Generative Adversarial Networks (GAN): A Gentle Introduction [UPDATED]

Presentation · April 2017

CITATIONS
0

READS
8,886

1 author:

Su Wang
University of Texas at Austin
15 PUBLICATIONS **88** CITATIONS

SEE PROFILE

Generative Adversarial Networks (GAN)
A Gentle Introduction

Su Wang
Department of Statistics and Data Science
University of Texas at Austin

Abstract

In this tutorial, I present an intuitive introduction to the *Generative Adversarial Network* (GAN) [1], invented by Ian Goodfellow of Google Brain, overview the general idea of the model, and describe the algorithm for training it as per the original work. I further briefly introduce the application of GAN in Natural Language Processing to show its flexibility and strong potential as a neural network architecture. In lieu of the discussion, I also present simple `Tensorflow` code[1] for the original GAN [1] and an important variant — *Wasserstein GAN* [26,27], to help the reader getting a quick start in practical applications.

1 Overview

Generative Adversarial Networks (GAN) [1] is a deep learning framework in which two models, a generative model G and a discriminative model D, are trained simultaneously. The objective of G is to capture the distribution of some target data (e.g. distributions of pixel intensity in images). D aids the training of G by examining the data generated by G in reference to "real" data, and thereby helping G learn the distribution that underpins the real data. GAN is fleshed out in Goodfellow et al. (2014) [1] as a pair of simple neural networks. However in practice, the models can in principle be any generative-discriminative pairs[2]

In the original work [1,2] and elsewhere [3], GAN has been given the analogy as a process of making counterfeit money: G plays the role of a counterfeiter in-training, while D the bank strives to identify fake bills and in the process (hopefully unintentionally) helps G honing its bill-making skills. More concretely, let $\boldsymbol{x} \sim p_{data}$ be characterizing features for real bills, and $G(\boldsymbol{z})$ be features G creates from some noise distribution $\boldsymbol{z} \sim p_z$. Further let J be some quantitative metric which measures the extent to which a bill is real. Then, D's job is to lower $J(G(\boldsymbol{z}))$ (the score of fake bill) while increase the score $J(\boldsymbol{x})$ (the score of real bill) for more successful identification. G, on the other hand, aims at increasing $J(G(\boldsymbol{z}))$ (i.e. improving the quality of fake bill) by learning from "observing" how D make differentiations. As the game of "busting fake bill" and "making better fake bill" proceeds, the model distribution p_G draws closer to p_{data}, and eventually reaches an *equilibrium* [2] where D can no longer classfiy better than chance (i.e. $D(\boldsymbol{x}) = D(G(\boldsymbol{z})) = \frac{1}{2}$). Now we say G has arrived at an optimal point[3] in counterfeiting.

GAN has gained massive attraction in computer vision [4,5,6], feature representation [7], and more recently in *Natural Language Processing* (NLP) tasks: Document Modeling [8], Dialogue Generation [9], Sentiment Analysis [10], and Domain Adaptation [11]. In Section 5 I briefly exemplify the successful application of GAN in NLP [8,9].

[1] `https://github.com/suwangcompling/GAN-tutorials`.

[2] These can be any generative and discriminative models, in a much wider sense than the term "G-D pair" is used in the literature [12].

[3] I will show that the optimal point is reached iff $p_G = p_{data}$ (Section 2)

2 Model

Formal description. To begin, I describe a barebone variant of GAN in its original formulation, in which both D and G are simple *Multi-Layer Perceptron*s (MLP). Let p_{data} be our target distribution which the generator G will learn by approximiating it with the model distribution p_G. G is associated with a noise prior p_z, from which G draws sample $\boldsymbol{z}$, and create fake datum[4] $G(\boldsymbol{z};\theta_G)$, where θ_G are model parameters. The discriminator $D(\boldsymbol{x};\theta_D)$ takes $\boldsymbol{x}$ or $G(\boldsymbol{z})$ as input, and returns a binary judgment as to whether the input is from p_{data} or p_G.

D evaluates the quality of input $\boldsymbol{x}$ with the following metric:

$$J(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{1}$$

The first term of (1) evaluates the expectation of the log probability that $\boldsymbol{x}$ is drawn from real data; the second term, that $\boldsymbol{x}$ is drawn from fake data (i.e. noise). D's objective is to maximize J, i.e. pushing the both terms towards 0 (i.e. $D(\boldsymbol{x}) \to 1$, $D(G(\boldsymbol{z})) \to 0$). G, on the other hand, wants to minimize J by lower the second term[5] (i.e. $D(G(\boldsymbol{z})) \to 1$). The "conflicting objectives" of the two models results in a *Minmax Game* [1] which is described as follows:

$$\min_G \max_D J(D,G) \tag{2}$$

Algorithmically, the training takes the form of an alternation process between minimization and maximization, which is described in Algorithm 1. In practice, Eq. 1 often does not bring the model to equilibrium, this is because $\log(1 - D(G(\boldsymbol{z})))$ rapidly saturates in the early stage of training, where D easily rejects $G(\boldsymbol{z})$ because G generates fake data of poor quality, such that they conspicuously differ from the real data. Therefore, rather than evaluating how bad fake data are (G's objective in the second term of Eq. 1), we instead evaluate how good they are by setting G's goal to maximizing $J_G(G) = \log D(G(\boldsymbol{z}))$. In so doing we end up with two objective functions:

$$\begin{aligned} &\max_D J_D(D,G) \\ &\max_G J_G(G) \end{aligned} \tag{3}$$

To illustrate the training process graphically, we observe (a) how p_G changes over time, and consequently (b) how the discrimination boundary of D changes accordingly (Figure 1, Figure 1 in [1]).

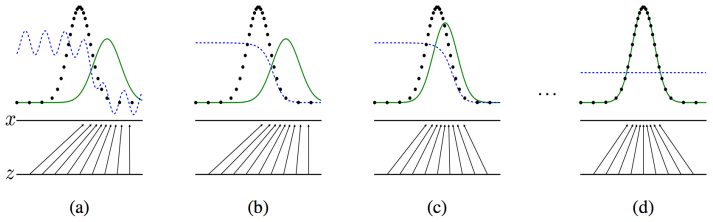

Figure 1: Training Process of GAN (green solid: p_G; black dotted: p_{data}; blue dash: D's discrimination boundary; arrows: generation of fake data)

In (a) through (d), $\boldsymbol{z}$ are sampled uniformly from the noise prior p_z, and p_G draws closer to p_{data}. In the process, the descrimination boundary changes accordingly, and finally morphs into a flat line (i.e. $D(\boldsymbol{x}) = D(G(\boldsymbol{z})) = \frac{1}{2}$) which indicates D is now unable to tell fake and real data apart. Specifically, Figure 1 shows a scenario where the model is near convergence: (a) p_G is similar to p_{data}, and D is now partially accurate[6]; (b) D is updated: based on the relative distribution of p_G and p_{data}, D converges in the inner loop of Algorithm 1 to $D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$; (c) G is updated: p_G

[4]The generator is essentially a function that maps a noise datum into the space of a real datum, i.e. $G : \boldsymbol{z} \mapsto \boldsymbol{x}$. Note that $\boldsymbol{z}$ and $\boldsymbol{x}$ therefore do not have to be equal in dimensionality.

[5]The first term is independent of G.

[6]In the late stage of training, G starts to generate high quality fakes, to the effect that D's classification performance suffers (but not entirely down to the level of randomness, i.e. $D(\boldsymbol{x}) = D(G(\boldsymbol{z})) = \frac{1}{2}$).

Algorithm 1 Minmax Game

1: **for** specified # of training iterations **do**

▷ TRAINING DISCRIMINATOR

2: **for** specified k steps* **do**

3: Draw minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\} \sim p_z$.

4: Draw minibatch of m data samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\} \sim p_{data}$.

5: Update D's parameters by gradient ascent:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(\boldsymbol{x}^{(i)}) + \log \Big(1 - D(G(\boldsymbol{z}^{(i)})) \Big) \Big]$$

6: **end for**

▷ TRAINING GENERATOR

7: Draw minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\} \sim p_z$.

8: Update G's parameters by gradient descent:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \Big(1 - D(G(\boldsymbol{z}^{(i)})) \Big)$$

9: **end for**

*k is a tunable hyperparameter which is usually set to 1 to lower the training cost of each iteration.

is drawn closer to p_{data} under the guidance of the gradient of D; (d) final convergence: assuming sufficient model capacity, the adversarial pair reach the equilibrium $p_G = p_{data}$, where $D(\boldsymbol{x}) = \frac{1}{2}$.

Analysis. We now look at (b)-(d) in Figure 1 analytically to understand why the training scheme works. We begin by addressing (b): how does D converge to $D^*(\boldsymbol{x})$ in the inner loop?

Statement 1. *For fixed G, the optimal discriminator D is*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})} \tag{4}$$

Proof. Given a fixed G, the training objective for D is to maximize $J(D, G)$, where

$$\begin{aligned} J(G, D) &= \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \\ &= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x})) dx + \int_{\boldsymbol{z}} p_z(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z}))) dz \\ &= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx \end{aligned} \tag{5}$$

The last equality employs change of variable for the second term of the penultimate formula: as $G : \boldsymbol{z} \mapsto \boldsymbol{x}$, (i) integrating over $p_z(\boldsymbol{z})$ is equivalent to integrating over $p_G(\boldsymbol{x})$, and (ii) integrating over $G(\boldsymbol{z})$ is equivalent to integrating over $\boldsymbol{x}$. Further, we have[7]

$$\begin{aligned} D^* &= \operatorname*{argmax}_D J(G, D) \\ &= \operatorname*{argmax}_D \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx \end{aligned} \tag{6}$$

where the integrand can be abstracted in the form $f(D) = a\log(D) + b\log(1 - D)$. By setting $\frac{\partial f}{\partial D} \triangleq 0$ and solving for D, we have $D^* = \frac{a}{a+b}$, satisfying Eq. 4, concluding the proof. □

Next we look at (c,d) in the figure and ask: how does updating G get us to $p_G = p_{data}$?

Statement 2. *Let $C(G) = \max_D J(G, D)$, i.e. $C(G)$ is G's minimization objective (cf. Eq. 2). The global minimum of $C(G)$ is reached iff $p_G = p_{data}$, at which point $C(G) = -\log 4$.*

[7] The last equality in Eq. 6 is copied from the results of Eq. 5.

Proof. First we prove $p_G = p_{data} \Rightarrow C(G) = -\log 4$. We know that

$$
\begin{aligned}
C(G) &= \max_D J(G, D) \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_G}[\log(1 - D_G^*(\boldsymbol{x}))] \quad \text{(by change of variable (cf. Eq. 5))} \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G}\left[\log \frac{p_G(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] \qquad (7)
\end{aligned}
$$

We also know that if $p_{data} = p_G$, then $D_G^*(\boldsymbol{x}) = \frac{1}{2}$ (by Eq. 4), which we plug in Eq. 7 to obtain $C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$. That is, $p_G = p_{data} \Rightarrow C(G) = -\log 4$. We now show that $C(G) = -\log 4 \Rightarrow p_G = p_{data}$.

$$
\begin{aligned}
C(G) &= \mathbb{E}_{\boldsymbol{x} \sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G}\left[\log \frac{p_G(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] \\
&= -\log 4 + \log 4 + \mathbb{E}_{\boldsymbol{x} \sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G}\left[\log \frac{p_G(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] \\
&= -\log 4 + \mathbb{E}_{\boldsymbol{x} \sim p_{data}}\left[\log \left(2 \cdot \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right)\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G}\left[\log \left(2 \cdot \frac{p_G(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right)\right] \\
&= -\log 4 + \mathbb{E}_{\boldsymbol{x} \sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G}\left[\log \frac{p_G(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}}\right] \\
&= -\log 4 + KL\left(p_{data} \,\middle\|\, \frac{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}\right) + KL\left(p_G \,\middle\|\, \frac{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}\right) \\
&= -\log 4 + 2 \cdot JS(p_{data} \,\|\, p_G) \quad \text{(by the def. of Jensen-Shannon Divergence)} \qquad (8)
\end{aligned}
$$

Now given $C(G) = \log 4$, we must have $JS(p_{data} \,\|\, p_G) = 0$, which is only true when $p_G = p_{data}$. Thus we have shown $C(G) = -\log 4 \Rightarrow p_G = p_{data}$. □

Statement 1 and 2 show the optima in the alternating updates in Algorithm 1 lead us to the equilibrium of the Minmax Game. The original work [1] also proves the convergence of Algorithm 1. However the derivation involves advanced knowledge of optimization, I thus refer the interested reader to the proof there.

3 Basic Implementation

This section presents a demo implementation with MNIST image reconstruction[8]. Specifically, we have the discriminator D as a convolutional net, and the generator G a deconvolutional net [24]. A simple GAN graph is as follows (the precise setup of the networks is omitted to defer to the full code Jupyter notebook):

```
import tensorflow as tf

# input images to the discriminator
x_placeholder = tf.placeholder(tf.float32, shape = [None,28,28,1])
# input noise vectors to the generator
z_placeholder = tf.placeholder(tf.float32, [None, z_dimensions])

# D(x): predicted logits/probabilities for real image.
Dx = D(x_placeholder)
# G(z): fake image generated by the generator.
Gz = G(z_placeholder, BATCH_SIZE, z_dimensions)
# D(G(z)): predicted logits/probabilities for generated image.
Dg = D(Gz, reuse=True)

# Discriminator loss
```

[8] For full code see https://github.com/suwangcompling/GAN-tutorials/blob/master/Basic GAN (MNIST demo).ipynb

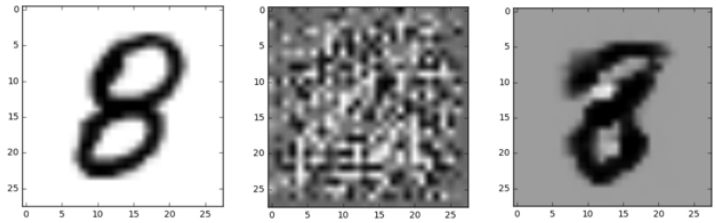

Figure 2: **Left**: real image; **Center**: generated image before training; **Right**: generated image after training. The convolutional net has two conv-avgpool layers and two fully-connected layers; the deconvolutional net has four layers and does exponential upscaling.

```
#   J(D,G) = E[log(Dx)] + E[log(1-D(Gz))]
#   NB: only update D's params in training.
d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dx,
labels = tf.ones_like(Dx)))
    # the above maximizes E[log(Dx)] by pushing Dx to label=1 (recognizing real
        image).
d_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg,
labels = tf.zeros_like(Dg)))
    # the above maximizes E[log(1-D(Gz))] by pushing D(Gz) to label=0 (recognizing
        fake image).
d_loss = d_loss_real + d_loss_fake

# Generator loss
#   J(D,G) = E[log(1-D(Gz))]
#   NB: only update G's params in training.
g_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Dg,
labels = tf.ones_like(Dg)))
    # the above minimizes E[log(1-D(Gz))] by pushing D(G(z)) to label=1 (fool
        discriminator).

# Separate training params
#   Ensure the correct params are updated in training.
tvars = tf.trainable_variables()
d_vars = [var for var in tvars if 'd_' in var.name]
g_vars = [var for var in tvars if 'g_' in var.name]

# Create optimizers for D & G
adam = tf.train.AdamOptimizer(2e-4)
trainer_D = adam.minimize(d_loss, var_list=d_vars)
trainer_G = adam.minimize(g_loss, var_list=g_vars)
```

It should be noted that GAN in its vanilla form is notoriously difficult to train and sensitive to hyper-parameter or architectural setup. In the next section we briefly discuss the *Wasserstein GAN* [26,27], a massive improvement on the original GAN which is more amenable to practical applications.

4 Improvement: Wassertein GAN

This section is my attempt to give a simple and intuitive description of *Wasserstein GAN* (WGAN) as a strong and theory-backed improvement. I defer the gory details and mathematical arguments to the original paper [26] and stay focused on the main storyline of the argument.

Despite the sexiness of the idea of adversarial training, the original GAN exhibits fickle behavior in training and is notoriously difficult to tune correctly. Arjovsky et al. [26] argue the observation to a large extent stems from the KL-based distance metric it uses. Specifically, strong deviation between the distribution of the real data and the generator leads to non-convergence of the model under KL-based distance (one among many other distance metrics such as *total variation*). The WGAN, which operates with the *Earth Mover* (EM) distance, on the other hand, does not suffer from

these issues. It is *"... continuous everywhere and differentiable (almost) everywhere."* (Theorem 1.2, [26]). Let p_{data}, p_g be the distributions of the real data and the generator respectively, the EM distance can be understand as the minimal effort required to move probability mass from p_{data} to transform it into p_g. It is defined as follows:

$$W(p_{data}, p_g) = \inf_{\gamma \in \Pi(p_{data}, p_g)} \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \gamma} \left[\| \boldsymbol{x} - \boldsymbol{y} \| \right] \tag{9}$$

where $\Pi(p_{data}, p_g)$ is the set of all joint distributions γ whose marginals are p_{data} and p_g. Because the EM distance is intractable in exact, Arjovsky et al. propose to approximate it by applying the *Kantorovich-Rubinstein duality*, which says W is equivalent to the following:

$$W(p_{data}, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \left[f(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{x} \sim p_g} \left[f(\boldsymbol{x}) \right] \tag{10}$$

where the supremum is taken over all $\mathbb{1}$-Lipschitz functions, and f is a general notation for a function, which, in the context of GAN, denotes the discriminator D (it is also known as the *critic*). Sidestepping the sophisticated mathematical ideas involved, one only needs to understand that the K-Lipschitz condition provides guarantee that the EM distance is continuous and differentiable everywhere with only minor assumptions (cf. Theorem 1. [26]). With this approximation, the gradient is now easily calculated:

$$\begin{aligned} \nabla_\theta W(p_{data}, p_g) &= \nabla_\theta \left\{ \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \left[D_w(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{z} \sim p_z} D_w(G_\theta(\boldsymbol{z})) \right\} \\ &= -\mathbb{E}_{\boldsymbol{z} \sim p_z} \left[\nabla_\theta D_w(G_\theta(\boldsymbol{z})) \right] \end{aligned}$$

where θ are the parameters of the generator G, $\boldsymbol{z} \sim p_z$ the random noise. Finally, Arjovsky et al. also propose a weight clipping technique which guarantees the family of distributions the discriminator can take on, i.e. $D_w, w \in \mathcal{W}$ is K-Lipschitz — the weights are constrained to lie within $[-c, c]$, where c is some (usally small) parameter (e.g. 0.01 is a popular default). The weight-clipping takes place after the weight update.

Despite of the complexity in the mathematical argument of WGAN, implementation-wise[9] it is quite simple: we simply replace the loss calculation in the code of the previous section with the following:

```
# Approximating the Earth Mover (EM) distance
d_loss = tf.reduce_mean(Dx) - tf.reduce_mean(Dg)
g_loss = tf.reduce_mean(Dg)
```

and then add the weight-clipping to the discriminator:

```
d_clip = [v.assign(tf.clip_by_value(v,-0.01,0.01)) for v in d_vars]
```

While theoretically sound and general being superior to the vanilla GAN, WGAN as presented above still exhibits undesired behavior in practice. In particular it is prone to (i) experience exploding or vanishing gradients; (ii) fails to match higher-order moments in pulling p_{data} and p_g close. As a remedy which is very robust in practice, Gulrajani et al. [27] suggest a *gradient penalty* method to replace the weight clipping method (to which they attribute the behaviors above). The following penalty term is added in the loss function:

$$\lambda \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{x}}} \left[(\| \nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}}) \|_2 - 1)^2 \right] \tag{11}$$

where $\hat{\boldsymbol{x}} = \epsilon \cdot \boldsymbol{x} + (1 - \epsilon) \cdot G(\boldsymbol{z})$, an interpolation of the true data and the generated data. We will not explore the details of this method. Interested readers are encouraged to read the original paper which is very well and clearly written [27]. The gradient penalty is implemented as follows (added after the WGAN implementation and get rid of weight-clipping)[10]:

```
# Gradient penalty
#   Gulrajani et al. (2017), Algorithm 1.
```

[9] https://github.com/suwangcompling/GAN-tutorials/blob/master/Wasserstein GAN (original weight-clipping, MNIST demo).ipynb.

[10] https://github.com/suwangcompling/GAN-tutorials/blob/master/Wasserstein GAN (gradient-penalty, MNISTdemo).ipynb.

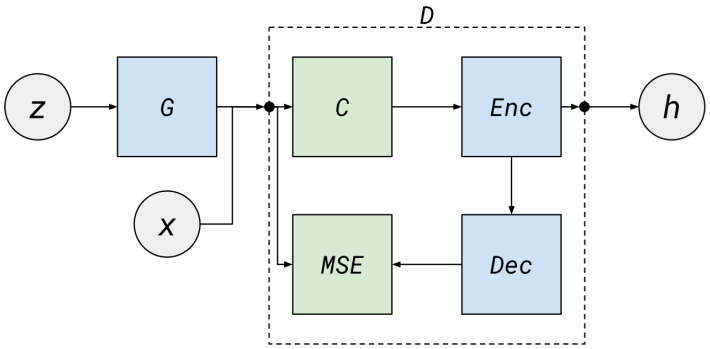

Figure 3: Adversarial Document Model: a variant of Energy-based GAN

```
#   \lambda * (||grad(D(x_hat))||_2 - 1.0)**2
epsilon = tf.random_uniform([], 0.0, 1.0)       # get random number.
x_hat = epsilon*x_placeholder + (1-epsilon)*Gz  # compute x_hat by interpolation.
Dx_hat = d_net(x_hat, reuse=True)               # compute D_w(x_hat).
scale = 10.0                                    # set scale=\lambda.
dDx = tf.gradients(Dx_hat, x_hat)[0]            # compute grad(D(x_hat)).
dDx = tf.sqrt(tf.reduce_sum(tf.square(dDx), axis=1)) # compute ||grad(D(x_hat))||_2.
dDx = tf.reduce_mean(tf.square(dDx - 1.0) * scale) # compute final penalty term.
d_loss = d_loss + dDx                           # apply to original loss.
```

Finally two training tips: (i) the scaling factor λ works well empirically across a wide range of tasks when set to 10.0, as per the recommendation in the paper; (ii) the results are generally better if we replace batch norm with layer norm for the discriminator/critic.

5 Applications in NLP

As briefly mentioned in Section 1, recent years have seen applications of GAN in the space of NLP. In this section, I take Document Modeling [8] and Dialogue Generation [9] as examples to demonstrate the potential of GAN as a flexible and versatile learning framework.

Document modeling. Glover (2016) [8] proposes an *Energy-based* GAN [13] (ADM)[11] where the generator is a regular MLP, while the discrimator is a *Denoising Autoencoder* (DAE) [14] (as the energy function [15]) instead. The model is pitched against a highly well-tuned *Restricted Boltzmann Machine* (RBM) [17] based system DocNADE [17], and a baseline (a stand-alone discriminative DAE classifier). The architecture is illustrated graphically in Figure 2.

In ADM, a document is modeled as a binary bag-of-words vector $\boldsymbol{x} \in \{0, 1\}^V$, where V is the size of vocabulary. D (the DAE discriminator) takes two inputs: (i) a "real" document vector $\boldsymbol{x}$, and (ii) a fake document vector $G(\boldsymbol{z})$ generated by G (an MLP). In input vector is first processed through a corruption process[12] C to obtain vector $\boldsymbol{x}^c$ and feed $\boldsymbol{x}^c$ to a regular encode-decode component [14] (Enc, Dec).

$$\boldsymbol{h} = f(\boldsymbol{W}^e \boldsymbol{x}^c + \boldsymbol{b}_e) \tag{12}$$

$$\boldsymbol{y} = \boldsymbol{W}^d \boldsymbol{h} + \boldsymbol{b}_d \tag{13}$$

where $(\boldsymbol{W}^e, \boldsymbol{b}_e), (\boldsymbol{W}^d, \boldsymbol{b}_d)$ are the parameters of the encoder/decoder, respectively. The quality of the input is evaluated by *Mean Square Error* (MSE) as a metric for reconstruction error.

$$MSE = \frac{1}{V} \sum_{i=1}^{V} (\boldsymbol{x}_i - \boldsymbol{y}_i)^2 \tag{14}$$

[11] The model is listed in [8] with the name *Adversarial Document Model*.

[12] Vincent et al. (2010) [14] show that formulating a autoencoder reconstruction task as a denoising task (with corrupted input) helps the autoencoder to generalize better.

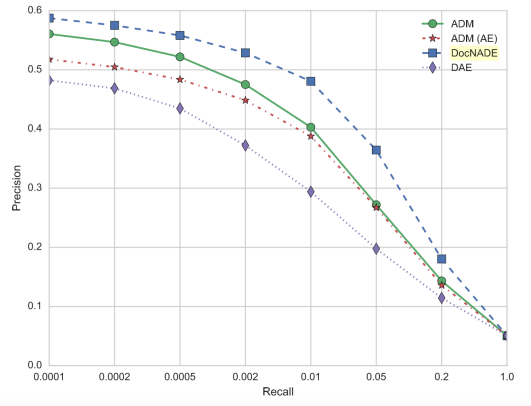

Figure 4: ADM results

In training, G and D strive to minimize their respective energy functions[13]:

$$f_D(\boldsymbol{x}, \boldsymbol{z}) = D(\boldsymbol{x}) + \max(0, m - D(G(\boldsymbol{z}))) \tag{15}$$

$$f_G(\boldsymbol{z}) = D(G(\boldsymbol{z})) \tag{16}$$

The process amounts to the same effect as Minmax Game (Eq. 3), in that D increases the score it assigns the real data (i.e. $\boldsymbol{x}$) while lowering the score for the fake data (i.e. $G(\boldsymbol{z})$) generated by G, and G increases the score of the fake data.

In his experiments, [8] formulates a document classification task with the 20 Newsgroup Dataset [18], where the model takes a query document $\boldsymbol{d}$ as input and produces a set of output documents that are closest[14] to $\boldsymbol{d}$. The results are shown in Figure 3 as a precision-recall curve. While producing overall weaker performance than the state-of-the-art DocNADE, ADM demonstrates its power by defeating the strong baseline DAE models by a respectable margin, showing its promise[15] in more sophisticated formulation.

Dialogue generation. Li et al., (2017) [9] presents an interesting GAN-based dialogue generation system. They formulate a reinforcement learning [20] based *Turning Test* where the goal is for the generator G, under the guidance of a discriminator D, to learn to generate realistic responses to input sentences that are indistinguishable from responses given by humans. D guides G by giving it feedbacks in the form of reinforcement rewards: Positive reward for realistic responses, and negative reward for non-realistic ones.

Concretely, G takes the form of a sequence-to-sequence *Recurrent Neural Network* (RNN) [21] which generates response y based on a dialogue history x. D is an autoencoder-based binary classifier [22] that takes a (dialogue history, response) pair $\{x, y\}$, and produces a label indicating whether the input is generated by a human (i.e. real data) or G (i.e. fake data). It further returns a reward score to guide G: $Q_+(\{x, y\}), Q_-(\{x, y\})$ for positive and negative rewards, respectively.

Deferring the details of the model to the original work [9], I only show some sample outputs from its two variants[16] evaluated therein to demonstrate the quality of the generated responses (Figure 4). Here, the *input* is a dialogue probe, and the following lines are the respective responses of the models give to the probe. The two GAN's produce apparently superior responses by human judgment. In addition to generating good responses, [9] also shows the reliability of their model (see Table 3 in the original work).

[13] One may consider an energy function as a family of loss function [15].

[14] By cosine similarity.

[15] In image generation tasks, while sometimes fall short behind VAE [19] in classification tasks [2], GAN often times generates more realistic images by human judgment. This thus leads us to reasonably believe in its potential in document-based tasks.

[16] REGS improves on REINFORCE by ameliorating its tendency for *mode collapose* — for G to generate the same fake data over and over again [2].

Input	tell me ... how long have you had this falling sickness ?
Vanilla-MLE	i 'm not a doctor .
Vanilla-Sample	well everything you did was totally untrue .
REINFORCE	i don 't know how long it 's been .
REGS Monte Carlo	A few months, I guess .
Input	so i had the doctors test sammy 's response to conditioning .
Vanilla-MLE	sammy wrote the test sammy wrote the test .
Vanilla-Sample	objects pick objects up objects objects objects objects objects
REINFORCE	i 'm not sure that 's relevant .
REGS Monte Carlo	so he took the pills .
Input	what are you going to do ?
Vanilla-MLE	i 'm going to the movies .
Vanilla-Sample	get him outta here first !
REINFORCE	i 'm going to get you .
REGS Monte Carlo	i 'm going to get you a new car .
Input	they fear your power your intellect .
Vanilla-MLE	you 're the only one who knows what 's going on .
Vanilla-Sample	when they are conquered and you surrender they will control all of us .
REINFORCE	i 'm afraid i 'm not ready yet .
REGS Monte Carlo	i 'm not afraid of your power .

Figure 5: Dialogue Generation (REINFORCE and REGS are GANs)

6 Conclusion

In this tutorial, I started by giving an in-detail description of GAN (Section 1) and step-by-step derivation in related proofs (Section 2). In Section 3 and 4 I presented simple implementations of the original GAN and the *Wasserstein GAN*. Finally I demonstrated the flexibility of GAN as a novel neural net architecture with examples of its application in NLP (Section 5).

References

[1] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014) *Generative Adversarial Nets*. NIPS 2014.

[2] Goodfellow, I.J. (2016) *NIPS 2016 Tutorial: Generative Adversarial Networks*. NIPS 2016. CoRR, arXiv:1701.00160.

[3] Jang, E. (2015) *Generative Adversarial Nets in Tensorflow*. Eric Jang's Blog: `blog.evjang.com/2016/06/generative-adversarial-nets-in.html`.

[4] Lotter, W., Kreiman, G. & Cox, D. (2015) *Unsupervised Learning of Visual Structure Using Predictive Generative Networks*. CoRR, arXiv:1151.06380.

[5] Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2016) *Photo-realistic single Image Super-resolution Using a Generative Adversarial Network*. CoRR, arXiv:1609.04802.

[6] Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016). *Image-to-image Translation with Conditional Adversarial Networks*. CoRR, arXiv:1611.07004.

[7] Donahue, J., Krähenbühl, P. & Darrell, T. (2017) *Adversarial Feature Learning*. CoRR, arXiv:1605.09782.

[8] Glover, J. (2016) *Modeling Documents with Generative Adversarial Networks*. CoRR, arXiv:1612.09122.

[9] Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A. & Jurafsky, D. (2017) *Adversarial Learning for Neural Dialogue Generation*. CoRR, arXiv:1701.06547.

[10] Chen X., Athiwaratkun, B., Sun, Y., Weinberger, K. & Cardie, C. (2016) *Adversarial Deep Averaging Networks for Cross-lingual Sentiment Classification*. CoRR, arXiv:1606.01614.

[11] Zhang, Y., Barzilay, R. & Jaakkola, T. (2017) *Aspect-augmented Adversarial Networks for Domain Adaptation*. CoRR, arXiv:1701.00188.

[12] Ng, A. & Jordan, M.I. (2002) *On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naïve Bayes*. NIPS 2002.

[13] Zhao, J., Mathieu, M. & LeCun Y. (2016) *Energy-based Generative Adversarial Network*. CoRR, arXiv:1609.03126.

[14] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P-A. (2010) *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*. JMLR 2010.

[15] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M-A. & Huang, F.J. (2006) *A Tutorial on Energy-based Learning*. In Bakir, G., Hofman, T., Schölkopf, B., Smola, A. & Taskar, B. (eds.) Predicting Structured Data. MIT Press.

[16] Hinton, G.E. (2002) *Training Products of Experts by Minimizing Contrastive Divergence*. Neural Computation, vol. 14, no. 8, pp. 1607–1614.

[17] Larochelle, H. & Lauly, S (2012) *A Neural Autoregressive Distribution Estimator*. NIPS 2012.

[18] Lang, K. (1995) *Newsweeder: Learning to Filter News*. ICML 1995.

[19] Kingma, D.P. & Welling, M. (2014) *Auto-encoding Variational Bayes*. ICLR 2014.

[20] Williams, R.J. (1992) *Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning*. Machine Learning, vol. 8 (3-4): 229–256.

[21] Sutskever, I., Vinyals, O. & Quoc, V.L. (2014) *Sequence to Sequence Learning with Neural Networks*. NIPS 2014.

[22] Li, J., Luong, M-T. & Jurafsky, D. (2015) *A Hierarchical Neural Autoencoder for Paragraphs and Documents*. CoRR, arXiv:1506.01057.

[23] Gutmann, M. & Hyvärinen A. (2010) *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models*. In Proceedings of AISTATS. Sardina, Italy.

[24] Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R. (2010) *Deconvolutional Networks*. In Proceedings of CVPR.

[25] Radford, A., Metz, L., Chintala, S. (2016) *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. In Proceedings of ICLR.

[26] Arjovsky, M., Chintala, S., Bottou, L. (2017) *Wasserstein GAN*. CoRR.

[27] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A. (2017) *Improved Training of Wasserstein GANs*. In Proceedings of NIPS.