Towards Robust VAE Defense Against Multiple Adversaries Through Adversarial Training and Biased Sampling Process

Aeberli Andreas, Elmiger Tobias,Jakob von Kalle, Xuran Li

***Abstract*—Variation auto encoders (VAE) are widely applied in the deep learning literature as a defense mechanism against adversarial attacks. Recent research, however, have shown adversarial input against VAE continue to exist despite this wide spread application. In this paper we explored the aspect of training a robust VAE against multiple attacks using adversarial training and a biased sampling schedule of generated adversarial examples from replay buffer during the training process.**

I. Literature Research

Deep neural network have been widely applied across a variety of tasks in the area of computer vision. However, building a robust network against adversarial attack remain a challenging goal despite substantial research. For image classification task, various classical methods of attack have been proposed to reduce classification accuracy by slightly modifying the image without changing its semantic, including projected gradient descent (PGD) and Fast Gradient Sign Method (FGSM).[1] Autoencoders have been used as a defense mechanism in the past to recognise adversarial sample, and reduce inserted noise created by the attacker [2], [3], [4]. However, noval attack pattern against autoencoders have been discovered in recent years. For instance, Tabacof et al. showed that by minimising the difference between the latent representation of target and source image attacker can generate images from different classes in MNIST dataset [4]. On the other hand, Kos et al. showed that minimising the loss between reconstructed image from adversary input and reconstructed target image produces comparable result [5]. Athalye et al. showed that BPDA algorithm can be used to successfully attack state-of-the-art autoencoder even without access to model gradient[6]. Thus, autoencoders are considered ineffective against these targeted attacks [7]. Despite these alarming discoveries, autoencoders and its variations remains as one of most the common defense approaches across various field of applications among the latest publications. Huang[8] detects adversarial samples with a discriminator in the decoding and purifying them in latent space by introduction of a gradient based search towards a reconstruction image closest to to the input. In the field of Variational Graph Autoencoders (VGAE) Ao Zhang [9] proposes to reconstruct the adjacency matrix which in turn reduces the effects of the perpetuation by the attacker. Houchao Gan [10] uses a VAE trained to reconstruct known adversarial samples for Autonomous Vehicles. We therefore feel the need to design a robust VAE trained to defend against both classical attacks and novel approaches. To this end, we propose to follow the practise of formalising the process of training robust VAE as a minimax algorithm aimed to minimise the maximum of expected loss achievable by the attacker [11]. This approach, however, comes at the price of increased training time. To reduce training time and increasing training efficiency, we further propose to utilise prioritised replay buffer [12] to store and sample adversarial inputs generated by the inner maximization problem, and propose several approaches to guide the sampling process from the replay buffer based on signals generated during the training process.

II. background

A. Variational Auto Encoder

Autoencoders and its variations are commonly used in the computer vision literature to perform tasks including image reconstruction, learning latent representation and image generation. The main components of autoencoders are two neural networks, encoder $f_{enc}(x)$ and decoder $f_{dec}(z)$. Given a image x as a input tensor,the encoder compresses the image into a low dimensional latent representation $z = f_{enc}(x)$ and the decoder uses z to reconstruct the image $x^{'} = f_{dec}(z)$.

The variational autoencoder (VAE) assumes that the input images are generated under a probability distribution determined by a hidden factor, which is modeled by the latent variable z. Given a input image, encoder of VAE returns z as a Gaussian probabilistic distribution over the latent space, which decoder used to sampled a particular latent value for image reconstruction. The training is guided by the following loss function,

$$L(x, x^{'}) = E_q[logp(x|z)] - D_{KL}[q(z|x)||p(z)] \qquad (1)$$

where $q(z|x)$ is the approximation of the true posterior distribution $p(z|x)$, and p(z) is the prior distribution of hidden factor. D_{KL} is the KL divergence between the learned approximation and prior distribution of the underlying distribution. In the context of image reconstruction, $E_q[logp(x|z)]$ is equivalent to $H(x, x^{'})$, the binary cross entropy between $x^{'}$ and x . For each x, the encoder returns the mean μ and variance σ^2 of $q(z|x)$, and the decoder samples z from $q(z|x)$ using $z = \mu + n * \sigma$, where n is sampled from a normal distribution. Decoder then use it to reconstruct the original image. By modeling and regularizing the latent space, VAE is less likely to overfit the training data during image generation and thus presents a harder challenge to attacks compared with vanilla autoencoders. This is because instead of reducing the

latent space to a fixed vector you reduce it to a distribution. So for decoding one can simply draw from the distribution using the mean and standard deviation.

B. Adversarial Attacks

Adversarial attacks have been an active research area over the past decade. Attacks can be either targeted or untargeted. We present targeted attack as follows: given input x from class c, attacker present a slightly perturbed version of x, x*, such that $x^* \in c$ by human standard. An attack is successful if the network reconstructs an image from the target class t image reconstruction. For untargeted attack, the attack is considered successful if the reconstructed image of x* is not of class c. In this work we focused on the defense of white box attack where attacker have full access to the model.

C. Replay Buffer and Prioritised Experience Replay

In the context of Reinforcement Learning, many agents (programs) utilise neural networks to train its policy, which decides the next action given observation from the environment. Deep Q network agents use neural network to update a Q value function, which takes the current state s and action a and returns a estimation of total reward that can be obtained by taking this action at the current state Q(s,a). Replay buffer is widely used to store past experience of a Deep Q Network agent, which would be later sampled and used to update the policy of the agent. Prioritised Experience Replay (PER) [13] is a special type of replay buffer that samples each experience according to its probability $P(x_i) = p(x_i)^{\alpha_i} / \sum_{k=1}^{\infty} p(x_k)^{\alpha_k}$ where $p(x_i)$ is the priority of its sample and α_k is a hyperparameter ranging from 0 to 1 that determines the level of prioritization applied. The priority of i-th sample is given as the absolute value of Temporal Difference (TD) error δ_i plus a small constant, i.e. $p(x_i) = |\delta_i| + \epsilon_{cst}$. Temporal Difference (TD) error is the difference between previous and current estimation of the Q value function Q(s,a). For each sampled experience, a importance sampling weight is multiplied with the calculated loss to balance the effect of over sampling. The weight is given by $w_i = 1/(N * P(i))^{\beta}$, where N is total number of experiences stored in the replay buffer, and β is a hyperparameter within 0 to 1. Using Prioritised Experience Replay allows a reinforcement learning agent to update the priority of sampled experience, while using these experience to update the policy, thus allowing the agent to focus on valuable experience that leads to further learning.

III. Methodology

In this work we present our model as a VAE trained using adversarial training techniques against multiple adversaries with experiences stored and sampled from a customized replay buffer.

For the ease of comparison with existing work, the models are trained on inputs from MNIST dataset,[14] where images consists of photos of digits from 0 to 9. The training set contains 50000 labeled images, while the validation and test set have 10000 each. Each image is of size 1*28*28. The batch size of input images are set to 128 for all training and testing runs.

In the following sections we will describe each part of the model in details.

A. Variational Auto Encoder

We present 2 versions of VAE: fully connected (fc) and cnn-based (cnn). The encoder of the fully connected VAE takes the flattened input tensor of size 1*784 and applies a fully connected linear layer to map it to a hidden tensor of size h_dim =400. the hidden tensor is then passed through relu activation and mapped to output mean μ and variance σ^2 of the posterior distribution $q(z|x)$, each of size z_dim=20. The decoder samples z using the mean and variance, then applies the following modules in linear order: 1. linear layer to map z back to a tensor of size h_dim, 2. relu activation 3. linear layer to map tensor of size h_dim to a tensor of size 784, 4. Sigmoid activation to map the pixel range back to 0 to 1.

The cnn-based VAE takes a image of size 1*28*28, and pass it through one layer of convolutional layer with channels =16, kennel size =3, stride =2 and padding =1, followed by relu activation. The tensor is then flattened and passed down through fully connected layers with relu activation to produce the mean and variance vector of $q(z|x)$. Decoder then samples z using the mean and variance, map it to a tensor of size h_dim, before applying 2d deconvolution layers (nn.ConvTranspose2d) to reconstruct the target image of size 1*28*28.

As shown in section IV, cnn-based VAE out-performs the fully connected VAE in most testing scenarios.

B. Adversarial Attacks

We select one or multiple of the following attackers as the adversary to generate attack samples (Fig. 2). The attack samples were used in the VAE's training (section III-C) or/and in the evaluating of a trained VAE (Fig. 1).

1) Fast Gradient Sign Method (FGSM): FGSM is a white box attack that performs one step of gradient update along the direction of the gradient. Given step size ϵ, the attack image can be generated using the following formula:

$$x^* = x + \epsilon * sign(\nabla_x l(x, x^{'})) \tag{2}$$

2) Projected Gradient Descent (PGD): Projected Gradient Descent on the other hand is a multi-step gradient update process, where during each step FGSM is used to create x^*, and the result is projected back into a predefined perturbation region S(x). The region S(x) is specified by a l_p norm and a ϵ_{range} range. In our work we only examined the untargeted version of FGSM and PGD attacks, where the objective is to increase the training loss while remaining in the bounded region S(x).

3) Gaussian Noise Attack (GNA): A gradient free and simple method to possibly fool networks is the Gaussion Noise Attack. It adds Gaussian noise to the attacked picture I_o, where the mean μ and variance σ^2 was chosen in a range of ± 0.1 individually for each picture. The parameters have been chosen

as large as possible (Fig. 2), to still have a clear visible number for the human eye. The sample attack I_a is described using equation-(3)

$$I_a = I_o + \mathcal{N}(\mu, \sigma^2) \tag{3}$$

4) Geometric Transformation Attack (GTA): Another gradient free and simple method is the used geometric transformation attack. Deep convolutional neural networks have been shown vulnerable to geometric transformation attack [15]. In the GTA, the selected picture is rotated in a range of $\alpha = \pm 30$ degree, scaled in a range of s=80-120 $\%$ and translated in x and y direction up to d=1 $\%$ of its original size, whereas the pixel size remains the same. Similar to the GNA, the parameters in the GTA have been chosen to be recognizable as a specific number for the human eye (Fig. 2).

5) VAEA: A method using the VAE loss function itself this adversarial approach perturbs a source image x into x^* so that its reconstruction $\hat{x}^*$ imitates the reconstruction $\hat{t}$ of a target image t. The adversary can precompute the reconstructions $\hat{t} = f_{dec}(f_{enc}(t))$ by evaluating it with the model. The reconstruction $\hat{x}^*$ on the other hand needs to be computed in each iteration of the perpetuation steps because the loss is computed on $\hat{x}*$ and $\hat{t}$ instead of x and $\hat{x}$ as is done in (1). At the same time we want to minimize the distance between x and x^*. Thus the resulting optimization problem as mentioned in literature [5] is (4)

$$\text{argmin}_{x*}\, \delta L(x, x^*) + \mathcal{L}_{VAE}(\hat{x}^*, \hat{t}) \tag{4}$$

6) LatentA: Similar to the VAE-Attack this adversary attack the generation process of the VAE. This time though the attack is focused on the latent space $z = f_{enc}(x)$. Again a source x_s and a target x_t is picked. The latent space of the target can precomputed with $z_t = f_{enc}(x_t)$. Iteratively perturbing x_S into x^* we try to minimize latent space distance between z_t and z_{x^*}. Having the similarity constraint already seen in VAE-Attack, the optimazitionproblem boils down to (5).

$$\begin{aligned} &\text{argmin}_{x*}\, \delta L(x, x^*) + \mathcal{L}_{Latent}(x^*, z_t) \\ &where\ L_{Latent}(x^*, z_t) = L(f_{enc}(x^*), z_t) \end{aligned} \tag{5}$$

In both the VAE-Attack and the Latent Attack L can be the squared loss function. In both we used PGD to iterate towards a viable x^*.

7) RANDOM: In the RANDOM method, the attack samples are generated by choosing randomly FGMS, PGD, GNA or GTA by equal probability.

C. Adversarial Training

During Adversarial Training, our model utilise minimax algorithm to minimise the maximal expected loss achievable by the attacker, such that the model parameter $\theta = argmin_\theta L(\theta)$ given a bound on the perturbation region around x, S(x)[11]:

$$L(\theta) = E_{(x,x')\sim D}[max_{x^{ad} \in S(x)} L_{VAE}(x, x^{'}_{ad})] \tag{6}$$

where $x^{'}_{ad}$ is the reconstructed image generated by the model given the attack image, and L_{VAE} is the binary cross entropy loss. For GNA attacks , S(x) is bounded by the maximum variance of the noise. For GTA attacks, S(x) is bounded by the maximum rotation angle and translation. For FGSM and PGD attacks, S(x) is specified by the ϵ_{range} and l_p norm. For Latent Attack and VAEA attack, there are no limitation on boundary S(x).

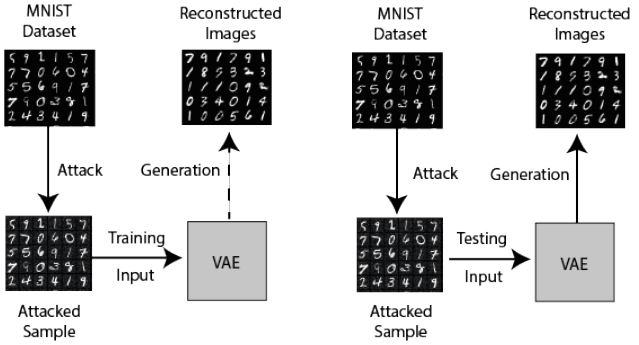

Fig. 1: Training of the VAE (left) and testing (right).

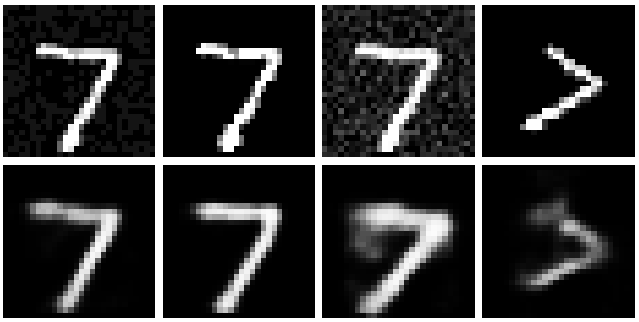

Fig. 2: Attack samples of the generated images (top) and the reconstruction using the fc VAE (bottom). From left to right: FGSM, PGD, GNA($\mu = 0.1$,$\sigma^2 = 0.1$), GTA($\alpha = 30, s = 120, d = 1\%$).

During each episode of the training process, one of the attacks are selected from the list of specified attacks, and the adversaries images of this type are then generated for the current batch of sample images, The model then calculates the loss between the original image and the reconstructed attack image $L_{vae}(x, x^{'}_{adv})$ and perform a gradient step on the loss.

D. Customized Replay Buffer and training against multiple adversaries

We present our own customized replay buffer as a modified version of the PER that better suits the process and requirement of supervised training. The replay buffer stores each entry as a tuple of original images x, previous loss loss, and the type of the attack to perform on these images attack_type, as well as other information that we didn't utilise in the scope of this project. The TD error is calculated as the absolute difference of previous and current loss instead of estimated Q values, i.e. $\delta = abs(l_{new}(x, x^{'}_{adv}) - l_{prev})$, where x is the sampled image and $x^{'}_{adv}$ is gernerated using the type of attack stored in the replay buffer. In addition, instead of using $p(x_i) = |\delta_i| + \epsilon_{cst}$ to set the priority of each example, we

instead set $p(x_i) = |\delta_i| + \epsilon_{cst} + w_l * loss_{new}(i)$ where w_l is a hyperparater to adjust the amount of influence the loss have on priority. This is particularly set as we observe that unlike the case of reinforcement learning where Q value doesn't directly influence the reward, minimising loss is the objective of the model and thus it make sense to focus more on examples with higher loss.

Doing so allows the training schedule to utilise multiple adversarials for the inner maximization problem, and sample attacks porpotionally to the loss model have on these adversarial samples. In theory this would allows the model to learn faster against attacks that have a higher loss, and thus reduce its training time and gives it an advantage when compared with adversarial training with RANDOM attacks. However our early testing shows this is not the case and the result is discussed in section IV.

E. Critic

While the TD error of the loss can be used as an effective baseline for the priority of the training samples of the replay buffer, it doesn't utilise all the information generated by the training process. In order to better incorporate the information and produce priority closer to the true usefulness of each sample, we propose to use a critic to predict the priority of each sample. Given a sampled batch of tuples of (original images, loss l, attack_type) from the replay buffer during the sampling process, the critic returns a single value l_{max} for each sample image, that indicates the predicted maximum loss achievable by the attacker. This predicted loss is then used to calculate the TD error $\delta_i = L_{TD}(l_{max}, l)$ and the predicted priority $p(i) = |\delta_i| + \epsilon_{cst} + w_l * loss_{new}(i)$ of each sample in the batch. Then the network selects the top 10 percent of samples that have the highest priority as the real sampled images to train on. After the loss of the adversarial attacks on the real sample images is calculated, the critic is then trained with a mean squared error loss between l_{max} and $l_{vae}(x, x'_{adv})$. Due to the small image size and the tight time limit of this project, the critic is modeled as a fully connected network that only takes in the original image as input.

Algorithm 1 contains the pseudo code for training process of the full model:

Algorithm 1 adversarial training with custom replay buffer

Init: training set, VAE f, epochs, batch size n_1,critic c, list of adversaries advs, sample size of the replay buffer n_2, replay buffer rb, train loss =0
for $iteration$ e=1,2,. . . , epochs **do**
    sample a mini-batch of n_1 (x,label) from the training set
    choose a attack type from advs and obtain the attack images x_{adv},
    x_{recon}, μ, σ^2 = $f(x_{adv})$
    l_{vae} = f.loss($x_{recon}, x, \mu, \sigma^2$)
    perform a gradient update on l_{vae}
    train_critic (x,attack type, l_{vae})
    rb.add($x, l_{vae}, attacktype$)
    train loss += sum(losses)
    t= e * len(train_loader) + idx, where idx is the current idx in the loader of training set.
    if t mod train_freq == 0 :
    xs,losses,att_type = rb.sample(n_2)
    td_errors,losses_new,new_idxes = train_replay_buffer(c, xs,losses,att_type)
    rb.update_priority(td_errors, losses_new,new_idxes)
    end if
end for

Algorithm 2 train_replay_buffer

Init: critic c, original image sampled from replay buffer xs,previous losses of the samples l_{prev},previous attack type of the sample att_type
pred_losses=c.forward(xs,attack type)
new_idxes=select_best(losses,pred_losses),
obtain the attack images x_{adv} for xs[idxes]
x_{recon}, μ, σ^2 = $f(x_{adv})$
l_{new} = f.loss($x_{recon}, x, \mu, \sigma^2$)
perform a gradient update on l_{new}
train_critic (x,attack type, l_{new})
TD error = absolute(l_{new}-l_{prev})
return TD error, l_{new}, idxes

IV. Results

A. Evaluation

The following models are evaluated: vanilla VAE, VAE with adversarial training, VAE with adversarial training and replay buffer with single adversarial(RB), VAE with adversarial training and replay buffer with multiple adversarial(RB Multi), and replay buffer with multiple adversarial and critic(RB Multi Critic). We use the Vanilla Variational autoencoder to serve as a baseline.

The models are evaluated on the following metrics: accuracy without adversarial attack, accuracy while under one of FGSM, PGD, GNA, GTA attack. Due to the tight time limit and our teammate falling ill, we didn't have time to test LatentA and VAEA attack from[5]

Figure 3 shows the train loss and validation loss of VAE with vanilla adversarial training using RANDOM attack without replay buffer, adversarial training with multiple adversarial replay buffer, and adversarial training with multiple adversarial replay buffer and critic. Generally, the cnn type (second column) and fc type (first column) models achieve very similar loss, with CNN usually end up slightly better at the end of the training. We also found that during testing the loss of the cnn vanilla VAE is generally smoother. Comparing the validation losses, the GTA loss in all models tend to be significantly higher, whereas all the other adversarial attacks have no significant variations. The additional features using multi adversarial attacks and critic did not reduce the validation and test loss. In Figure 4, the both version of VAE are trained using GNA attacks. Interestingly, We found that training with

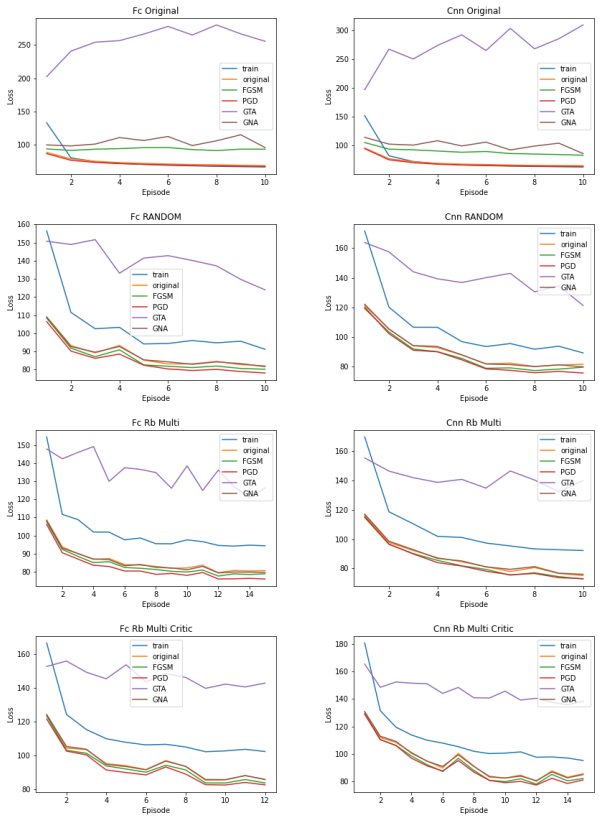

Fig. 3: Fc VAE (left) and CNN VAE (right) from top to bottom: original, random, rb multi, rb multi with critic.

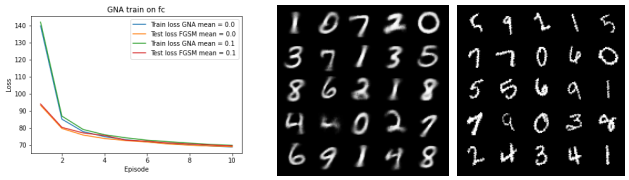

Fig. 4: GNA training on fc with mean = 0.0 and 0.1 (left). Generated images of cnn vanilla VAE trained with GTA attack samples (center) and PGD attack samples (right).

GNA attack only also consistently reduce the loss of FGSM and PGD attack on the validation set. Removing the mean of the GNA attack during the training had no effect on the loss (Fig. 4).

Table I shows the time of adversarial training on a single method using fully connected VAE in minutes and seconds, where the PGD's training time is much larger than the other methods. Additionally, the CNN models require more than 6 times higher training time.

VAE	original	FGSM	PGD	GNA	GTA	RANDOM
fc	02:32	02:34	07:25	02:39	02:37	03:53
cnn	18:03	17:34	55:16	17:37	20:08	30:08

TABLE I: Training Time

V. Discussion and Future Work

In this section we discuss the potential reasons that our proposed model did not out perform baseline and outline future work. Performance of our proposed model might not be optimal for the following reason: 1. we didn't have time to perform extensive hyperparameter search. most parameters are set as default for the example shown in section IV. 2. the critic is defined as a very simplistic fc model that only takes the current image into account, which might be insufficient for the prediction task. 3. the small image dimension restricts the complexity of the models we are capable of training, as well as amplifies the overhead of maintaining a replay buffer compared with common practise in reinforcement learning.

Due to the tight timeline (I only have 5 days to test my model and rush this report) and limited computational resources, we leave the following to future work:

1. perform a thorough hyper parameter search over important model parameters to find the optimal parameter for max performance.

2. explore the possibility of writing a better critic for the replay buffer, for example utilising the TD error of past losses.

3. perform testing on datasets with images of higher resolution to allow the utilization of deep cnn model, report l0 l2 loss for comparison with existing work.

4. test the model performance on image classification task.

Though this project has come to an end, I personally intend to continue on improving the model, assuming that I can find a Ph.D. who' s willing to supervise me on this :).

VI. Summary

Through this project we explored the transferability of the classic minimax algorithm, and showed that adversarial training can be successfully applied to VAE for image reconstruction. We implemented attacks using gradient based approaches, graphical attacks and also latent and VAEA attacks, however we didn't have time to test out latnet or VAEA attacks. We demonstrated that by using customized Prioritized Experience Replay we were able to successfully defend against multiple adversaries,In addition, we proposed and evaluated the usage of critic to guide the sampling process of the replay buffer. However Adversarial training with replay buffer, whether using critic or not, did not perform better than the baseline that samples all attacks randomly during adversarial training. We discussed the various potential reasons behind this and outline guideline for future research works. The code and models for this project can be found at: https://github.com/elmigert/VAE_ATTACKS_ANALYSES

References

[1] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[2] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[3] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1778–1787.

[4] P. Tabacof, J. Tavares, and E. Valle, "Adversarial images for variational autoencoders," *arXiv preprint arXiv:1612.00155*, 2016.

[5] J. Kos, I. Fischer, and D. Song, "Adversarial examples for generative models," in *2018 ieee security and privacy workshops (spw)*. IEEE, 2018, pp. 36–42.

[6] A. Athalye and N. Carlini, "On the robustness of the cvpr 2018 white-box adversarial example defenses," *arXiv preprint arXiv:1804.03286*, 2018.

[7] L. Schott, J. Rauber, M. Bethge, and W. Brendel, "Towards the first adversarially robust neural network model on mnist," *arXiv preprint arXiv:1805.09190*, 2018.

[8] W. Huang, S. Tu, and L. Xu, "Defense against adversarial examples by encoder-assisted search in the latent coding space," 2019.

[9] A. Zhang and J. Ma, "Defensevgae: Defending against adversarial attacks on graph data via a variational graph autoencoder," *arXiv preprint arXiv:2006.08900*, 2020.

[10] H. Gan and C. Liu, "An autoencoder based approach to defend against adversarial attacks for autonomous vehicles," in *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*. IEEE, 2020, pp. 43–44.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[14] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15] C. Kanbak, S. M. Moosavi-Dezfooli, and P. Frossard, "Geometric Robustness of Deep Networks: Analysis and Improvement," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4441–4449, 2018.