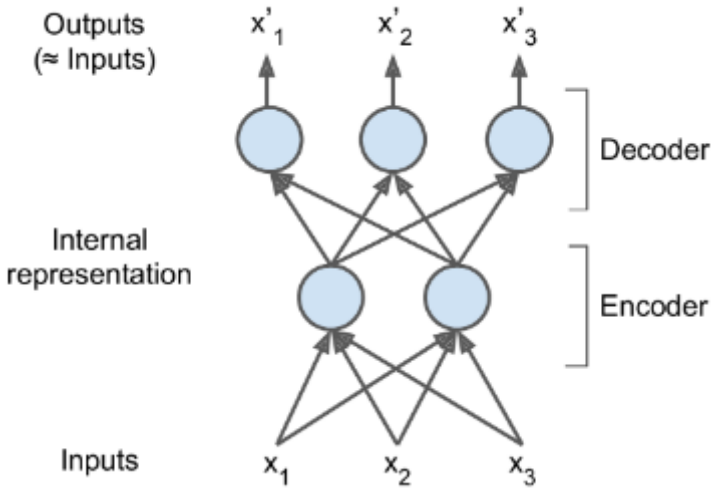
$$\mathcal{L}_D^L = \max_D H_{\mathcal{X}}\Big[p(y\mid D)\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\Big[H\big[p(y\mid\mathbf{x},D)\big]\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y\mid G(\mathbf{z}),D)\big]\Big]$$
$$+\lambda\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{X}^L}\Big[CE\big[\mathbf{y},p(y\mid\mathbf{x},D)\big]\Big],$$

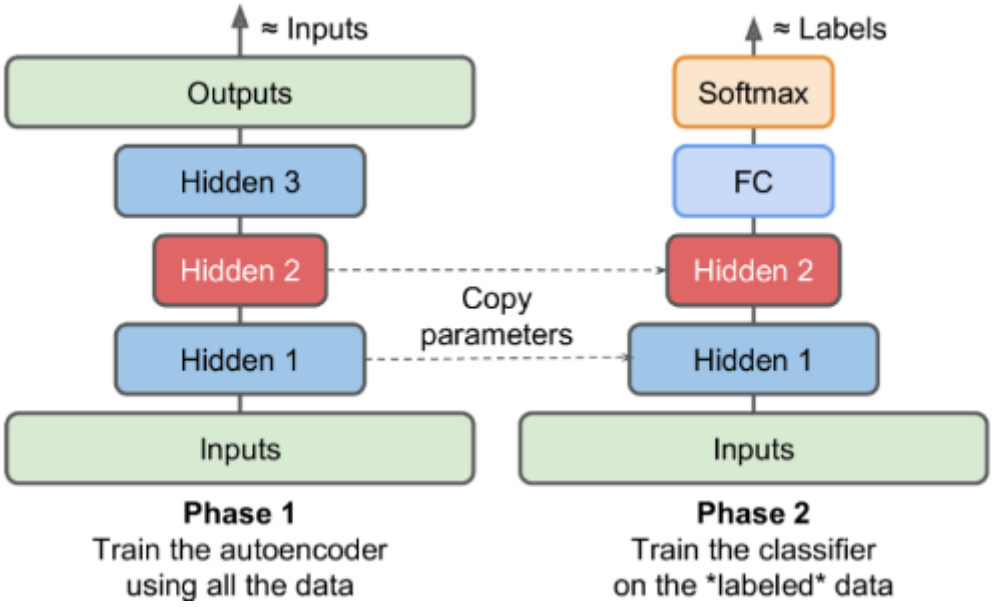
$$\mathcal{L}_G = \min_G -H_G\Big[p\big(y \mid D\big)\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\Big[p\big(y \mid G(\mathbf{z}), D\big)\Big]\Big]$$

Semi-supervised VAE

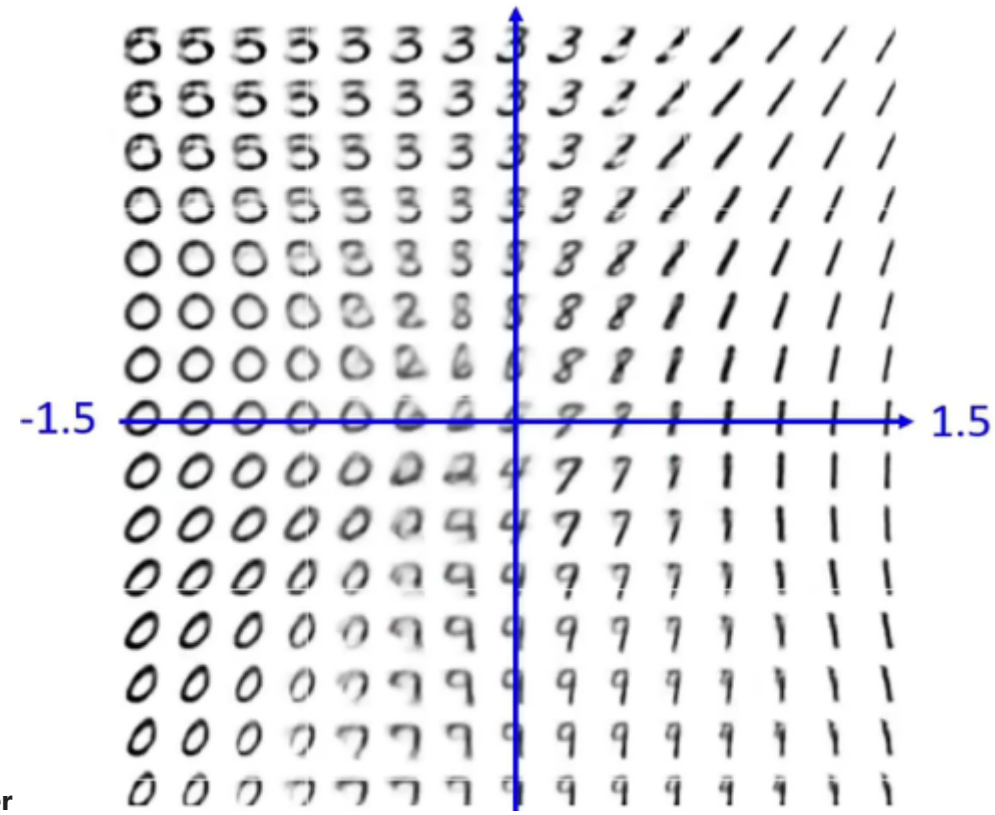
AutoEncoder

The purpose of the autoencoder is to perform unsupervised feature learning, using unlabeled data to find an efficient low-dimensional feature extractor.





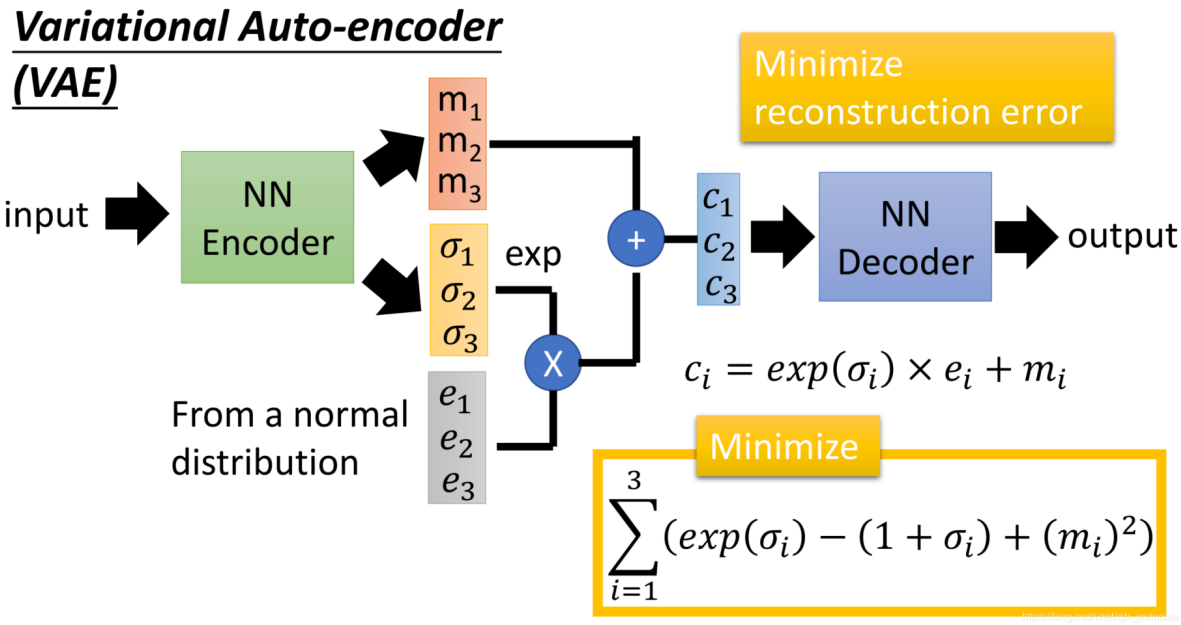
Encoder



Decoder

Variational AutoEncoder(VAE)

Variational AutoEncoders(VAEs)are flexible models which combine deep autoencoders with generative latent-variable models. The generative model captures representations of the distributions rather than the observations of the dataset.



The greater the probability that the original image in the training set appears in the output image set, the more similar the output image is to the original image.

Only the decoder is considered：

$$\max \quad L = \sum_{x} \log P(x)$$

$$while \quad P(x) = \int_{z} P(z)P(x|z)dz$$

Considering the encoder:

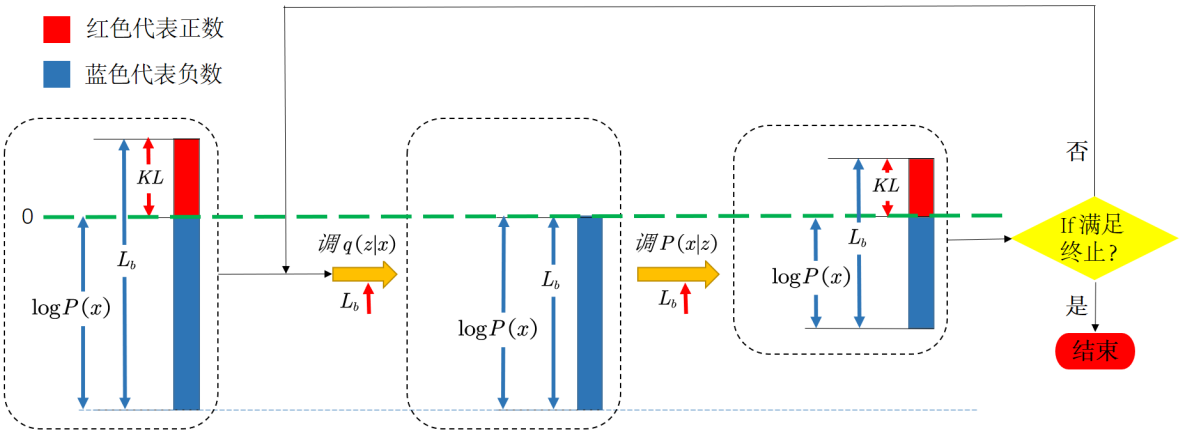
$$\begin{cases} \log P(x) & = \int_z q(z|x) \log P(x) dz \\ & = \int_z q(z|x) \log(\frac{P(z,x)}{P(z|x)}) dz = \int_z q(z|x) \log(\frac{P(z,x)}{q(z|x)} \frac{q(z|x)}{P(z|x)}) dz \\ & = \underbrace{\int_z q(z|x) \log(\frac{P(z,x)}{q(z|x)}) dz}_{lower\ bound\ L_b} + \underbrace{\int_z q(z|x) \log(\frac{q(z|x)}{P(z|x)}) dz}_{KL(q(z|x)||P(z|x) \geq 0} \\ & \geq \underbrace{\int_z q(z|x) \log(\frac{P(x|z)P(z)}{q(z|x)}) dz}_{lower\ bound\ L_b} \end{cases}$$

$$\begin{cases} \log P(x) = \{L_b + KL(q(z|x)||P(z|x))\} \leq 0 \\ KL(q(z|x)||P(z|x)) \geq 0 \\ L_b = \int_z q(z|x) \log(\frac{P(x|z)P(z)}{q(z|x)}) dz \leq 0 \end{cases}$$

Decompose L_b:

$$\begin{cases} L_b & = \int q(z|x) \log(\frac{P(z,x)}{q(z|x)}) dz = \int q(z|x) \log(\frac{P(x|z)P(z)}{q(z|x)}) dz \\ & = \underbrace{\int q(z|x) \log(\frac{P(z)}{q(z|x)}) dz}_{-KL(q(z|x)||P(z))} + \int q(z|x) \log P(x|z) dz \end{cases}$$

The training process of VAE



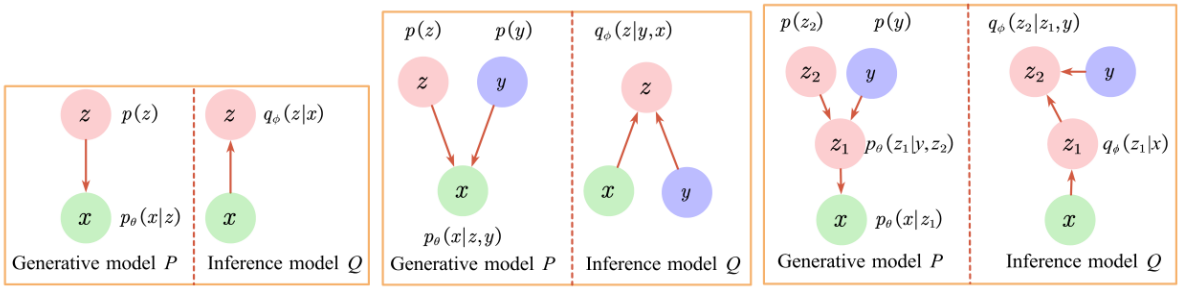
VAE训练流程图

STEP1: Adjust the encoder($q(z|x)$) to increase L_b.
The second term in the L_b decomposition formula is not easy to quantify training, so it is adjusted by reducing $KL(q(z|x)||P(z))$, that is, making $P(z)$ close to $q(z|x)$.Generally, $P(z)$ is a standard normal distribution.

STEP2: Adjust the decoder($P(x|z)$) to increase L_b.
The first term in the L_b decomposition formula has nothing to do with $P(x|z)$, so we can maximize the second term using Monte Carlo.

$$\begin{cases} \max \quad loss \quad = -loss_1 + loss_2 = -KL(q(z|x)||P(z)) + \int q(z|x)\log P(x|z)dz \\ \\ \qquad\qquad \simeq \sum_{i=1}^{3}(exp(\sigma_i) - (1+\sigma_i) + (m_i)^2) + \frac{1}{L}\sum_{l=1}^{L}\log P(x^{(i)}|z^{(i,l)}) \end{cases}$$

Semi-supervised learning with deep generative models——SSVAE



(1) Latent feature discriminative model (M1)　(2) Generative semi-supervised model (M2)　(3) Stacked generative semi-supervised model (M1 + M2)

(1)A clustering of related observations in a latent feature space that allows for accurate classification, even with a limited number of labels.Approximate samples from the posterior distribution over the latent variables p(z|x) are used as features to train a classifier that predicts class labels y.

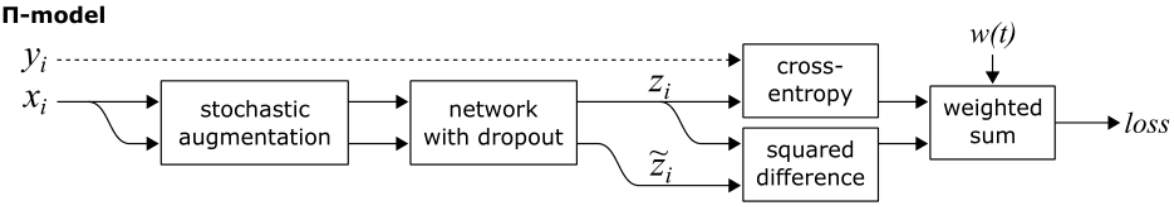
(2)The class labels y are treated as latent variables if no class label is available and z are additional latent variables.Since most labels y are unobserved, we integrate over the class of any unlabelled data during the inference process, thus performing classification as inference.The inferred posterior distribution $p_\theta(y|x)$ can predict any missing labels.

(3)Using the generative model M1 to learn the new latent representation z_1, and uses the embedding from z_1 instead of the raw data x to learn a generative semi-supervised model M2.

Consistency regularization

The main idea of Consistency Regularization is that for an input, even with small disturbances, its predictions should be consistent.In these methods, a consistency regularization term is applied to the final loss function to specify the prior constraints assumed by researchers.

Π Model



During training, we evaluate the network for each training input x_i twice, resulting in prediction vectors z_i and $\widetilde{z}_i$.

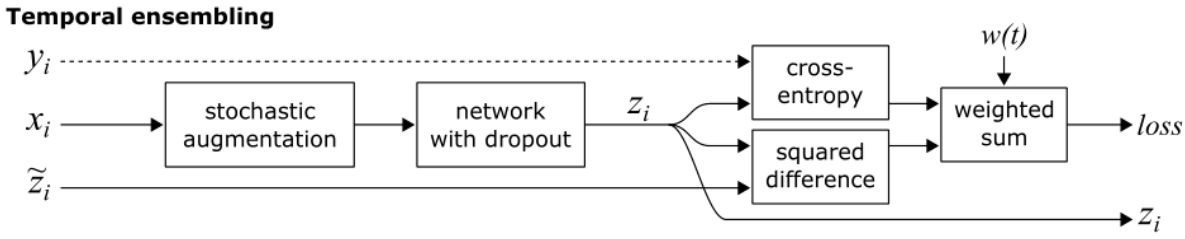
Our loss function consists of two components. The first component is the standard cross-entropy loss, evaluated for labeled inputs only. The second component, evaluated for all inputs, penalizes different predictions for the same training input x_i by taking the mean square difference between the prediction vectors z_i and $\widetilde{z}_i$.

Time-dependent weighting function $w(t)$ used to control the weight of consistency loss. Since the network parameters are random at the beginning of training, the loss of the unsupervised part will be particularly large, and a large $w(t)$ will affect the normal training of the network. Therefore, $w(t)$ increases slowly over time.

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

Temporal Ensembling



An obvious problem with the π-model is that each input requires two forward propagations to calculate the consistency loss, which may be inefficient.Temporal ensembling alleviates this by aggregating the predictions of multiple previous network evaluations into an ensemble prediction. It also lets us evaluate the network only once during training, gaining an approximate 2x speedup over the Π-model.

The main difference to the Π-model is that the network and augmentations are evaluated only once per input per epoch, and the target vectors $\tilde{z}$ for the unsupervised loss component are based on prior network evaluations instead of a second evaluation of the network.

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

Graph-based methods

The basic assumption in GSSL is that a graph can be extracted from the raw dataset where each node represents a training sample, and each edge denotes some similarity measurement of the node pair.

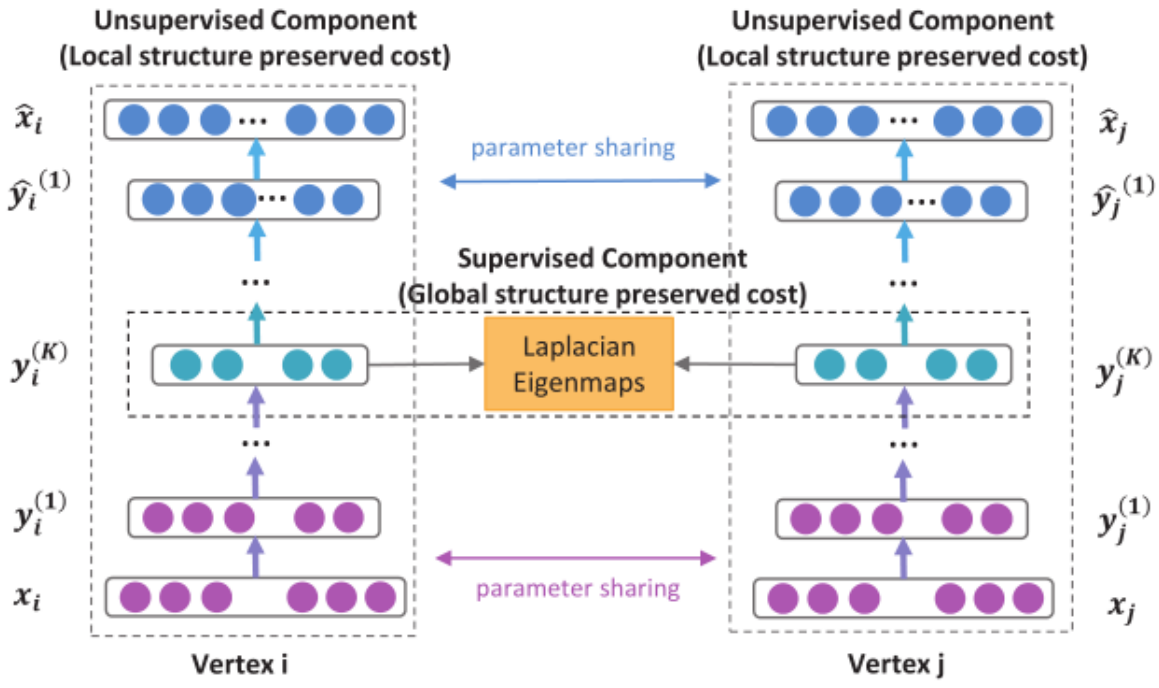
Structural deep network embedding (SDNE)

Defination 1 (Graph) A graph is denoted as *G = (V,E)*, where *V = {*$v_1, v_2, \ldots, v_n$*}* represents *n* vertexes and *E =* $\{e_{i,j}\}_{i,j=1}^{n}$ represents the edges. Each edge $e_{i,j}$ is associated with a weight $s_{i,j}$ ≥0. For v_i and v_j not linked by an edge, $s_{i,j} = 0$. Otherwise, for unweighted graph $s_{i,j} = 1$ and for weighted graph $s_{i,j} > 0$

Defination 2 (First-Order Proximity) The first-order proximity describes the pairwise proximity between vertexes. For any pair of vertexes, if $s_{i,j}$ > 0, there exists positive first-order proximity between v_i and v_j. Otherwise, the first-order proximity between v_i and v_j is 0.

Defination 3 (Second-Order Proximity) The second-order proximity between a pair of vertexes describes the proximity of the pair's neighborhood structure. Let $N_u = \{s_{u,1}, \ldots, s_{u,|V|}\}$ denote the first-order proximity between v_u and other vertexes. Then, second-order proximity is determined by the similarity of N_u and N_v.

Defination 4 (Network Embedding) Given a graph denoted as *G = (V,E)*, network embedding aims to learn a mapping function $f : v_i \rightarrow y_i \in R^d$, where $d << |V|$. The objective of the function is to make the similarity between y_i and y_j explicitly preserve the first-order and second-order proximity of v_i and v_j.



Given a network $G = (V, E)$, we can obtain its adjacency matrix S, which contains n instances

$s_1, \ldots, s_n$. For each instance $s_i = \{s_{i,j}\}_{j=1}^n, s_{i,j} > 0$ if and only if there exists a link between v_i and v_j. Therefore, s_i describes the neighborhood structure of the vertex v_i and S provides the information of the neighborhood structure of each vertex.

Considering our case that if we use the adjacency matrix S as the input to the autoencoder, i.e. $x_i = s_i$, since each instance s_i characterizes the neighborhood structure of the vertex v_i, the reconstruction process will make the vertexes which have similar neighborhood structures have similar latent representations.

In the networks, we can observe some links but simultaneously many legitimate links are not observed, which means that the links between vertexes do indicate their similarity but no links do not necessarily indicate their dissimilarity. Moreover, due to the sparsity of networks, the number of non-zero elements in S is far less than that of zero elements. Then if we directly use S as the input to the traditional autoencoder, it is more prone to reconstruct the zero elements in S.

$$\begin{aligned}\mathcal{L}_{2nd} &= \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b_i}\|_2^2 \\ &= \|(\hat{X} - X) \odot B\|_F^2\end{aligned}$$

$b_i = \{b_{i,j}\}_{j=1}^n$. If $s_{i,j} = 0, b_{i,j} = 1$, else $b_{i,j} = \beta > 1$. Now by using the revised deep autoencoder with the adjacency matrix S as input, the vertexes which have similar neighborhood structure will be mapped near in the representations space, guaranteed by the reconstruction criterion. In other words, the unsupervised component of our model can preserve the global network structure by reconstructing the second-order proximity between vertexes.

It is not only necessary to preserve the global network structure, but also essential to capture the local structure. We use the first- order proximity to denote the local network structure. The first-order proximity can be regarded as the supervised information to constrain the similarity of the latent representations of a pair of vertexes.

$$\begin{aligned}\mathcal{L}_{1st} &= \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 \\ &= \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2\end{aligned}$$

The objective function borrows the idea of Laplacian Eigenmaps, which incurs a penalty when similar vertexes are mapped far away in the embedding space.

$$\begin{aligned}\mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg} \\ &= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu\mathcal{L}_{reg}\end{aligned}$$

where L_{reg} is an L2-norm regularizer term to prevent overfitting, which is defined as follows:

$$\mathcal{L}_{reg} = \frac{1}{2}\sum_{k=1}^{K}(\|W^{(k)}\|_F^2 + \|\hat{W}^{(k)}\|_F^2)$$

Pseudo-labeling methods

The pseudo-labeling methods differ from the consistency regularization methods in that the consistency regularization methods usually rely on consistency constraint of rich data transformations. In contrast, pseudo-labeling methods rely on the high confidence of pseudo-labels, which can be added to the training data set as labeled data. There are two main patterns, one is to improve the performance of the whole framework based on the disagreement of views or multiple networks, and the other is self-training, in particular, the success of self-supervised learning in unsupervised domain makes some self-training self-supervised methods realized.

Pseudo-label

In this article we propose the simpler way of training neural network in a semi-supervised fashion. Basically, the proposed network is trained in a supervised fashion with labeled and unlabeled data simultaneously.For unlabeled data, Pseudo-Labels, just picking up the class which has the maximum predicted probability every weights update, are used as if they were true labels.

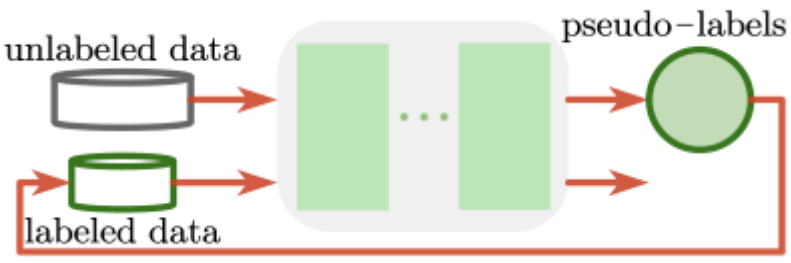
Step1: Given labeled data and unlabeled data

Step2: Train the model with labeled data

Step3: Use the trained model to predict unlabeled data and get pseudo-label

Step4: Take part of the data from the unlabeled data and add it to the labeled data set. Repeat Step2

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



Hybrid methods

Hybrid methods combine ideas from the above-mentioned methods such as pseudo-label, consistency regularization and entropy minimization for performance improvement.

MixMatch

Given a batch *X* of labeled examples with one-hot targets (representing one of L possible labels) and **an equally-sized batch *U* of unlabeled examples**, MixMatch produces a processed batch of augmented labeled examples *X'* and a batch of augmented unlabeled examples with "guessed" labels *U'*.*U'* and *X'* are then used in computing separate labeled and unlabeled loss terms.

Data Augmentation

As is typical in many SSL methods, we use data augmentation both on labeled and unlabeled data. For each x_b in the batch of labeled data *X* , we generate a transformed version $\widehat{x}_b$ = Augment(x_b). For each u_b in the batch of unlabeled data *U*, we generate *K* augmentations $\widehat{x}_{b,k}$ = Augment(x_b), k ∈ (1, . . . , K). We use these individual augmentations to generate a "guessed label" q_b for each u_b.

Label Guessing

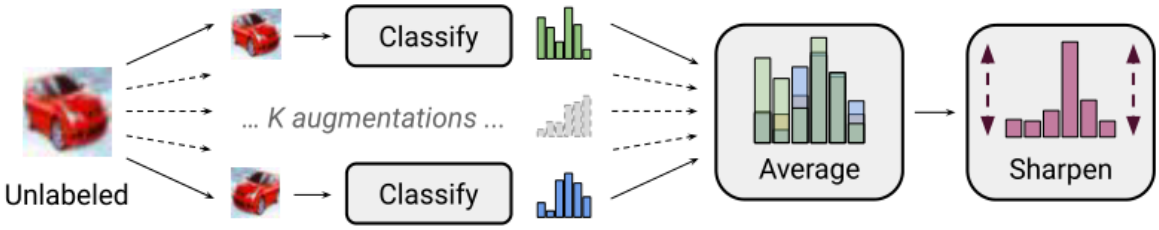
For each unlabeled example in *U*, MixMatch produces a "guess" for the example's label using the model's predictions. This guess is later used in the unsupervised loss term. To do so, we compute the average of the model's predicted class distributions across all the *K* augmentations of u_b by

$$\bar{q}_b = \frac{1}{K} \sum_{k=1}^{K} \mathrm{p}_{\mathrm{model}}(y \mid \hat{u}_{b,k}; \theta)$$

Sharpening

Given the average prediction over augmentations $\overline{q_b}$, we apply a sharpening function to reduce the entropy of the label distribution.

$$\mathrm{Sharpen}(p, T)_i := p_i^{\frac{1}{T}} \Big/ \sum_{j=1}^{L} p_j^{\frac{1}{T}}$$



MixUp

We mix both labeled examples and unlabeled examples with label guesses.For a pair of two examples with their corresponding labels probabilities $(x_1, p_1), (x_2, p_2)$ we compute (x', p') by

$$
\begin{aligned}
\lambda &\sim \mathrm{Beta}(\alpha, \alpha) \\
\lambda' &= \max(\lambda, 1-\lambda) \\
x' &= \lambda' x_1 + (1-\lambda') x_2 \\
p' &= \lambda' p_1 + (1-\lambda') p_2
\end{aligned}
$$

To apply MixUp, we first collect all augmented labeled examples with their labels and all unlabeled examples with their guessed labels into

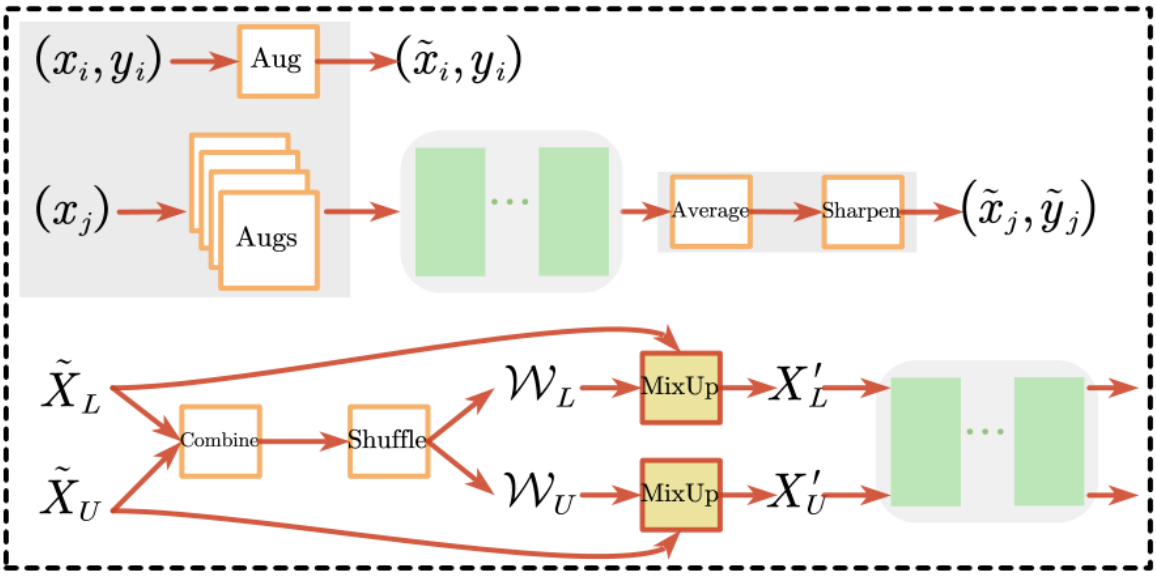
$$
\begin{aligned}
\hat{\mathcal{X}} &= \big((\hat{x}_b, p_b); b \in (1, \ldots, B)\big) \\
\hat{\mathcal{U}} &= \big((\hat{u}_{b,k}, q_b); b \in (1, \ldots, B), k \in (1, \ldots, K)\big)
\end{aligned}
$$

Then, we combine these collections and shuffle the result to form *W* which will serve as a data source for MixUp.

$$
\begin{aligned}
&\mathcal{W} = \mathrm{Shuffle}\big(\mathrm{Concat}(\hat{\mathcal{X}}, \hat{\mathcal{U}})\big) \quad \textit{// Combine and shuffle labeled and unlabeled data} \\
&\mathcal{X}' = \big(\mathrm{MixUp}(\hat{\mathcal{X}}_i, \mathcal{W}_i); i \in (1, \ldots, |\hat{\mathcal{X}}|)\big) \quad \textit{// Apply MixUp to labeled data and entries from } \mathcal{W} \\
&\mathcal{U}' = \big(\mathrm{MixUp}(\hat{\mathcal{U}}_i, \mathcal{W}_{i+|\hat{\mathcal{X}}|}); i \in (1, \ldots, |\hat{\mathcal{U}}|)\big) \quad \textit{// Apply MixUp to unlabeled data and the rest of } \mathcal{W}
\end{aligned}
$$

Loss Function

$$
\begin{aligned}
\mathcal{X}', \mathcal{U}' &= \mathrm{MixMatch}(\mathcal{X}, \mathcal{U}, T, K, \alpha) \\
\mathcal{L}_{\mathcal{X}} &= \frac{1}{|\mathcal{X}'|} \sum_{x,p \in \mathcal{X}'} \mathrm{H}(p, \mathrm{p}_{\mathrm{model}}(y \mid x; \theta)) \\
\mathcal{L}_{\mathcal{U}} &= \frac{1}{L|\mathcal{U}'|} \sum_{u,q \in \mathcal{U}'} \|q - \mathrm{p}_{\mathrm{model}}(y \mid u; \theta)\|_2^2 \\
\mathcal{L} &= \mathcal{L}_{\mathcal{X}} + \lambda_{\mathcal{U}} \mathcal{L}_{\mathcal{U}}
\end{aligned}
$$



Algorithm 1 MixMatch takes a batch of labeled data $\mathcal{X}$ and a batch of unlabeled data $\mathcal{U}$ and produces a collection $\mathcal{X}'$ (resp. $\mathcal{U}'$) of processed labeled examples (resp. unlabeled with guessed labels).

Input: Batch of labeled examples and their one-hot labels $\mathcal{X} = \big((x_b, p_b); b \in (1, \dots, B)\big)$, batch of unlabeled examples $\mathcal{U} = \big(u_b; b \in (1, \dots, B)\big)$, sharpening temperature T, number of augmentations K, Beta distribution parameter α for MixUp.
for $b = 1$ **to** B **do**
  $\hat{x}_b = \text{Augment}(x_b)$ *// Apply data augmentation to x_b*
  for $k = 1$ **to** K **do**
    $\hat{u}_{b,k} = \text{Augment}(u_b)$ *// Apply k^{th} round of data augmentation to u_b*
  end for
  $\bar{q}_b = \frac{1}{K} \sum_k \text{p}_{\text{model}}(y \mid \hat{u}_{b,k}; \theta)$ *// Compute average predictions across all augmentations of u_b*
  $q_b = \text{Sharpen}(\bar{q}_b, T)$ *// Apply temperature sharpening to the average prediction (see eq. (7))*
end for
$\hat{\mathcal{X}} = \big((\hat{x}_b, p_b); b \in (1, \dots, B)\big)$ *// Augmented labeled examples and their labels*
$\hat{\mathcal{U}} = \big((\hat{u}_{b,k}, q_b); b \in (1, \dots, B), k \in (1, \dots, K)\big)$ *// Augmented unlabeled examples, guessed labels*
$\mathcal{W} = \text{Shuffle}\big(\text{Concat}(\hat{\mathcal{X}}, \hat{\mathcal{U}})\big)$ *// Combine and shuffle labeled and unlabeled data*
$\mathcal{X}' = \big(\text{MixUp}(\hat{\mathcal{X}}_i, \mathcal{W}_i); i \in (1, \dots, |\hat{\mathcal{X}}|)\big)$ *// Apply MixUp to labeled data and entries from $\mathcal{W}$*
$\mathcal{U}' = \big(\text{MixUp}(\hat{\mathcal{U}}_i, \mathcal{W}_{i+|\hat{\mathcal{X}}|}); i \in (1, \dots, |\hat{\mathcal{U}}|)\big)$ *// Apply MixUp to unlabeled data and the rest of $\mathcal{W}$*
return $\mathcal{X}', \mathcal{U}'$