Semi-Supervised Sequence Modeling with Cross-View Training

(2018)

Kevin Clark et al.
Resume

May 7, 2019

1 Introduction

The authors propose Cross-View Training (CVT), a semi-supervised learning algorithm that improves the representations of a sentence encoders using a mix of labeled and unlabeled data. On labeled examples, standard supervised learning is used. On unlabeled examples, CVT teaches auxiliary prediction modules that see restricted views of the input (e.g., only part of a sentence) to match the predictions of the full model seeing the whole input. Since the auxiliary modules and the full model share intermediate representations, this in turn improves the full model, they also find that CVT is also effective when doing multi-task leaning.

CVT works by improving the models representation learning. The auxiliary prediction modules can learn from the full models predictions because the full model has a better, unrestricted view of the input. As the auxiliary modules learn to make accurate predictions despite their restricted views of the input, they improve the quality of the representations they are built on top of. This inturn improves the full model, which uses the same shared representations.

2 Cross-View Training

Let $\mathcal{D}_l = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ represent a labeled dataset and $\mathcal{D}_{ul} = \{x_1, x_2, \ldots, x_M\}$ represent an unlabeled dataset We use $p_\theta(y|x_i)$ to denote the output distribution over classes produced by the model with parameters θ on input x_i. During CVT, the model alternates between learning on a mini-batch of labeled examples and learning on a mini-batch of unlabeled examples. For labeled examples, CVT uses standard cross-entropy loss:

$$\mathcal{L}_{\text{sup}}(\theta) = \frac{1}{|\mathcal{D}_l|} \sum_{x_i, y_i \in \mathcal{D}_l} CE\left(y_i, p_\theta\left(y|x_i\right)\right)$$

CVT adds k auxiliary prediction modules to the model, which are used when learning on unlabeled examples. A prediction module is usually a small neural network (e.g., a hidden layer followed by a softmax layer). Each one takes as input an intermediate representation $h^j(x_i)$ produced by the model (e.g., the outputs of one of the LSTMs in a Bi-LSTM model). It outputs a distribution over labels $p_\theta^j(y|x_i)$. Each h^j is chosen such that it only uses a part of the input xi; the particular choice can depend on the model and the task, on unlabeled examples, the model first produces soft targets $p_\theta^j(y|x_i)$ by performing inference. CVT trains the auxiliary prediction modules to match the primary prediction module on the unlabeled data by minimizing the distance D between the distribution (e.g. KL-divergence):

$$\mathcal{L}_{\text{CVT}}(\theta) = \frac{1}{|\mathcal{D}_{ul}|} \sum_{x_i \in \mathcal{D}_{ul}} \sum_{j=1}^{k} D\left(p_\theta\left(y|x_i\right), p_\theta^j\left(y|x_i\right)\right)$$

The primary model is hold fixed during unlabeled training so that the auxillary modules can imtate it and inchance the model representation learning, the total loss is $\mathcal{L} = \mathcal{L}_{\text{sup}} + \mathcal{L}_{\text{CVT}}$, in training we alternate between minimizing the $\mathcal{L}_{\text{sup}}$ for a labled mini batch of example and $\mathcal{L}_{\text{CVT}}$ for an unlabled mini-batch.

CVT can also be used for multitask leaning, during supervision we can randomly select a task and update the primary prediction module responsible of this task, and then for an unlabled example we can create prediction from all the primary module and train all the auxillary ones, so we are able to create artificial labels for all examples of the dataset, even if the labeled example are only for a specific task only.

Side note: KL-Divergence Messages are composed of bits, but not all bits are useful, some of them are redundent and some are errors, and our objective is to transmit as much information as we can with maximum efficency, in Claude Shannon definition, one usefull bit of information reduces the uncertainty by two, say we have 50%/50% change of sunny/rainny, with one bit we'll able to reduce the uncertainy by 2 and be certain if it is sunny/rainy, so the weather channel can only send one bit of information independently of the message length, if we have 8 equally possible weather conditions, then the weather channel will reduce our uncertainty by a factor of 8, and so we'll get a maximum of 3 bit of useful information ($\log(8) = 3$), but if the possibilities are not equal, say sunny 75% and 25% rainny, if the weather channels say that it is rainy, the uncertainty will be reduced by a factor of 4 and we'll get two bit of information, and if it is sunny only 3/4 of uncertainty reduction and only 0.41 bit of information ($\log2(4/3) = 0.41$), so in average we'll get; $0.25 \times 2 + 0.75 \times 0.41 = 0.81$ bits of information from the weather station, this is the entropy, giving us the average amount of information that we get from one sample drawn from a given probability distribution P and how predictible the probability distribution is, the larger the uncertainty the larger the entropy, in a dersert, we'll be quite sure that it'll be sunny so the entropy will quite small, and vice versa:

$$H(p) = -\sum_i p_i \log_2 (p_i)$$

Cross entropy: is the average message length, in this case we'll measure the uncertainly of our predicted probability with regard to the true probability (the number of bits which is the predicted probability q used to transmit a given message/information about the underlining true probability distribution which is the weather condition p):

$$H(p, q) = -\sum_i p_i \log_2 (q_i)$$

If our predictions are perfect, than cross entropy is equal to the entropy, and if not, the different between the predicted and true distribution is the KL divergence (or relative entropy), CR = Entropy + KL-divergence, so KL divergence is:

$$\mathrm{D_{KL}(p\|q) = H(p, q) - H(p)}$$

3 CVT models

Encoder The encoder used in a two layer CNN-BiLSTM, taking as inputs the word embeddings $x_i = \left[x_i^1, x_i^2, \ldots, x_i^T\right]$, then each word is represented as the sum of its embedding vector and charecter-level CNN resulting in $v = \left[v^1, v^2, \ldots, v^T\right]$, the encoder takes v^t ; $v^{T-t} + 1$ in the forward ; backward direction and produces a sequence of states $\left[\vec{h}_1^1, \vec{h}_1^2, \ldots \vec{h}_1^T\right]$ and $\left[\overleftarrow{h}_1^1, \overleftarrow{h}_1^2, \ldots h_1^T\right]$ for the forward ; backward, the second layer works in the same way, taking as inputs the states of the first layer and producing h_2 states.

CVT for Sequence Tagging In sequence tagging, each token x_i^t has a corresponding label y_i^t, it works like a neural language model that, instead of predicting which token comes next in the sequence, predicts whichclass of token comes next. The primary prediction module for sequence tagging produces a probability distribution over classes for the t^{th} label using a one-hidden-layer neural network applied to the corresponding encoder outputs

$$p\left(y^t | x_i\right) = \mathrm{NN}\left(h_1^t \oplus h_2^t\right) = \mathrm{softmax}\left(U \cdot \mathrm{ReLU}\left(W\left(h_1^t \oplus h_2^t\right)\right) + b\right)$$

The auxiliary prediction modules takes $\overrightarrow{h_1}(x_i)$ and $\overleftarrow{h}_1(x_i)$, the outputs of the forward and backward LSTMs in the first encoder layer at different positions, here are the four auxillary modules used:

$$p_\theta^{\text{fwd}}\left(y^t|x_i\right) = \text{NN}^{\text{fwd}}\left(\vec{h}_1^t\left(x_i\right)\right)$$
$$p_\theta^{\text{bwd}}\left(y^t|x_i\right) = \text{NN}^{\text{bwd}}\left(\overleftarrow{h}_1^t\left(x_i\right)\right)$$
$$p_\theta^{\text{future}}\left(y^t|x_i\right) = \text{NN}^{\text{future}}\left(\vec{h}_1^{t-1}\left(x_i\right)\right)$$
$$p_\theta^{\text{past}}\left(y^t|x_i\right) = \text{NN}^{\text{past}}\left(\overleftarrow{h}_1^{t+1}\left(x_i\right)\right)$$

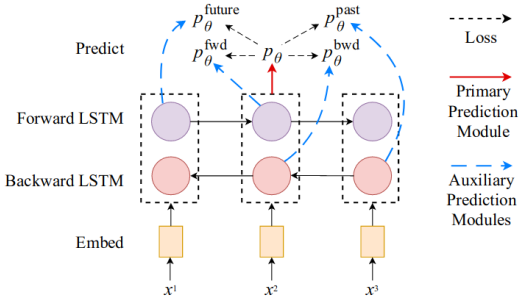

Figure 2: Auxiliary prediction modules for sequence tagging models. Each one sees a restricted view of the input. For example, the "forward" prediction module does not see any context to the right of the current token when predicting that token's label. For simplicity, we only show a one layer Bi-LSTM encoder and only show the model's predictions for a single time step.

The forward module makes each prediction without seeing the right context of the current token. The future module makes each prediction without the right context or the current token itself.

4 Results

Details and Baselines Dropout is applied during training but not for the auxillary model that produces the soft targets, and we also have an auxillary module that sees the whole input but dropout is applied this time, we compare CVT to these methods:

- Word Dropout: In this method, we only train the primary prediction module. When acting as a teacher it is run as normal, but when acting as a student, we randomly replace some of the input words with a REMOVED token. This is similar to CVT in that it exposes the model to a restricted view of the input but it is restriced to the number of possible views and less data efficient.
- Virtual Adversarial Training (VAT): works like word dropout, but adds noise to the word embeddings of the student instead of dropping out words, the noise is chosen adversarially so it most changes the models prediction.
- ELMo: ELMo incorporates the representations from a large separately-trained language model into a task-specific model.

Method	CCG Acc.	Chunk F1	NER F1	FGN F1	POS Acc.	Dep. Parse UAS	Dep. Parse LAS	Translate BLEU
Shortcut LSTM (Wu et al., 2017)	95.1				97.53			
ID-CNN-CRF (Strubell et al., 2017)			90.7	86.8				
JMT† (Hashimoto et al., 2017)		95.8			97.55	94.7	92.9	
TagLM* (Peters et al., 2017)		96.4	91.9					
ELMo* (Peters et al., 2018)			92.2					
Biaffine (Dozat and Manning, 2017)						95.7	94.1	
Stack Pointer (Ma et al., 2018)						95.9	94.2	
Stanford (Luong and Manning, 2015)								23.3
Google (Luong et al., 2017)								26.1
Supervised	94.9	95.1	91.2	87.5	97.60	95.1	93.3	28.9
Virtual Adversarial Training*	95.1	95.1	91.8	87.9	97.64	95.4	93.7	–
Word Dropout*	95.2	95.8	92.1	88.1	97.66	95.6	93.8	29.3
ELMo (our implementation)*	95.8	96.5	92.2	88.5	97.72	96.2	94.4	29.3
ELMo + Multi-task*†	95.9	96.8	92.3	88.4	**97.79**	96.4	94.8	–
CVT*	95.7	96.6	92.3	88.7	97.70	95.9	94.1	**29.6**
CVT + Multi-task*†	96.0	96.9	92.4	88.4	97.76	96.4	94.8	–
CVT + Multi-task + Large*†	**96.1**	**97.0**	**92.6**	**88.8**	97.74	**96.6**	**95.0**	–

Table 1: Results on the test sets. We report the mean score over 5 runs. Standard deviations in score are around 0.1 for NER, FGN, and translation, 0.02 for POS, and 0.05 for the other tasks. See the appendix for results with them included. The +Large model has four times as many hidden units as the others, making it similar in size to the models when ELMo is included. * denotes semi-supervised and † denotes multi-task.

5 CVT for Image Recognition

Method	CIFAR-10 4000 labels	CIFAR-10+ 4000 labels
GAN (Salimans et al., 2016)	–	18.63 ± 2.32
Stochastic Transformations (Sajjadi et al., 2016)	–	11.29 ± 0.24
Π model (Laine and Aila, 2017)	16.55 ± 0.29	12.36 ± 0.31
Temporal Ensemble (Laine and Aila, 2017)	–	12.16 ± 0.24
Mean Teacher (Tarvainen and Valpola, 2017)	–	12.31 ± 0.28
Complement GAN (Dai et al., 2017)	14.41 ± 0.30	–
VAT (Miyato et al., 2017b)	13.15	10.55
VAdD (Park et al., 2017)	–	11.68 ± 0.19
VAdD + VAT (Park et al., 2017)	–	$\mathbf{10.07 \pm 0.11}$
SNGT + Π model (Luong et al., 2017)	13.62 ± 0.17	11.00 ± 0.36
SNGT + VAT (Luong et al., 2017)	$\mathbf{12.49 \pm 0.36}$	$\mathbf{9.89 \pm 0.34}$
Consistency + WGAN (Wei et al., 2018)	–	$\mathbf{9.98 \pm 0.21}$
Manifold Mixup (Verma et al., 2018)	–	$\mathbf{10.26 \pm 0.32}$
Supervised	23.61 ± 0.60	19.61 ± 0.56
VAT (ours)	13.29 ± 0.33	10.90 ± 0.31
CVT, no input perturbation	14.63 ± 0.20	12.44 ± 0.27
CVT, random input perturbation	13.80 ± 0.30	11.10 ± 0.26
CVT, adversarial input perturbation	$\mathbf{12.01 \pm 0.11}$	$\mathbf{10.11 \pm 0.15}$

Table 7: Error rates on semi-supervised CIFAR-10. We report means and standard deviations from 5 runs. CIFAR-10+ refers to results where data augmentation (random translations of the input image) was applied. For some of our models we add a random or adversarially chosen perturbation to the student model's inputs, which is done in most consistency regularization methods.

The image recognition models are based on Convolutional Neural Networks, which produce a set of features $H(x_i) \in \mathbb{R}^{n \times n \times d}$ from an image x_i. The first two dimensions of H index into the spatial coordinates of feature vectors and dis the size of the feature vectors. The primary prediction layers of the CNNs used take as input the mean of H over the first two dimensions, which results in a d-dimensional vector that is fed into a softmax layer:

$$p_\theta(y|x_i) = \text{softmax}(W\text{global\_overage\_pool}(H) + b)$$

We add n^2 auxiliary prediction layers to the top of the CNN. The j_{th} layer takes a single feature vector as input:

$$p_\theta^j(y|x_i) = \text{softmax}\left(W^j H_{\lfloor j/n \rfloor, j \bmod n} + b_j\right)$$