



Domain Adaptation

October 14, 2021

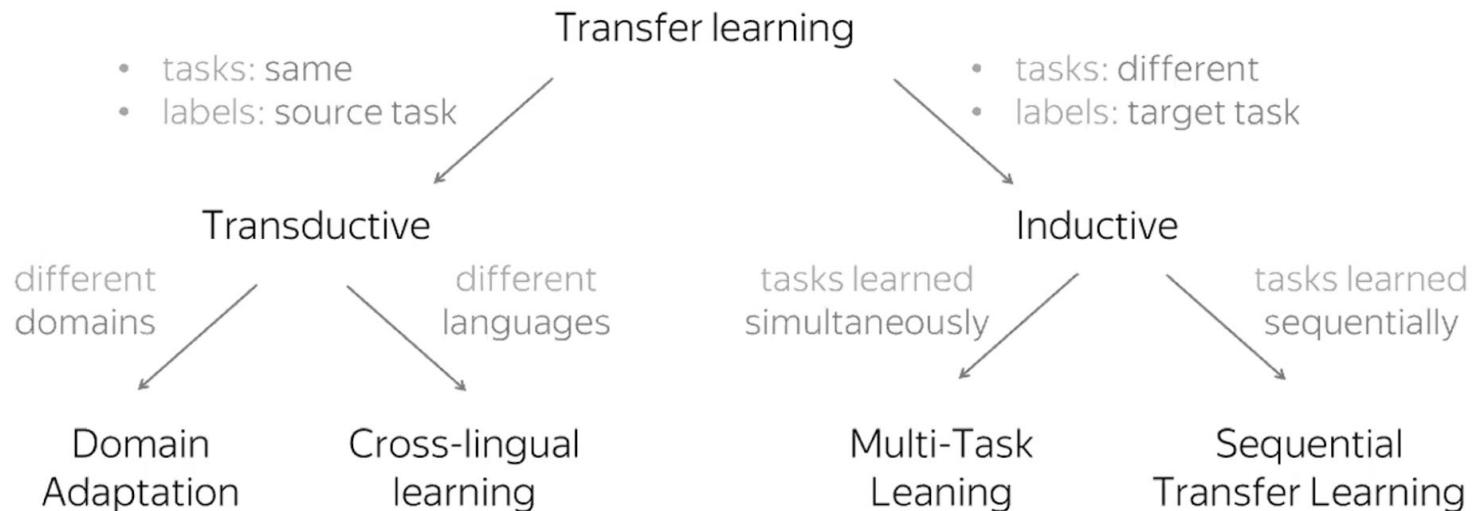
What is going to happen

- Domain adaptation idea
- Motivation
- Data-centric approaches for domain adaptation
- Model-centric approaches for domain adaptation

What is going to happen

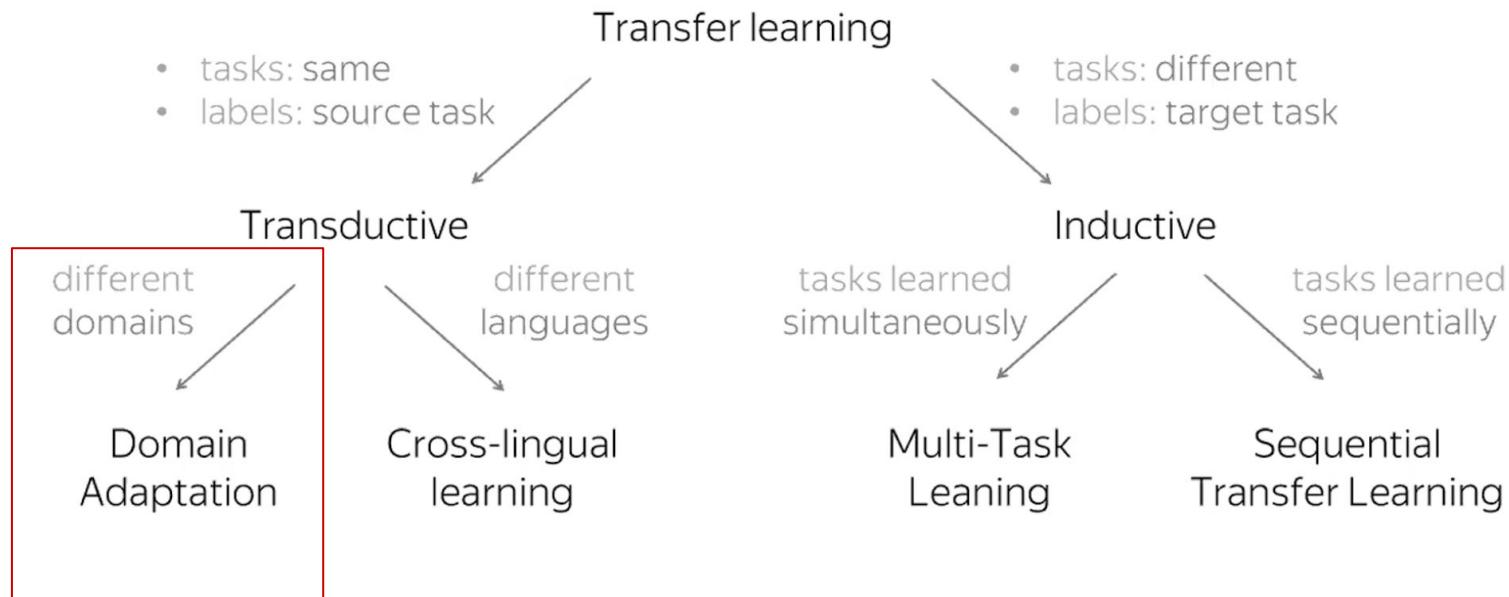
- Domain adaptation idea
- Motivation
- Data-centric approaches for domain adaptation
- Model-centric approaches for domain adaptation

A Taxonomy of Transfer Learning in NLP



This taxonomy is from [Ruder, 2019](#)

A Taxonomy of Transfer Learning in NLP

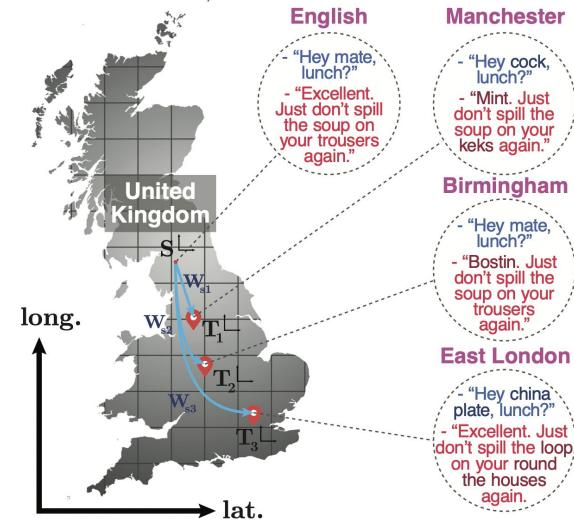
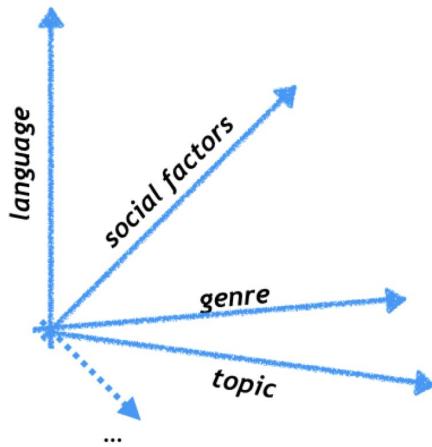


This taxonomy is from [Ruder, 2019](#)

What is a domain?

“A domain is a coherent type of corpus (predetermined by the given dataset)” (Plank, 2011)

“A domain is a manifold in a high dimensional variety space” (Plank, 2016)



Domain visualization

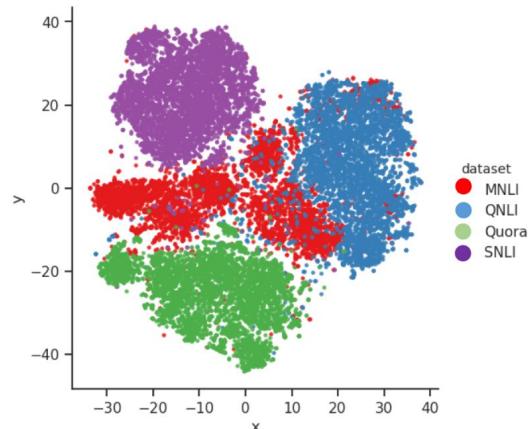


Figure 3: t-SNE plot of [CLS] token activations from the second-to-last encoder of BERT for four datasets in this paper. Second-to-last layer is used since the last layer embeddings may be biased to the target of BERT pre-training tasks.

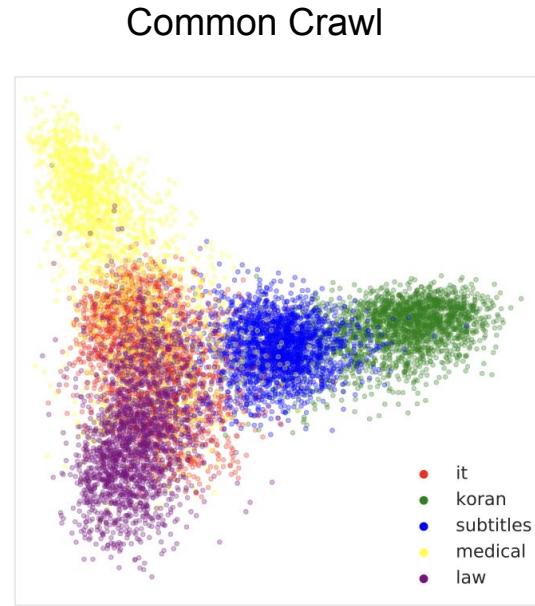


Figure 1: A 2D visualization of average-pooled BERT hidden-state sentence representations using PCA. The colors represent the domain for each sentence.

Standard supervised learning setting

“model” = model + data + task

Standard supervised learning setting

“model” =  model + data + task

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

Standard supervised learning setting

“model” \equiv model data task

- Model (CNN vs RNN, GNMT vs Transformer and so on) $\mathcal{H} = \{h_\theta : X \rightarrow Y\}$
- 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 (CNN vs RNN, GNMT vs Transformer and so on) $\mathcal{H} = \{h_\theta : X \rightarrow Y\}$
- 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) $h_\theta^* = \arg \min_{\theta} R^S(h_\theta) = \arg \min_{\theta} \mathbb{E}_{(x,y) \sim P_S} [L(h_\theta(x), y)]$

Standard learning assumptions

Standard learning assumptions

- Sample points are IID

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

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

Standard learning assumptions

- Sample points are IID

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

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

- Same distributions for training and test

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

Standard learning assumptions

- Sample points are IID

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

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

- 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

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

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

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

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

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

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

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

Standard learning assumptions

- Sample points are IID

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

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

- Same distributions for training and test $P_S(x, y) = P_T(x, y)$
- Fixed distributions (they don't change with time)

EXPECTATIONS

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:

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

What is going to happen

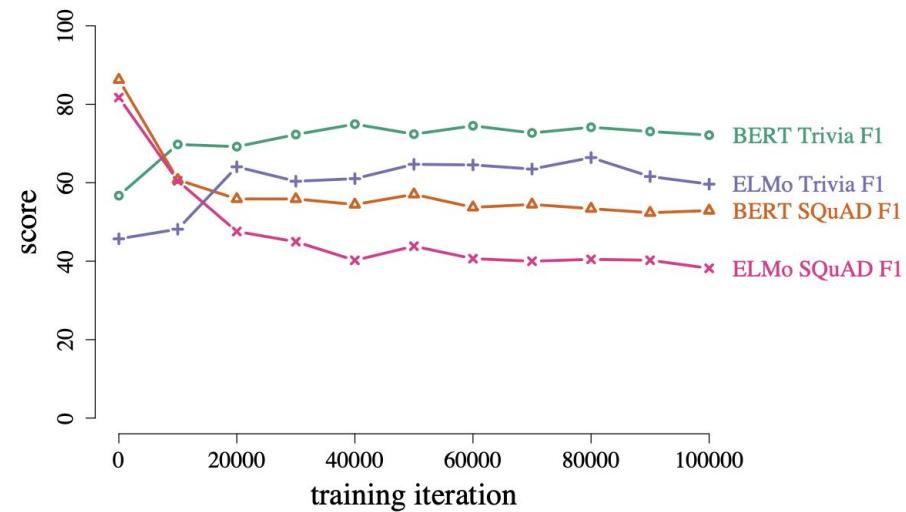
- Domain adaptation idea
- Motivation
- Data-centric approaches for domain adaptation
- Model-centric approaches for domain adaptation

In-domain vs. out-of-domain generalization

On SQuAD Dani Yogatama et al., “Learning and Evaluating General Linguistic Intelligence,”
<http://arxiv.org/abs/1901.11373>

	SQuAD	Trivia	QuAC	QA-SRL	QA-ZRE
BERT	86.5 (78.5)	35.6 (13.4)	56.2 (43.9)	77.5 (65.0)	55.3 (40.0)
ELMo	81.8 (72.2)	32.9 (12.6)	45.0 (34.5)	68.7 (52.3)	60.2 (42.0)

Table 2: F_1 (exact match) scores of the best BERT and ELMo models trained on SQuAD and evaluated on other question answering datasets.



In-domain vs. out-of-domain generalization

On MNLI: R. Thomas McCoy, Junghyun Min, and Tal Linzen, “BERTs of a Feather Do Not Generalize Together: Large Variability in Generalization across Models with Similar Test Set Performance,” <http://arxiv.org/abs/1911.02969>

Heuristic	Definition	Example
Lexical overlap	Assume that a premise entails all hypotheses constructed from words in the premise	The doctor was paid by the actor . → The doctor paid the actor. WRONG
Subsequence	Assume that a premise entails all of its contiguous subsequences.	The doctor near the actor danced. → The actor danced. WRONG
Constituent	Assume that a premise entails all complete subtrees in its parse tree.	If the artist slept, the actor ran. → The artist slept. WRONG

Figure 1: The heuristics targeted by the HANS dataset, along with examples of incorrect entailment predictions that these heuristics would lead to. (Figure from McCoy et al. (2019).)

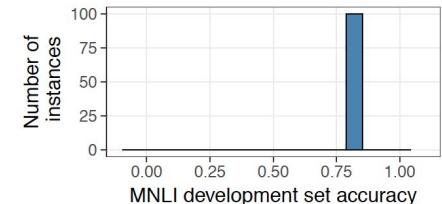


Figure 2: In-distribution generalization: Performance on the MNLI development set

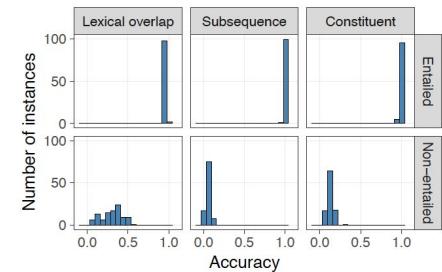


Figure 3: Out-of-distribution generalization: Performance on the HANS evaluation set, broken down into six categories of examples based on which syntactic heuristic each example targets and whether the correct label is *entailment* or *non-entailment*. The non-entailed lexical overlap cases (lower left plot) display a large degree of variability across instances.

In-domain vs. out-of-domain generalization

On MNLI: R. Thomas McCoy, Junghyun Min, and Tal Linzen, “BERTs of a Feather Do Not Generalize Together: Large Variability in Generalization across Models with Similar Test Set Performance,” <http://arxiv.org/abs/1911.02969>

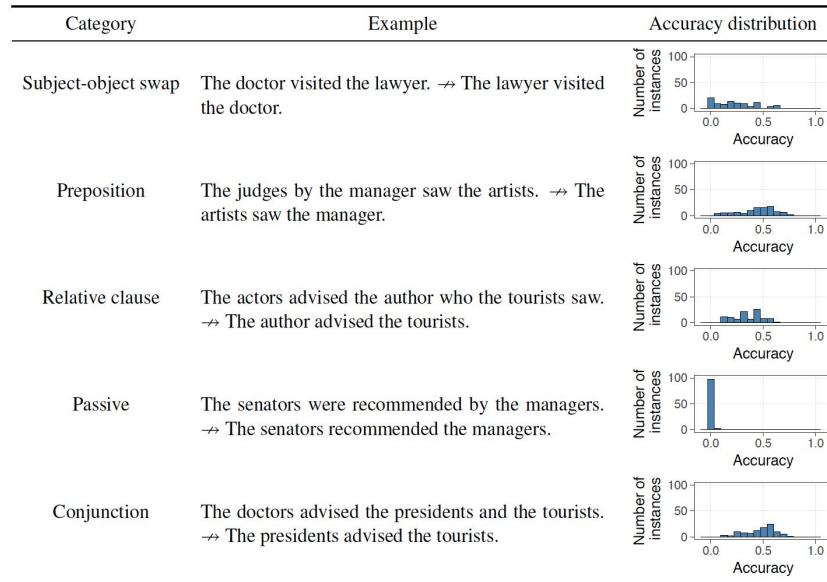
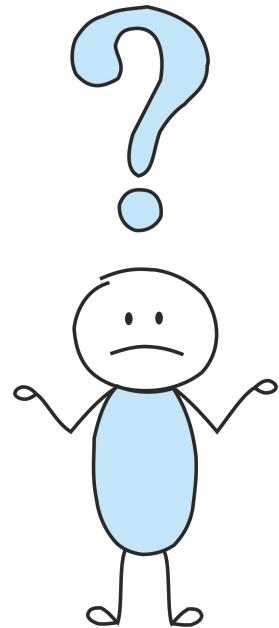


Figure 4: Results on the various subcategories within the non-entailed lexical overlap examples of the HANS dataset. We do not include the other 25 subcategories of the HANS dataset in this figure as there was little variability across instances for those subcategories.

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:

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

The divergence between
source and target domain

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

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?

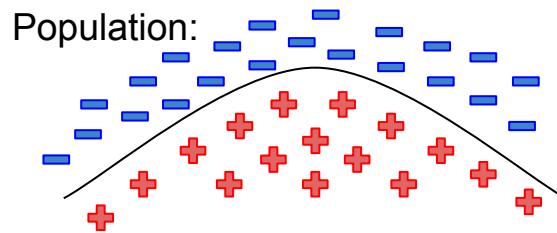
$$P_S(x) \neq P_T(x)$$

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

Why does a covariate shift make problems?

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

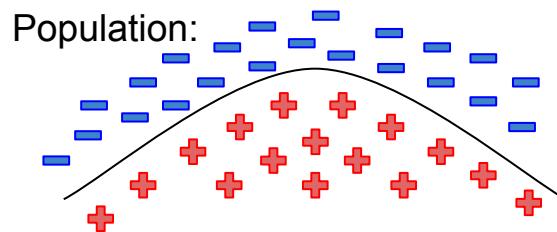
$$P_S(y|x) = P_T(y|x)$$



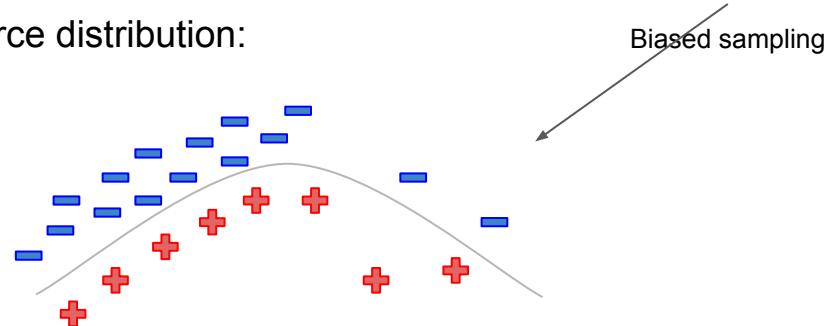
Why does a covariate shift make problems?

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

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



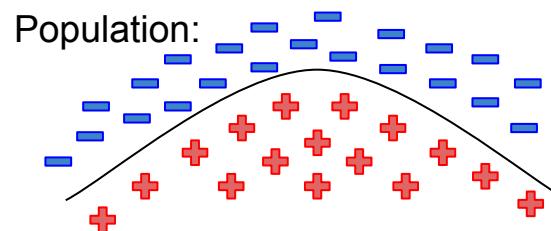
Source distribution:



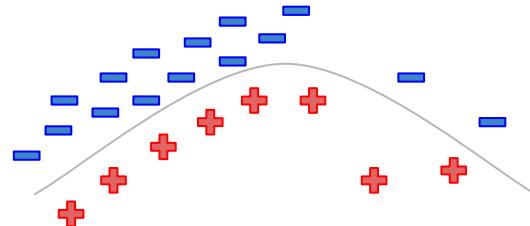
Why does a covariate shift make problems?

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

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

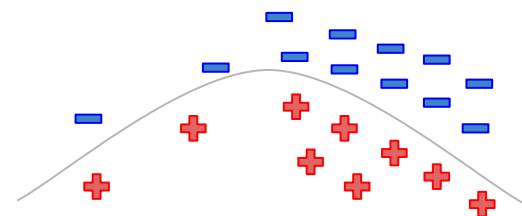


Source distribution:



Biased sampling

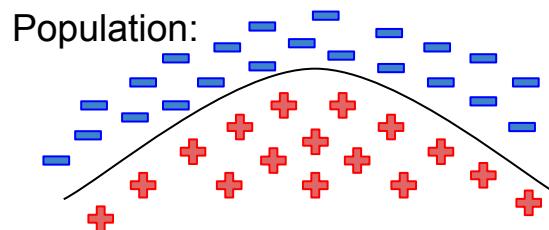
Target distribution:



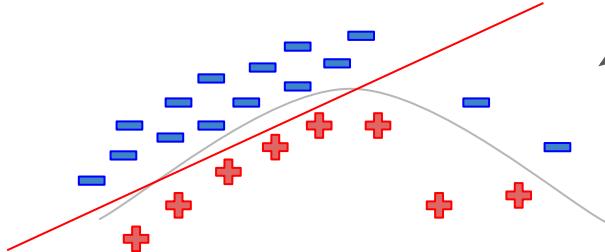
Why does a covariate shift make problems?

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

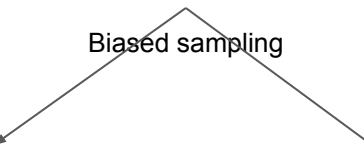
$$P_S(y|x) = P_T(y|x)$$



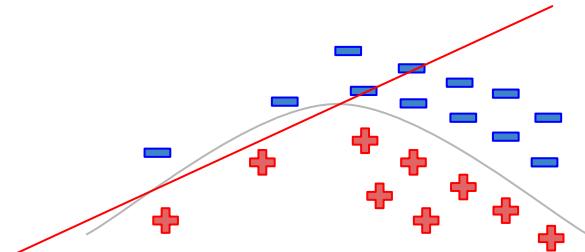
Source distribution:



Biased sampling



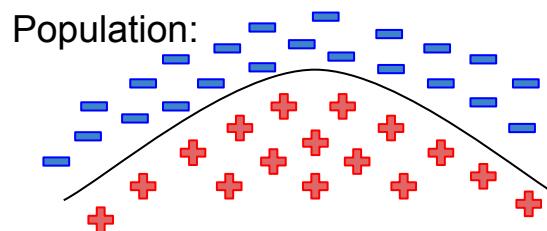
Target distribution:



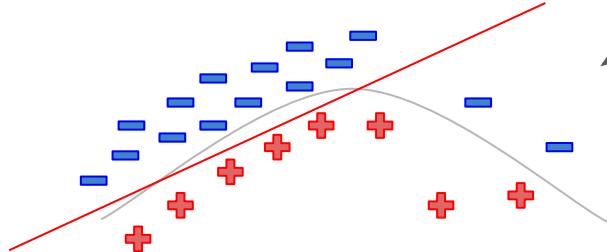
Why does a covariate shift make problems?

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

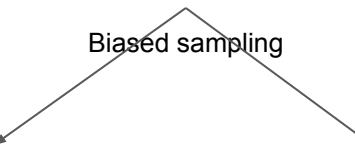
$$P_S(y|x) = P_T(y|x)$$



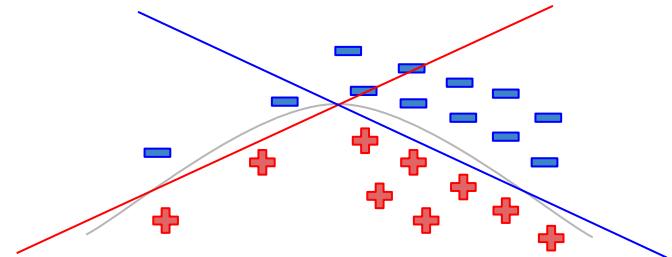
Source distribution:



Biased sampling



Target distribution:



Not optimal decision rule!

Unsupervised domain adaptation

Unsupervised domain adaptation

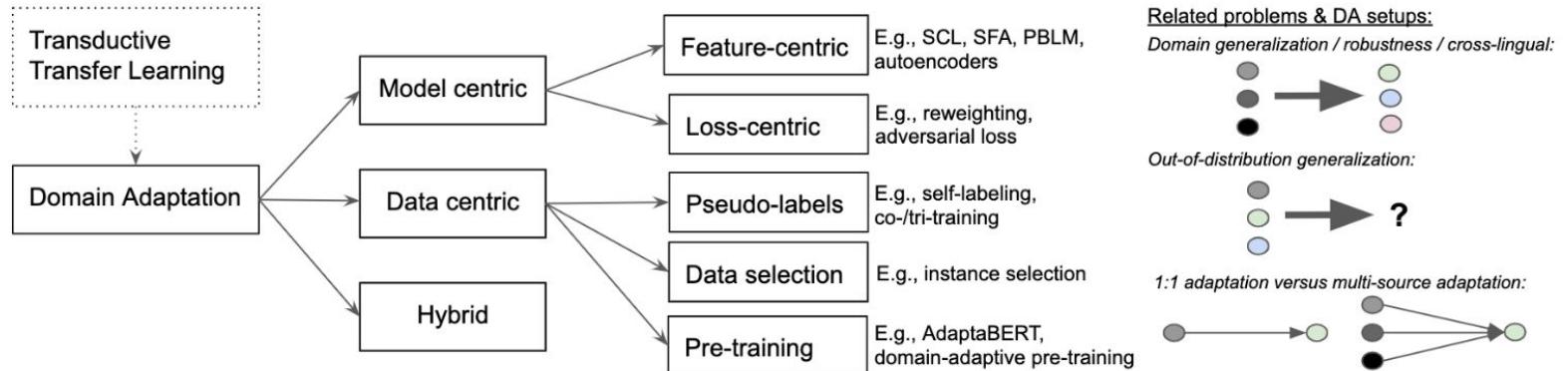
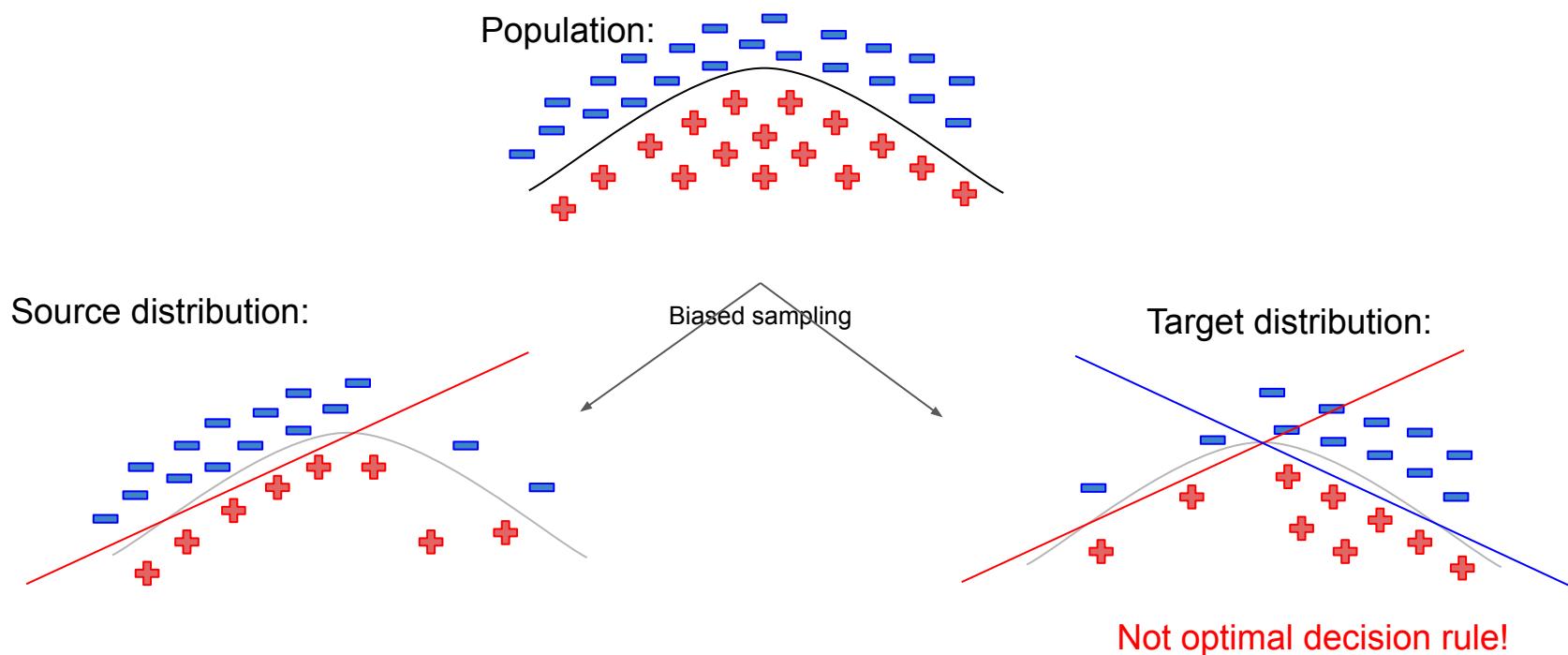


Figure 1: Taxonomy of DA as special case of transductive transfer learning, DA setups (1:1 versus multi-source adaptation) and related problems (domain and out-of-distribution generalization).

[Neural Unsupervised Domain Adaptation in NLP—A Survey](#)

What is going to happen

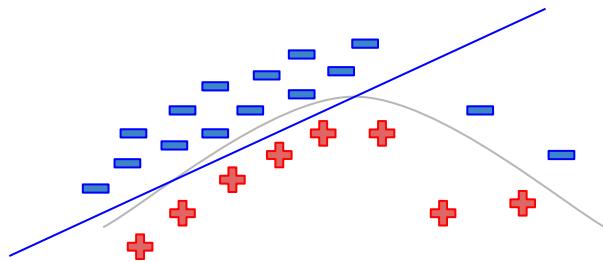
- Domain adaptation idea
- Motivation
- **Data-centric approaches for domain adaptation**
- Model-centric approaches for domain adaptation



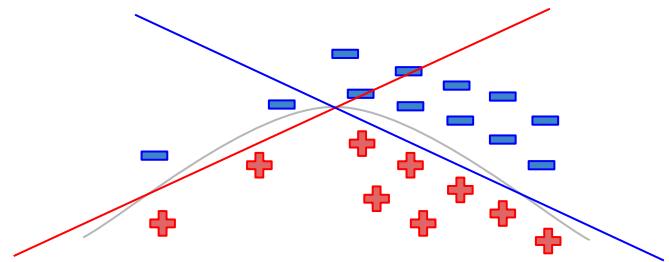
Instance weighting

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

Source distribution:



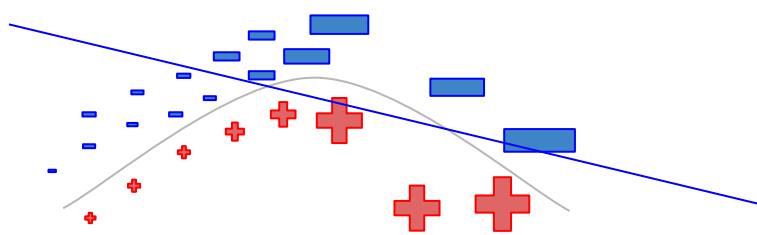
Target distribution:



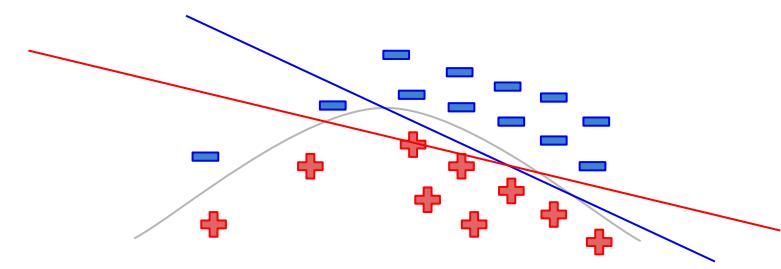
Not optimal decision rule!

Instance weighting

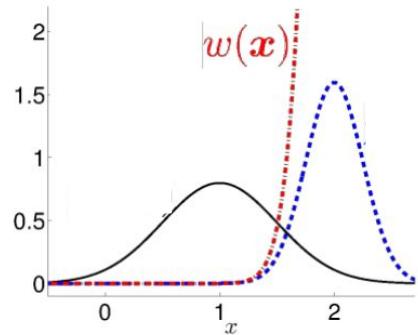
Source distribution:



Target distribution:



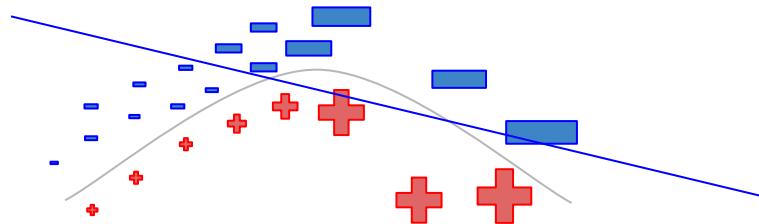
Instance weighting



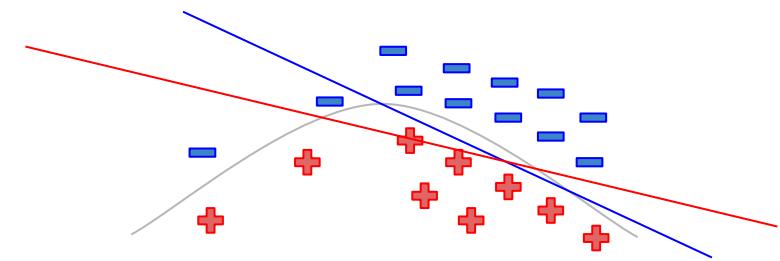
$$h_{\theta}^{*} = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N w(x_i) L(h_{\theta}(x_i), y_i)$$

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

Source distribution:



Target distribution:



Instance weighting: NLP applications

- Neural Machine Translation

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

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

Instance weighting: NLP applications

- Neural Machine Translation

$x = (v_1, v_2, \dots, v_k)$: 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

$x = (v_1, v_2, \dots, v_k)$: 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

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

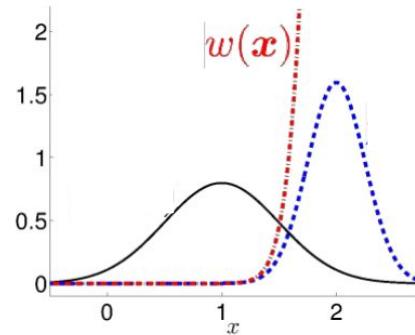
$P_T(x), P_S(x)$ can be defined using source and target language models

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

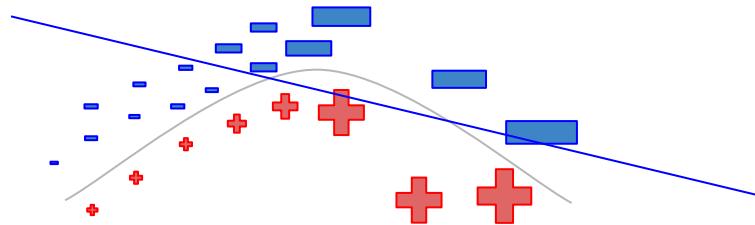
$$w_i = (1 + p_d(x_i))$$

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

Instance weighting: problems



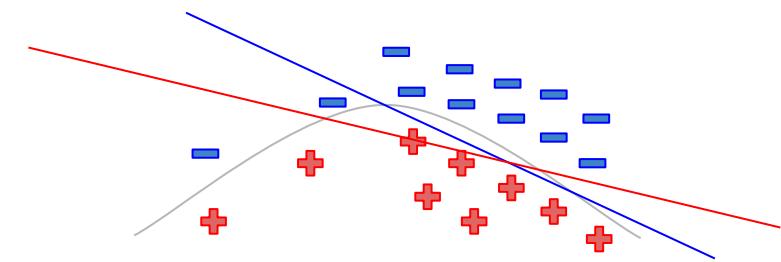
Source distribution:



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

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

Target distribution:

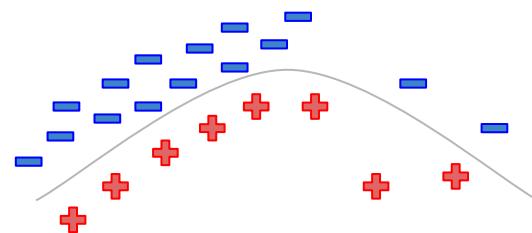


Data selection

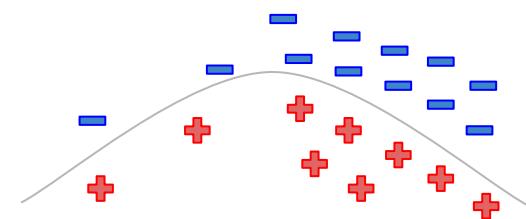
Instead of weighting, we can drop all observations that are too dissimilar from target domain.

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

Source distribution:



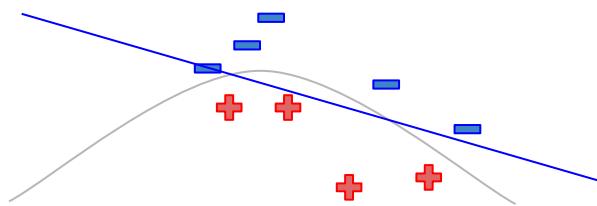
Target distribution:



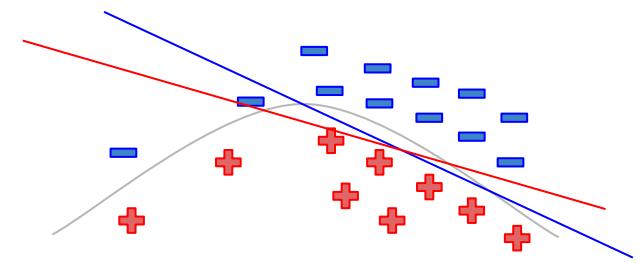
Data selection

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

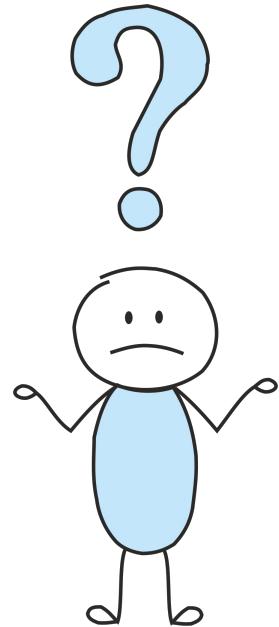
Source distribution:



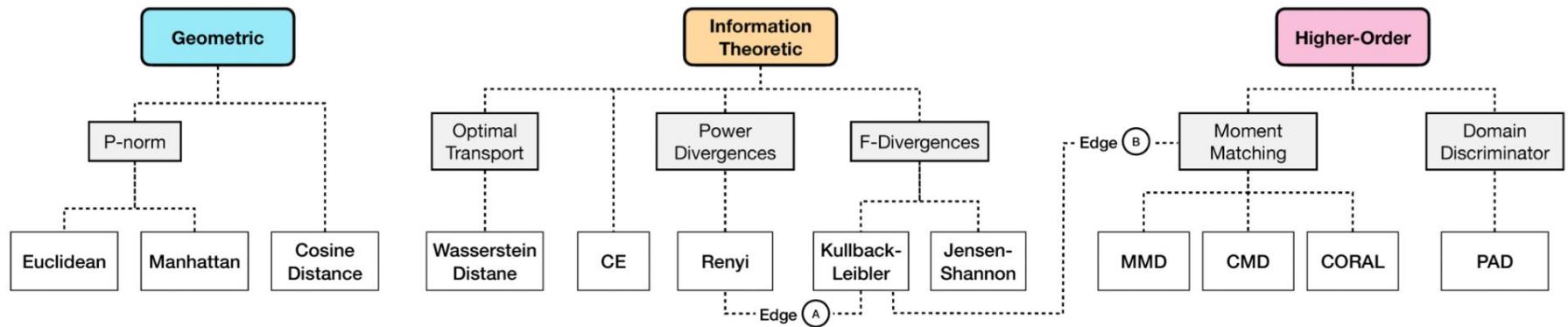
Target distribution:



Data selection: how to choose right samples from source domain?



Domain similarity measures

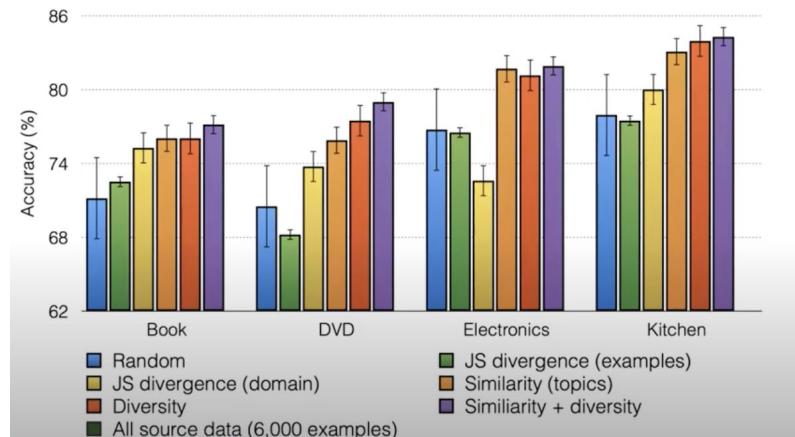


Data selection: how to choose right samples from source domain?

Criterion = Similarity + Diversity ([Ruder, 2019](#))

Sentiment Analysis Results

Selecting m=2,000 from 6,000 source domain examples



Data selection: how to choose right samples from source domain?

Criterion = Similarity + Diversity ([Ruder, 2019](#))

Similarity measures:

- Domain-target binary classifier score
- Target domain language model score / ratio
- Cosine similarity for sentence/document embeddings
- Rényi divergence
- Jensen-Shannon divergence
- Bhattacharyya distance

Diversity measures:

- Number of word types: $\#types$.
- Type-token ratio: $\frac{\#types}{\#tokens}$.
- Entropy (Shannon, 1948): $-\sum_i p_i \ln(p_i)$.
- Simpson's index (Simpson, 1949): $-\sum_i p_i^2$.
- Rényi entropy (Rényi, 1961):
$$\frac{1}{\alpha-1} \log(\sum_i p_i^\alpha)$$
- Quadratic entropy (Rao, 1982):
$$\sum_{i,j} \cos(v_{t_i}, v_{t_j}) p_i p_j.$$

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.

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

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

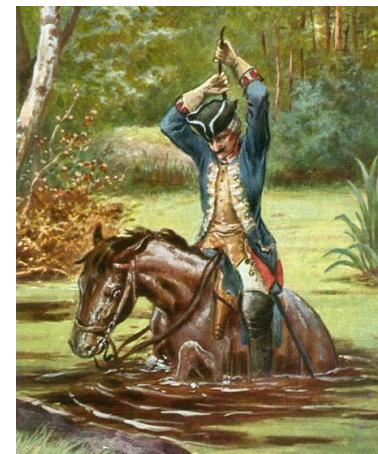
Self-training

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)

```
1: repeat
2:    $m \leftarrow \text{train\_model}(L)$ 
3:   for  $x \in U$  do
4:     if  $\max m(x) > \tau$  then
5:        $L \leftarrow L \cup \{(x, p(x))\}$ 
6: until no more predictions are confident
```

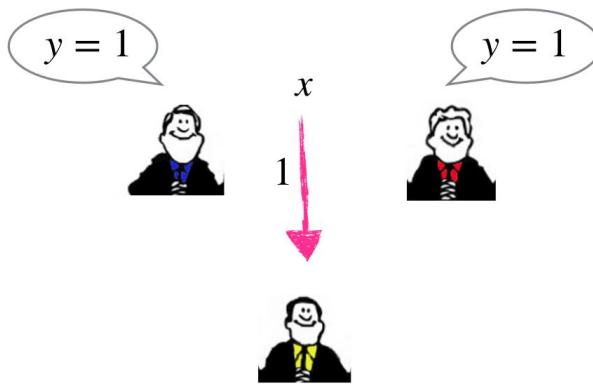


Error is amplified cause a model can not correct its own mistakes.

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



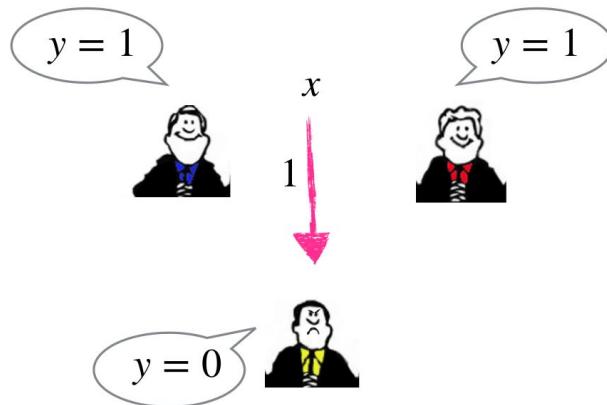
Algorithm 2 Tri-training (Zhou and Li, 2005)

```
1: for  $i \in \{1..3\}$  do
2:    $S_i \leftarrow bootstrap\_sample(L)$ 
3:    $m_i \leftarrow train\_model(S_i)$ 
4: repeat
5:   for  $i \in \{1..3\}$  do
6:      $L_i \leftarrow \emptyset$ 
7:     for  $x \in U$  do
8:       if  $p_j(x) = p_k(x) (j, k \neq i)$  then
9:          $L_i \leftarrow L_i \cup \{(x, p_j(x))\}$ 
 $m_i \leftarrow train\_model(L \cup L_i)$ 
10: until none of  $m_i$  changes
11: 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.

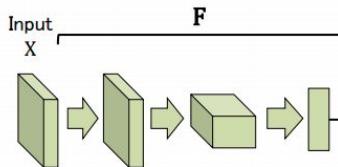
Asymmetric Tri-training

S : source samples

T_l: pseudo-labeled target samples

\hat{y} : Pseudo-label for target sample

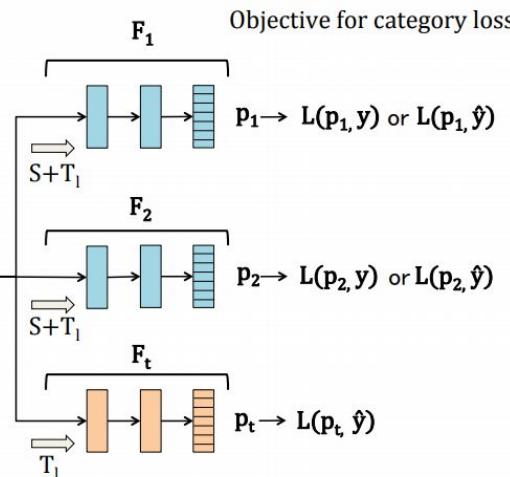
y : Label for source sample



F : Shared network

F_1, F_2 : Labeling networks

F_t : Target specific network



Objective for category loss

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

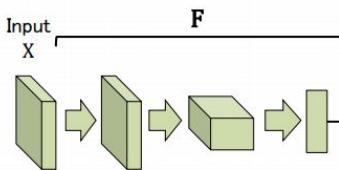
Asymmetric Tri-training

S : source samples

T_1 : pseudo-labeled target samples

\hat{y} : Pseudo-label for target sample

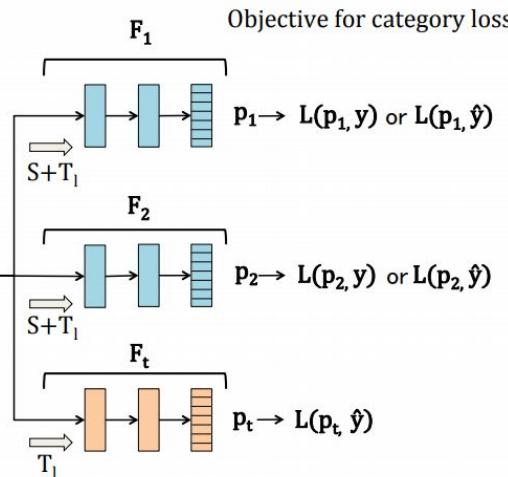
y : Label for source sample



F : Shared network

F_1, F_2 : Labeling networks

F_t : Target specific network



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

Input: data

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

$\mathcal{T}_l = \emptyset$

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

 Train F, F_1, F_2, F_t with mini-batch from training set

\mathcal{S}

end for

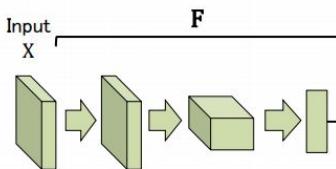
Asymmetric Tri-training

S : source samples

T_1 : pseudo-labeled target samples

\hat{y} : Pseudo-label for target sample

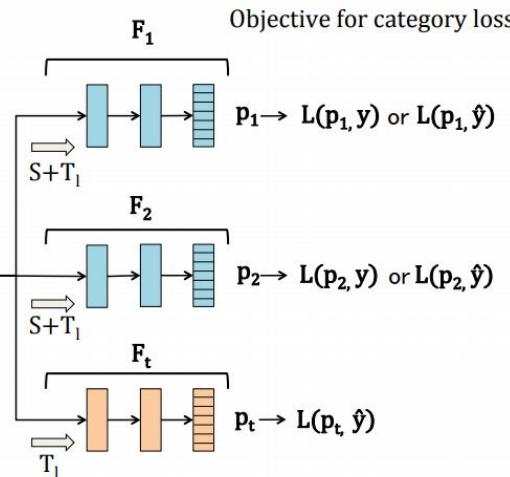
y : Label for source sample



F : Shared network

F_1, F_2 : Labeling networks

F_t : Target specific network



$$E(\theta_F, \theta_{F_1}, \theta_{F_2}) = \frac{1}{n} \sum_{i=1}^n [L_y(F_1 \circ F(x_i)), y_i)$$

$$+ L_y(F_2 \circ (F(x_i)), y_i)] + \boxed{\lambda |W_1^T W_2|}$$

To force F_1 and F_2 to learn from different features.

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

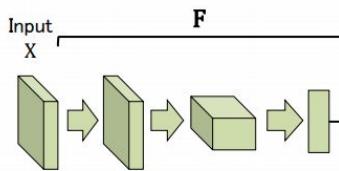
Asymmetric Tri-training

S : source samples

T_1 : pseudo-labeled target samples

\hat{y} : Pseudo-label for target sample

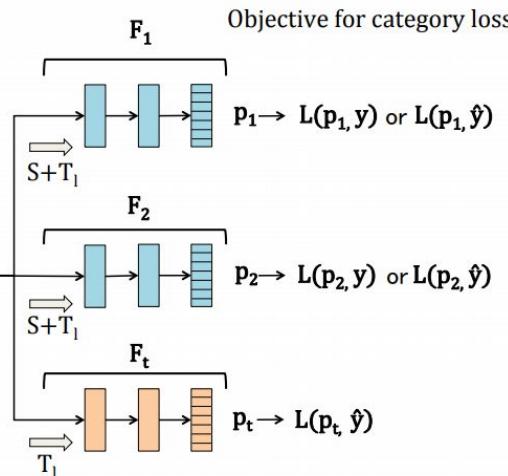
y : Label for source sample



F : Shared network

F_1, F_2 : Labeling networks

F_t : Target specific network



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

Input: data

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

$\mathcal{T}_l = \emptyset$

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

 Train F, F_1, F_2, F_t with mini-batch from training set S

end for

$N_t = N_{init}$

$\mathcal{T}_l = \text{Labeling}(F, F_1, F_2, \mathcal{T}, N_t)$

$\mathcal{L} = S \cup \mathcal{T}_l$

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

$\mathcal{T}_l = \emptyset, N_t = k/20 * n$

$\mathcal{T}_l = \text{Labeling}(F, F_1, F_2, \mathcal{T}, N_t)$

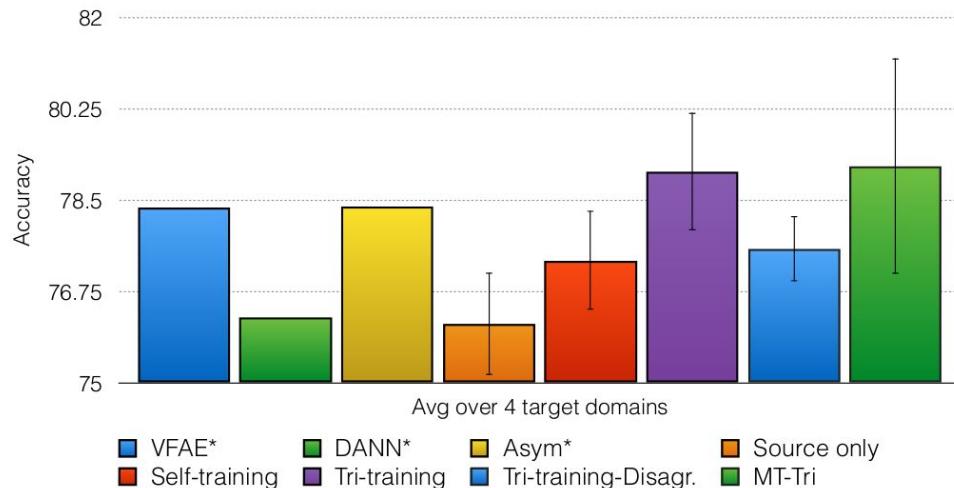
$\mathcal{L} = S \cup \mathcal{T}_l$

end for

On effectiveness of proxy-label methods

Sentiment Analysis Results

Based on [Ruder's presentation at ACL'18](#)



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

Back-translation

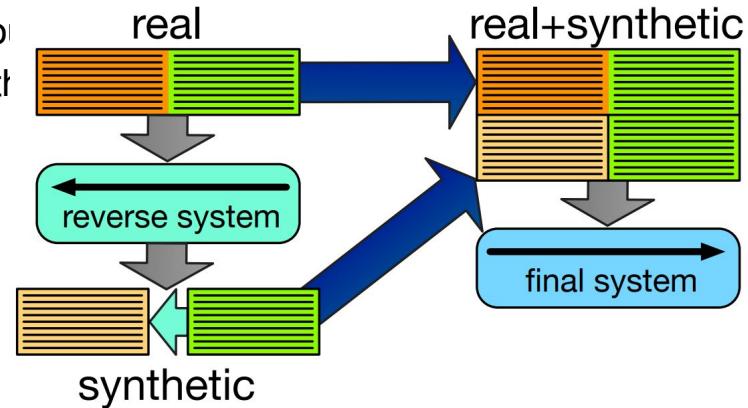
[Sennrich, 2016b](#)

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-source) then used to translate target-side monolingual data back into the language.

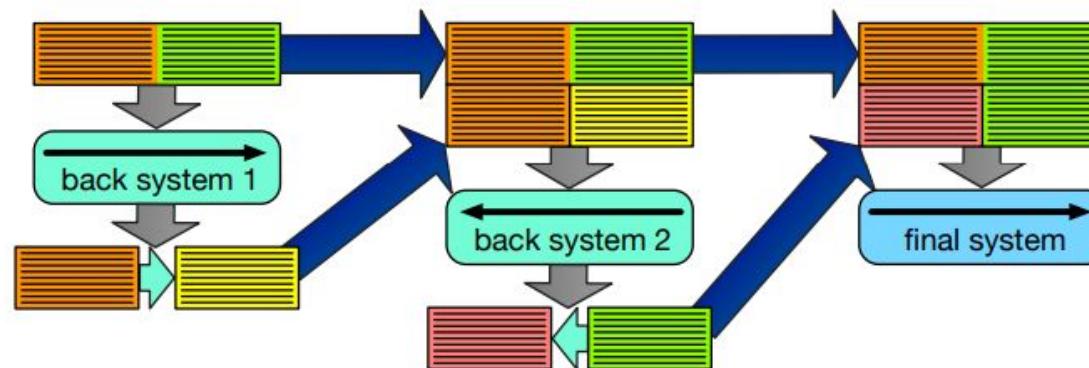
Picture from [Hoang et al., 2018](#)



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



Wait a second...

Why do we need proxy-labels?

What about **unsupervised pre-training**?

Unsupervised in-domain pre-training is effective

Don't Stop Pretraining: Adapt Language Models to Domains and Tasks

[Gururangan et al., 2020](#)

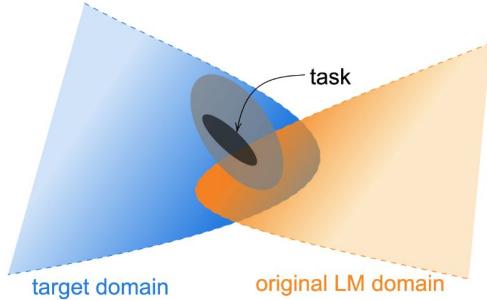


Figure 1: An illustration of data distributions. Task data is comprised of an observable task distribution, usually non-randomly sampled from a wider distribution (light grey ellipsis) within an even larger target domain, which is not necessarily one of the domains included in the original LM pretraining domain – though overlap is possible. We explore the benefits of continued pretraining on data from the task distribution and the domain distribution.

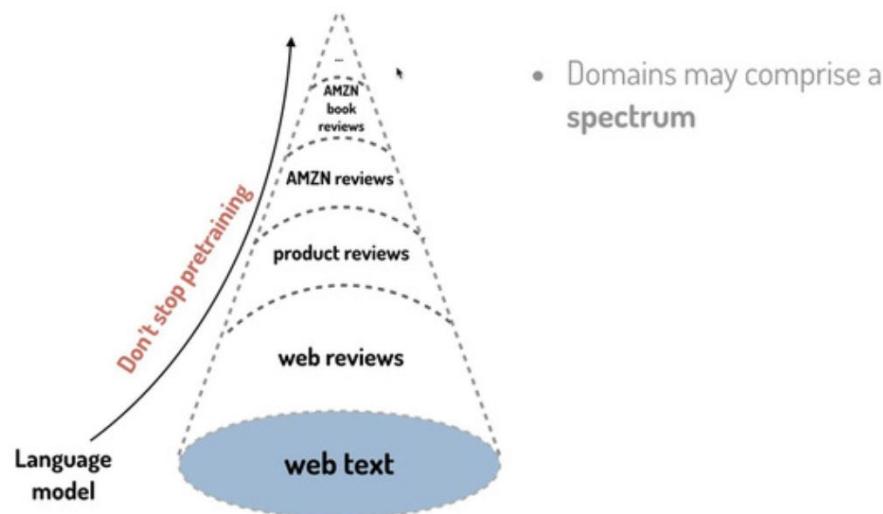
Domain	Task	ROBERTA	Additional Pretraining Phases		
			DAPT	TAPT	DAPT + TAPT
BIOMED	CHEMPROT	81.9 _{1.0}	84.2 _{0.2}	82.6 _{0.4}	84.4 _{0.4}
	†RCT	87.2 _{0.1}	87.6 _{0.1}	87.7 _{0.1}	87.8 _{0.1}
CS	ACL-ARC	63.0 _{5.8}	75.4 _{2.5}	67.4 _{1.8}	75.6 _{3.8}
	SCIERC	77.3 _{1.9}	80.8 _{1.5}	79.3 _{1.5}	81.3 _{1.8}
NEWS	HYPERPARTISAN	86.6 _{0.9}	88.2 _{5.9}	90.4 _{5.2}	90.0 _{6.6}
	†AGNEWS	93.9 _{0.2}	93.9 _{0.2}	94.5 _{0.1}	94.6 _{0.1}
REVIEWS	†HELPFULNESS	65.1 _{3.4}	66.5 _{1.4}	68.5 _{1.9}	68.7 _{1.8}
	†IMDB	95.0 _{0.2}	95.4 _{0.1}	95.5 _{0.1}	95.6 _{0.1}

Table 5: Results on different phases of adaptive pretraining compared to the baseline ROBERTA (col. 1). Our approaches are DAPT (col. 2, §3), TAPT (col. 3, §4), and a combination of both (col. 4). Reported results follow the same format as Table 3. State-of-the-art results we can compare to: CHEMPROT (84.6), RCT (92.9), ACL-ARC (71.0), SCIERC (81.8), HYPERPARTISAN (94.8), AGNEWS (95.5), IMDB (96.2); references in §A.2.

Unsupervised in-domain pre-training is effective

Don't Stop Pretraining: Adapt Language Models to Domains and Tasks

[Gururangan et al., 2020](#)



Unsupervised in-domain pre-training is effective

Unsupervised Domain Adaptation of Contextualized Embeddings for Sequence Labeling

[Han et al., 2020](#)

AdaptaBERT:

1. Domain tuning: (50/50 source+target data unsupervised pretraining.)
2. Task tuning using source data

System	WNUT			CoNLL	
	domain adaptation data	Precision	Recall	F1	F1
<i>Unsupervised domain adaptation</i>					
1. Task-tuned BERT	n/a	50.9	66.6	57.7	97.8
2. AdaptaBERT	WNUT training	52.8	66.7	58.9	97.6
3. AdaptaBERT	+ 1M tweets	53.6	68.3	60.0	97.8
4. AdaptaBERT	WNUT train+test	57.7	68.9	62.8	97.8
<i>Supervised in-domain training</i>					
5. Fine-tuned BERT	n/a	66.3	62.3	64.3	80.9
6. Limsopatham and Collier (2016)	n/a	73.5	59.7	65.9	

Table 4: Named entity segmentation performance on the WNUT test set and CoNLL test set A. Limsopatham and Collier (2016) had the winning system at the 2016 WNUT shared task. Their results are reprinted from their paper, which did not report performance on the CoNLL dataset.

Unsupervised in-domain pre-training is effective

Unsupervised Domain Adaptation of Contextualized Embeddings for Sequence Labeling

[Han et al., 2020](#)

AdaptaBERT:

1. Domain tuning: (50/50 source+target data unsupervised pretraining.)
2. Task tuning using source data

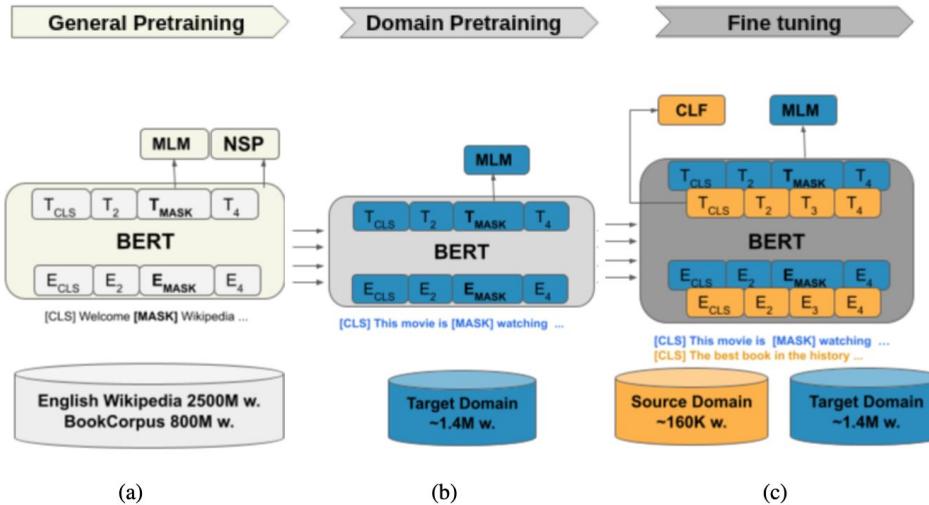
System	WNUT			CoNLL	
	domain adaptation data	Precision	Recall	F1	F1
<i>Unsupervised domain adaptation</i>					
1. Task-tuned BERT	n/a	50.9	66.6	57.7	97.8
2. AdaptaBERT	WNUT training	52.8	66.7	58.9	97.6
3. AdaptaBERT	+ 1M tweets	53.6	68.3	60.0	97.8
4. AdaptaBERT	WNUT train+test	57.7	68.9	62.8	97.8
<i>Supervised in-domain training</i>					
5. Fine-tuned BERT	n/a	66.3	62.3	64.3	80.9
6. Limsopatham and Collier (2016)	n/a	73.5	59.7	65.9	

Table 4: Named entity segmentation performance on the WNUT test set and CoNLL test set A. Limsopatham and Collier (2016) had the winning system at the 2016 WNUT shared task. Their results are reprinted from their paper, which did not report performance on the CoNLL dataset.

Unsupervised in-domain pre-training is effective

UDALM: Unsupervised Domain Adaptation through Language Modeling

[Carouzos et al., 2021](#)



$$L(\mathbf{s}, \mathbf{t}) = \lambda L_{CLF}(\mathbf{s}) + (1 - \lambda) L_{MLM}(\mathbf{t}) \quad (2)$$

We process n labeled source samples $\mathbf{s} \sim D_S$ and m unlabeled target samples $\mathbf{t} \sim D_T$ on a batch. The weighting factor λ is selected as the ratio of labeled source data over the sum of labeled source and unlabeled target data, as stated in Eq. 3:

$$\lambda = \frac{n}{n + m} \quad (3)$$

(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

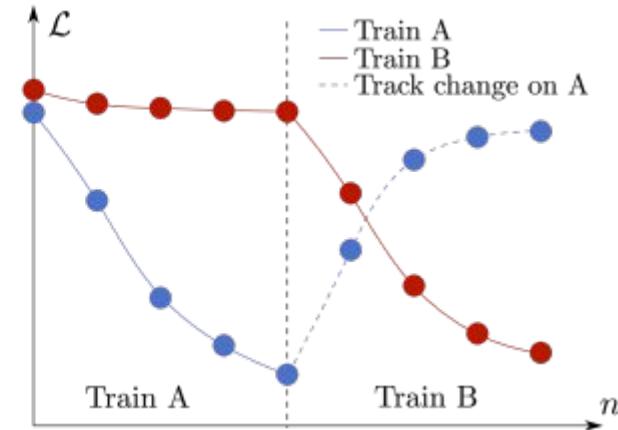
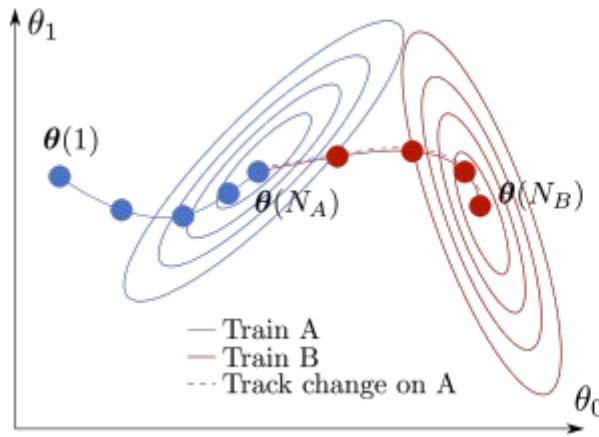
- Train model on the labeled target domain data
- Train with lower learning rate on the labeled source domain data

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

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

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

Fine-tuning: catastrophic forgetting



Picture from [Felix Wiewel, Bin Yang](#), 2019

Fine-tuning: catastrophic forgetting

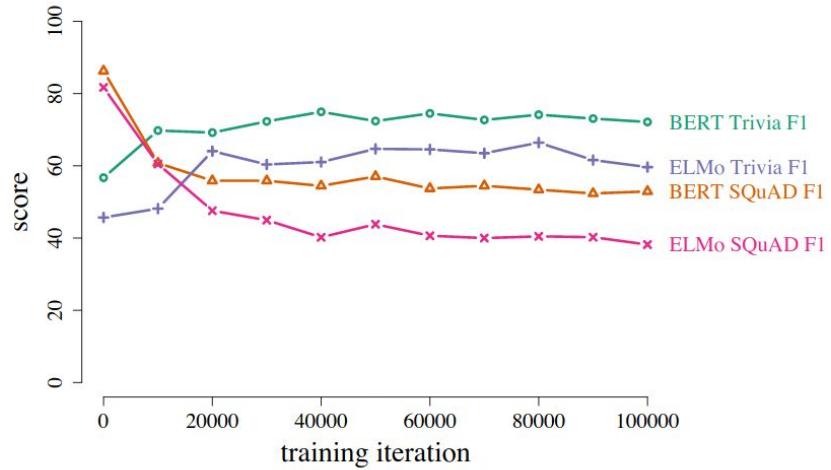
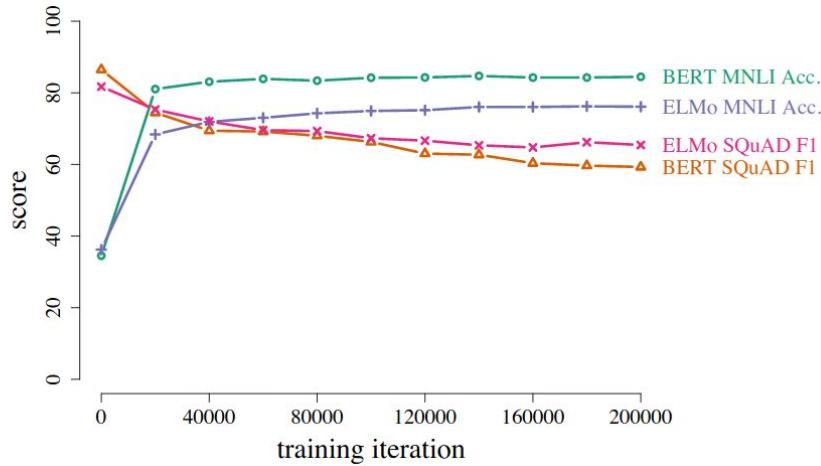


Figure 4: Catastrophic forgetting in a continual learning setup on unsupervised → SQuAD → MNLI (top) and unsupervised → SQuAD → TriviaQA (bottom).

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) $L_W = \lambda \cdot \|W - \hat{W}\|_2^2$

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			avg
		tst2011	tst2012	tst2013	
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.

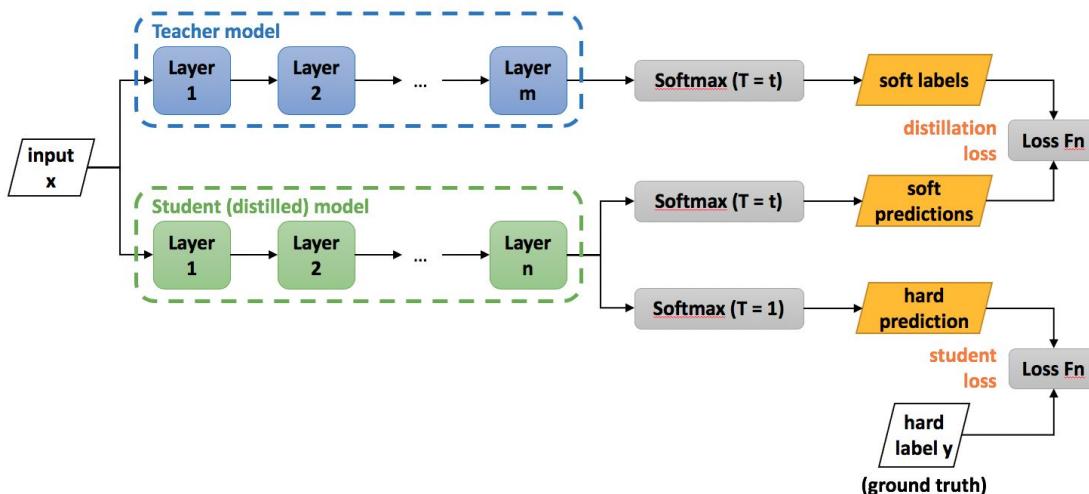
Limited degradation on out-of-domain

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

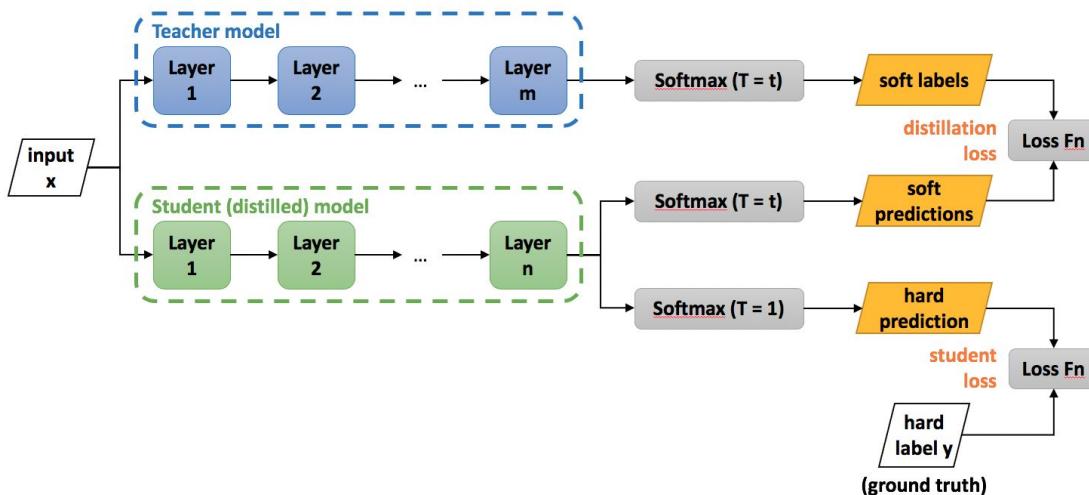
“Knowledge distillation” framework of (Hinton et al., 2014): a smaller “student” network learns to mimic a large “teacher” network by minimizing the loss between the output distributions of the two networks.



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.



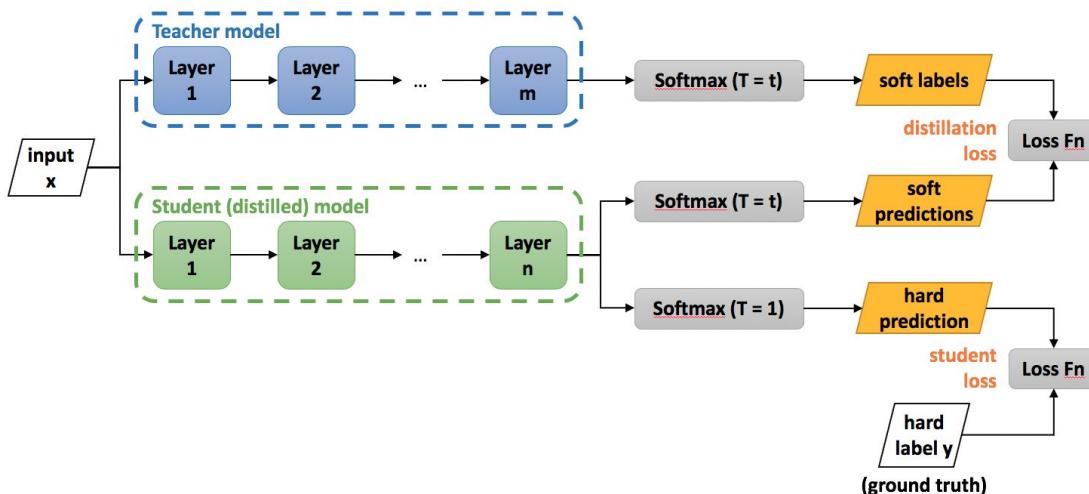
Originally, to compress multiple models (ensemble) to a smaller model.

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.

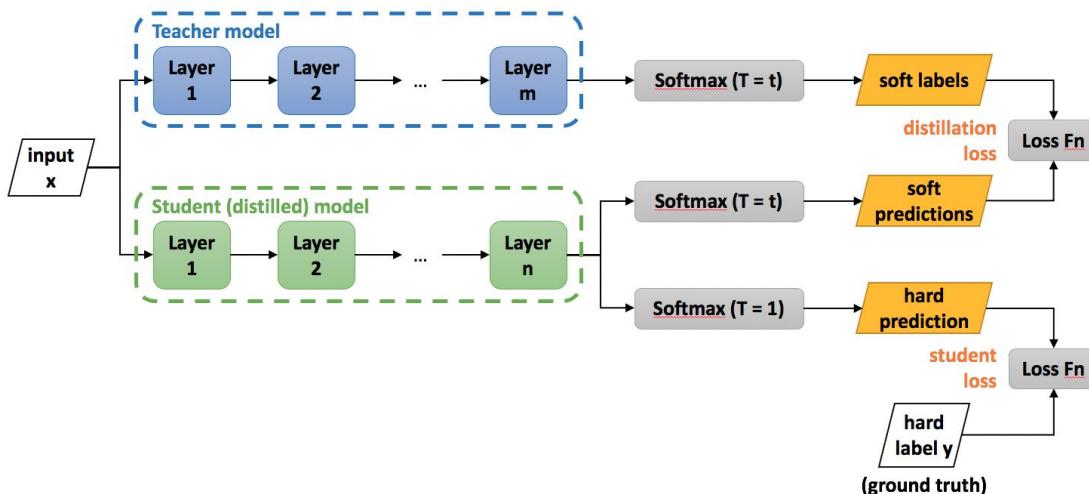
Picture from nervanasystems.github.io

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 .

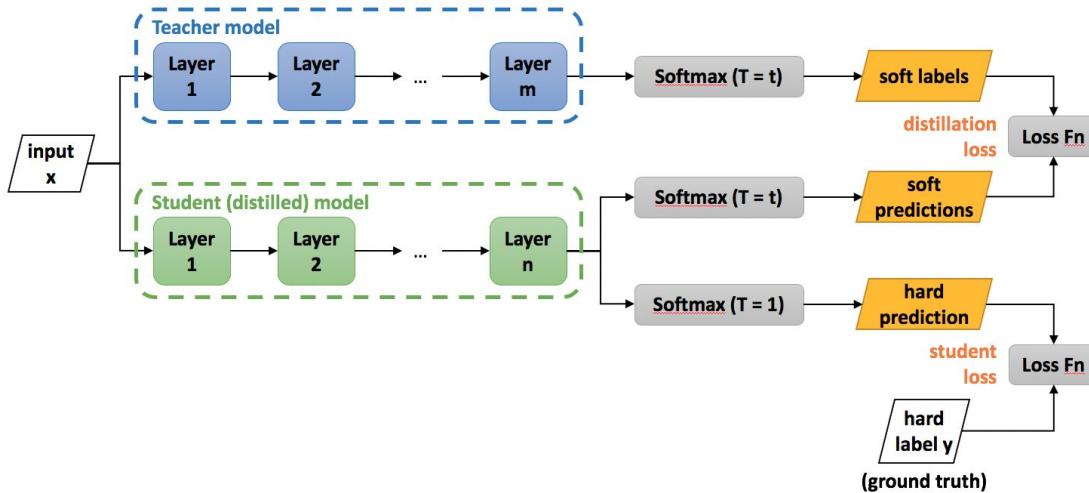


$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

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

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

What is going to happen

- Domain adaptation idea
- Motivation
- Data-centric approaches for domain adaptation
- Model-centric approaches for domain adaptation

Should we minimize distribution divergence directly?

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

The divergence between
source and target domain

Should we minimize distribution divergence directly?

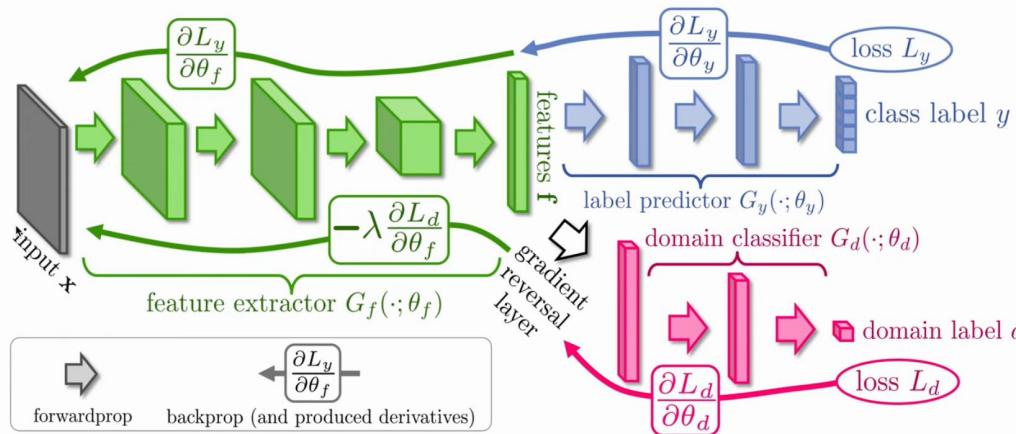
Deep distribution alignment! -> adversarial methods, variational bayes methods and other cool stuff.

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

The divergence between
source and target domain

Domain adversarial neural networks

- Train a classifier for the **domain** of an image based on deep convolutional features.
- Try to maximize the loss of this classifier when training the CNN (**confusion loss**).
- Simultaneously, minimize the classification loss on the source domain using the same convolutional features.
- Train the digit classifier with source domain data, and the domain classifier with both domains' data.



[Ganin, Lempitsky 2015](#)

Domain adversarial neural networks

DAAT = Domain-aware adversarial training

[Du et al. 2020](#)

DAAT:

1. Domain-distinguish task + target domain MLM
2. Adversarial training

Input = [CLS] The mouse is smooth and great
[SEP] The screen is plain [SEP]

Label = TargetDomain

Input = [CLS] This book is boring [SEP] The
system of the laptop is stable [SEP]

Label = MixDomain

Domain adversarial neural networks

DAAT = Domain-aware adversarial training

[Du et al. 2020](#)

DAAT:

1. Domain-distinguish task + target domain MLM
2. Adversarial training

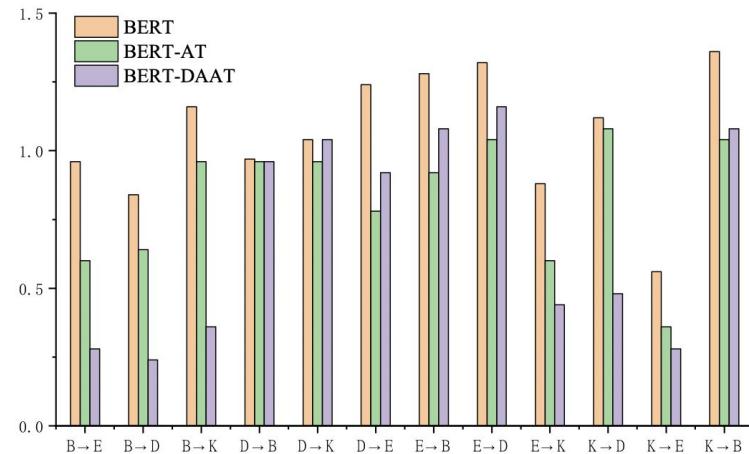


Figure 2: Comparison of \mathcal{A} -distance of different models.

Domain adversarial neural networks

DAAT = Domain-aware adversarial training

[Du et al. 2020](#)

S → T	Previous Models					BERT				
	DANN	PBLM	HATN	ACAN	IATN	BERT	HATN-BERT	BERT-AT	BERT-DA	BERT-DAAT
D → B	81.70	82.50	86.30	82.35	87.00	89.40	89.81	89.55	90.40	90.86
E → B	78.55	71.40	81.00	79.75	81.80	86.50	87.10	87.15	88.31	88.91
K → B	79.25	74.20	83.30	80.80	84.70	87.55	87.88	87.65	87.90	87.98
B → D	82.30	84.20	86.10	83.45	86.80	88.96	89.36	89.70	89.75	89.70
E → D	79.70	75.00	84.00	81.75	84.10	87.95	88.81	88.20	89.03	90.13
K → D	80.45	79.80	84.50	82.10	84.10	87.30	87.89	87.72	88.35	88.81
B → E	77.60	77.60	85.70	81.20	86.50	86.15	87.21	87.30	88.11	89.57
D → E	79.70	79.60	85.60	82.80	86.90	86.55	86.99	86.05	88.15	89.30
K → E	86.65	87.10	87.00	86.60	87.60	90.45	90.31	90.25	90.59	91.72
B → K	76.10	82.50	85.20	83.05	85.90	89.05	89.41	89.55	90.65	90.75
D → K	77.35	83.20	86.20	78.60	85.80	87.53	87.59	87.69	88.55	90.50
E → K	83.95	87.80	87.90	83.35	88.70	91.60	92.01	91.91	92.75	93.18
Average	80.29	80.40	85.10	82.15	85.90	88.25	88.69	88.56	89.37	90.12