Virtual Adversarial Training for Semi-Supervised Text Classification

Takeru Miyato, Andrew M. Dai, Ian Goodfellow

Slides By Roee Aharoni
BIU NLP summer 2016 reading group
Motivation

ML Hipster @ML_Hipster · 17h
Want to be the best PhD student you can be? Simply make an infinitesimal change to your inputs then take a step in the resulting direction.
Outline

• Introduction to (virtual) adversarial training
• Virtual adversarial training for text classification
• Experimental Setup
• Results (and some analysis)
• Conclusions
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure
A basic NN training procedure

[Diagram showing the process of training a neural network with nodes labeled as 'Training Example', 'Prediction', 'Compute Loss', and 'True Label'. Arrows indicate the flow of data and operations.]
Adversarial Examples

Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet’s classification of the image. Here our $\epsilon$ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet’s conversion to real numbers.
Adversarial Examples

Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet’s classification of the image. Here our $\epsilon$ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet’s conversion to real numbers.
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s
Adversarial Training for NN’s

“panda”
Adversarial Training for NN's

- Compute Loss Function
- True Label
  - “panda”

Prediction

Perturbed Example

Training Example

Images of a panda and a perturbed image are shown.
Adversarial Training for NN’s

Training Example

True Label

“panda”

Compute Loss Function

Prediction

Backpropagate loss

Perturbed Example

Training Example
Adversarial Training for NN’s

- The general addition to the cost function for adversarial training:

\[- \min_{r, \|r\| \leq \varepsilon} \log p(y \mid x + r, \theta)\]

- In practice, take a change depending on the gradient:

\[r_{adv} = -\varepsilon g / \|g\|_2 \text{ where } g = \nabla_x \log p(y \mid x, \theta).\]
Virtual Adversarial Training

- Extends adversarial training to the semi-supervised regime
- The key idea - make the output distribution for an original and perturbed example close to each other
- Enables the use of large amounts of unlabeled data
Virtual Adversarial Training for NN’s
Virtual Adversarial Training for NN’s
Virtual Adversarial Training for NN’s
Virtual Adversarial Training for NN's
Virtual Adversarial Training for NN’s
Virtual Adversarial Training for NN’s

Measure KL Divergence-based loss

Prediction Distribution

Perturbed Example

Unlabeled Example
Virtual Adversarial Training for NN’s

Measure KL Divergence-based loss

Prediction Distribution

Backpropagate loss

Perturbed Example

Unlabeled Example
Virtual Adversarial Training for NN’s

• The general addition to the cost function for virtual adversarial training:

\[
\max_{r, \|r\| \leq \epsilon} \text{KL}[p(\cdot | x, \theta)\|p(\cdot | x + r, \theta)]
\]

\[
D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}
\]

• Again, in practice, there is an efficient way to approximate this (as detailed in Miyato et. al., 2016)
Model - Adversarial Training for Text Classification

• Adversarial perturbations typically consist of making **small modifications** to very many real-valued inputs (i.e. pixels in the previous examples).

• For text classification, the input is discrete, and usually represented as a series of high-dimensional one-hot vectors (where such small modifications are impossible).

• Solution: define the perturbation on continuous word embeddings instead of discrete word inputs.
The perturbation is introduced to normalized embeddings to avoid the network from learning to ignore them:

\[
\bar{v}_k = \frac{v_k - E(v)}{\sqrt{\text{Var}(v)}} \quad \text{where} \quad E(v) = \sum_{j=1}^{K} f_j v_j, \quad \text{Var}(v) = \sum_{j=1}^{K} f_j (v_j - E(v))^2
\]

where \( f_i \) is the frequency of the \( i \)-th word, calculated within all training examples.
Adversarial Training for Text Classification

- As we model the input text as:

\[ s = [\tilde{v}^{(1)}, \tilde{v}^{(2)}, \ldots, \tilde{v}^{(T)}] \]

- The perturbation is defined as:

\[ r_{adv} = -\epsilon g / \|g\|_2 \text{ where } g = \nabla_s \log p(y \mid s, \theta) \]

- And the addition to the loss function is:

\[ L_{adv}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \log p(y_n \mid s_n + r_{adv,n}, \theta) \]
Virtual Adversarial Training for Text Classification

• Here, the perturbation is defined as:

\[ r_{v-adv} = \epsilon g / \| g \|_2 \quad \text{where} \quad g = \nabla_{s+d} KL [p(\cdot \mid s, \theta) \| | p(\cdot \mid s + d, \theta)] \]

• And the addition to the loss function is then:

\[ L_{v-adv}(\theta) = \frac{1}{N'} \sum_{n'=1}^{N'} KL [p(\cdot \mid s_{n'}, \theta) \| | p(\cdot \mid s_{n'} + r_{v-adv,n', \theta})] \]
Experimental Settings

- 5 datasets:
  - Sentiment classification (binary): IMDB, Rotten Tomatoes, Elec
  - Topic classification (multiclass): DBpedia, RCV1

Table 1: Summary of datasets. Note that unlabeled examples for the Rotten Tomatoes dataset are not provided so we instead use the unlabeled Amazon reviews dataset.

<table>
<thead>
<tr>
<th></th>
<th>Classes</th>
<th>Train</th>
<th>Test</th>
<th>Unlabeled</th>
<th>Avg. $T$</th>
<th>Max $T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB [17]</td>
<td>2</td>
<td>25,000</td>
<td>25,000</td>
<td>50,000</td>
<td>239</td>
<td>2,506</td>
</tr>
<tr>
<td>Elec [9]</td>
<td>2</td>
<td>24,792</td>
<td>24,897</td>
<td>197,025</td>
<td>110</td>
<td>5,123</td>
</tr>
<tr>
<td>Rotten Tomatoes [23]</td>
<td>2</td>
<td>9596</td>
<td>1066</td>
<td>7,911,684</td>
<td>20</td>
<td>54</td>
</tr>
<tr>
<td>DBpedia [14]</td>
<td>14</td>
<td>560,000</td>
<td>70,000</td>
<td>–</td>
<td>49</td>
<td>953</td>
</tr>
</tbody>
</table>
Experimental Settings - Preprocessing

• Treat punctuation as spaces
• Convert words to lower case
• Remove words which appear in only one document
• RCV1 - remove stop words
Pre-Training Tricks and Hyperparams

• Initialize word embeddings and LSTM weights with RNNLM on labeled and unlabeled examples

• Single layer LSTM, 1024 units (512 for BiLSTM)

• Embedding size: 256(IMDB, BiLSTM)/512(Rest)

• Sampled softmax loss with 1024 candidate samples (?)

• Adam optimization, 256 samples per batch

• 0.5 dropout rate on the word embeddings
Classification Model Tricks and Hyperparams

- 1 Hidden layer before softmax, 30(IMDB, Elec, Rotten)/128(Rest) units
- ReLU activation function
- batch size - 64(IMDB, Elec, RCV1)/128(Rest)
- 10k-20k training steps for each model
- Truncated back propagation - stop back propagating after 400 steps
- Generate perturbation after dropout
- Optimize epsilon, dropout rate on validation set
Results - IMDB

- Adversarial and virtual adversarial training show lower negative log-likelihood
- Virtual adversarial training also improves the adversarial training loss
Results - IMDB

Table 2: Test performance on the IMDB sentiment classification task.

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (without embedding normalization)</td>
<td>7.33%</td>
</tr>
<tr>
<td>Baseline</td>
<td>7.39%</td>
</tr>
<tr>
<td>Random perturbation with labeled examples</td>
<td>7.20%</td>
</tr>
<tr>
<td>Random perturbation with labeled and unlabeled examples</td>
<td>6.78%</td>
</tr>
<tr>
<td>Adversarial</td>
<td>6.21%</td>
</tr>
<tr>
<td>Virtual Adversarial</td>
<td>5.91%</td>
</tr>
<tr>
<td>Adversarial + Virtual Adversarial</td>
<td>6.09%</td>
</tr>
<tr>
<td>Virtual Adversarial (on bidirectional LSTM)</td>
<td>5.91%</td>
</tr>
<tr>
<td>Adversarial + Virtual Adversarial (on bidirectional LSTM)</td>
<td>6.02%</td>
</tr>
<tr>
<td>Full+Unlabeled+BoW [17]</td>
<td>11.11%</td>
</tr>
<tr>
<td>Paragraph Vectors [13]</td>
<td>7.42%</td>
</tr>
<tr>
<td>SA-LSTM [4]</td>
<td>7.24%</td>
</tr>
<tr>
<td>One-hot bi-LSTM (with pretrained embeddings of CNN and bi-LSTM) [10]</td>
<td>5.94%</td>
</tr>
</tbody>
</table>
Embedding-Based Analysis

Table 3: 10 top nearest neighbors to ‘good’ and ‘bad’ with the word embeddings trained on each method. We used cosine distance for the metric. ‘Baseline’ means training with embedding dropout and ‘Random’ means training with random perturbation with labeled examples. ‘Adversarial’ and ‘Virtual Adversarial’ mean adversarial training and virtual adversarial training.

<table>
<thead>
<tr>
<th>‘good’</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Random</td>
<td>Adversarial</td>
<td>Virtual Adversarial</td>
<td>Baseline</td>
<td>Random</td>
<td>Adversarial</td>
<td>Virtual Adversarial</td>
</tr>
<tr>
<td>1</td>
<td>great</td>
<td>great</td>
<td>decent</td>
<td>decent</td>
<td>terrible</td>
<td>terrible</td>
<td>terrible</td>
</tr>
<tr>
<td>2</td>
<td>decent</td>
<td>decent</td>
<td>great</td>
<td>great</td>
<td>awful</td>
<td>awful</td>
<td>awful</td>
</tr>
<tr>
<td>3</td>
<td>×bad</td>
<td>excellent</td>
<td>nice</td>
<td>nice</td>
<td>horrible</td>
<td>horrible</td>
<td>horrible</td>
</tr>
<tr>
<td>4</td>
<td>excellent</td>
<td>nice</td>
<td>fine</td>
<td>fine</td>
<td>×good</td>
<td>×good</td>
<td>poor</td>
</tr>
<tr>
<td>5</td>
<td>Good</td>
<td>Good</td>
<td>entertaining</td>
<td>entertaining</td>
<td>Bad</td>
<td>poor</td>
<td>BAD</td>
</tr>
<tr>
<td>6</td>
<td>fine</td>
<td>×bad</td>
<td>interesting</td>
<td>interesting</td>
<td>BAD</td>
<td>BAD</td>
<td>stupid</td>
</tr>
<tr>
<td>7</td>
<td>nice</td>
<td>fine</td>
<td>Good</td>
<td>Good</td>
<td>poor</td>
<td>Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>8</td>
<td>interesting</td>
<td>interesting</td>
<td>Excellent</td>
<td>cool</td>
<td>stupid</td>
<td>stupid</td>
<td>laughable</td>
</tr>
<tr>
<td>9</td>
<td>solid</td>
<td>entertaining</td>
<td>solid</td>
<td>enjoyable</td>
<td>Horrible</td>
<td>Horrible</td>
<td>lame</td>
</tr>
<tr>
<td>10</td>
<td>entertaining</td>
<td>solid</td>
<td>cool</td>
<td>excellent</td>
<td>Horrendous</td>
<td>Horrendous</td>
<td>Horrible</td>
</tr>
</tbody>
</table>
Results - Topic classification: Elec, RCV1

- Improved SOTA on Elec, RCV1, without using CNN’s

Table 4: Test performance on the Elec and RCV1 classification tasks

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Elec</td>
</tr>
<tr>
<td>Baseline</td>
<td>6.24%</td>
</tr>
<tr>
<td>Adversarial</td>
<td>5.61%</td>
</tr>
<tr>
<td>Virtual Adversarial</td>
<td>5.54%</td>
</tr>
<tr>
<td>Adversarial + Virtual Adversarial</td>
<td><strong>5.40%</strong></td>
</tr>
<tr>
<td>Virtual Adversarial (on bidirectional LSTM)</td>
<td>5.55%</td>
</tr>
<tr>
<td>Adversarial + Virtual Adversarial (on bidirectional LSTM)</td>
<td>5.45%</td>
</tr>
<tr>
<td>One-hot CNN (with pretrained embeddings of CNN)</td>
<td>6.27%</td>
</tr>
<tr>
<td>One-hot CNN (with pretrained embeddings of CNN and bi-LSTM)</td>
<td>5.82%</td>
</tr>
<tr>
<td>One-hot bi-LSTM (with pretrained embeddings of CNN and bi-LSTM)</td>
<td>5.55%</td>
</tr>
</tbody>
</table>
Results - Sentiment analysis: Rotten Tomatoes

- Adversarial + Virtual adv. performs equally to SOTA
- Virtual adversarial is weaker than baseline - could be due to small amount of supervised examples, short sentences

Table 5: Test performance on the Rotten Tomatoes sentiment classification task

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>17.9%</td>
</tr>
<tr>
<td>Adversarial</td>
<td>16.8%</td>
</tr>
<tr>
<td>Virtual Adversarial</td>
<td>19.1%</td>
</tr>
<tr>
<td>Adversarial + Virtual Adversarial</td>
<td><strong>16.6%</strong></td>
</tr>
<tr>
<td>NBSVM-bigrams[28]</td>
<td>20.6%</td>
</tr>
<tr>
<td>CNN (with pretrained embeddings from word2vec Google News)[11]</td>
<td>18.5%</td>
</tr>
<tr>
<td>AdaSent (with pretrained embeddings from word2vec Google News)[31]</td>
<td>16.9%</td>
</tr>
<tr>
<td>SA-LSTM (with unlabeled data from Amazon reviews)[4]</td>
<td>16.7%</td>
</tr>
</tbody>
</table>
Results - DBpedia

- Baseline itself improves over SOTA, virtual adversarial performs best
Related Work

- Dropout/Random Noise
- Generative Models
- Pre-Training as semi-supervised learning
Conclusion

• Adversarial and virtual adversarial training provides good regularization performance for text classification with RNN’s

• Provides SOTA results or on-par results for the examined datasets

• “Improved Quality” of word embeddings