

Generative Adversarial Networks (GANs)

From **Ian Goodfellow et al.**

Presented by:- Ali Farooq

To:- Prof. Dr. Hauke Schramm

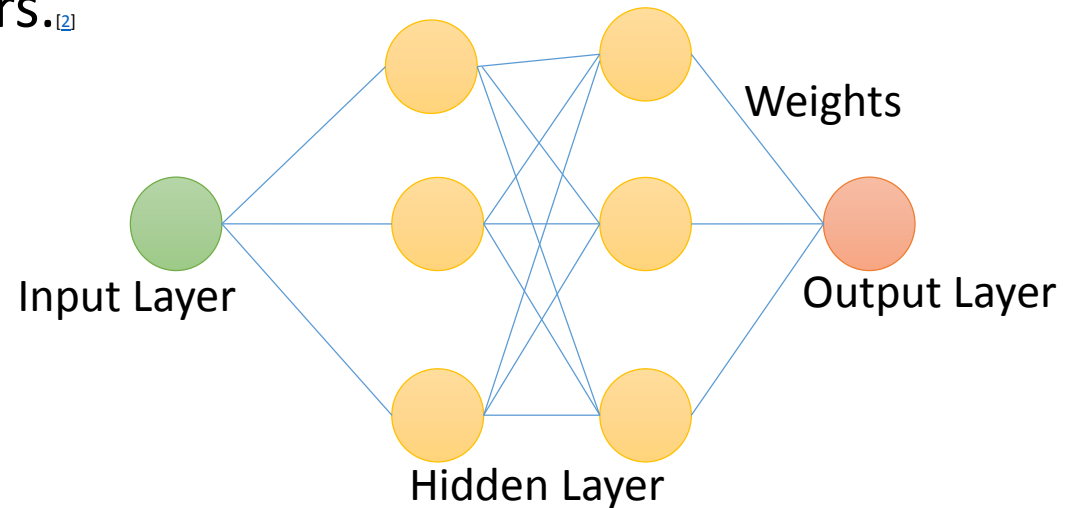
Contents

- Introduction
- What is GAN?
- Working of GAN
- GAN Function
- GAN Architectures
- GAN Applications
- Project Plan
- Reference

Introduction

- **Neural networks:**

- These are computing system
- Consist of many elements and layers.^[2]



- **Deep Neural Network:**

- Network with more than one or two hidden layers.
- This is inspired by the structure and functions of brain called artificial neural networks.^[1]

Introduction

- Networks learn from an examples.
- Networks use learning rules to learn and extract things from examples.
- **Learning rule** is a logic which improves the performance and results of network.
- Types of learning rules:
 - Supervised learning
 - Unsupervised learning
 - Reinforcement learning
- These rules are apply all over the network and updates the weights of the network.[\[3\]](#)

Introduction

- There are two types of models in Deep learning which are^[4]:
 - **Generative models** (model distribution of each class).
 - **Discriminative models** (learn boundary between classes).
- Generative model are difficult to approximate and estimate.
- (Ian J. Goodfellow et al., 2014) propose new generative model estimation procedure which sidesteps these difficulties.
- The model is Generative Adversarial Network (GAN).

What is GAN?

- GAN (Generative Adversarial Network) is the most interesting idea in the last 10 years in Machine Learning.[\[5\]](#)
- GAN can learn to mimic any distribution of data.
- GAN consist of two deep learning neural networks generative and discriminative.

Working of GAN^{[5][6][7][8][9][10]}

- In GAN two deep neural networks are in competition with each other.
- One network which is called a generator it generates forgeries.
- Other network is called discriminator which receive both forgeries and real images and its aim to differentiate them.
- The discriminator network is a standard convolutional network.^[16]
- The generator network is an inverse convolutional network.^[17]

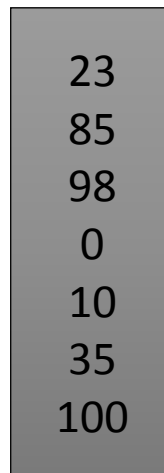
Working of GAN

- Both networks are trying to optimize itself.
- If any one network changes its behavior its effect on other and vice versa.
- Both networks consist of multi and fully connected multilayers perceptron.
- Training of these networks are done one by one.
- Both networks are playing minimax game.

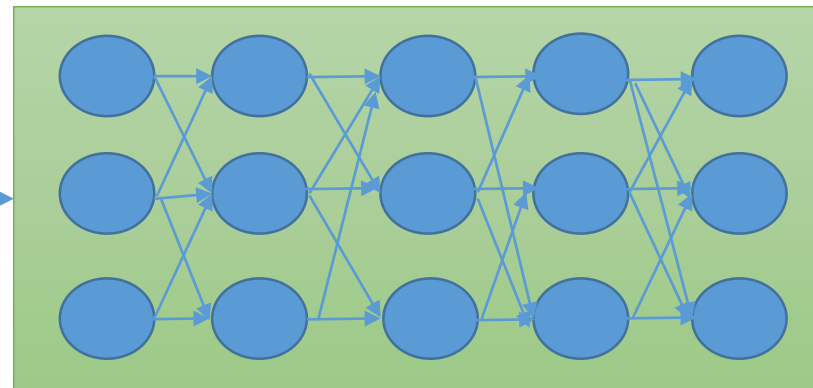
Working of GAN

- Steps^[11]
 1. Define the problem
 2. Define architecture of GAN: There are many architectures of GAN. Each architecture has his own features and best fit according to the nature of problem.
 3. Random number of vector feed to generator network to generate fake images.

D-dimensional Random Vector

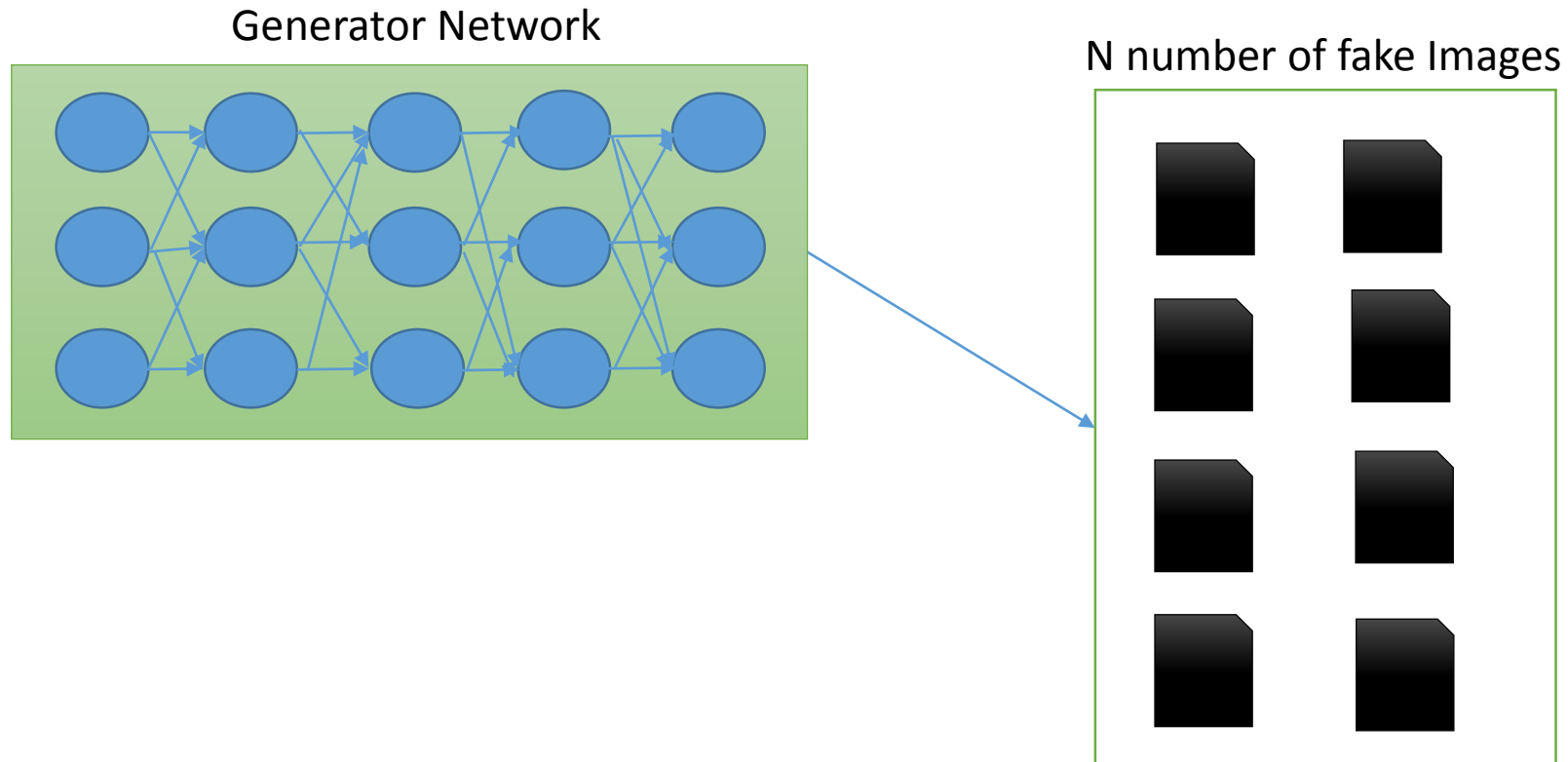


Generator Network



Working of GAN

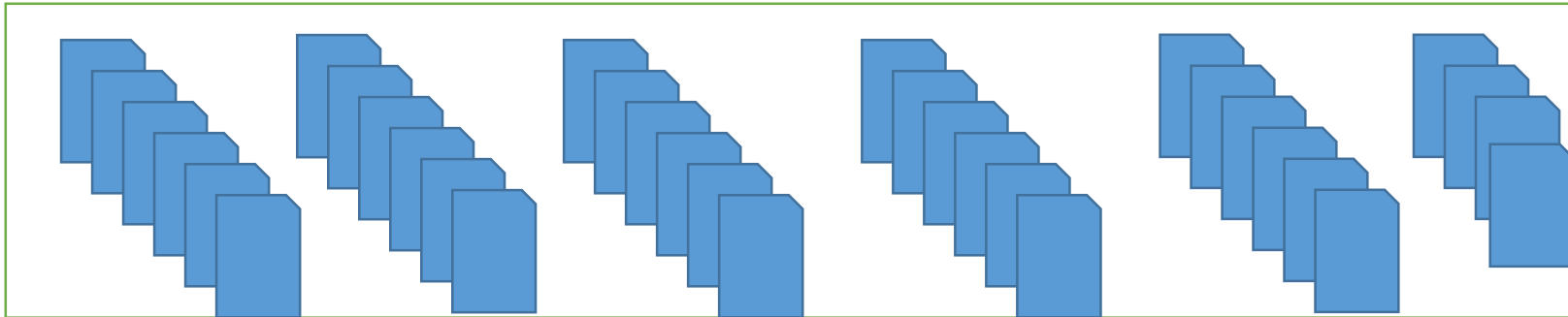
4. Generator network generate fake images.



Working of GAN

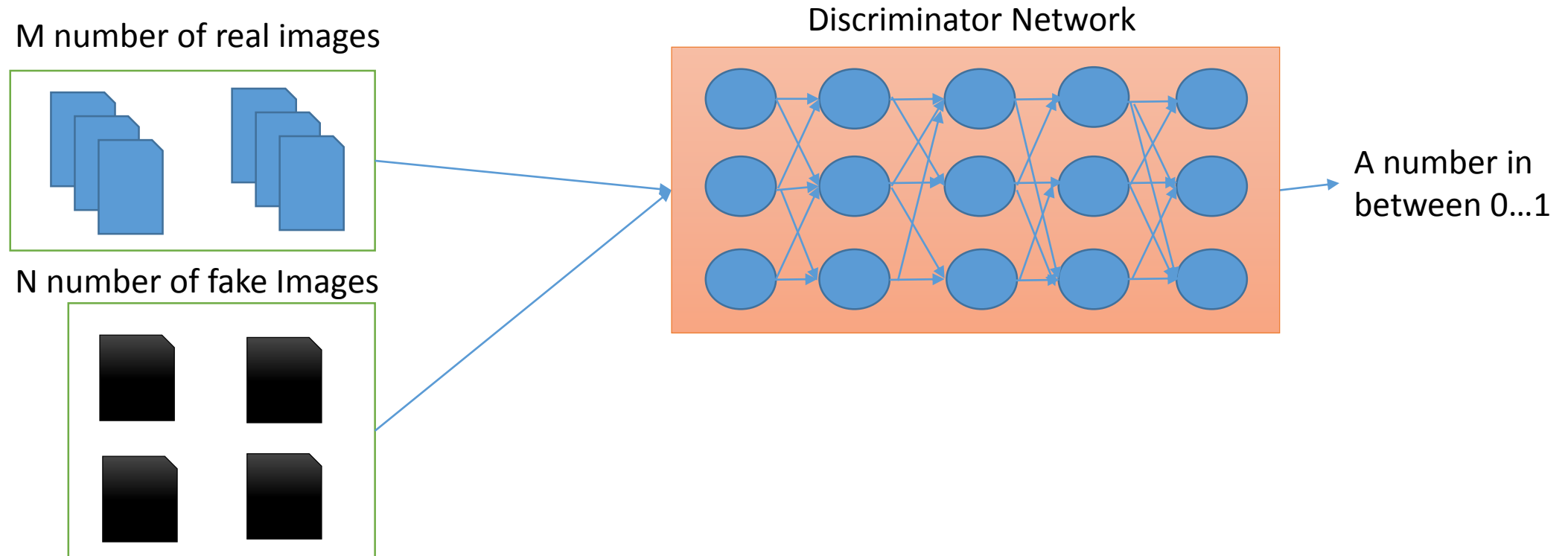
5. Now take a large sample of real images which are related to the problem.

M number of real images



Working of GAN

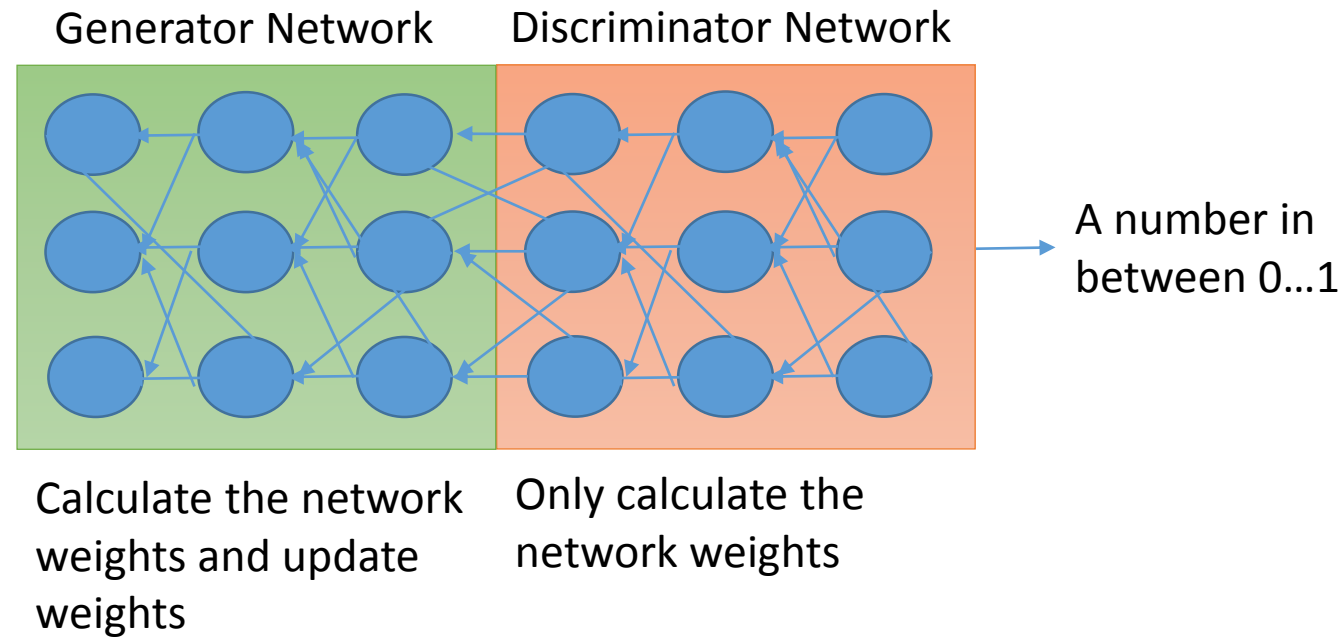
6. First train discriminator on real and fake generated images. Discriminator label images as fake or real based on their probabilities which varies in between 0 and 1. 0 means fake and 1 means real. The input of this network is either real or fake image in the form of vector and output is the number.



Working of GAN

7. Train until discriminator network become optimal.
8. Now stop training discriminator and backpropagate the output of the discriminator to update the parameters of generator to generate images which are near to real images.
 - Discriminator output $\rightarrow \text{dis}(x) \rightarrow x = \text{gen}(\text{random vector}) \rightarrow \text{Random vector}$
9. Lets assume both networks as a single network and backpropagate the discriminator output by only calculating not updating the weights of discriminator network and calculating and updating the weights of generator network.[\[12\]](#)

Working of GAN



10. For new parameters of generator generate new images.

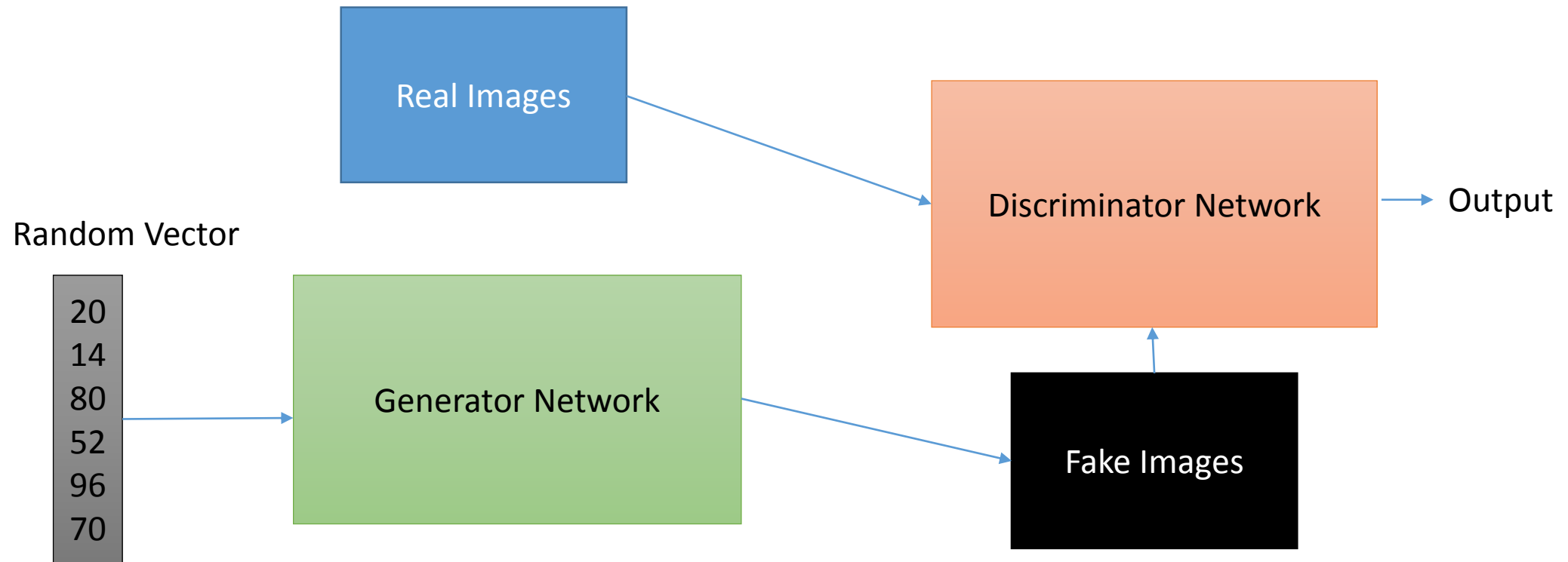
Working of GAN

11. Train generator and discriminator in this feed back loop for n epochs.

- i. Random vector $\rightarrow x = \text{gen}(\text{random vector}) \rightarrow \text{dis}(x) \rightarrow \text{Discriminator output}$
- ii. Discriminator output $\rightarrow \text{dis}(x) \rightarrow x = \text{gen}(\text{random vector}) \rightarrow \text{Random vector}$

12. Check fake data manually if it seems legit/real then stop training otherwise train both networks until they are optimal or reach the probability of 0.5

Working of GAN



GAN Functions^[12]

- Functions which are used in real paper for training of both networks are:
 - Train discriminator to maximize the probability of real images, when generator is fixed
 - $\max_D V(D, G^*) = E_{x \sim p_{\text{data}}}[\log D(x)] + E_{z \sim p_Z(z)}[\log(1 - D(G^*(z)))]$
 - Train generator to minimize the probability of real images, when discriminator is fixed
 - $\min_G V(D^*, G) = E_{x \sim p_{\text{data}}}[\log D^*(x)] + E_{z \sim p_Z(z)}[\log(1 - D^*(G(z)))]$

GAN Architectures^[13]

- There are many architectures of GAN:
 - Fully Connected GAN
 - Convolutional GAN
 - Laplacian pyramid of GAN
 - Deep convolutional GAN
 - Conditional GAN
 - Info GAN
 - Adversarial Autoencoders
 - GAN with Inference model
 - Bidirectional GAN

GAN Applications

- GAN's is a very vast field and has a lot of application to detect create and analyze things in different fields.
 - Image retrieval from historical archives
 - Text translation into images
 - Drug Discovery
 - Shadow detection
 - Draw Human Faces
 - Realistic Paintings
 - Designing

GAN Example

- Gaussian Data generation implementation using GAN's:
 - Discriminator input Gaussian data and generator generated data.
 - Generator input random data.
 - Find the optimal Weight values for both networks to generate Gaussian data
 - Sigmoid Activation function for discriminator.
 - Tanh activation function for generator.
 - Binary cross entropy function for loss.
 - Torch library for network implementation.

Python code:

```
train() # Run the train function to start the program
```

```
def train():
```

```
    # Model parameters
```

```
    g_input_size = 1    # Random noise dimension coming into generator, per output vector
```

```
    g_hidden_size = 5   # Hidden layers of generator
```

```
    g_output_size = 1   # Size of generated output vector
```

```
    d_input_size = 500  # Mini batch size
```

```
    d_hidden_size = 10  # Hidden layers of discriminator
```

```
    d_output_size = 1   # Single dimension for 'real' vs. 'fake' classification
```

```
    minibatch_size = d_input_size # Size of the mini batch
```

```
    d_learning_rate = 1e-3 # Discriminator learning rate
```

```
    g_learning_rate = 1e-3 # Generator learning rate
```

```
    sgd_momentum = 0.9 # Momentum
```

```
    num_epochs = 5000 # Number of epochs
```

```
    print_interval = 1000 # Print interval for output
```

```
    d_steps = 20 # Training of discriminator for one epoch
```

Cont.

```
dfe, dre, ge = 0, 0, 0 # Variables to store errors
```

```
d_real_data, d_fake_data, g_fake_data = None, None, None # Variables to store data
```

```
discriminator_activation_function = torch.sigmoid # Activation function for discriminator
```

```
generator_activation_function = torch.tanh # Activation function for generator
```

```
d_sampler = get_distribution_sampler(data_mean, data_stddev) # Generate data for discriminator
```

```
# parameters for Discriminator data
```

```
data_mean = 4
```

```
data_stddev = 1.25
```

```
# Data for discriminator
```

```
# Uncomment only one of these to define what data is actually sent to the Discriminator
```

```
(name, preprocess, d_input_func) = ("Only 4 moments", lambda data: get_moments(data), lambda x: 4)
```

```
##(name, preprocess, d_input_func) = ("Raw data", lambda data: data, lambda x: x)
```

```
##(name, preprocess, d_input_func) = ("Data and variances", lambda data: decorate_with_diffs(data, 2.0), lambda x: x * 2)
```

```
##(name, preprocess, d_input_func) = ("Data and diffs", lambda data: decorate_with_diffs(data, 1.0), lambda x: x * 2)
```

```
# Discriminator data generator for input
```

```
def get_distribution_sampler(mu, sigma):
```

```
    return lambda n: torch.Tensor(np.random.normal(mu, sigma, (1, n))) # Gaussian
```

```
def get_moments(d):
```

```
    # Return the first 4 moments of the data provided
```

```
    mean = torch.mean(d)
```

```
    diffs = d - mean
```

```
    var = torch.mean(torch.pow(diffs, 2.0))
```

```
    std = torch.pow(var, 0.5)
```

```
    zscores = diffs / std
```

```
    skews = torch.mean(torch.pow(zscores, 3.0))
```

```
    kurtoses = torch.mean(torch.pow(zscores, 4.0)) - 3.0 # excess kurtosis, should be 0 for Gaussian
```

```
    final = torch.cat((mean.reshape(1,), std.reshape(1,), skews.reshape(1,), kurtoses.reshape(1,)))
```

```
    return final
```

```
def decorate_with_diffs(data, exponent, remove_raw_data=False):
```

```
    mean = torch.mean(data.data, 1, keepdim=True)
```

```
    mean_broadcast = torch.mul(torch.ones(data.size()), mean.tolist()[0][0])
```

```
    diffs = torch.pow(data - Variable(mean_broadcast), exponent)
```

```
    if remove_raw_data:
```

```
        return torch.cat([diffs], 1)
```

```
    else:
```

```
        return torch.cat([data, diffs], 1)
```

Cont.

```
gi_sampler = get_generator_input_sampler() # Generate data for generator
```

```
def get_generator_input_sampler():  
    return lambda m, n: torch.rand(m, n) # Uniform-dist data into generator, _NOT_ Gaussian
```

Generator model define

```
G = Generator(input_size=g_input_size,  
             hidden_size=g_hidden_size,  
             output_size=g_output_size,  
             f=generator_activation_function)
```

Generator model

```
class Generator(nn.Module):  
    # Constructor of the generator class  
    def __init__(self, input_size, hidden_size, output_size, f):  
        super(Generator, self).__init__()  
        self.map1 = nn.Linear(input_size, hidden_size)  
        self.map2 = nn.Linear(hidden_size, hidden_size)  
        self.map3 = nn.Linear(hidden_size, output_size)  
        self.f = f  
  
    # define the sequence of the layers with activation functions  
    def forward(self, x):  
        x = self.map1(x)  
        x = self.f(x)  
        x = self.map2(x)  
        x = self.f(x)  
        x = self.map3(x)  
        return x
```

Cont.

Discriminator model define

```
D = Discriminator(input_size=d_input_func(d_input_size),  
                 hidden_size=d_hidden_size,  
                 output_size=d_output_size,  
                 f=discriminator_activation_function)
```

Discriminator model

class Discriminator(nn.Module):

Constructor of the discriminator class

```
def __init__(self, input_size, hidden_size, output_size, f):  
    super(Discriminator, self).__init__()  
    self.map1 = nn.Linear(input_size, hidden_size)  
    self.map2 = nn.Linear(hidden_size, hidden_size)  
    self.map3 = nn.Linear(hidden_size, output_size)  
    self.f = f
```

define the sequence of the layers with activation functions

```
def forward(self, x):  
    x = self.f(self.map1(x))  
    x = self.f(self.map2(x))  
    return self.f(self.map3(x))
```

criterion = nn.BCELoss() # Loss function to calculate loss

d_optimizer = optim.SGD(D.parameters(), lr=d_learning_rate, momentum=sgd_momentum) # Discriminator parameters optimization

g_optimizer = optim.SGD(G.parameters(), lr=g_learning_rate, momentum=sgd_momentum) # Generator parameters optimization

Cont.

Start of training

```
for epoch in range(num_epochs):
```

```
    for d_index in range(d_steps):
```

```
        # 1. Train D on real + fake
```

```
        D.zero_grad() # Define weights of discriminator network with zero
```

```
        # 1A: Train D on real
```

```
        d_real_data = Variable(d_sampler(d_input_size)) # Put real data into discriminator
```

```
        d_real_decision = D(preprocess(d_real_data)) # Put real data decisions one = true and zero = false
```

```
        d_real_error = criterion(d_real_decision, Variable(torch.ones([1,1]))) # Run the network and calculate the error
```

```
        d_real_error.backward() # Compute and store gradients, but don't change params why?????
```

```
        # 1B: Train D on fake
```

```
        d_gen_input = Variable(gi_sampler(minibatch_size, g_input_size)) # Divide data into mini batch size
```

```
        d_fake_data = G(d_gen_input).detach() # Put fake data into generator. Detach to avoid training G on these labels
```

```
        d_fake_decision = D(preprocess(d_fake_data.t())) # Put fake data decisions one = true and zero = false
```

```
        d_fake_error = criterion(d_fake_decision, Variable(torch.zeros([1,1]))) # Run the network and calculate the error
```

```
        d_fake_error.backward() # Compute and store gradients
```

```
        d_optimizer.step() # Only optimizes D's parameters; changes based on stored gradients from backward()
```

Cont.

```
for g_index in range(g_steps):  
    # 2. Train G on D's response (but DO NOT train D on these labels)  
    G.zero_grad() # Define weights of generator network with zero  
  
    gen_input = Variable(gi_sampler(minibatch_size, g_input_size)) # Divide data into mini batch size  
    g_fake_data = G(gen_input) # Put fake data into generator.  
    dg_fake_decision = D(preprocess(g_fake_data.t())) # Put fake data decisions from discriminator  
    g_error = criterion(dg_fake_decision, Variable(torch.ones([1,1]))) # Run the network and calculate the error  
  
    #Train G to pretend it's genuine  
    g_error.backward() # Compute and store gradients  
    g_optimizer.step() # Only optimizes G's parameters  
    ge = extract(g_error)[0] # Store generator data errors.
```

Cont.

```
# Print the output after specified interval
```

```
if epoch % print_interval == 0:
```

```
    # Print epoch, all errors, and data
```

```
    print("Epoch %s: D (%s real_err, %s fake_err) G (%s err); Real Dist (%s), Fake Dist (%s) " %  
          (epoch, dre, dfe, ge, stats(extract(d_real_data)), stats(extract(d_fake_data))))
```

```
# Print data on graph
```

```
print("Plotting the generated distribution...")
```

```
values = extract(g_fake_data)
```

```
print(" Values: %s" % (str(values)))
```

```
plt.hist(values, bins=50)
```

```
plt.xlabel('Value')
```

```
plt.ylabel('Count')
```

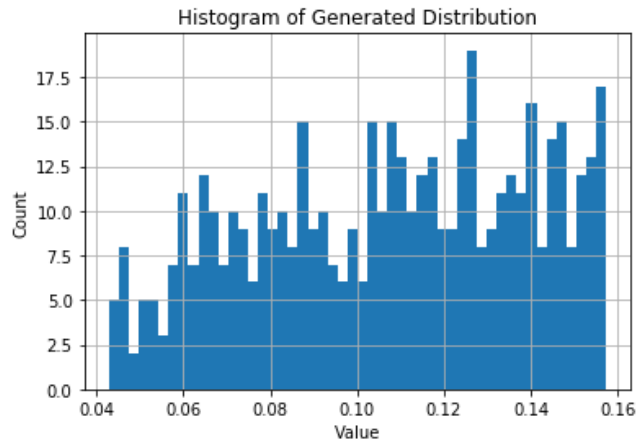
```
plt.title('Histogram of Generated Distribution')
```

```
plt.grid(True)
```

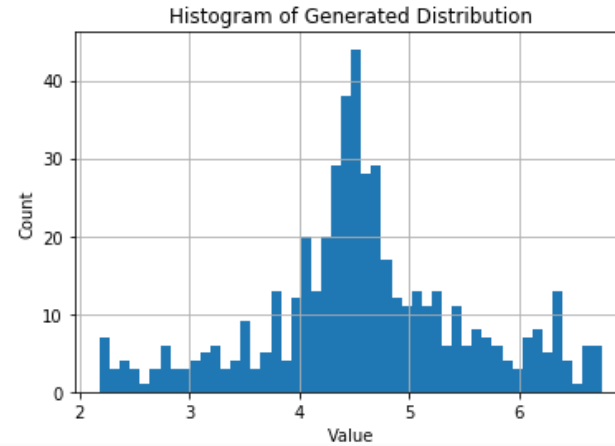
```
plt.show()
```

Results:

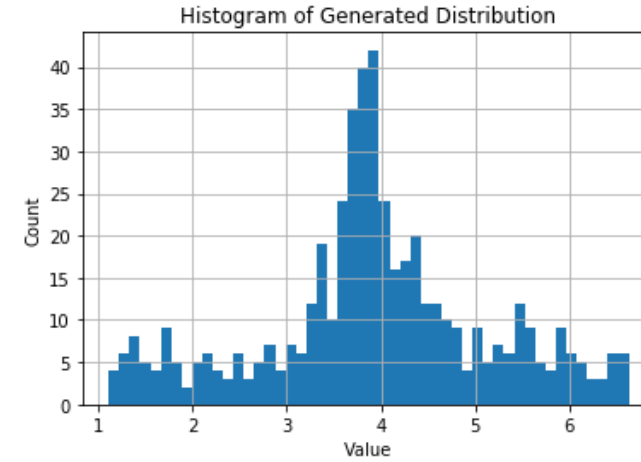
1000 Iterations:



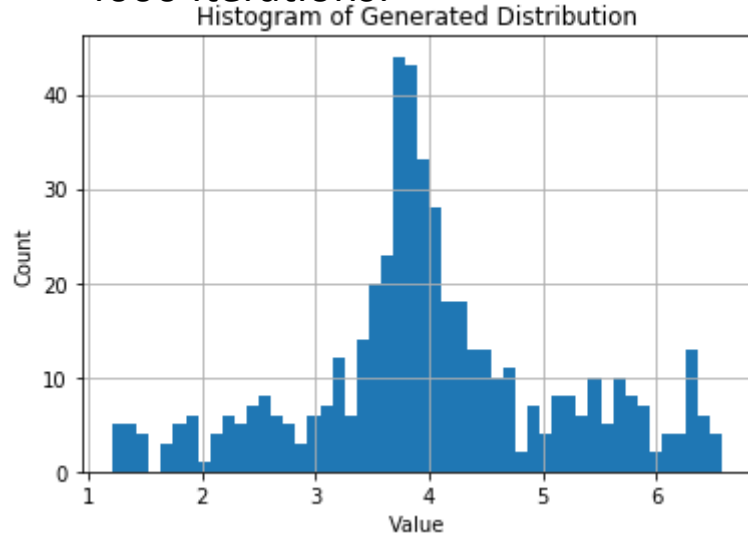
2000 Iterations:



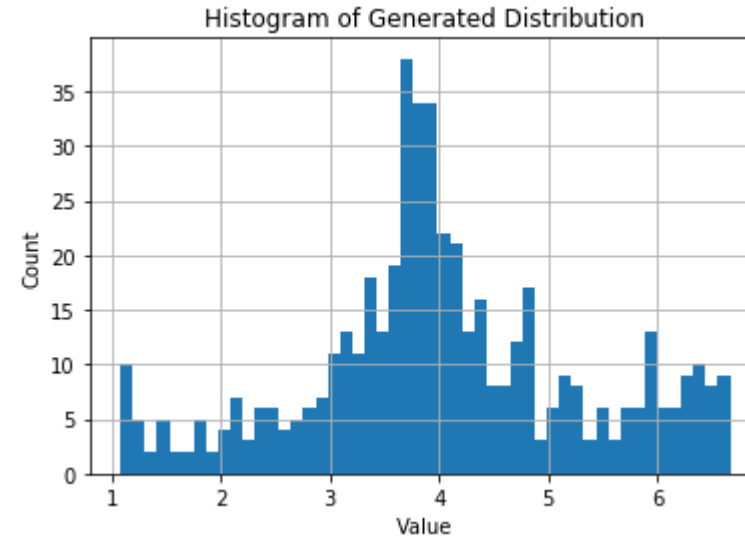
3000 Iterations:



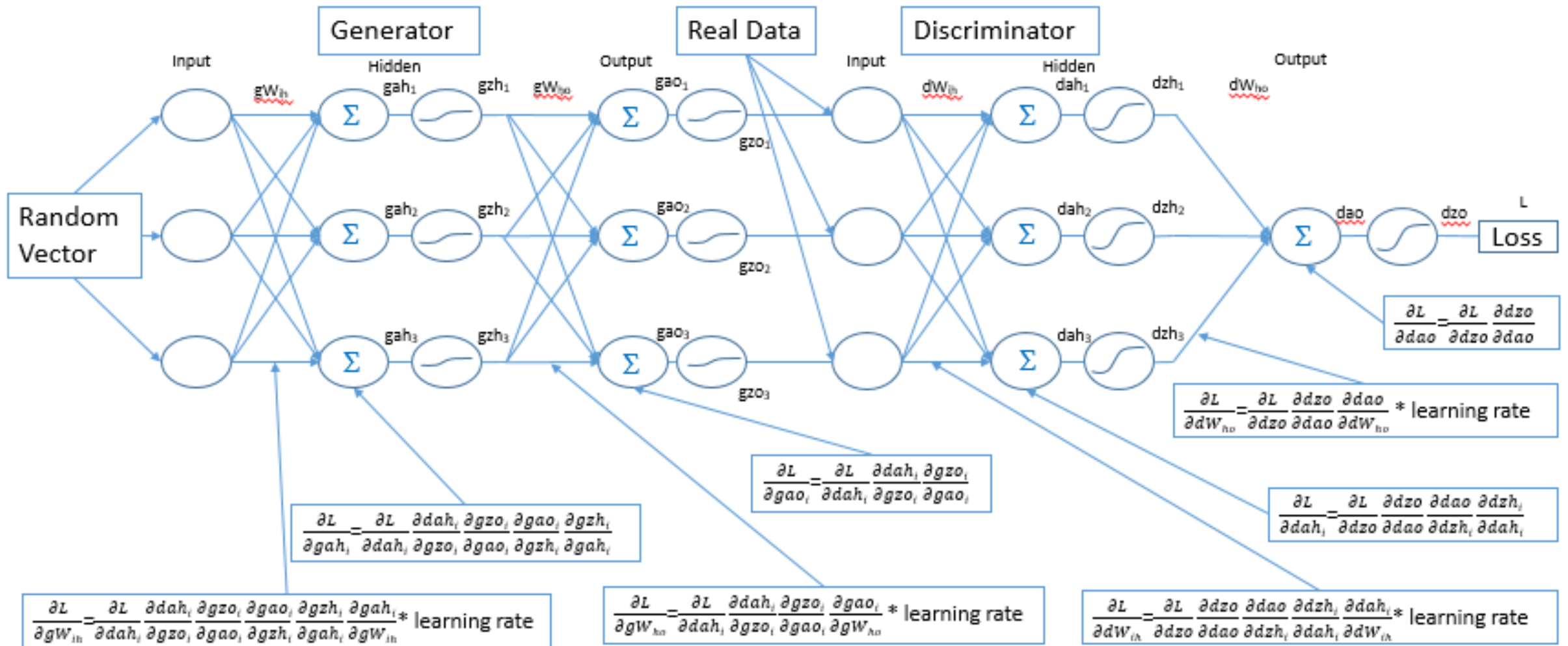
4000 Iterations:



5000 Iterations:



Structure of GAN:



Project Plan

- Understand generative and discriminative models
- Train simple network on some dummy data
- Learn backpropagation
- Learn GAN architectures and algorithm
- Choose GAN architecture and problem
- Gather real data specific to problem domain
- Select parameters for both networks
- Select loss/objective function for both networks
- Implement discriminative network
- Implement generative network
- Train both networks by running GAN
- Check results

GAN example for Name

- Generate human names using GAN's.
- Not possible using simple GAN architecture.
- Hard to learn the relation between the characters of names.
- Use codes for the letters.
- Learn the relation between these letters.
- Apply CNN filters to learn the relation between the characters.

References

1. Flach, P., 2012. Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press.
2. A Basic Introduction To Neural Networks. 2018. *A Basic Introduction To Neural Networks*. [ONLINE] Available at: <http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>. [Accessed 09 April 2018].
3. What is Machine Learning? A definition - Expert System. 2018. *What is Machine Learning? A definition - Expert System*. [ONLINE] Available at: <http://www.expertsystem.com/machine-learning-definition/>. [Accessed 09 April 2018].
4. DataFlair. 2018. *Introduction to Learning Rules in Neural Network - DataFlair*. [ONLINE] Available at: <https://data-flair.training/blogs/learning-rules-in-neural-network/>. [Accessed 09 April 2018].
5. AYLIEN. 2018. *An introduction to Generative Adversarial Networks (with code in TensorFlow) - AYLIEN*. [ONLINE] Available at: <http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/>. [Accessed 09 April 2018].
6. Chris V. Nicholson, Adam Gibson, Skymind team. 2018. *GAN: A Beginner's Guide to Generative Adversarial Networks - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM*. [ONLINE] Available at: <https://deeplearning4j.org/generative-adversarial-network>. [Accessed 09 April 2018].
7. OpenAI Blog. 2018. *Generative Models*. [ONLINE] Available at: <https://blog.openai.com/generative-models/>. [Accessed 09 April 2018].
8. Goodfellow, I., 2016. Deep Learning (Adaptive Computation and Machine Learning series). The MIT Press.
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative Adversarial Nets.
10. Goodfellow, I. (2017). NIPS 2016 Tutorial: Generative Adversarial Networks.
11. Analytics Vidhya. 2018. *Introductory guide to Generative Adversarial Networks (GANs)*. [ONLINE] Available at: <https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/>. [Accessed 09 April 2018].
12. machine learning - Generative Adversarial Networks: how the generator is trained with the output of discriminator - Cross Validated. 2018. *machine learning - Generative Adversarial Networks: how the generator is trained with the output of discriminator - Cross Validated*. [ONLINE] Available at: <https://stats.stackexchange.com/questions/287242/generative-adversarial-networks-how-the-generator-is-trained-with-the-output-of-d>. [Accessed 09 April 2018].
13. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B. and Bharath, A. (2018). Generative Adversarial Networks. *IEEE Signal Processing Magazine*.
14. Towards Data Science. 2018. *Implementing a Generative Adversarial Network (GAN/DCGAN) to Draw Human Faces*. [ONLINE] Available at: <https://towardsdatascience.com/implementing-a-generative-adversarial-network-gan-dcgan-to-draw-human-faces-8291616904a>. [Accessed 09 April 2018].
15. SYNTHETIC APERTURE RADAR SHIP DISCRIMINATION, GENERATION AND LATENT VARIABLE EXTRACTION USING INFORMATION MAXIMIZING GENERATIVE ADVERSARIAL NETWORKS
16. Image Processing - Downsampling | GIASSA.NET. 2018. *Image Processing - Downsampling | GIASSA.NET*. [ONLINE] Available at: https://www.giassa.net/?page_id=174. [Accessed 09 April 2018].
17. Image Processing - Upsampling & Interpolation | GIASSA.NET. 2018. *Image Processing - Upsampling & Interpolation | GIASSA.NET*. [ONLINE] Available at: https://www.giassa.net/?page_id=200. [Accessed 09 April 2018].
18. https://github.com/devnag/pytorch-generative-adversarial-networks/blob/master/gan_pytorch.py