Distributional Smoothing with Virtual Adversarial Training

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, Shin Ishii (Kyoto Univ.)
Presenter: Takeru Miyato
Classification

Input: $x$

Model: $p(y|x, \theta)$

Output: $y$

"Cat"
Purpose

- Get a good classifier
  - A good classifier: a classifier which can predict a label correctly on a example *not included in training sets*. (汎化性能が高い)

- How to get a good classifier?
  - Data augmentation
  - Bayes inference
  - *Regularization* (正則化)<- I am working on this and propose a new method.
What are the properties of good classifier?

- This shall NOT happen!

What are the properties of a good classifier?
Adversarial training [Goodfellow, 2015]

Adversarial Perturbation:

\[ r_{adv} = -\epsilon \text{sign}(\nabla_x \log p(y|x, \theta)) \]
Adversarial examples from overfitting
Adversarial examples from underfitting
[proposal] Adversarial examples for unlabeled inputs

- Define the difference between $p(y|x, \theta)$ and $p(y|x+r, \theta)$ as:

$$\Delta_{KL}(r, x^{(n)}, \theta) \equiv KL[p(y|x^{(n)}, \theta) \| p(y|x^{(n)} + r, \theta)]$$

- Define *virtual* adversarial perturbation as:

$$r_{v-adv}^{(n)} \equiv \arg \max_r \{\Delta_{KL}(r, x^{(n)}, \theta); \|r\|_2 \leq \epsilon\}$$

($\epsilon$ : norm constraint)
Virtual Adversarial Training (VAT)

- Local Distributional Smoothness

\[
\text{LDS}(x^{(n)}, \theta) \equiv -\Delta_{\text{KL}}(r^{(n)}_{\text{v-adv}}, x^{(n)}, \theta)
\]

- Maximize:

\[
\frac{1}{N} \sum_{n=1}^{N} \log p(y^{(n)}|x^{(n)}, \theta) + \lambda \frac{1}{M} \sum_{m=1}^{M} \text{LDS}(x^{(m)}, \theta)
\]

(\lambda: balance factor)
Computation of LDS

- We will need to compute

\[ r_{v-adv} = \arg \max_r \{ \Delta_{KL}(r, x, \theta); \|r\|_2 \leq \epsilon \} \]

- For can be compute \( r_{v-adv} \) fast?
  - Our method allows for fast approximation!
Approximation of $r_{v-adv}$

- Approximate $\Delta_{KL}$ with 2nd Taylor expansion:

$$\Delta_{KL}(r, x, \theta) \approx \frac{1}{2} r^T H(x, \theta) r$$

$$H(x, \theta) \equiv \nabla \nabla_r \Delta_{KL}(r, x, \theta)|_{r=0}$$
Approximation of \( r_{v-adv} \)

- In 2nd Taylor approximation, we see that \( r_{v-adv} \) is the first dominant eigenvector of \( H(x, \theta) \) of magnitude \( \epsilon \):

\[
r_{v-adv}(x, \theta) \approx \arg \max_r \left\{ r^T H(x, \theta) r; \| r \|_2 \leq \epsilon \right\} \\
= \frac{\epsilon u(x, \theta)}{}
\]

\( u(x, \theta) \) is 1st eigen vector of \( H(x, \theta) \)
Approximation of $r_{v-adv}$

- Power iteration method:

$$d \leftarrow H d$$

- Finite difference method:

$$H d \equiv \frac{\nabla_r \Delta_{KL}(r + \xi d, x, \theta)|_{r=0} - \nabla_r \Delta_{KL}(r, x, \theta)|_{r=0}}{\xi}$$

$$= \frac{\nabla_r \Delta_{KL}(r + \xi d, x, \theta)|_{r=0}}{\xi},$$
Algorithm for the generation of approximated $r_{v-adv}^{(n)}$

Algorithm 1 Generation of $r_{v-adv}^{(n)}$

Function GenVAP($\theta, x^{(n)}, \epsilon, I_p, \xi$)

1. Initialize $d \in R^I$ by a random unit vector.
2. Repeat For i in 1...$I_p$ (Perform $I_p$-times power method)
   \[ d \leftarrow \nabla_r \Delta_{KL}(r + \xi d, x^{(n)}, \theta)|_{r=0} \]
3. Return $\epsilon d$
Demo: 2D synthetic dataset

- Use 1 hidden layer NN.
- Use gradient descent with Momentum.
Learning curves

(a) Moons

(b) Circles

- -- MLE(train)  -- MLE(test)  -- VAT(train)  -- VAT(test)
Visualization of $p(y|x, \theta)$, MLE and L2 regularization

average LDS

Method

no regularization

L$_2$ regularization
Visualization of $p(y|x, \theta)$, MLE and VAT (proposed)

average LDS

Method

no regularization

VAT (proposed)
Visualization of $p(y|x, \theta)$

Method

<table>
<thead>
<tr>
<th>Method</th>
<th>no regularization</th>
<th>$L_2$ regularization</th>
<th>VAT (proposed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDS</td>
<td>−1.588</td>
<td>−1.351</td>
<td>−0.257</td>
</tr>
</tbody>
</table>
Benchmark results: Supervised learning for MNIST

- MNIST (permutation invariant task)
  - 60000 training samples
  - Use ADAM [Kingma, 2015] optimizer
Supervised learning for MNIST

- MNIST (permutation invariant task)
Semi-supervised learning (permutation invariant task)

- MNIST
- CIFAR10
  - 4000 labeled and 50000 unlabeled
- SVHN (street view housing numbers)
  - 1000 labeled and 70000 unlabeled
Semi-supervised learning (permutation invariant task)

- MNIST (1000 labeled samples)

**Diagram:**
- T SVM: Transductive SVM
- MTC: Manifold Tangent Classifier
- DG: Deep Generative Model
- VAT (ours)
- Ladder networks

**Error rate (%)**
- T SVM: 6%
- MTC: 3%
- DG: 1.5%
- VAT (ours): 1%
- Ladder networks: 0%
Semi-supervised learning on CIFAR10

- CIFAR10 (4000 labeled)

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ladder network (Rasmus et al., 2015)</td>
<td>20.40%</td>
</tr>
<tr>
<td>CatGAN (Springenberg, 2015)</td>
<td>19.58%</td>
</tr>
<tr>
<td>GAN with feature matching (Salimans et al., 2016)</td>
<td>18.63%</td>
</tr>
<tr>
<td>Baseline</td>
<td>23.71%</td>
</tr>
<tr>
<td>Virtual Adversarial Training (ours)</td>
<td><strong>17.60%</strong></td>
</tr>
</tbody>
</table>
Table 3: SVHN with 1000 labeled examples.

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSVM (Kingma et al., 2014)</td>
<td>66.55 %</td>
</tr>
<tr>
<td>SWWAE (Zhao et al., 2015)</td>
<td>23.56 %</td>
</tr>
<tr>
<td>Skip Generative Model (Maaløe et al., 2016)</td>
<td>16.30 %</td>
</tr>
<tr>
<td>GAN with feature matching (Salimans et al., 2016)</td>
<td>8.11%</td>
</tr>
<tr>
<td>Baseline</td>
<td>11.66%</td>
</tr>
<tr>
<td>Virtual Adversarial Training (ours)</td>
<td><strong>7.09%</strong></td>
</tr>
</tbody>
</table>
Conclusion

- Our approach was effective for supervised and semi-supervised learning for benchmark datasets.
- With optimizing only 1 hyperparameter $\varepsilon$, our method achieved good performance.
References

  ○ github: [https://github.com/takerum/vat](https://github.com/takerum/vat)