Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning

Ali Abbasi

Information Theory, Statatistics, and Learning
Prof. Yassaee
Sharif University of Technology

Outline

Adversarial Attacks

- Cause a machine learning model to make incorrect predictions.
- Achieved by perturbations that maximally degrade the information contained in an input signal.

Adversarial Attacks

- Cause a machine learning model to make incorrect predictions.
- Achieved by perturbations that maximally degrade the information contained in an input signal.
- Deep learning systems have been shown to be vulnerable to adversarial attacks
- Their outputs can be manipulated with imperceptibly small perturbations applied to the inputs

Adversarial Attacks

stop sign
Confidence: 0.9153

Adversarial perturbation

flowerpot
Confidence: 0.8374

Optimal Adversarial Attacks

Some Notations:
U: the quantity (or label) of interest
X: data generated from U

Goal?
Adding random perturbation E to X, producing random variable $Y = X + E$, such that the mutual information between U and $Y = X + E$ is minimized

Optimal Adversarial Attacks

Some Notations:
U: the quantity (or label) of interest
X: data generated from U

Goal?
Adding random perturbation E to X, producing random variable $Y = X + E$, such that the mutual information between U and $Y = X + E$ is minimized

NOTE: Since adding perturbations often introduces costs, we put constraints on the perturbation E

Optimal Adversarial Attacks

the problem can be formulated as:

$$\min_{p(E|U,X)\in\mathcal{P}} I(U;X+E)$$

where $\mathcal{P}$ is a set of admissible probability distributions.

Optimal Adversarial Attacks

the problem can be formulated as:

$$\min_{p(E|U,X)\in\mathcal{P}} I(U; X + E)$$

where $\mathcal{P}$ is a set of admissible probability distributions.
for example, it can be:

$$\mathcal{P} = \{p(E|U,X) : \mathbb{E}[\|E\|^2] \leq \epsilon, X \in \Omega\}$$

Robustness

The property of a model to perform well and produce reliable results even when faced with unexpected or anomalous inputs

A robust loss (say, absolute error) may be preferred over a non-robust loss (say, squared error) due to its reduced sensitivity to large errors

Semi-Supervised Learning

- Labeling is hard, but there are often a lot of data
- Want to use unlabeled data to help training

Regularization

Adding a penalty term to the loss function
$\Rightarrow$ Discourages the model from having large weights for any of its features
$\Rightarrow$ Prevent overfitting
$\Rightarrow$ More robust to variations in the input data (Improve robustness)

Regularization

Can be interpreted as a prior distribution that reflects our educated a priori knowledge or belief regarding the model (from a Bayesian standpoint)

Regularization

Can be interpreted as a prior distribution that reflects our educated a priori knowledge or belief regarding the model (from a Bayesian standpoint)

A popular a priori belief $\rightarrow$ the outputs of systems are smooth with respect to spatial and/or temporal inputs

Other Methods

- Label Propagation:

  Assign class labels to unlabeled training samples based on the belief that close input data points tend to have similar class labels

Other Methods

- Label Propagation:

  Assign class labels to unlabeled training samples based on the belief that close input data points tend to have similar class labels

- Artificial Input:

  Apply random perturbations to each input in order to generate artificial input points and encourage the model to assign similar outputs to the set of artificial inputs derived from the same point

Other Methods

? But, what is the problem with these methods?

They often leave the predictor particularly vulnerable to a small perturbation in a specific direction called adversarial direction $\rightarrow$ it's the direction in the input space in which the label probability $p(y = k|x)$ of the model is most sensitive

Other Methods

- Contractive Loss:

  Impose constraints on the Frobenius norm of the Jacobian matrix of the output with respect to the input

  Deep Contractive Network, which imposes a layer-wise contraction penalty in a feed-forward neural network. The layer-wise penalty approximately minimizes the network outputs variance with respect to perturbations in the inputs, enabling the trained model to achieve “flatness” around the training data points.

Other Methods

? But, what is the problem with this method?

Computing the full Jacobian is computationally expensive and therefore they approximate it.
However, possibly because of their layer-wise approximation, their method was not successful in significantly decreasing the test error

Other Methods

- Generative adversarial networks (GANs)

  Do not require an explicit definition of smoothness.

  Based on an objective function that trades off mutual information between observed examples and their predicted categorical class distribution, against the robustness of the classifier to an adversarial generative model. The resulting algorithm can be interpreted as a natural generalization of the generative adversarial networks (GAN) framework to robust classification against an optimal adversary.

Other Methods

? But, what is the problem with this method?

In practice, these methods often require careful tuning of many hyperparameters in the generative model, and are usually not easy to implement without high expertise in its optimization process.

Other Methods

- Adversarial Training:

  Trains the model to assign to each input data a label that is similar to the labels to be assigned to its neighbors in the adversarial direction.
  Adversarial Direction $\rightarrow$ direction that can most greatly "deviate" the prediction of the model from the correct label.

Other Methods

? But, what is the problem with this method?

Labeled data are needed for calculating the adversarial direction.

Other Methods

? But, what is the problem with this method?

Labeled data are needed for calculating the adversarial direction.

Hence, the best method would be a method that considers Anisotropic smoothing and does not need labeled data to calculate the adversarial direction

Some Notations

I: Input dimension

Q: Space of all labels

Input vector: $x \in R^I$

Output label: $y \in Q$

$p(y|x, \theta)$: Output distribution parameterized by θ

$\hat{\theta}$: vector of the model parameters at a specific iteration step of the training process.

Some Notations

Labeled Dataset: $\mathcal{D}_l = \{x_l^n, y_l^n | n = 1, ..., N_l\}$

Unlabeled Dataset: $\mathcal{D}_{ul} = \{x_{ul}^m | m = 1, ..., N_{ul}\}$

Adversarial Training

The loss function of adversarial training:

$$L_{adv}(x_l, \theta) = \mathrm{D}\left[q(y|x_l), p(y|x_l + r_{adv}, \theta)\right]$$
$$r_{adv} = \underset{r;\|r\| \leq \epsilon}{\arg\max}\, \mathrm{D}\left[q(y|x_l), p(y|x_l + r, \theta)\right]$$

$D[p, p']$:divergence between two distributions p and p'
$q(y|x_l)$: true distribution of the output label, which is unknown.

Goal: approximate the true distribution by a parametric model $p(y|x_l, \theta)$ that is robust against adversarial attack to x.

Adversarial Training

When the norm is L_2, adversarial perturbation can be approximated by:

$$r_{adv} \approx \epsilon \frac{g}{\|g\|_2}$$

$$g = \nabla_{x_l} \mathrm{D}\left[h(y; y_l), p(y|x_l, \theta)\right]$$

g can be efficiently computed by backpropagation.
When the norm is L_∞:

$$r_{adv} \approx \epsilon \, \mathrm{sign}(g)$$

Virtual Adversarial Training

Let x_* represent either a labeled (x_l) or unlabeled (x_{ul}) data point. Our objective function is:

$$\mathrm{D}\left[q(y|x_*), p(y|x_* + r_{qadv}, \theta)\right]$$

$$\text{Where: } r_{qadv} = \underset{r;\|r\|\leq\epsilon}{\arg\max}\, \mathrm{D}\left[q(y|x_*), p(y|x_* + r, \theta)\right]$$

Virtual Adversarial Training

Let x_* represent either a labeled (x_l) or unlabeled (x_{ul}) data point. Our objective function is:

$$\mathrm{D}\left[q(y|x_*), p(y|x_* + r_{qadv}, \theta)\right]$$

$$\text{Where: } r_{qadv} = \underset{r;\|r\|\leq\epsilon}{\arg\max}\, \mathrm{D}\left[q(y|x_*), p(y|x_* + r, \theta)\right]$$

But we have no information about $q(y|x_{ul})$.

Virtual Adversarial Training

Let x_* represent either a labeled (x_l) or unlabeled (x_{ul}) data point.
Our objective function is:

$$\mathrm{D}\left[q(y|x_*), p(y|x_* + r_{qadv}, \theta)\right]$$

$$\text{Where: } r_{qadv} = \underset{r; \|r\| \leq \epsilon}{\arg\max}\, \mathrm{D}\left[q(y|x_*), p(y|x_* + r, \theta)\right]$$

But we have no information about $q(y|x_{ul})$.
We'll use its current estimate, $p(y|x_*, \theta)$ instead (hence the term virtual).

Virtual Adversarial Training

We denote the current parameters as $\hat{\theta}$. So the *Local Distributional Smoothness* is defined as:

$$\mathrm{LDS}(x_*, \theta) = \mathrm{D}\left[p(y|x_*, \hat{\theta}), p(y|x_* + r_{vadv}, \theta)\right]$$

$$r_{vadv} = \underset{r;\|r\|\leq\epsilon}{\arg\max}\, \mathrm{D}\left[p(y|x_*, \hat{\theta}), p(y|x_* + r, \theta)\right]$$

The regularization term proposed in the paper is simply the average of LDS over all data points:

$$\mathcal{R}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) = \frac{1}{N_l + N_{ul}} \sum_{x_* \in \mathcal{D}_l, \mathcal{D}_{ul}} \mathrm{LDS}(x_*, \theta)$$

And the full objective becomes:

$$\mathcal{L} = \ell(D_l, \theta) + \alpha \mathcal{R}(D_l, D_{ul}, \theta)$$

Approximating r_{vadv}

Assume twice differentiability of $p(y|x_*, \theta)$ (and as a result D) with respect to x_* and θ. D archives the minimum value at $r = 0$:

$$\mathrm{D}(r, x, \hat{\theta})|_{r=0} = 0 \implies \nabla_r \mathrm{D}(r, x, \hat{\theta})|_{r=0} = 0$$

Approximating r_{vadv}

Assume twice differentiability of $p(y|x_*, \theta)$ (and as a result D) with respect to x_* and θ. D archives the minimum value at $r = 0$:

$$\mathrm{D}(r, x, \hat{\theta})|_{r=0} = 0 \implies \nabla_r \mathrm{D}(r, x, \hat{\theta})|_{r=0} = 0$$

The evaluation of r_{vadv} cannot be performed with the linear approximation as in the original adversarial training since the gradient of $\mathrm{D}(r, x_*, \hat{\theta}) \triangleq \mathrm{D}\left[p(y|x_*, \hat{\theta}), p(y|x_* + r, \theta)\right]$ with respect to r is always zero at $r = 0$.

Approximating r_{vadv}

We can derive second order Taylor expansion of D as follows:

$$\mathrm{D}(r, x, \hat{\theta}) \approx \frac{1}{2} r^T H(x, \hat{\theta}) r$$

Where $H(x, \hat{\theta})$ is the Hessian matrix of D with respect to r.

Approximating r_{vadv}

We can derive second order Taylor expansion of D as follows:

$$\mathrm{D}(r, x, \hat{\theta}) \approx \frac{1}{2} r^T H(x, \hat{\theta}) r$$

Where $H(x, \hat{\theta})$ is the Hessian matrix of D with respect to r.
We know that the maximum value of D is achieved when r is the eigenvector $u(x, \hat{\theta})$ of the matrix H corresponding to the largest eigenvalue:

$$\begin{aligned} r_{vadv} &\approx \arg\max_{r} \left\{ r^T H(x, \hat{\theta}) r \mid \|r\| \leq \epsilon \right\} \\ &= \epsilon \overline{u(x, \hat{\theta})} \qquad \left(\overline{v} = \frac{v}{\|v\|}\right) \end{aligned}$$

Approximating r_{vadv}

What about $O(I^3)$ complexity of computing the eigenvector?

Approximating r_{vadv}

What about $O(I^3)$ complexity of computing the eigenvector?
We use the iterative *Power Method* to approximate the dominant eigenvector. Suppose d is a random unit vector. If d is not prependicular to the dominant eigenvector, then iterative calculation of

$$d \leftarrow \overline{Hd}$$

converges to the dominant eigenvector (see Appendix).

Approximating r_{vadv}

We can approximate Hd with difference method too:

$$
\begin{aligned}
Hd &\approx \frac{\nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=\xi d} - \nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=0}}{\xi} \\
&= \frac{\nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=\xi d}}{\xi}
\end{aligned}
$$

Approximating r_{vadv}

We can approximate Hd with difference method too:

$$Hd \approx \frac{\nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=\xi d} - \nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=0}}{\xi}$$
$$= \frac{\nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=\xi d}}{\xi}$$

$$\implies d \leftarrow \overline{\nabla_r \mathrm{D}(r, x_*, \hat{\theta})|_{r=\xi d}}$$

Thus, the approximation of r_{vadv} with K steps of the power method can be achieved with K sets of backpropagation.

Approximation for Derivative of Regularization Term

Algorithm 1: Mini-batch SGD for $\nabla_\theta \mathcal{R}$ with one time iteration of power method

1 Choose M random samples from dataset $\mathcal{D}$: $\left\{x^{(i)}\right\}_{i=1}^{M}$.

2 Sample unit vector $d^{(i)}$ from an iid Gaussian distribution.

3 Calculate r_{vadv} with one set of backpropagation:

$$g^{(i)} \leftarrow \nabla_r \mathrm{D}\left[p(y|x^{(i)}, \hat{\theta}), p(y|x^{(i)} + r, \theta)\right]\Big|_{r=\xi d^{(i)}}$$

$$r_{vadv}^{(i)} \leftarrow \epsilon \frac{g^{(i)}}{\left\|g^{(i)}\right\|}$$

4 **return** $\nabla_\theta \left(\frac{1}{M} \sum_{i=1}^{M} \mathrm{D}\left[p(y|x^{(i)}, \hat{\theta}), p(y|x^{(i)} + r_{adv}^{(i)}, \theta)\right]\right)$

Tuning Hyperparameters

We have two hyperparameters: ϵ and α.

$$\begin{aligned}\max_r \{\mathrm{D}(r, x, \theta) \mid \|r\|_2 \leq \epsilon\} &\approx \max_r \left\{\frac{1}{2} r^T H(x, \theta) r \mid \|r\|_2 \leq \epsilon\right\} \\ &= \frac{1}{2}\epsilon^2 \lambda_1(x, \theta)\end{aligned}$$

Tuning Hyperparameters

We have two hyperparameters: ϵ and α.

$$\begin{aligned}\max_r \{\mathrm{D}(r,x,\theta) \mid \|r\|_2 \le \epsilon\} &\approx \max_r \left\{\frac{1}{2} r^T H(x,\theta) r \mid \|r\|_2 \le \epsilon\right\} \\ &= \frac{1}{2}\epsilon^2 \lambda_1(x,\theta)\end{aligned}$$

$$\begin{aligned}\mathcal{L} &= \ell(\theta,\mathcal{D}_l) + \alpha \mathcal{R}_{vadv}(\theta,\mathcal{D}_l,\mathcal{D}_{ul}) \\ &= \ell(\theta,\mathcal{D}_l) + \alpha \frac{1}{N_l + N_{ul}} \sum_{x\in\mathcal{D}_l,\mathcal{D}_{ul}} \max_r \{\mathrm{D}(r,x,\theta) \mid \|r\|_2 \le \epsilon\} \\ &\approx \ell(\theta,\mathcal{D}_l) + \frac{1}{2}\epsilon^2 \alpha \frac{1}{N_l + N_{ul}} \sum_{x\in\mathcal{D}_l,\mathcal{D}_{ul}} \lambda_1(x,\theta)\end{aligned}$$

Experiments

$$
\begin{aligned}
\text{D} =& \text{KL Divergence} \\
\xi =& 10^{-6} \\
\alpha =& 1 \\
\text{Model} =& \text{Simple Fully Connected Network with 4 hidden layers} \\
& \text{for MNIST and Conv-Large for CIFAR-10}
\end{aligned}
$$

Experiments

Table 1: Test performance of supervised learning methods on MNIST with 60,000 labeled examples in the permutation invariant setting.

Method	Test error rate(%)
SVM (Gaussian kernel)	1.40
Dropout	1.05
Adversarial, L_∞ norm constraint	0.78
Ladder networks	0.57 (±0.02)
Baseline (MLE)	1.11 (±0.06)
RPT	0.84 (±0.03)
Adversarial, L_1 norm constraint	0.79 (±0.03)
Adversarial, L_2 norm constraint	0.71 (±0.03)
VAT	0.64 (±0.05)

Experiments

Table 2: Test performance of supervised learning methods implemented with CNN on CIFAR-10 with 50,000 labeled examples.

Network in Network	8.81
All-CNN	7.25
Deeply Supervised Net	7.97
Highway Network	7.72
ResNet (1001 layers)	4.62 (±0.20)
DenseNet (190 layers)	3.46
Baseline (only with dropout)	6.67 (±0.07)
RPT	6.30 (±0.04)
VAT	5.81 (±0.02)

Experiments

Table 3: Test performance of semi-supervised learning methods.

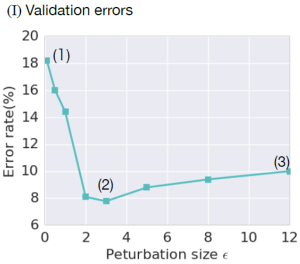
Method	Test error rate(%) SVHN $N_l = 1000$	CIFAR-10 $N_l = 4000$
SWWAE	23.56	
Auxiliary DGM	22.86	
Skip DGM	16.61 ($\pm$0.24)	
Ladder networks, Π model		20.40 ($\pm$0.47)
CatGAN		19.58 ($\pm$0.58)
GQAN with FM	8.11 ($\pm$1.3)	18.63 ($\pm$2.32)
model	5.43 ($\pm$0.25)	16.55 ($\pm$0.29)
(on Conv-Small)		
RPT	8.41 ($\pm$0.24)	18.56 ($\pm$0.29)
VAT	6.83 ($\pm$0.24)	14.87 ($\pm$0.13)
(on Conv-Large)		
VAT	5.77 ($\pm$0.32)	14.18 ($\pm$0.38)
VAT+EntMin	4.28 ($\pm$0.10)	13.15 ($\pm$0.21)

Conditional Entropy Minimization

VAT + EntMin:

$$\begin{aligned}\mathcal{R}_{cent} &= \mathcal{H}(Y|X) \\ &= -\frac{1}{N_l + N_{ul}} \sum_{x \in \mathcal{D}_l, \mathcal{D}_{ul}} \sum_y p(y|x,\theta) \log p(y|x,\theta)\end{aligned}$$

Performance of VAT with different ϵ



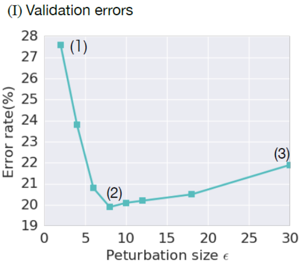


Figure 1: Validation error rate of VAT with different ϵ on SVHN and CIFAR-10.

Performance of VAT with different ϵ

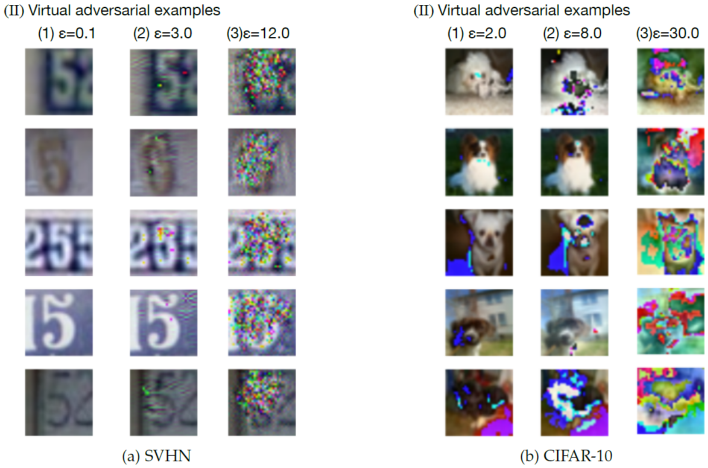


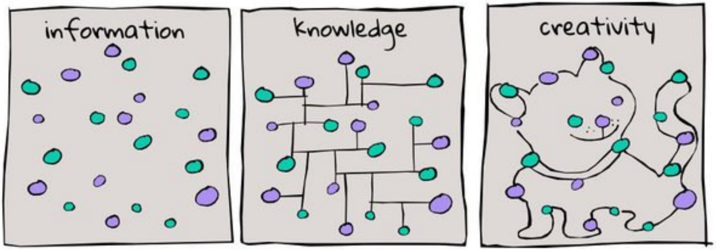
Figure 2: Images distorted with r_{vadv} with different ϵ on SVHN and CIFAR-10.

Conclusion

- Effective model for both supervised and semi-supervised learning on different datasets
- Small computational cost
- Model agnostic
- Simple:
  - Requires optimization of only one hyperparameter
  - Does not require training of additional models

Thank You!

Thanks for your attention :)



Any questions?

References

1. Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Shin Ishii. "Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning", 2018.
2. Jirong Yi, Raghu Mudumbai and Weiyu Xu. "Derivation of Information-Theoretically Optimal Adversarial Attacks with Applications to Robust Machine Learning", 2020.
3. Shixiang Gu, Luca Rigazio. "Towards deep neural network architectures robust to adversarial examples", 2015.
4. Jost Tobias Springenberg, "Unsupervised and semi-supervised learning with categorical generative adversarial networks", 2015.
5. Yves Grandvalet, Yoshua Bengio. "Semi-supervised learning by entropy minimization", NIPS, 2004.

Power Method

We want to find the dominant eigenvector of the matrix A with eigenvectors $v_1, v_2, \ldots, v_n$ sorted in descending order of their eigenvalues.
We will show that the iterative method $d_{t+1} = Ad_t$ will converge to a multiple of the dominant eigenvector v_1 provided that d_0 is not orthogonal to v_1.

$$
\begin{aligned}
\text{Suppose } d_0 &= c_1 v_1 + c_2 v_2 + \cdots + c_n v_n \\
\implies d_k &= Ad_{k-1} = A^k d_0 = (A^k c_1 v_1 + A^k c_2 v_2 + \cdots + A^k c_n v_n) \\
&= \lambda_1^k c_1 v_1 + \lambda_2^k c_2 v_2 + \cdots + \lambda_n^k c_n v_n \\
&= \lambda_1^k (c_1 v_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k c_2 v_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^k c_n v_n) \\
&\xrightarrow{k \to \infty} \lambda_1^k c_1 v_1
\end{aligned}
$$