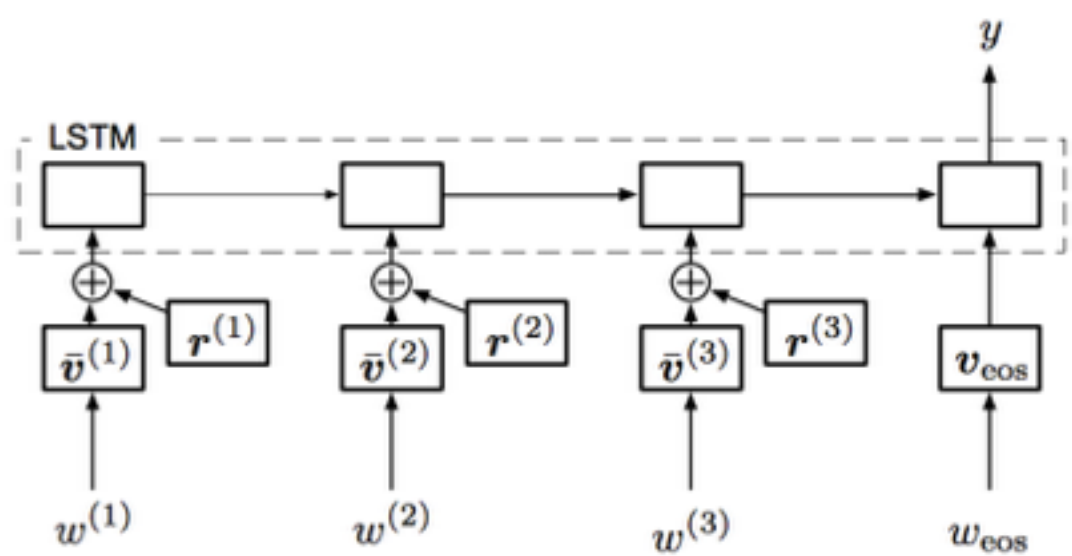
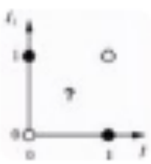
Virtual Adversarial Training for Semi-Supervised Text Classification

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.

24 31

Outline

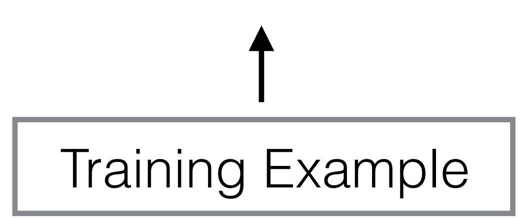
- Introduction to (virtual) adversarial training
- Virtual adversarial training for text classification
- Experimental Setup
- Results (and some analysis)
- Conclusions

A basic NN training procedure

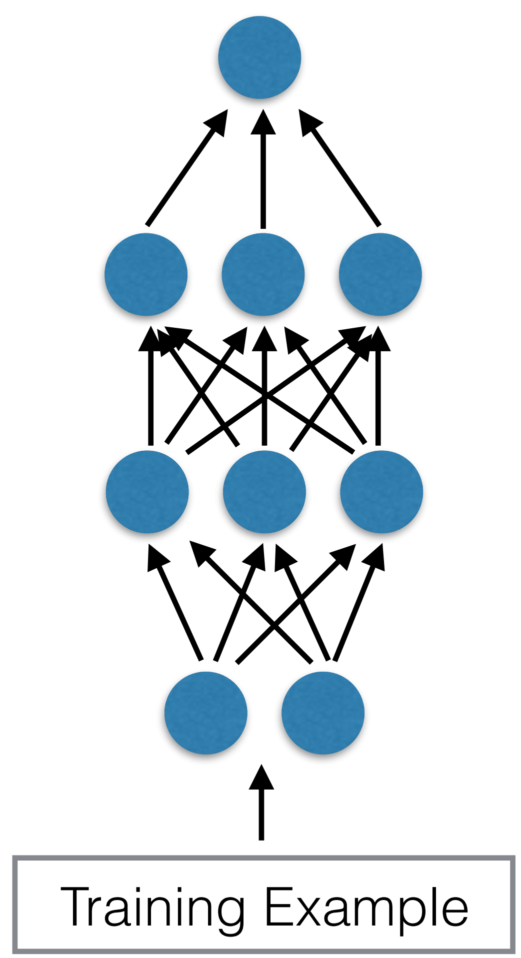
A basic NN training procedure

Training Example

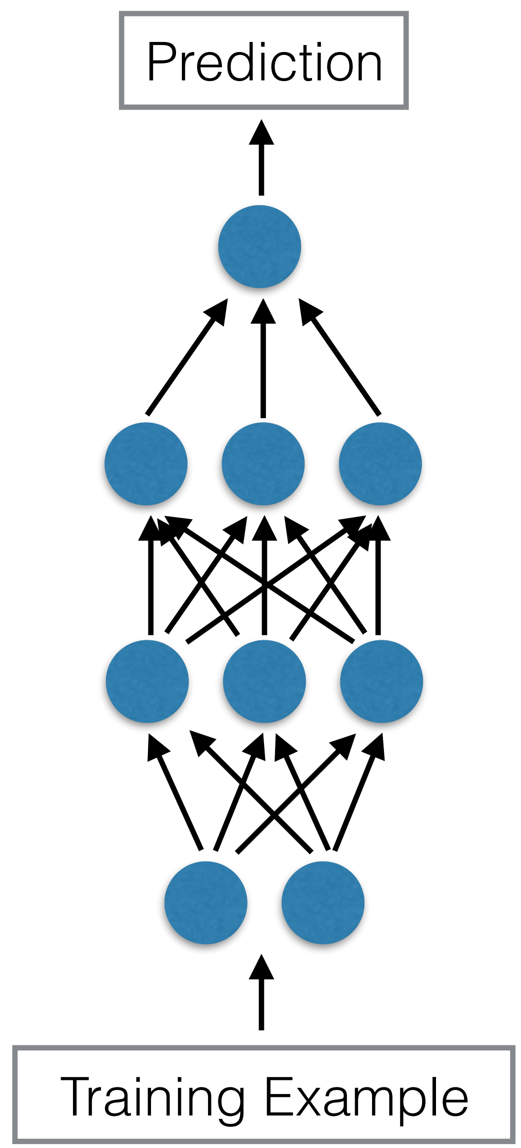
A basic NN training procedure



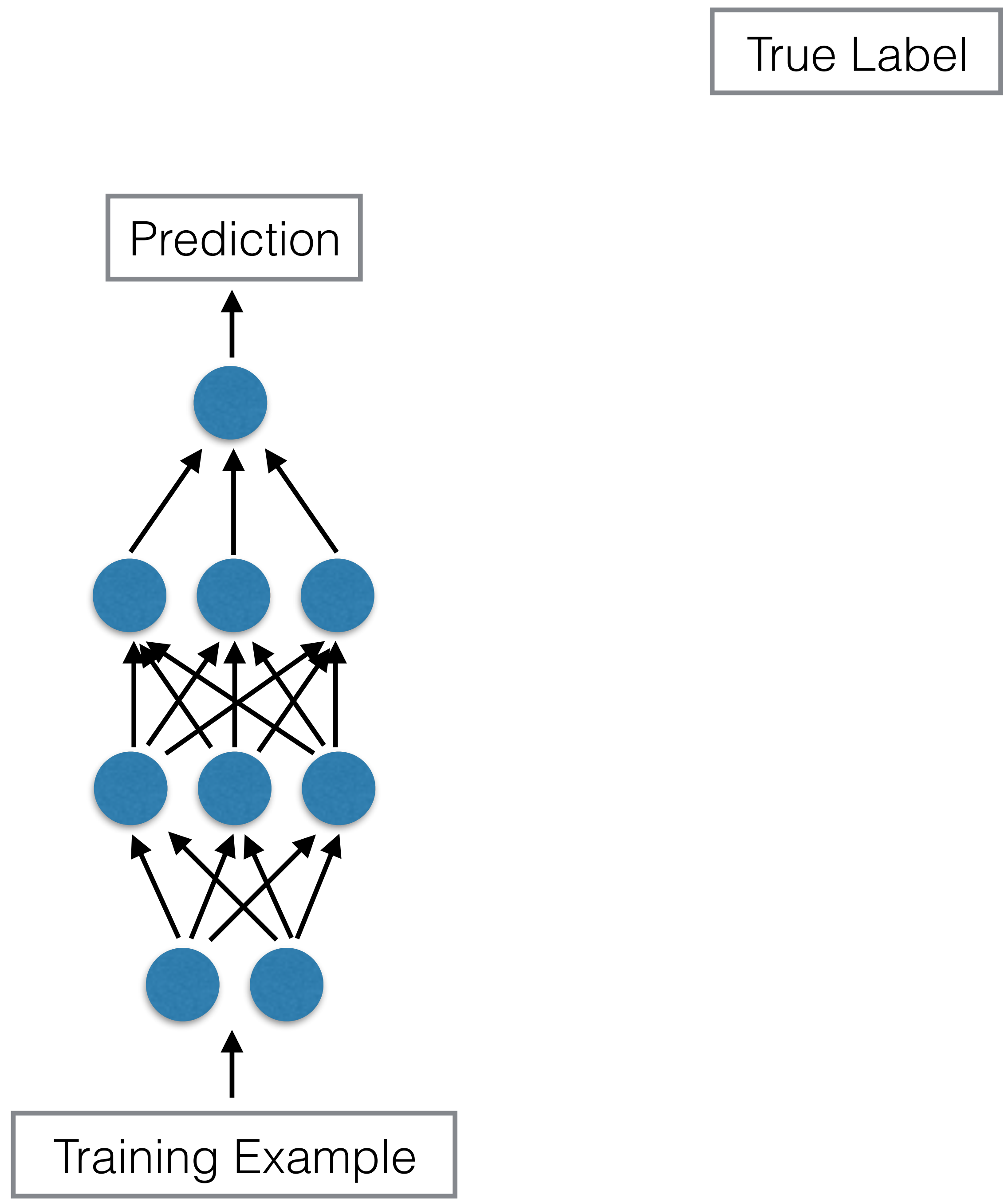
A basic NN training procedure



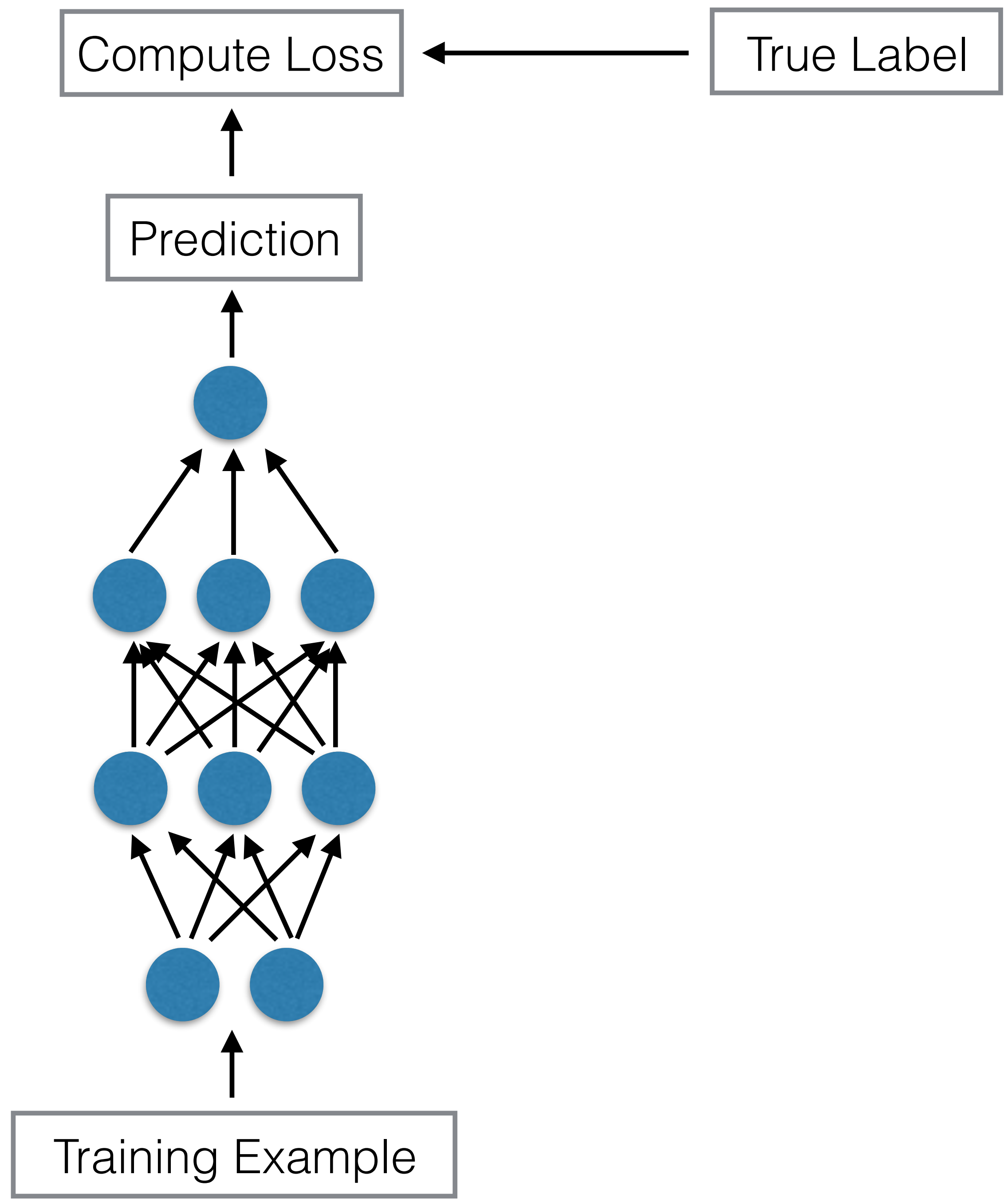
A basic NN training procedure



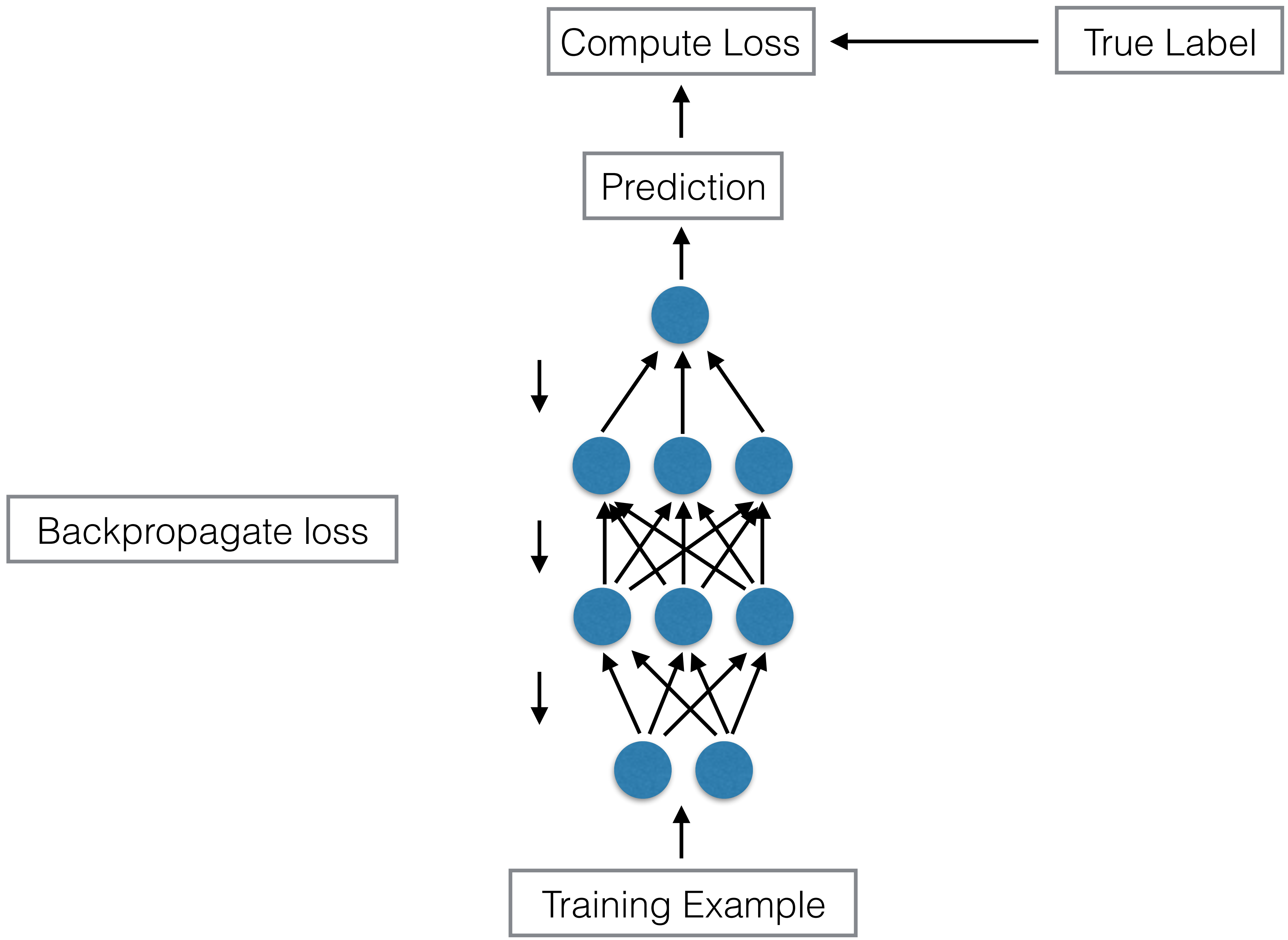
A basic NN training procedure



A basic NN training procedure



A basic NN training procedure



Adversarial Examples

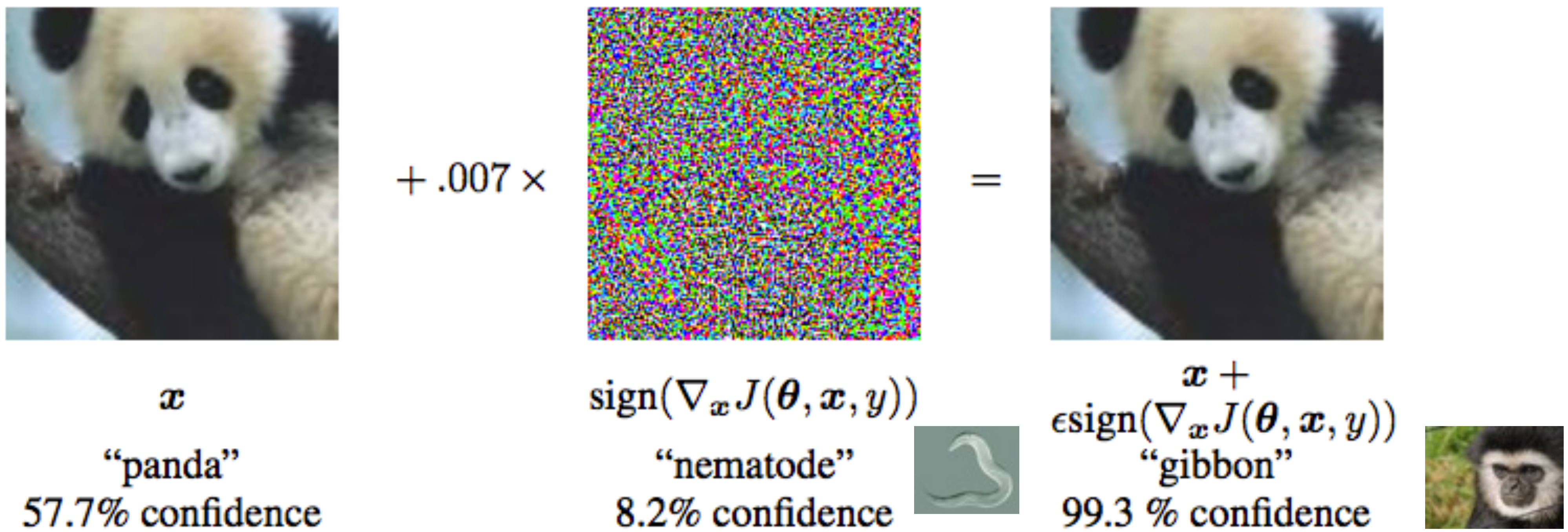
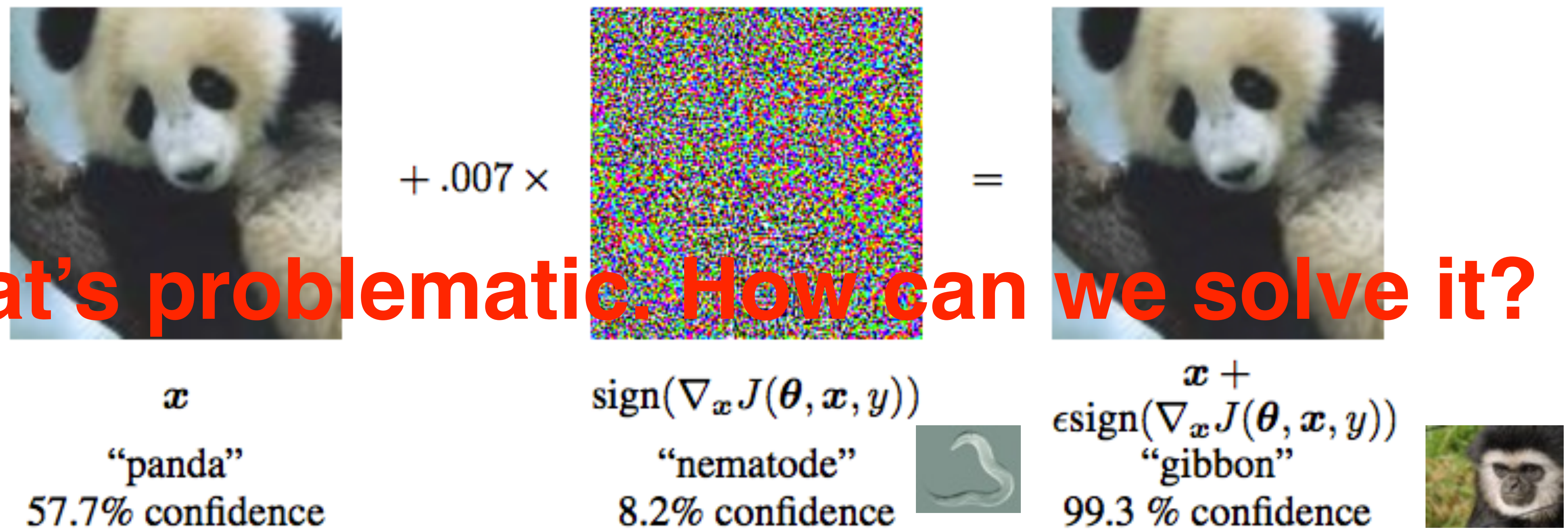


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ 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



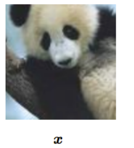
That's problematic. How can we solve it?

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 ϵ 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

Training Example



x

Adversarial Training for NN's

Training Example

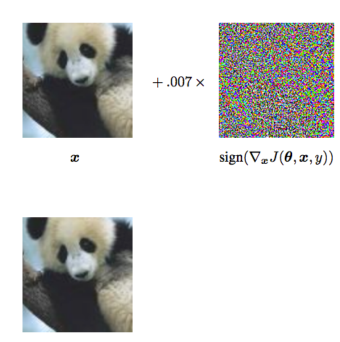
x

Adversarial Training for NN's

Perturbed Example

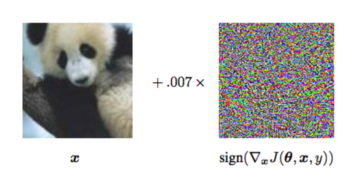
↑

Training Example

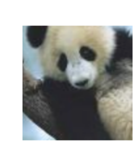


Adversarial Training for NN's

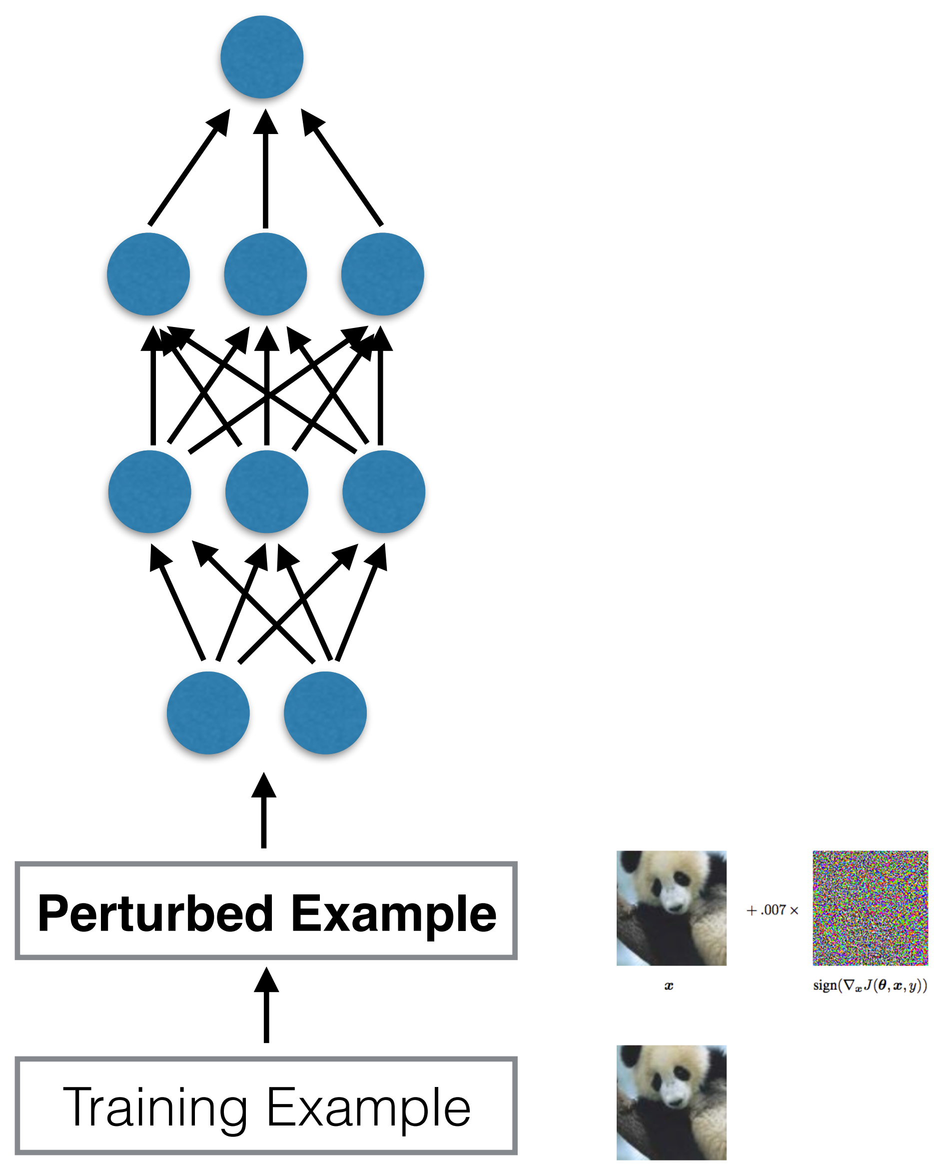
Perturbed Example



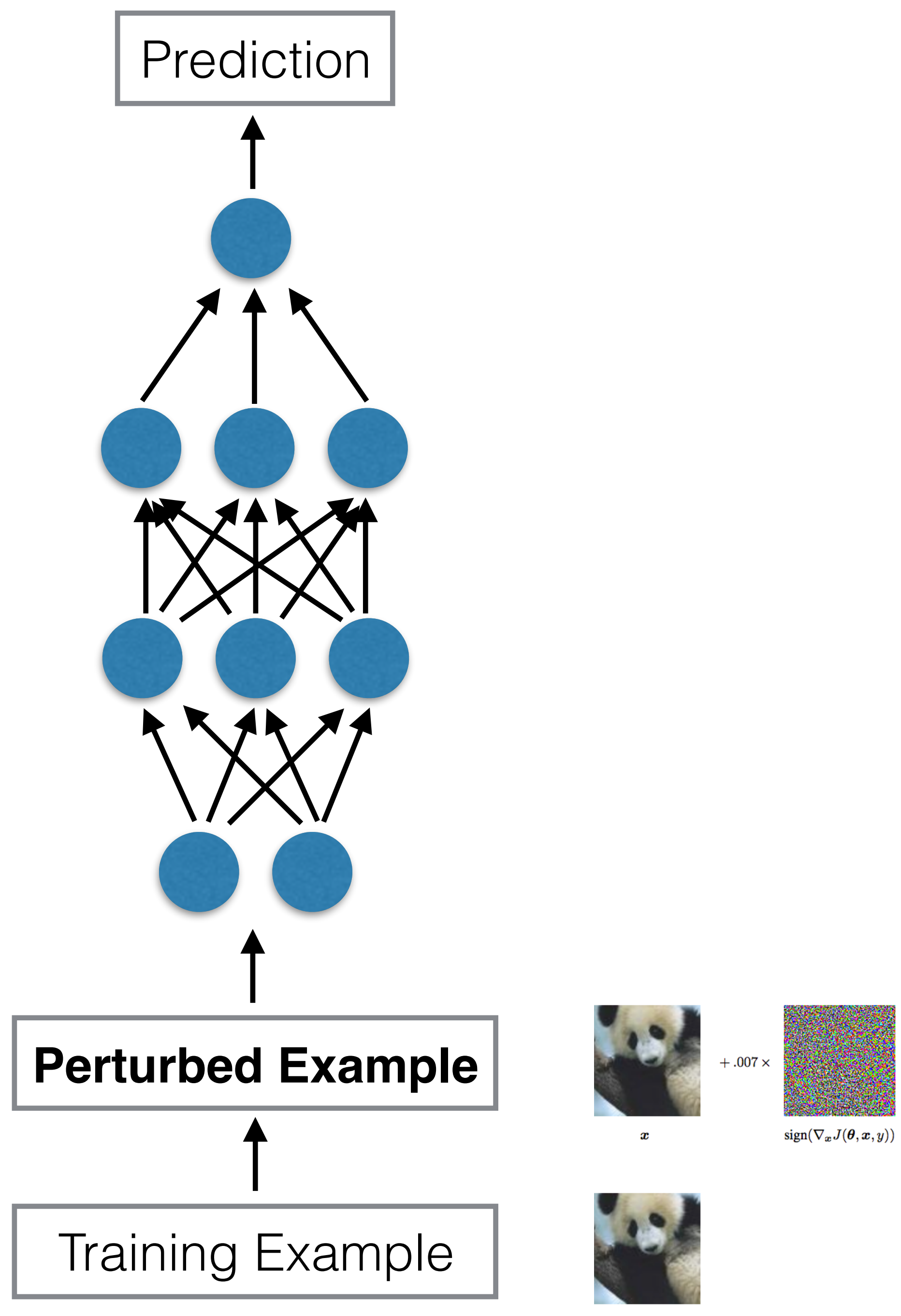
Training Example



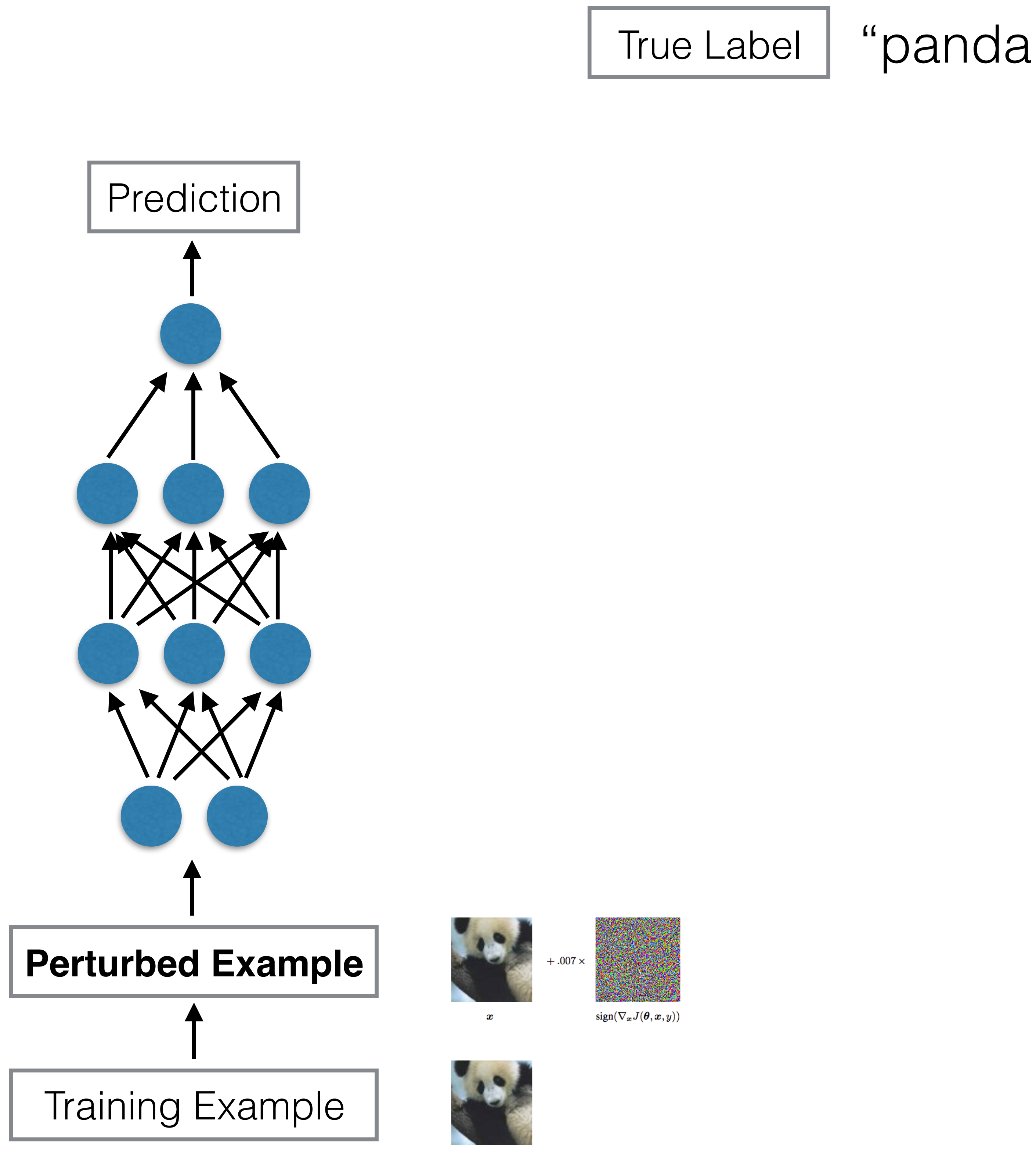
Adversarial Training for NN's



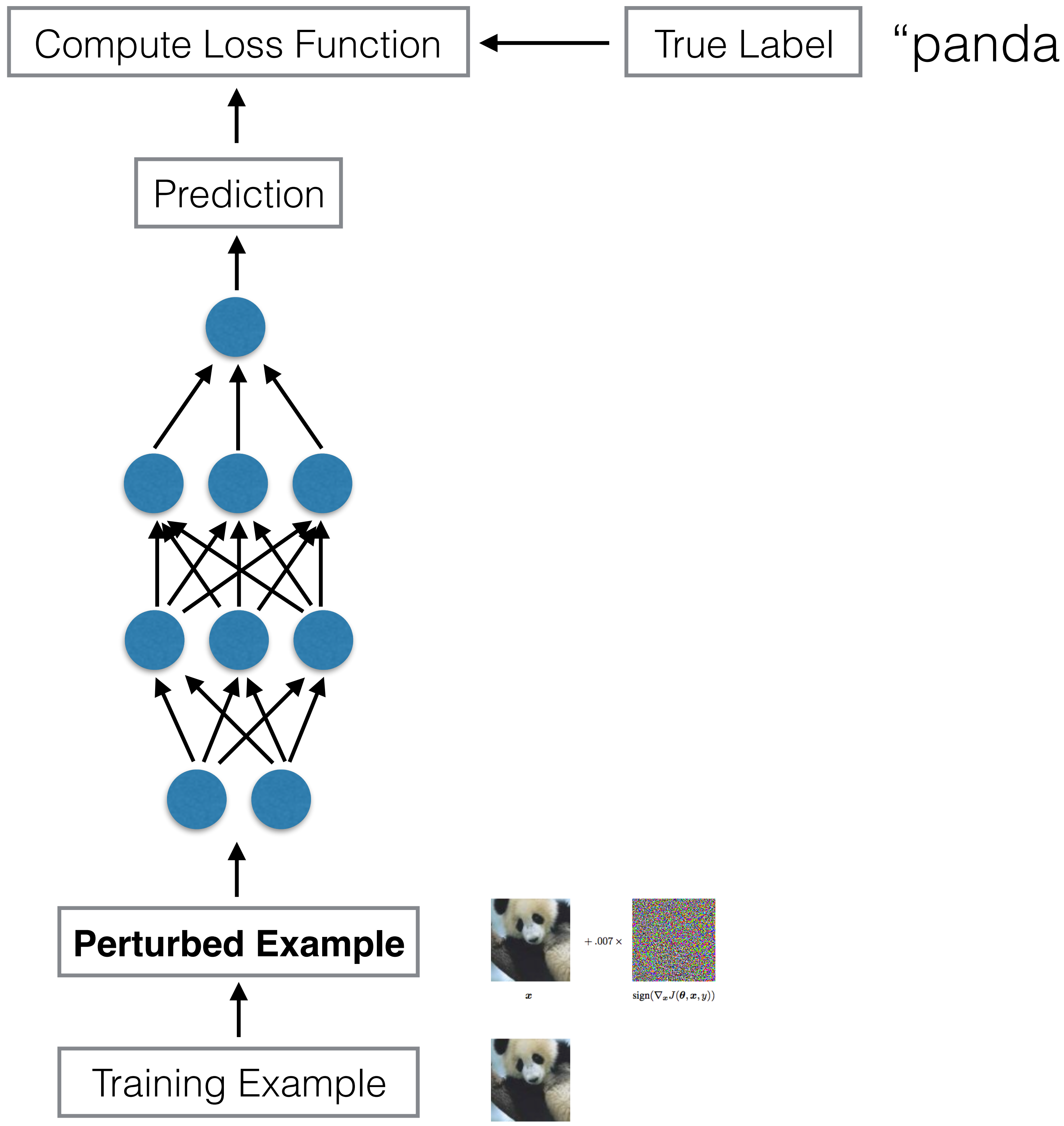
Adversarial Training for NN's



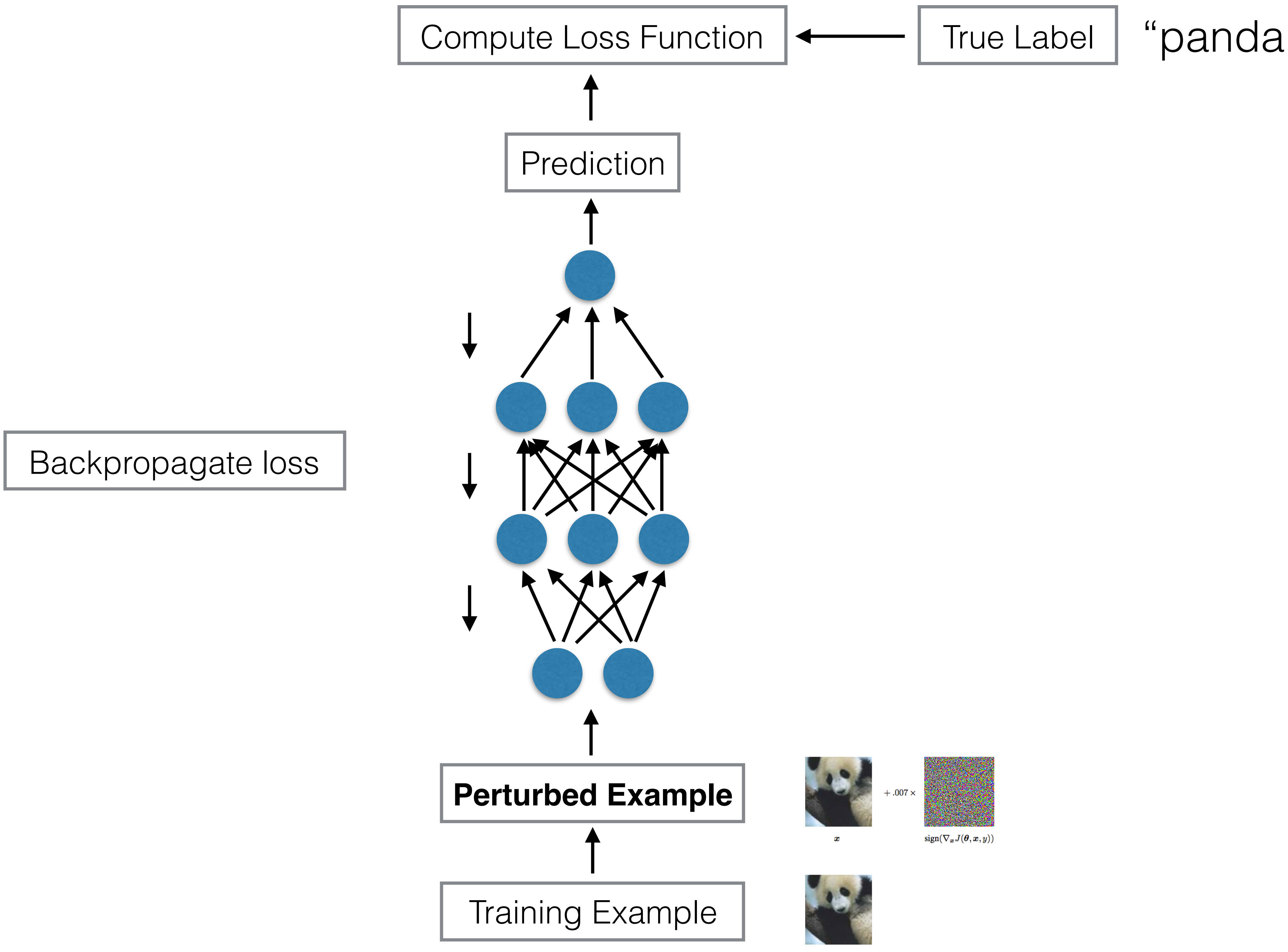
Adversarial Training for NN's



Adversarial Training for NN's



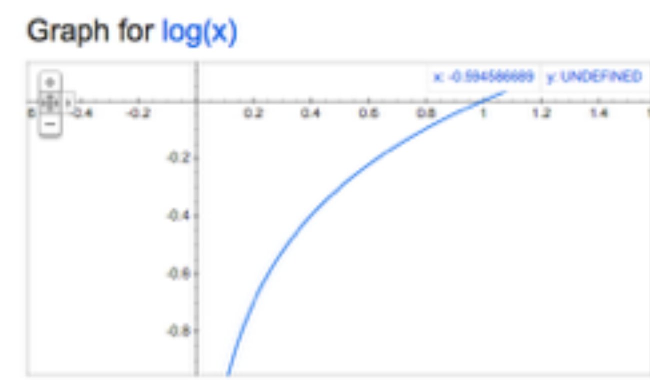
Adversarial Training for NN's



Adversarial Training for NN's

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

$$-\min_{\boldsymbol{r}, \|\boldsymbol{r}\| \leq \epsilon} \log p(y \mid \boldsymbol{x} + \boldsymbol{r}, \boldsymbol{\theta})$$



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

$$\boldsymbol{r}_{\text{adv}} = -\epsilon \boldsymbol{g} / \|\boldsymbol{g}\|_2 \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{x}} \log p(y \mid \boldsymbol{x}, \boldsymbol{\theta}).$$

Virtual Adversarial Training

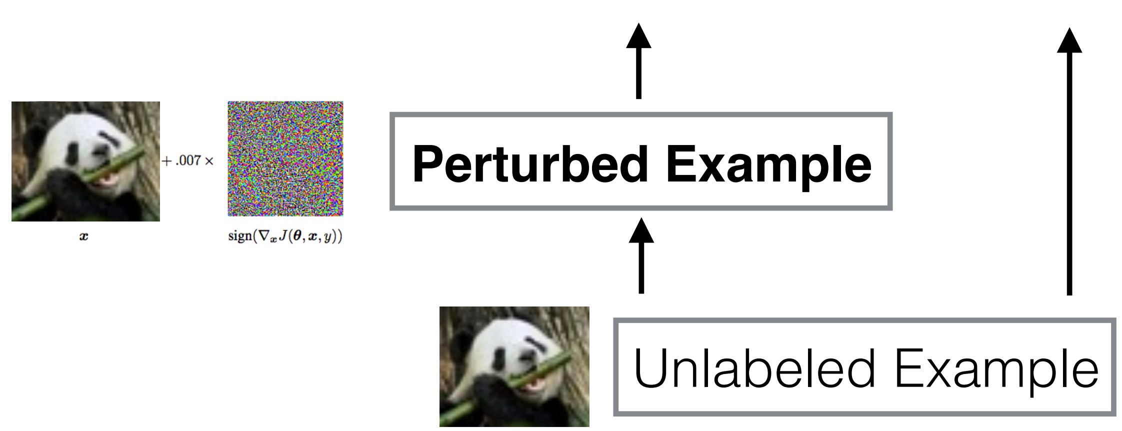
- Extends adversarial training to the semi-supervised regime
- The key idea - make the output **distribution** for an original and perturbrated example close to each other
- Enables the use of large amounts of unlabeled data

Virtual Adversarial Training for NN's

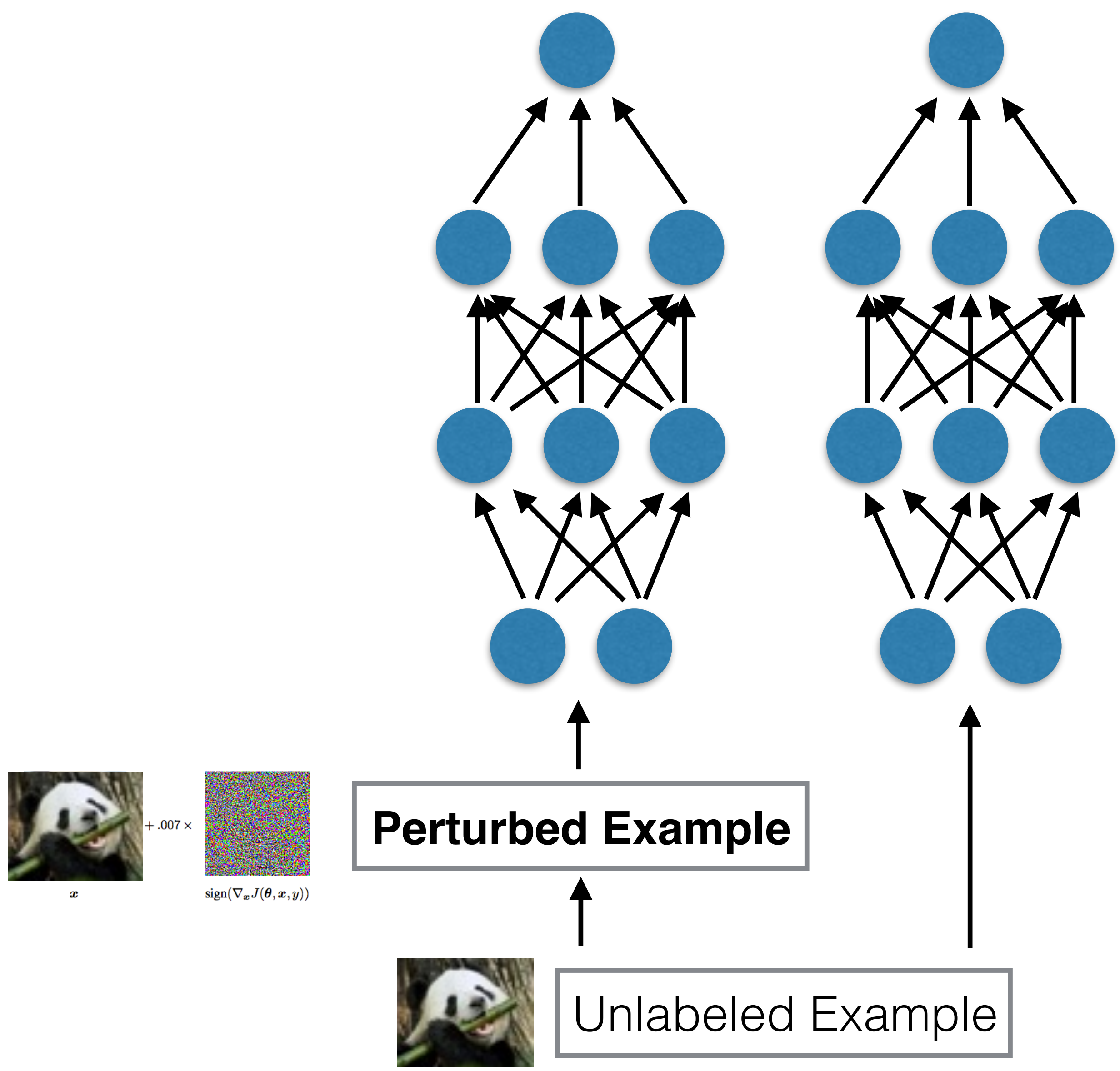
Virtual Adversarial Training for NN's

Unlabeled Example

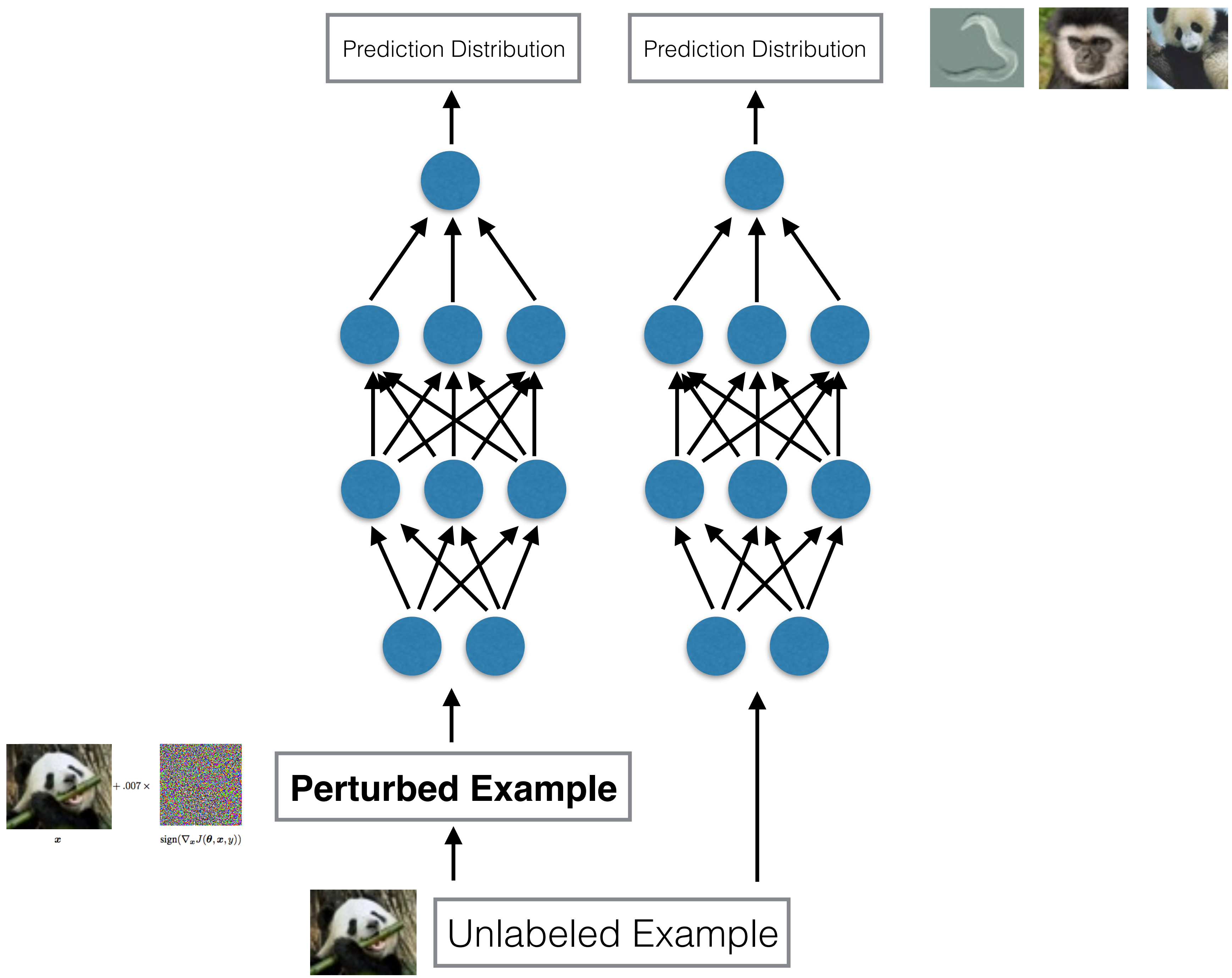
Virtual Adversarial Training for NN's



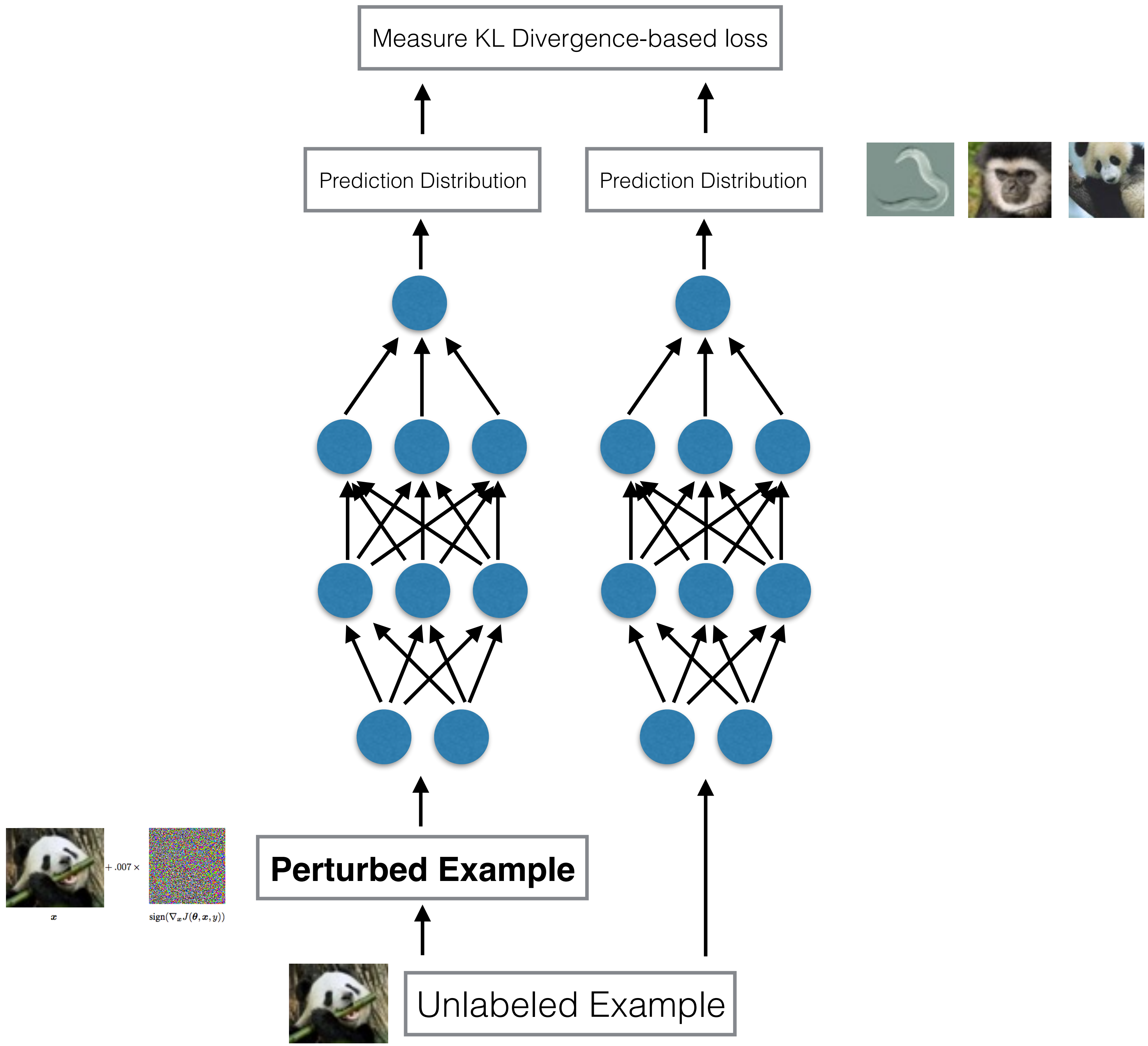
Virtual Adversarial Training for NN's



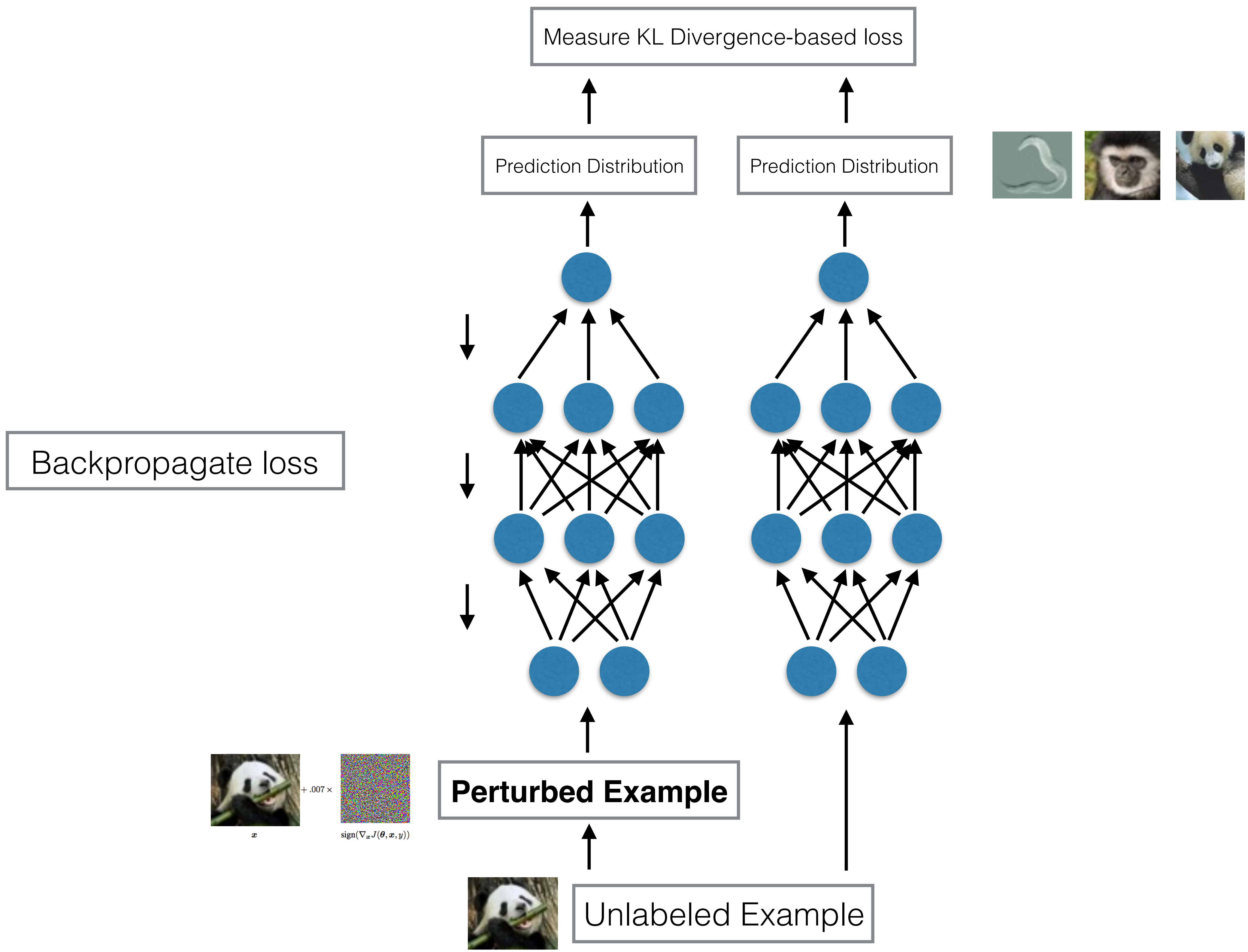
Virtual Adversarial Training for NN's



Virtual Adversarial Training for NN's



Virtual Adversarial Training for NN's



Virtual Adversarial Training for NN's

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

$$\max_{\boldsymbol{r}, \|\boldsymbol{r}\| \leq \epsilon} \mathrm{KL}[p(\cdot \mid \boldsymbol{x}, \boldsymbol{\theta}) || p(\cdot \mid \boldsymbol{x} + \boldsymbol{r}, \boldsymbol{\theta})]$$

$$D_{\mathrm{KL}}(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

- Again, in practice, there is an efficient way to approximate this (as detailed in Miyato et. al., 2016)

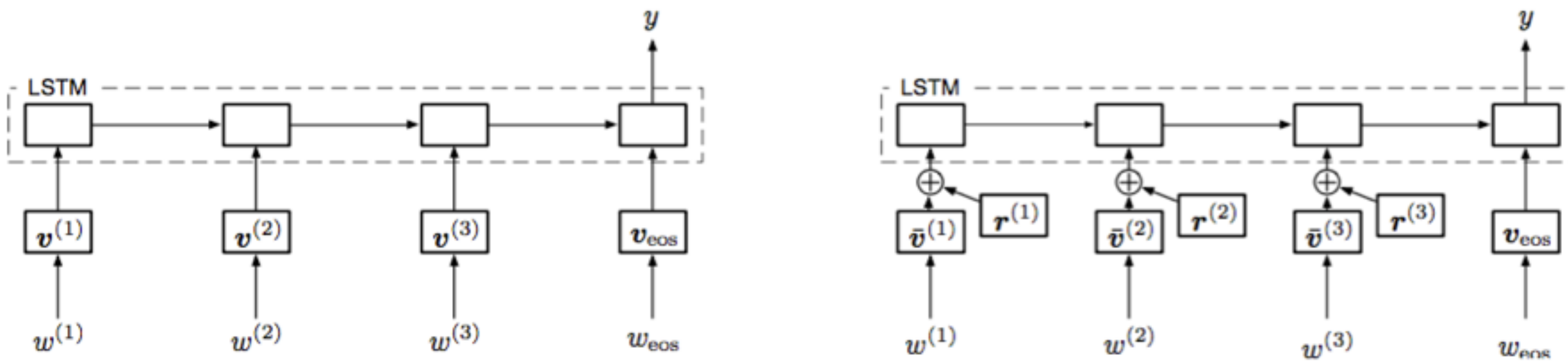
Model - Adversarial Training for Text Classification

- 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.

Model - Adversarial Training for Text Classification



(a) LSTM-based text classification model.   (b) The model with perturbed embeddings.

- The perturbation is introduced to <u>normalized embeddings</u> to avoid the network from learning to ignore them:

$$\bar{\boldsymbol{v}}_k = \frac{\boldsymbol{v}_k - \mathrm{E}(\boldsymbol{v})}{\sqrt{\mathrm{Var}(\boldsymbol{v})}} \text{ where } \mathrm{E}(\boldsymbol{v}) = \sum_{j=1}^{K} f_j \boldsymbol{v}_j, \mathrm{Var}(\boldsymbol{v}) = \sum_{j=1}^{K} f_j \left(\boldsymbol{v}_j - \mathrm{E}(\boldsymbol{v})\right)^2$$

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:

$$\boldsymbol{s} \quad = \quad [\bar{\boldsymbol{v}}^{(1)}, \bar{\boldsymbol{v}}^{(2)}, \dots, \bar{\boldsymbol{v}}^{(T)}]$$

- The perturbation is defined as:

$$\boldsymbol{r}_{\text{adv}} = -\epsilon \boldsymbol{g} / \|\boldsymbol{g}\|_2 \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{s}} \log p(y \mid \boldsymbol{s}, \boldsymbol{\theta})$$

- And the addition to the loss function is:

$$L_{\text{adv}}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{s}_n + \boldsymbol{r}_{\text{adv},n}, \boldsymbol{\theta})$$

Virtual Adversarial Training for Text Classification

- Here, the perturbation is defined as:

$$\boldsymbol{r}_{\text{v-adv}} = \epsilon \boldsymbol{g}/\|\boldsymbol{g}\|_2 \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{s}+\boldsymbol{d}} \mathrm{KL}\left[p(\cdot \mid \boldsymbol{s}, \boldsymbol{\theta}) || p(\cdot \mid \boldsymbol{s}+\boldsymbol{d}, \boldsymbol{\theta})\right]$$

- And the addition to the loss function is then:

$$L_{\text{v-adv}}(\boldsymbol{\theta}) = \frac{1}{N'} \sum_{n'=1}^{N'} \mathrm{KL}\left[p(\cdot \mid \boldsymbol{s}_{n'}, \boldsymbol{\theta}) || p(\cdot \mid \boldsymbol{s}_{n'} + \boldsymbol{r}_{\text{v-adv},n'}, \boldsymbol{\theta})\right]$$

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.

	Classes	Train	Test	Unlabeled	Avg. T	Max T
IMDB [17]	2	25,000	25,000	50,000	239	2,506
Elec [9]	2	24,792	24,897	197,025	110	5,123
Rotten Tomatoes [23]	2	9596	1066	7,911,684	20	54
DBpedia [14]	14	560,000	70,000	–	49	953
RCV1 [15]	55	15,564	49,838	668,640	153	9,852

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

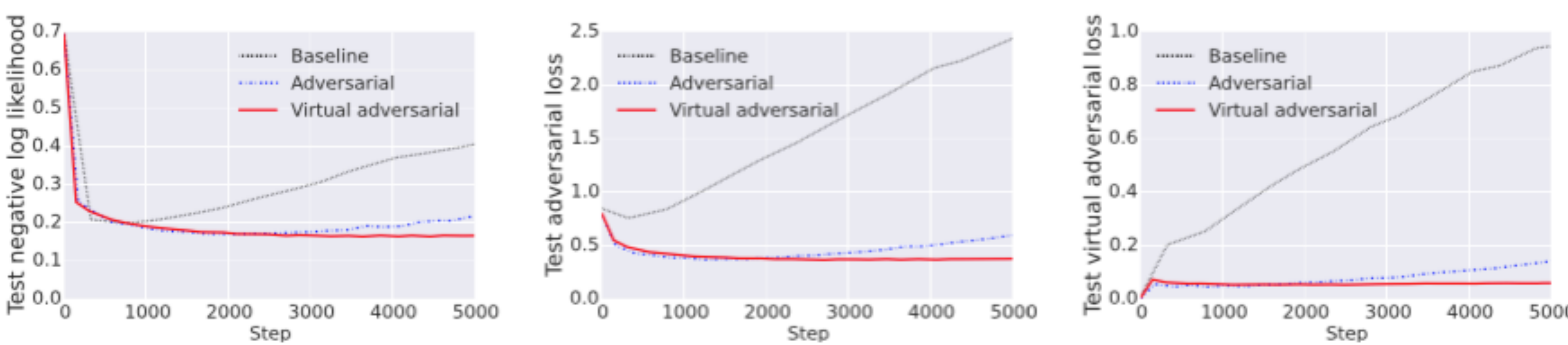
- Initialize word embeddings and LSTM weights with RNNLM on labeled and unlabeled examples
- Single layer LSTM, 1024 units (512 for BiLSTM)
- Embedding size: 256(IMDB, BiLSTM)/512(Rest)
- Sampled softmax loss with 1024 candidate samples (?)
- Adam optimization, 256 samples per batch
- 0.5 dropout rate 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



(a) Negative log likelihood (b) $L_{\mathrm{adv}}(\boldsymbol{\theta})$ (c) $L_{\mathrm{v\text{-}adv}}(\boldsymbol{\theta})$

Results - IMDB

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

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

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.

	‘good’				*‘bad’*			
	Baseline	**Random**	**Adversarial**	**Virtual Adversarial**	**Baseline**	**Random**	**Adversarial**	**Virtual Adversarial**
1	great	great	decent	decent	terrible	terrible	terrible	terrible
2	decent	decent	great	great	awful	awful	awful	awful
3	×bad	excellent	nice	nice	horrible	horrible	horrible	horrible
4	excellent	nice	fine	fine	×good	×good	poor	poor
5	Good	Good	entertaining	entertaining	Bad	poor	BAD	BAD
6	fine	×bad	interesting	interesting	BAD	BAD	stupid	stupid
7	nice	fine	Good	Good	poor	Bad	Bad	Bad
8	interesting	interesting	excellent	cool	stupid	stupid	laughable	laughable
9	solid	entertaining	solid	enjoyable	Horrible	Horrible	lame	lame
10	entertaining	solid	cool	excellent	horrendous	horrendous	Horrible	Horrible

Results - Topic classifiction: Elec, RCV1

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

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

Method	Test error rate	
	Elec	RCV1
Baseline	6.24%	7.40%
Adversarial	5.61%	7.12%
Virtual Adversarial	5.54%	7.05%
Adversarial + Virtual Adversarial	**5.40%**	6.97%
Virtual Adversarial (on bidirectional LSTM)	5.55%	6.71%
Adversarial + Virtual Adversarial (on bidirectional LSTM)	5.45%	**6.68%**
One-hot CNN (with pretrained embeddings of CNN) [9]	6.27%	7.71%
One-hot CNN (with pretrained embeddings of CNN and bi-LSTM) [10]	5.82%	7.20%
One-hot bi-LSTM (with pretrained embeddings of CNN and bi-LSTM) [10]	5.55%	8.52%

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

Method	**Test error rate**
Baseline	17.9%
Adversarial	16.8%
Virtual Adversarial	19.1%
Adversarial + Virtual Adversarial	**16.6%**
NBSVM-bigrams[28]	20.6%
CNN (with pretrained embeddings from word2vec Google News)[11]	18.5%
AdaSent (with pretrained embeddings from word2vec Google News)[31]	16.9%
SA-LSTM (with unlabeled data from Amazon reviews)[4]	16.7%

Results - DBpedia

- Baseline itself improves over SOTA, virtual adversarial performs best

Table 6: Test performance on the DBpedia topic classification task

Method	**Test error rate**
Baseline (without embedding normalization)	0.87%
Baseline	0.90%
Random perturbation	0.85%
Adversarial	0.79%
Virtual Adversarial	**0.76%**
Bag-of-words[4]	3.57%
Large-CNN(character-level) [4]	1.73%
SA-LSTM(word-level)[4]	1.41%
N-grams TFIDF [30]	1.31%
SA-LSTM(character-level)[4]	1.19%

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