Yandex

Domain Adaptation

November 15, 2018

Recap: “model” is never just “model”

“model” = model + data + task

Standard supervised learning setting

“model” = model + data + task

Standard supervised learning setting

$\boxed{\text{“model”}} = \text{model} + \text{data} + \text{task}$

- Model (CNN vs RNN, GNMT vs Transformer and so on) $\mathcal{H} = \{h_\theta : X \to Y\}$

Standard supervised learning setting

“model” = model + $\boxed{\text{data}}$ + task

- Model (CNN vs RNN, GNMT vs Transformer and so on) $\mathcal{H} = \{h_\theta : X \to Y\}$

- Data (source and target marginal distributions)
  source ~ training data
  target ~ testing data

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Standard supervised learning setting

"model" = model + data + task

- Model (CNN vs RNN, GNMT vs Transformer and so on) $\mathcal{H} = \{h_\theta : X \to Y\}$

- Data (source and target marginal distributions)
  source ~ training data
  target ~ testing data

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

- Task (loss function, risk)

$$h_\theta^* = \arg\min_\theta R^S(h_\theta) = \arg\min_\theta \mathop{\mathrm{E}}_{(x,y)\sim P_S} [L(h_\theta(x), y)]$$

Standard learning assumptions

Standard learning assumptions

- Sample points are IID

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \overset{iid}{\sim} P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \overset{iid}{\sim} P_T(x, y)$$

Standard learning assumptions

- Sample points are IID

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \overset{iid}{\sim} P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \overset{iid}{\sim} P_T(x, y)$$

- Same distributions for training and test

$$P_S(x, y) = P_T(x, y)$$

Standard learning assumptions

- Sample points are IID

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \overset{iid}{\sim} P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \overset{iid}{\sim} P_T(x, y)$$

- Same distributions for training and test

$$P_S(x, y) = P_T(x, y)$$

- Fixed distributions (they don't change with time)

Standard learning assumptions

- Sample points are IID

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \overset{iid}{\sim} P_S(x,y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \overset{iid}{\sim} P_T(x,y)$$

- Same distributions for training and test

$$P_S(x,y) = P_T(x,y)$$

- Fixed distributions (they don't change with time)

$$\epsilon_T = \underset{(x,y)\sim P_T}{\mathrm{E}}[L(h_\theta(x),y)]$$

$$\epsilon_T \leq \hat{\epsilon}_S + O(\frac{complexity(h)}{\sqrt{N}})$$

$$\hat{\epsilon}_S = \frac{1}{N}\sum_{i=1}^{N} L(h_\theta(x^S),y)$$

Standard learning assumptions

- Sample points are IID $\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \overset{iid}{\sim} P_S(x,y)$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \overset{iid}{\sim} P_T(x,y)$$

- Same distributions for training and test $P_S(x,y) = P_T(x,y)$

EXPECTATIONS

- Fixed distributions (they don't change with time)

Real-world problems

- Training sample is biased
- Noisy labels for training sample
- Sample points are not drawn IID
- Distributions drifts with time

REALITY

Domain shift:

$$P_S(x,y) \neq P_T(x,y)$$

Real-world examples

- Data from different sources
- Absence of good in-domain data
- Temporal (both short term and long term) changes of audience structure
- Synthetic training data

Real-world examples

- Data from different sources
- Absence of good in-domain data
- Temporal (both short term and long term) changes of audience structure
- Synthetic training data

Domain shift:

$$P_S(x,y) \neq P_T(x,y)$$

How to check whether there is a domain shift or not?

How to check whether there is a domain shift or not?

If domain distribution are different we can train classifier to discriminate them.

$$P_S(x, y) \neq P_T(x, y)$$

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \epsilon_S + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda$$

$$P_S(x, y) \neq P_T(x, y)$$

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\boxed{\epsilon_T} \leq \epsilon_S + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda$$

Expected target error

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \boxed{\epsilon_S} + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda$$

Expected source error

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \le \epsilon_S + \boxed{\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)} + \lambda$$

The divergence between source and target domain

$$d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) = 2 \sup_{(h,h')} \left| \mathop{\mathrm{E}}_{(x,y)\sim P_S} [h(x) \neq h'(x)] - \mathop{\mathrm{E}}_{(x,y)\sim P_T} [h(x) \neq h'(x)] \right|$$

H-divergence measures the worst case of the disagreement between a pair of hypothesis.
How much features discriminative for S and T

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \epsilon_S + \boxed{\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)} + \lambda$$

The divergence between
source and target domain

$$d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) \sim d_{\mathcal{A}} = 2(1 - 2\varepsilon)$$

As H-divergence approximation proxy A-distance (PAD) can be used.
ε is the generalization error of the domain classifier.

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \epsilon_S + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \boxed{\lambda}$$

Error of ideal joint hypothesis

$$\lambda = \min_h [\epsilon_S(h) + \epsilon_S(h)]$$

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \epsilon_S + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda$$

Generalization error under domain shift

Ben-David, 2010

Ben-David theory:

$$\epsilon_T \leq \epsilon_S + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda$$

Domain adaptation tend to minimize second and (third?) terms.

Domain adaptation

- Unsupervised DA:
  No labels for target domain

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{T} = \{x_i^T\}_{i=1}^M \sim P_T(x)$$

- (Semi)supervised DA:
  Target domain dataset is (partially) labeled

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Covariate shift:

$$P_S(x) \neq P_T(x)$$

$$P_S(y|x) = P_T(y|x)$$

Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

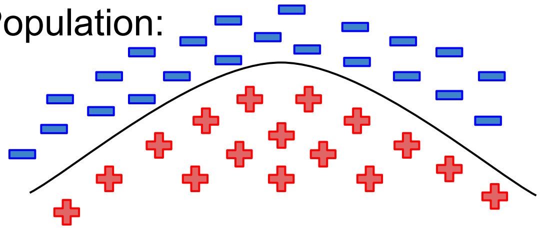
$$P_S(y|x) = P_T(y|x)$$

Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

$$P_S(y|x) = P_T(y|x)$$

Population:

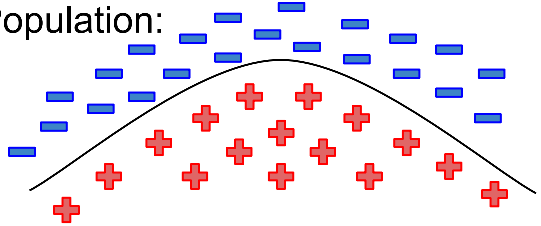


Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

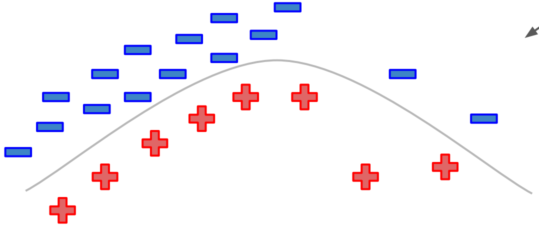
$$P_S(y|x) = P_T(y|x)$$

Population:



Biased sampling

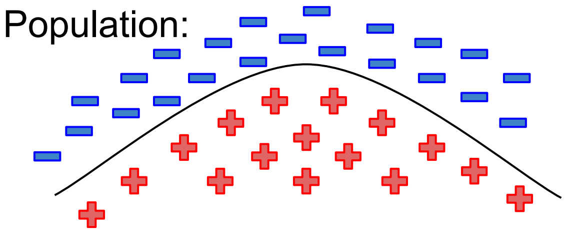
Source distribution:

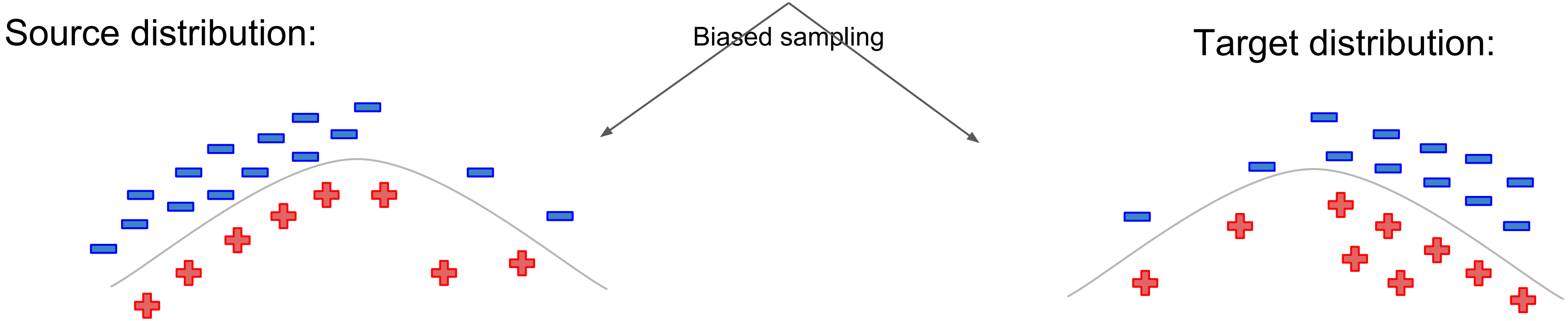


Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

$$P_S(y|x) = P_T(y|x)$$

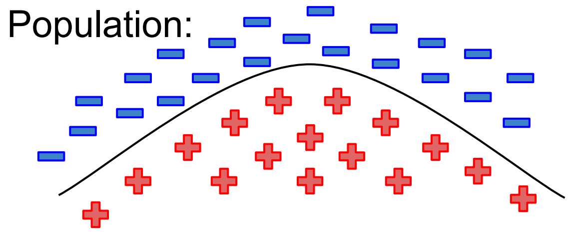


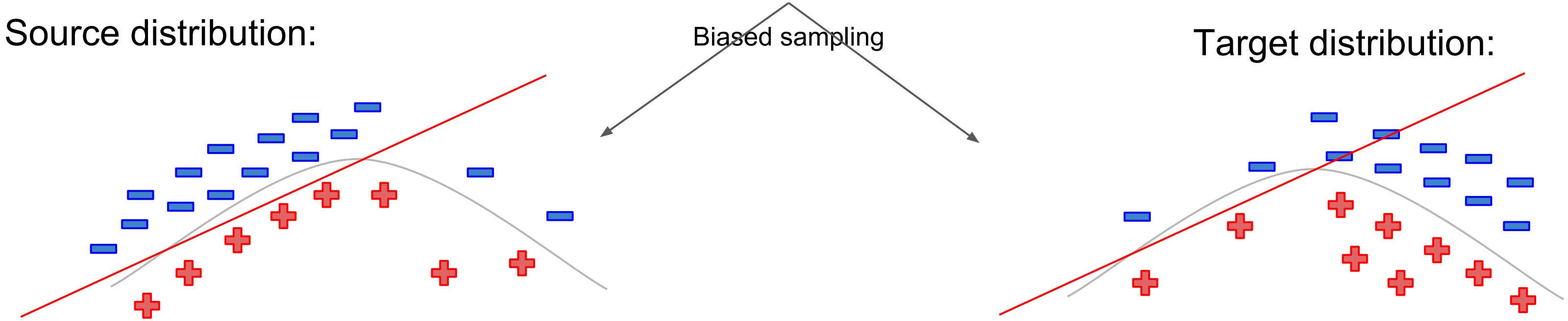


Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

$$P_S(y|x) = P_T(y|x)$$

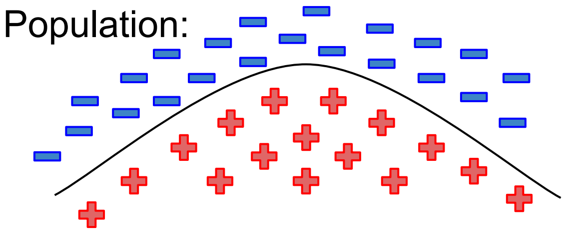


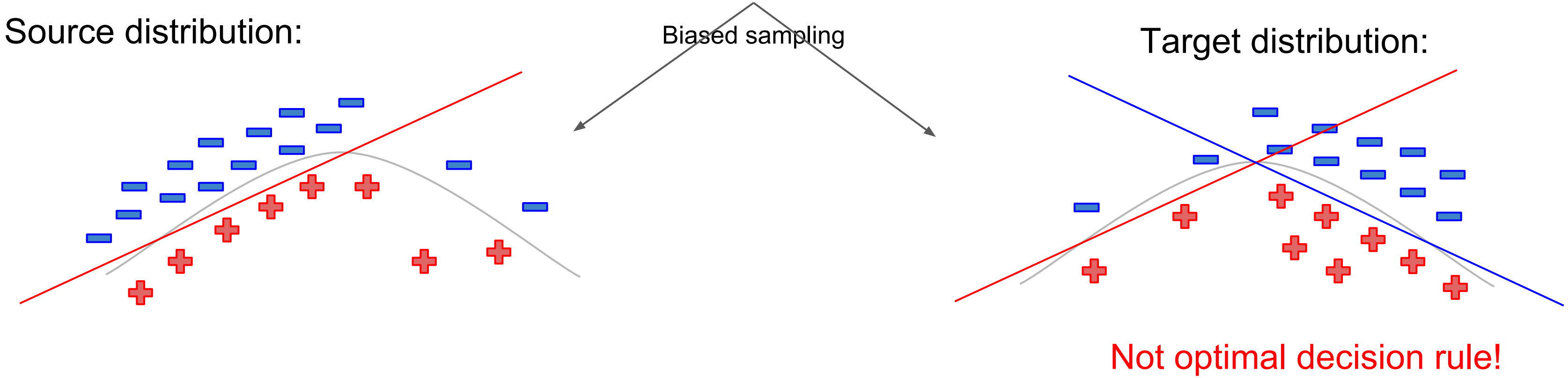


Why does a covariate shift make problems?

$$P_S(x) \neq P_T(x)$$

$$P_S(y|x) = P_T(y|x)$$





Unsupervised domain adaptation

Unsupervised domain adaptation

- Instance weighting/data selection

Unsupervised domain adaptation

- Instance weighting/data selection
- Proxy-labels methods

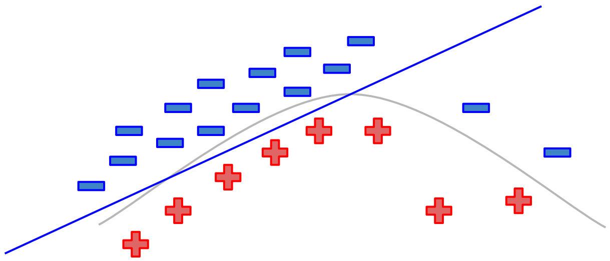
Unsupervised domain adaptation

- Instance weighting/data selection
- Proxy-labels methods
- Feature matching methods

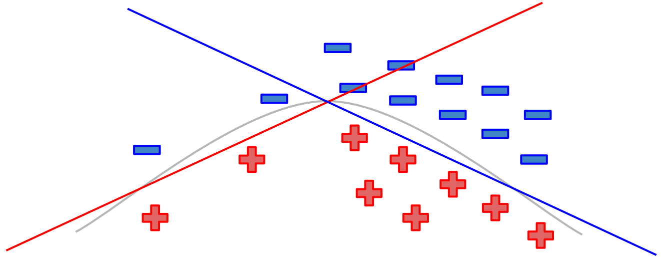
Instance weighting

$$h_\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} L(h_\theta(x), y)$$

Source distribution:



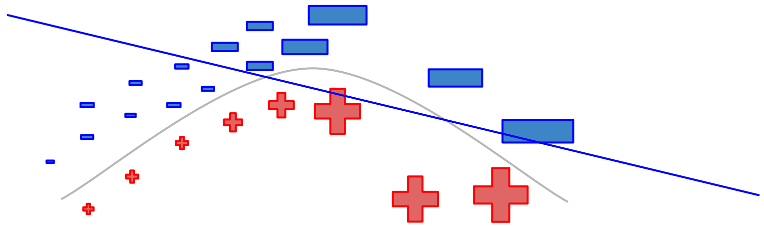
Target distribution:



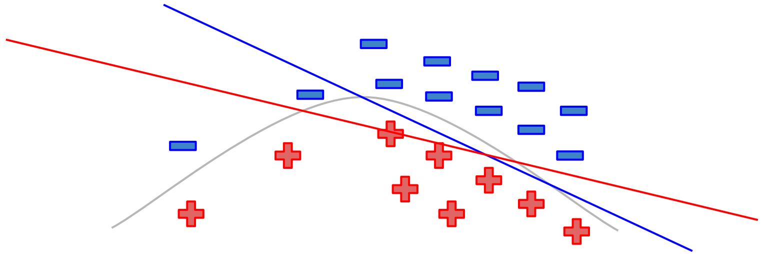
Not optimal decision rule!

Instance weighting

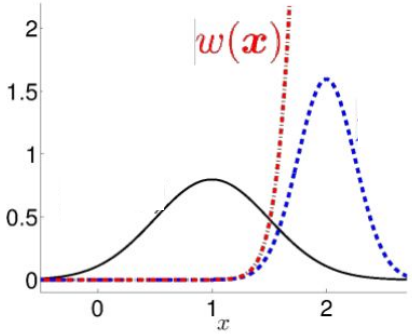
Source distribution:



Target distribution:



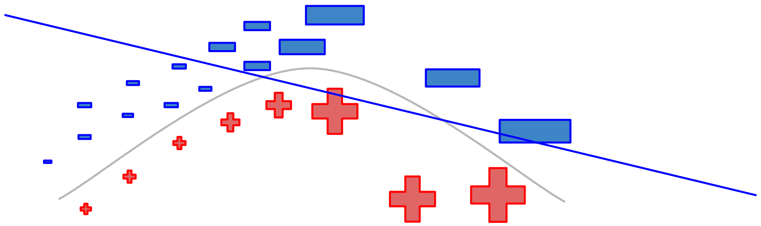
Instance weighting



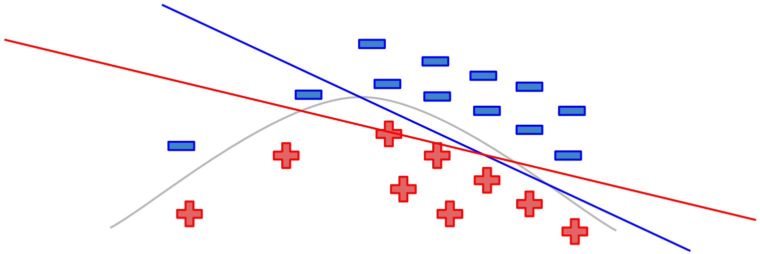
$$h_\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} w(x_i) L(h_\theta(x_i), y_i)$$

$$w(x) = \frac{p_T(x)}{p_S(x)}$$

Source distribution:



Target distribution:



Instance weighting: NLP applications

- Neural Machine Translation

  $x = (v_1, v_2, ..., v_k)$: Input sequence of tokens

  $P_T(x), P_S(x) = ?$

Instance weighting: NLP applications

- Neural Machine Translation

$x = (v_1, v_2, ..., v_k)$: Input sequence of tokens

$P_T(x), P_S(x)$ can be defined using source and target language models

Instance weighting: NLP applications

- Neural Machine Translation

$x = (v_1, v_2, ..., v_k)$: Input sequence of tokens

$P_T(x), P_S(x)$ can be defined using source and target language models

Weighting using language models (Wang et al. 2017):

$$w_i = \delta[H_S(x) - H_T(x)]$$

$$H = -\frac{1}{k} \log P(x)$$

(δ - min-max normalization)

Instance weighting: NLP applications

- Neural Machine Translation

$x = (v_1, v_2, ..., v_k)$: Input sequence of tokens

$P_T(x), P_S(x)$ can be defined using source and target language models

Weighting using domain classifier (Chen et al. 2017):

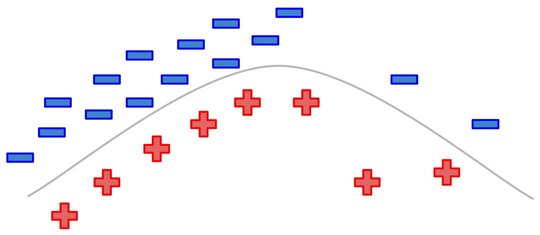
$$w_i = (1 + p_d(x_i))$$

$p_d(x)$ - probability of being from target

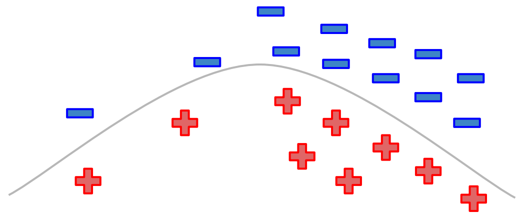
Data selection

Instead of weighting, we can train classifier and drop all observations that are too dissimilar from target domain.

Source distribution:



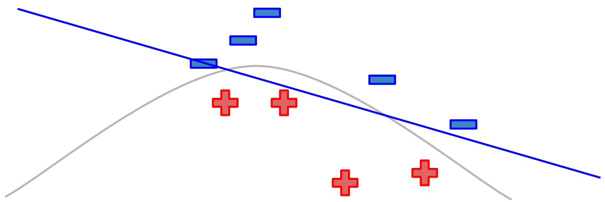
Target distribution:



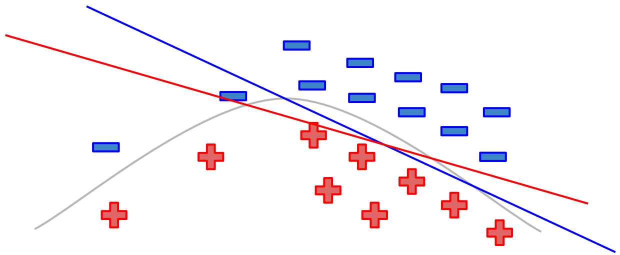
Data selection

Instead of weighting, we can train classifier and drop all observations that are too dissimilar from target domain.

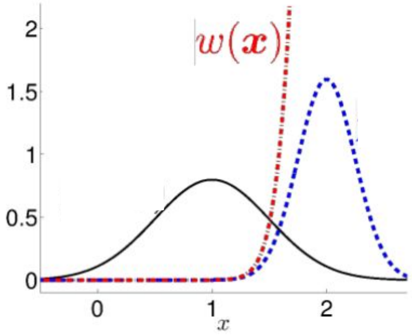
Source distribution:



Target distribution:



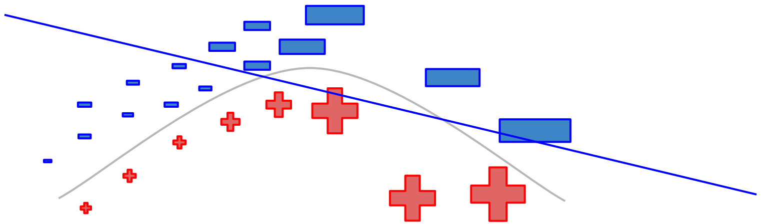
Instance weighting: problems



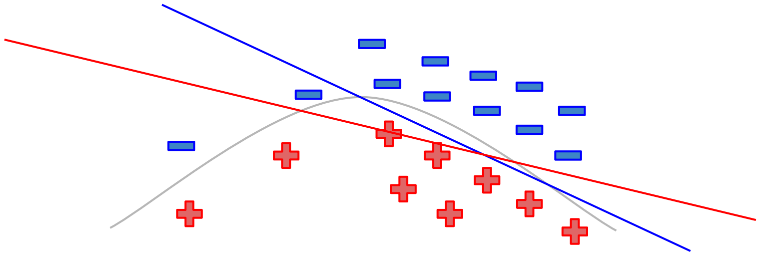
$$\hat{\epsilon}_S(w, x) = \frac{1}{N} \sum_{i=1}^{N} w(x_i) L(h_\theta(x_i^S), y_i^S)$$

$$\epsilon_T \leq \hat{\epsilon}_S(w, x) + \sqrt{\frac{O(\max_x w(x)^2)}{N}}$$

Source distribution:



Target distribution:



Proxy-labels methods

In previous case we use target domain data only for weight calculation. And do not use knowledge that target domain data contain.

It’s good to utilize somehow unlabelled data for training.

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{T} = \{x_i^T\}_{i=1}^M \sim P_T(x)$$

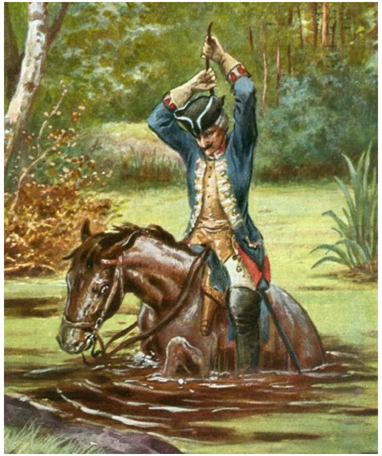
Self-training

Based on Ruder's presentation at ACL'18

1. Train model on labeled data.
2. Use confident predictions on unlabeled data as training examples. Repeat.

Algorithm 1 Self-training (Abney, 2007)

repeat
  $m \leftarrow train\_model(L)$
  for $x \in U$ **do**
    if $\max m(x) > \tau$ **then**
      $L \leftarrow L \cup \{(x, p(x))\}$
until no more predictions are confident

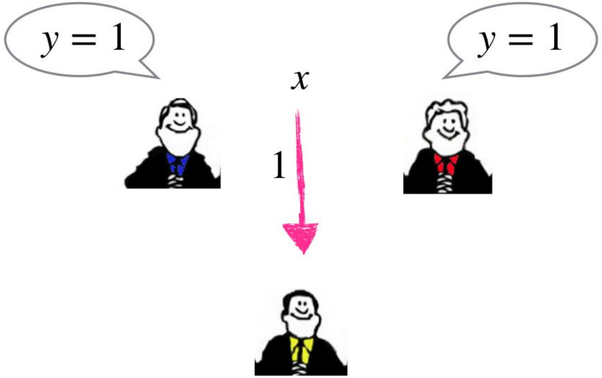


Error is amplified cause a model can not correct its own mistakes.

Tri-training

Based on [Ruder's presentation at ACL'18]

1. Train three models on bootstrapped samples.
2. Use predictions on unlabeled data for third if two agree.
3. Final prediction: majority voting



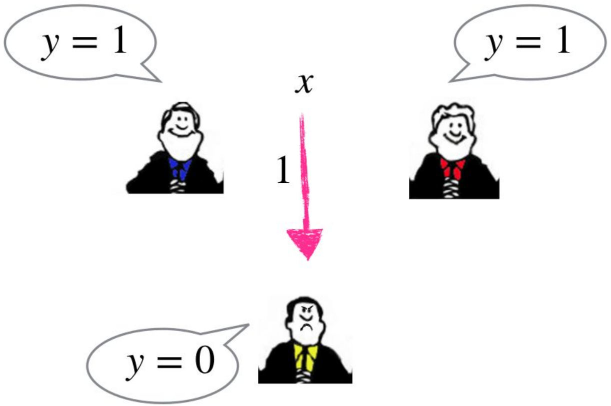
Algorithm 2 Tri-training (Zhou and Li, 2005)

for $i \in \{1..3\}$ **do**
  $S_i \leftarrow bootstrap\_sample(L)$
  $m_i \leftarrow train\_model(S_i)$
repeat
  for $i \in \{1..3\}$ **do**
    $L_i \leftarrow \emptyset$
    for $x \in U$ **do**
      if $p_j(x) = p_k(x) (j, k \neq i)$ **then**
        $L_i \leftarrow L_i \cup \{(x, p_j(x))\}$
    $m_i \leftarrow train\_model(L \cup L_i)$
until none of m_i changes
apply majority vote over m_i

Tri-training with disagreement

Based on [Ruder's presentation at ACL'18]

1. Train three models on bootstrapped samples.
2. Use predictions on unlabeled data for third if two agree and prediction differs.
3. Final prediction: majority voting

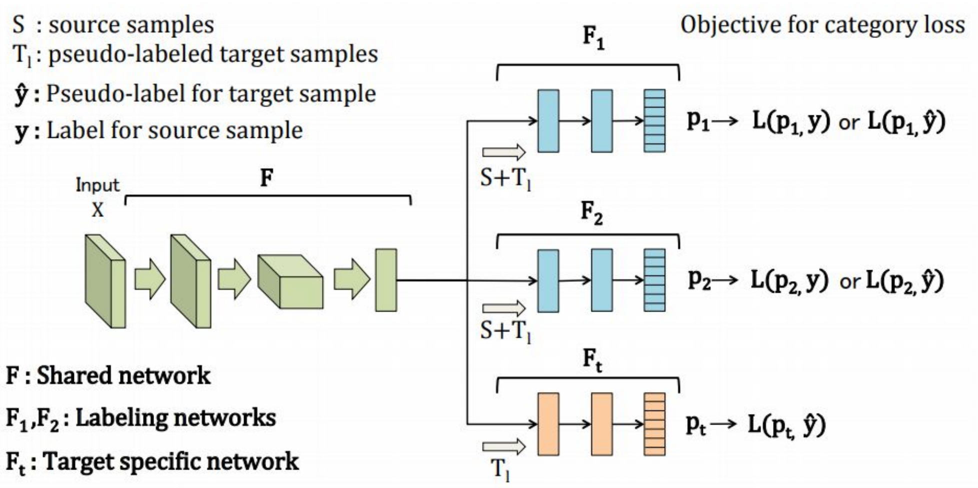


The problem with tri-training is that training three separate models can be too expensive.

Let's share parameters somehow.

[Saito et al., 2017]

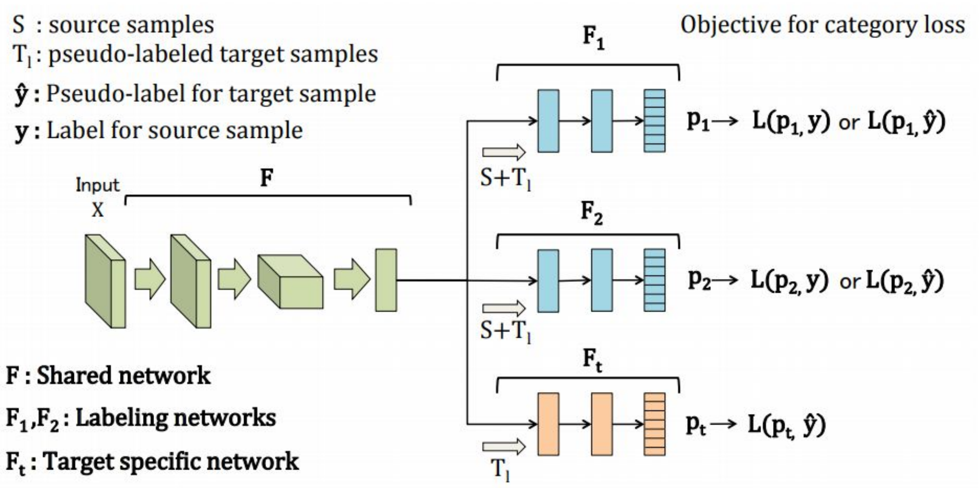
Asymmetric Tri-training



Input: data
$\mathcal{S} = \{(x_i, t_i)\}_{i=1}^{m}, \mathcal{T} = \{(x_j)\}_{j=1}^{n}$
$\mathcal{T}_l = \emptyset$

Asymmetric Tri-training

[Saito et al., 2017](#)



Input: data

$\mathcal{S} = \{(x_i, t_i)\}_{i=1}^{m}, \mathcal{T} = \{(x_j)\}_{j=1}^{n}$

$\mathcal{T}_l = \emptyset$

for $j = 1$ **to** $iter$ **do**

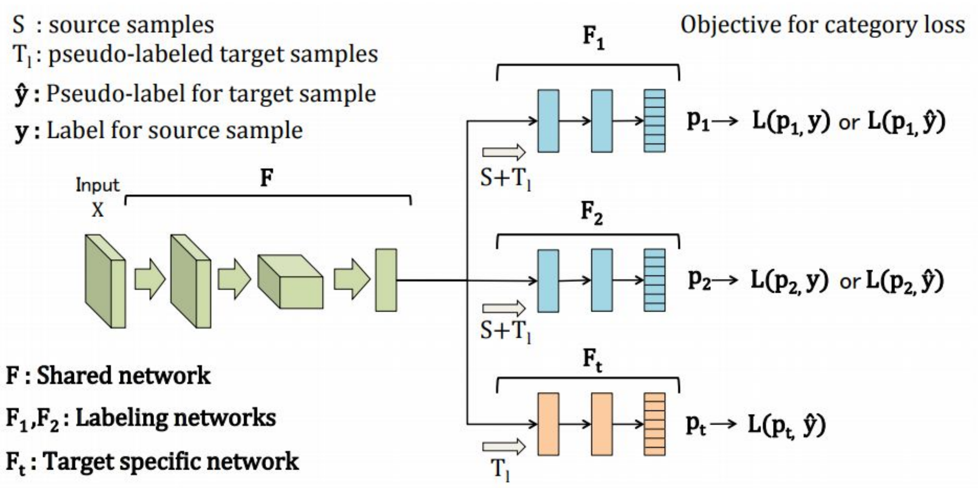
Train F, F_1, F_2, F_t with mini-batch from training set $\mathcal{S}$

end for

$$E(\theta_F, \theta_{F_1}, \theta_{F_2}) = \frac{1}{n}\sum_{i=1}^{n}\big[L_y(F_1 \circ F(x_i)), y_i)$$
$$+ L_y(F_2 \circ (F(x_i)), y_i)\big] + \lambda |W_1{}^T W_2|$$

Saito et al., 2017

Asymmetric Tri-training



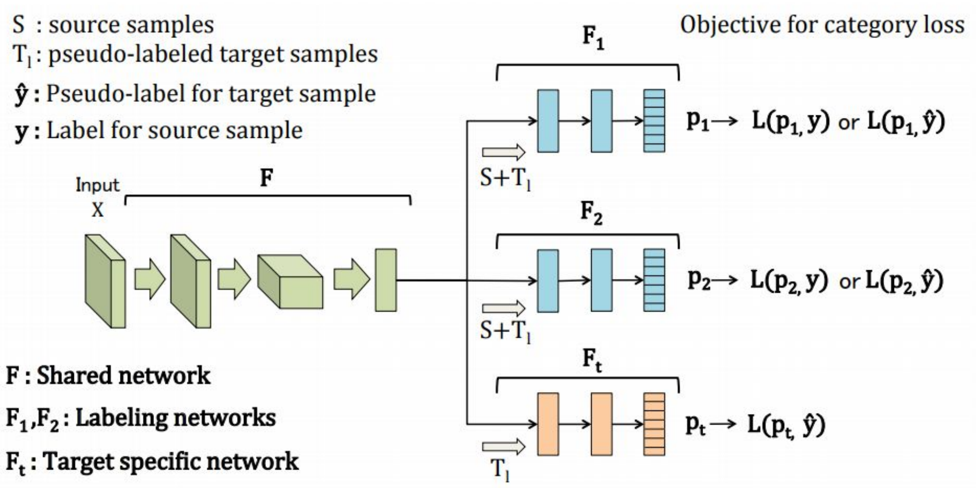
Input: data
$\mathcal{S} = \{(x_i, t_i)\}_{i=1}^{m}, \mathcal{T} = \{(x_j)\}_{j=1}^{n}$
$\mathcal{T}_l = \emptyset$
for $j = 1$ **to** *iter* **do**
  Train F, F_1, F_2, F_t with mini-batch from training set $\mathcal{S}$
end for

$$E(\theta_F, \theta_{F_1}, \theta_{F_2}) = \frac{1}{n}\sum_{i=1}^{n}\Big[L_y(F_1 \circ F(x_i)), y_i)$$
$$+ L_y(F_2 \circ (F(x_i)), y_i)\Big] + \boxed{\lambda |{W_1}^T W_2|}$$

To force F1 and F2 to learn from different features.

Asymmetric Tri-training

Saito et al., 2017

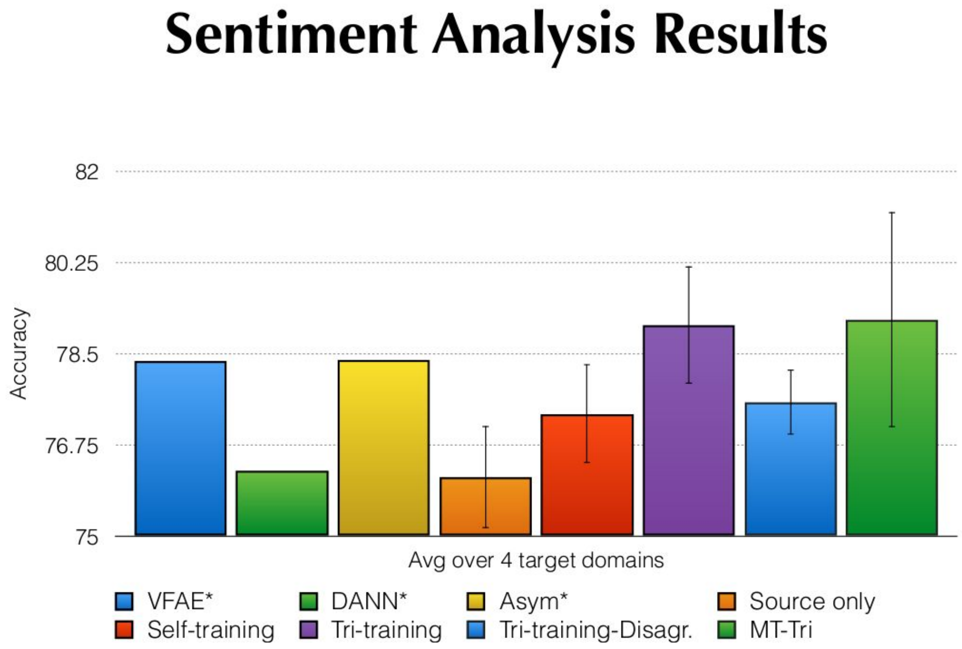


$$E(\theta_F, \theta_{F_1}, \theta_{F_2}) = \frac{1}{n}\sum_{i=1}^{n}\left[L_y(F_1 \circ F(x_i)), y_i) + L_y(F_2 \circ (F(x_i)), y_i)\right] + \lambda|W_1{}^T W_2|$$

Input: data
$\mathcal{S} = \{(x_i, t_i)\}_{i=1}^{m}, \mathcal{T} = \{(x_j)\}_{j=1}^{n}$
$\mathcal{T}_l = \emptyset$
for $j = 1$ **to** $iter$ **do**
  Train F, F_1, F_2, F_t with mini-batch from training set $\mathcal{S}$
end for
$N_t = N_{init}$
$\mathcal{T}_l$ = Labeling($F, F_1, F_2, \mathcal{T}, N_t$)
$\mathcal{L} = \mathcal{S} \cup \mathcal{T}_l$
for k steps **do**
  for $j = 1$ **to** $iter$ **do**
    Train F, F_1, F_2 with mini-batch from training set $\mathcal{L}$
    Train F, F_t with mini-batch from training set $\mathcal{T}_l$
  end for
  $\mathcal{T}_l = \emptyset, N_t = k/20 * n$
  $\mathcal{T}_l$ = Labeling($F, F_1, F_2, \mathcal{T}, N_t$)
  $\mathcal{L} = \mathcal{S} \cup \mathcal{T}_l$
end for

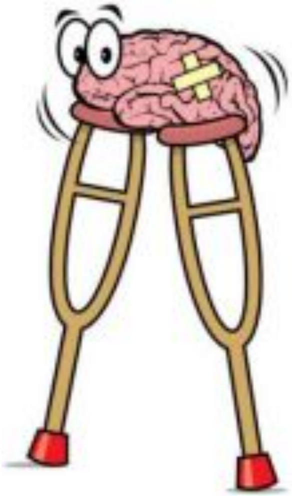
On effectiveness of proxy-label methods

Based on [Ruder's presentation at ACL'18]



Sentiment analysis on Amazon
reviews
dataset (Blitzer et al, 2006)

Hack of the day...



Hack of the day: back-translation

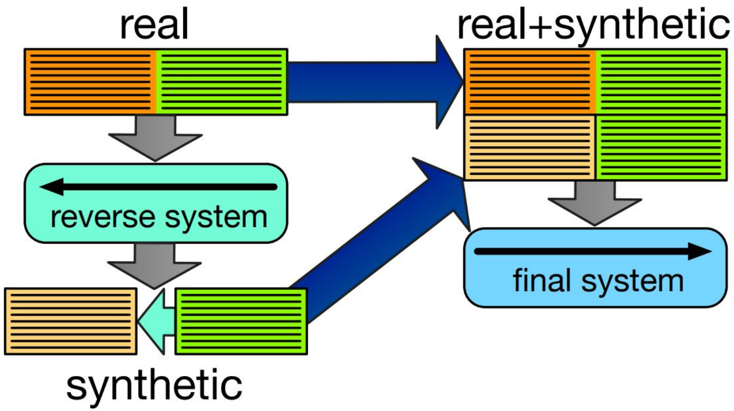
Sennrich, 2016b

In proxy-label methods we generate output labels based on unlabeled input data.

But what if we want to generate input data based on output? Does it make sense?

NMT is trained in the reverse translation direction (target-to-so
then used to translate target-side monolingual data back into th
language.

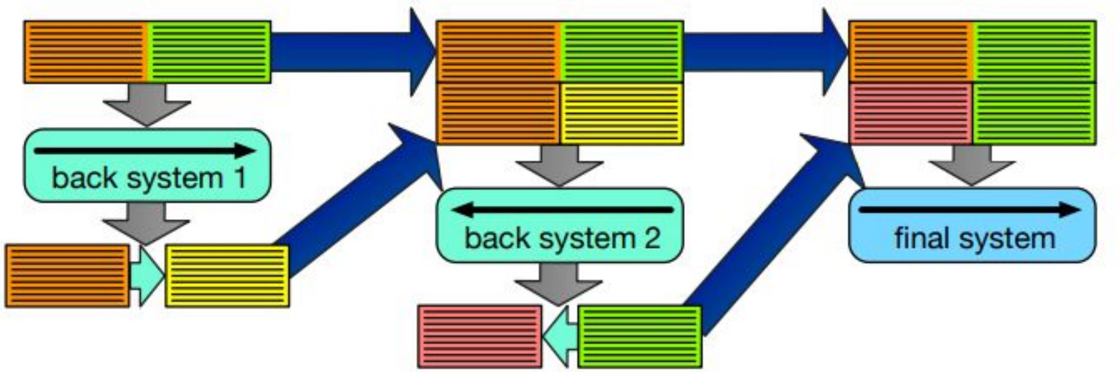
Picture from Hoang et al., 2018



Step further: iterative back-translation

[Hoang et al., 2018]

Iterative back-translation: back-translated data is used to build better translation systems in forward and backward directions, which in turn is used to reback-translate monolingual data



(Semi)-supervised domain adaptation

Both in source and target domain labels are known.

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Fine-tuning

- Train model on the target domain data
- Train with lower learning rate on the source domain

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Fine-tuning

- Train model on the target domain data
- Train with lower learning rate on the source domain

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Problems?

Fine-tuning

- Train model on the labeled target domain data
- Train with lower learning rate on the labeled source domain data

$$\mathcal{D}_S = \{x_i^S, y_i^S\}_{i=1}^N \sim P_S(x, y)$$

$$\mathcal{D}_T = \{x_i^T, y_i^T\}_{i=1}^M \sim P_T(x, y)$$

Ordinarily, in-domain data is limited, therefore we have generalization problems (over-fitting).

Fine-tuning: regularization techniques

Ordinarily, in-domain data is limited, therefore we have generalization problems (over-fitting). To prevent this:

Fine-tuning: regularization techniques

Ordinarily, in-domain data is limited, therefore we have generalization problems (over-fitting). To prevent this:

1. Weight decay $L_W = ||W||_{L_p}$

W is the in-domain parameter matrix to be learned

$\hat{W}$ is the corresponding fixed out-of-domain parameter matrix.

Fine-tuning: regularization techniques

Ordinarily, in-domain data is limited, therefore we have generalization problems (over-fitting). To prevent this:

1. Weight decay $L_W = ||W||_{L_p}$
2. Dropout

W is the in-domain parameter matrix to be learned

$\hat{W}$ is the corresponding fixed out-of-domain parameter matrix.

Fine-tuning: regularization techniques

Ordinarily, in-domain data is limited, therefore we have generalization problems (over-fitting). To prevent this:

1. Weight decay $L_W = ||W||_{L_p}$
2. Dropout
3. L2-distance from out-of-domain penalization (MAP-L2) $L_W = \lambda \cdot \left\| W - \hat{W} \right\|_2^2$

W is the in-domain parameter matrix to be learned

$\hat{W}$ is the corresponding fixed out-of-domain parameter matrix.

Fine-tuning: regularization techniques

Barone et al., 2017

Out-of-domain: WMT
In-domain: TED talks

Table 1: English-to-German translation BLEU scores

System	valid tst2010	test tst2011	test tst2012	test tst2013	avg
Out-of-domain only	27.19	29.65	25.78	27.85	27.76
In-domain only	25.95	27.84	23.68	25.83	25.78
Fine-tuning	30.53	32.62	28.86	32.11	31.20
Fine-tuning + dropout	30.63	33.06	28.90	32.02	31.33
Fine-tuning + MAP-L2	30.81	32.87	28.99	31.88	31.25
Fine-tuning + tuneout	30.49	32.07	28.66	31.60	30.78†
Fine-tuning + dropout + MAP-L2	30.80	**33.19**	**29.13**	**32.13**	**31.48**†

†: different from the fine-tuning baseline at 5% significance.

Limited degradation on out-of-domain

L2-distance from out-of-domain penalization (MAP-L2) limits quality degradation on the out-of-domain.

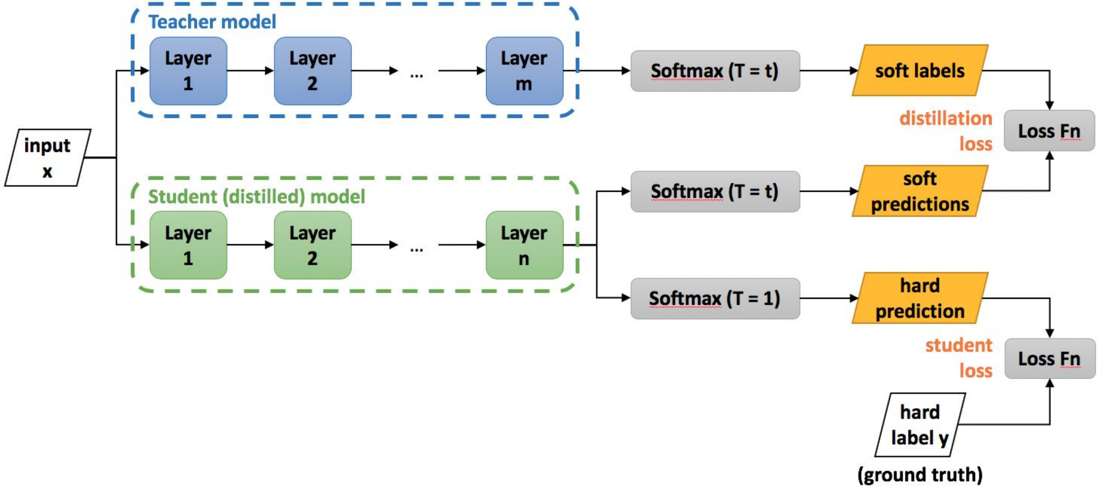
Limited degradation on out-of-domain

L2-distance from out-of-domain penalization (MAP-L2) limits quality degradation on the out-of-domain.

But we can directly force predictions of in-domain model and out-of-domain model to be closer to each other!

Distillation-like domain adaptation

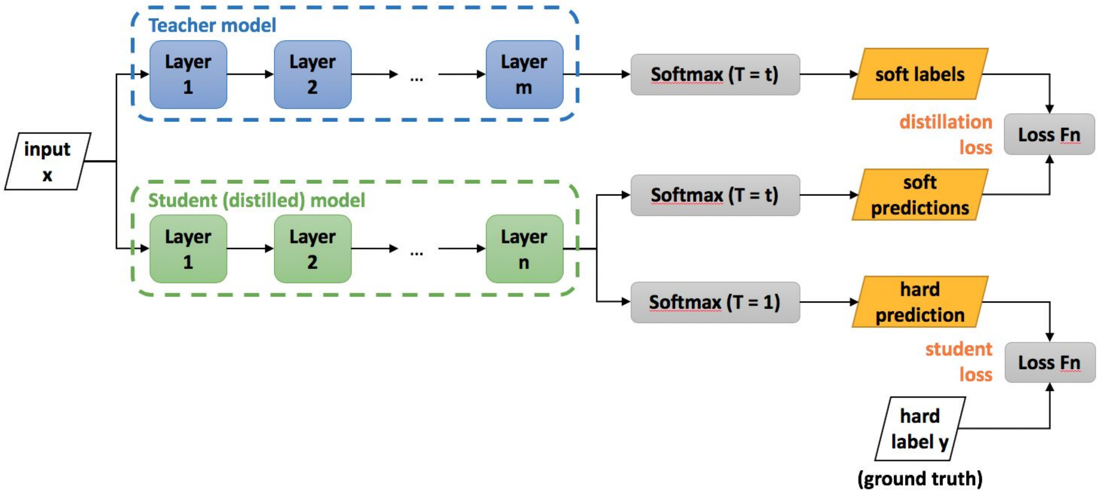
“Knowledge distillation” framework of (Hinton et al., 2014): a smaller “student” network learns to mimic a large “teacher” network by minimizing the loss between the output distributions of the two networks.



Picture from nervanasystems.github.io

Distillation-like domain adaptation

“Knowledge distillation” framework of (Hinton et al., 2014): a smaller “student” network learns to mimic a large “teacher” network by minimizing the loss between the output distributions of the two networks.



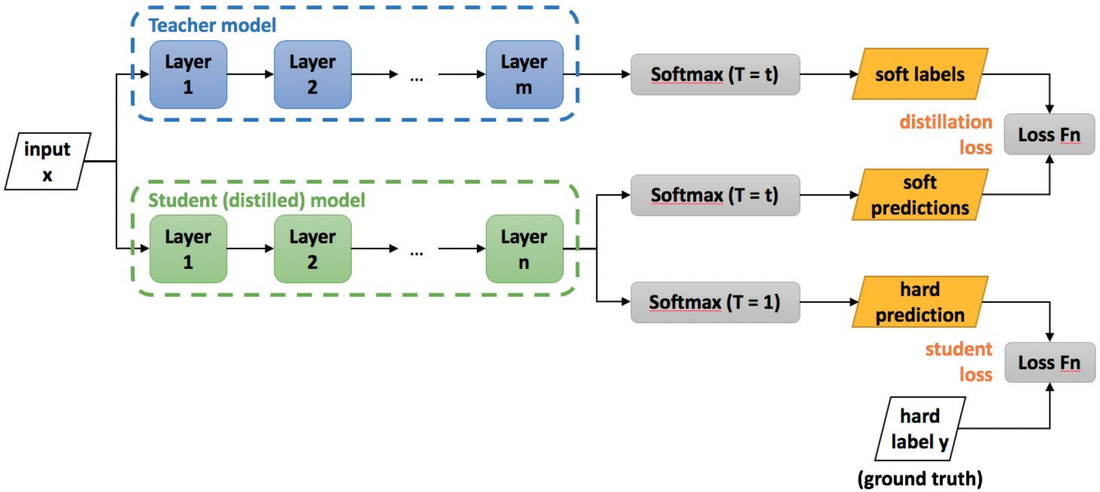
Originally, to compress multiple models (ensemble) to a smaller model.

Picture from nervanasystems.github.io

Distillation-like domain adaptation

Hinton et al., 2015

“Knowledge distillation” framework of (Hinton et al., 2014): a smaller “student” network learns to mimic a large “teacher” network by minimizing the loss between the output distributions of the two networks.



But this framework can be adapted for domain adaptation.

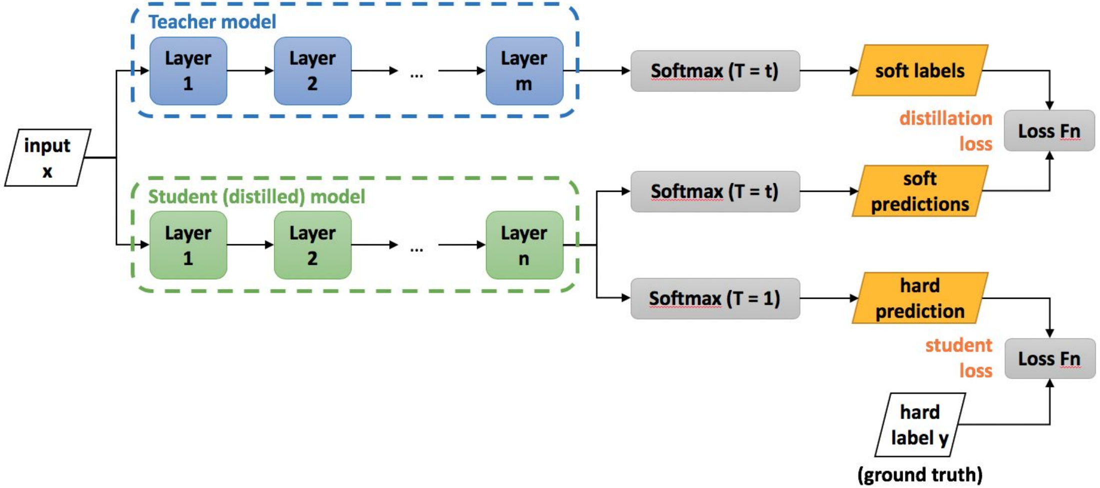
Picture from nervanasystems.github.io

Distillation-like domain adaptation

Dakwale et al., 2017

- Train teacher network
- Initialize student network by weights of teacher
- Train student with composite loss

Teacher produces distribution *p*,
Student produces distribution *q*.

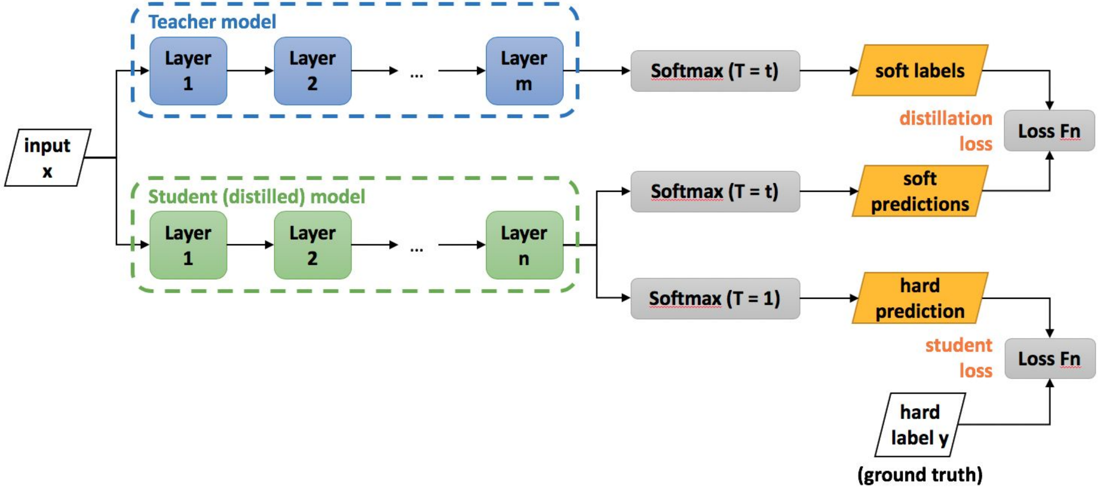


$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

Picture from nervanasystems.github.io

Distillation-like domain adaptation

- Train teacher network
- Initialize student network by weights of teacher
- Train student with composite loss



Composite loss consists of two terms:

1. Cross-entropy loss on *q*
2. KL-divergence between *p* and *q*

$$L_{total} = (1-\lambda)L(q_\theta) + \lambda D_{KL}(p||q_\theta)$$

Picture from nervanasystems.github.io

Distillation-like domain adaptation

Composite loss consists of two terms:

1. Cross-entropy loss between hard-label distribution and *q*
2. KL-divergence between *p* and *q*

$$L_{total} = (1 - \lambda)L(q_\theta) + \lambda D_{KL}(p||q_\theta)$$

Distillation-like domain adaptation

Composite loss consists of two terms:

1. Cross-entropy loss between hard-label distribution and *q*
2. KL-divergence between *p* and *q*

$$L_{total} = (1-\lambda)\boxed{L(q_\theta)} + \lambda D_{KL}(p||q_\theta)$$

$$L(q_\theta) = -\sum_k [l = y_k] \log q_\theta(y_k) = H(l, q_\theta)$$

Distillation-like domain adaptation

Composite loss consists of two terms:

1. Cross-entropy loss between hard-label distribution and *q*
2. KL-divergence between *p* and *q*

$$L_{total} = (1 - \lambda)L(q_\theta) + \lambda \boxed{D_{KL}(p||q_\theta)}$$

$$D_{KL}(p||q_\theta) = \sum_k p(y_k)(\log p(y_k) - \log q_\theta(y_k)) = H(p, q_\theta) - H(p)$$

Distillation-like domain adaptation

Composite loss consists of two terms:

1. Cross-entropy loss between hard-label distribution and *q*
2. KL-divergence between *p* and *q*

$$L_{total} = (1-\lambda)L(q_\theta) + \lambda D_{KL}(p||q_\theta)$$

$$L_{total} \sim (1-\lambda)H(l, q_\theta) + \lambda(H(p, q_\theta) - H(p)) \sim (1-\lambda)H(l, q_\theta) + \lambda H(p, q_\theta)$$

Distillation-like domain adaptation

Using composite loss we force internal representation to be tolerant to domain. Thus we implicitly minimize discrepancy between domain distributions in internal representation space.

$$\epsilon_T \leq \epsilon_S + \boxed{\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)} + \lambda$$

The divergence between
source and target domain

Should we minimize distribution divergence directly?

$$\epsilon_T \le \epsilon_S + \boxed{\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)} + \lambda$$

The divergence between
source and target domain

Should we minimize distribution divergence directly?

Deep distribution alignment! -> adversarial methods, variational bayes methods and other cool stuff.

$$\epsilon_T \leq \epsilon_S + \boxed{\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)} + \lambda$$

The divergence between
source and target domain

Batch normalization revisited

Li et al., 2017

Algorithm 1 Adaptive Batch Normalization (AdaBN)

for neuron j in DNN **do**

Concatenate neuron responses on all images of target domain t: $\mathbf{x}_j = [\ldots, x_j(m), \ldots]$

Compute the mean and variance of the target domain: $\mu_j^t = \mathbb{E}(\mathbf{x}_j^t)$, $\sigma_j^t = \sqrt{\mathrm{Var}(\mathbf{x}_j^t)}$.

end for

for neuron j in DNN, testing image m in target domain **do**

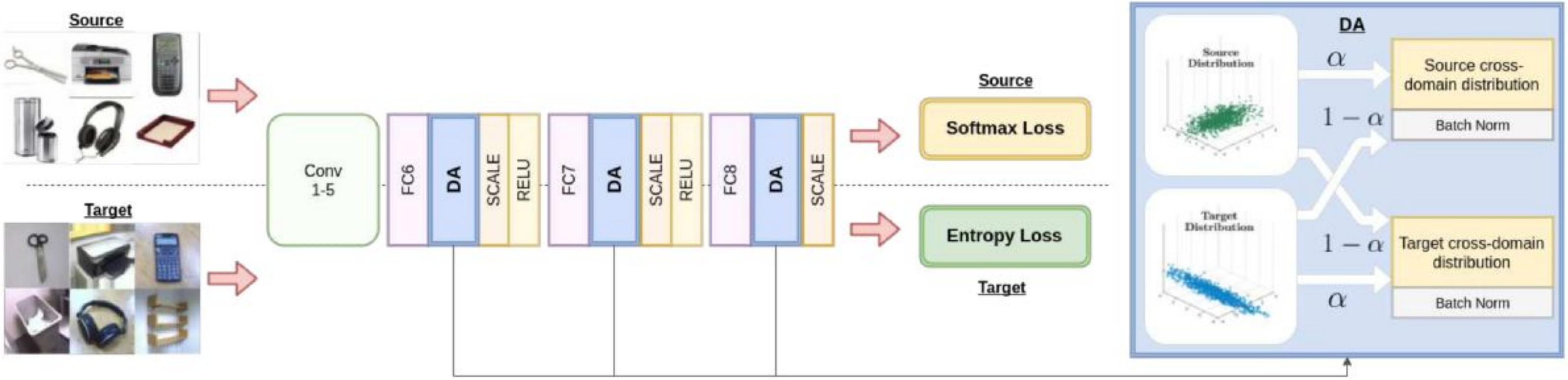
Compute BN output $y_j(m) := \gamma_j \frac{(x_j(m) - \mu_j^t)}{\sigma_j^t} + \beta_j$

end for

$$\mathtt{DA}(x_s; \alpha) = \frac{x_s - \mu_{st,\alpha}}{\sqrt{\epsilon + \sigma^2_{st,\alpha}}}, \quad \mathtt{DA}(x_t; \alpha) = \frac{x_t - \mu_{ts,\alpha}}{\sqrt{\epsilon + \sigma^2_{ts,\alpha}}},$$

Batch normalization revisited

Carlucci et al., 2017



$$\mathrm{DA}(x_s;\alpha) = \frac{x_s - \mu_{st,\alpha}}{\sqrt{\epsilon + \sigma^2_{st,\alpha}}}, \quad \mathrm{DA}(x_t;\alpha) = \frac{x_t - \mu_{ts,\alpha}}{\sqrt{\epsilon + \sigma^2_{ts,\alpha}}},$$

Assume q^s and q^t to be the distribution of x_s and x_t, respectively, and let $q^{st}_\alpha = \alpha q^s + (1-\alpha)q^t$ and, symmetrically, $q^{ts}_\alpha = \alpha q^t + (1-\alpha)q^s$ be cross-domain distributions mixed by a factor $\alpha \in [0.5, 1]$.

Auto-DIAL

Deep distribution alignment

Long et al., 2015

Domain adaptation network utilizes Maximum Mean Discrepancy (MMD)

$$d_k^2(p,q) \triangleq \left\| \mathbf{E}_p\left[\phi\left(\mathbf{x}^s\right)\right] - \mathbf{E}_q\left[\phi\left(\mathbf{x}^t\right)\right] \right\|_{\mathcal{H}_k}^2$$

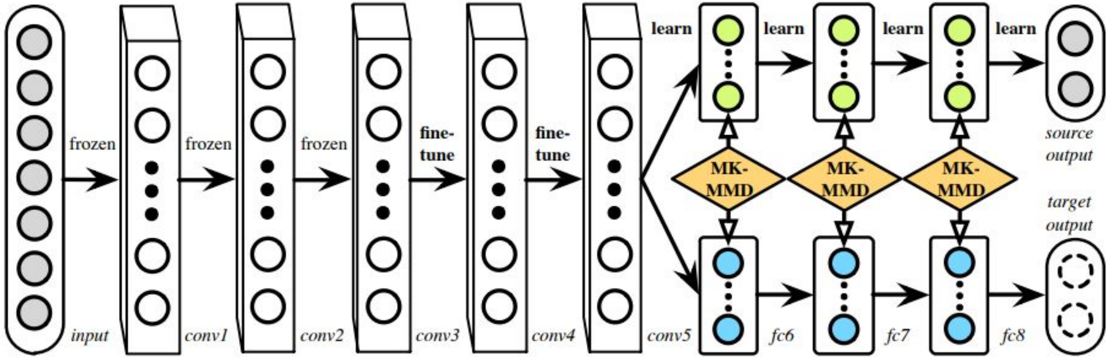


Figure 1. The DAN architecture for learning transferable features.

In two weeks...

We will try align distributions using adversarial methods!