GAN based Random Number Generator on Embedded system

https://youtu.be/yRFzHHRzBqM

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

Random Number Generator

random number

- 무작위로 선택된 수로 실제로 생성되기 전까지는 **예측할 수 없는 수**
- 키생성, nonce, salt 등에 사용

  → **예측가능한 난수가 생성되면 암호 시스템에 치명적**

- 조건

  1. **무작위성**

     특정 값에 편향되지 않고 균등하게 분포

  2. **예측불가능성**

     과거의 난수열로부터 다음 수 예측 불가

  3. **재현불가능성**

     같은 수열 재현 불가

entropy

- 데이터의 정보량을 측정한 값, 무작위성을 측정한 값
- random variable의 확률분포가 entropy 결정

  확률분포가 균등할 때 entropy 최대이며 불확실성이 높아짐

  entropy가 높다 → 무작위성이 높다 → 난수성이 높다 → 예측이 어렵다 → 안전

- DRBG의 안전성에 영향을 미침

  따라서 비밀로 지켜져야 하는 값

***결정론적 알고리즘**

: 동일한 입력에 대해 항상 동일한 출력 생성

***결정론적 난수발생기 (Deterministic Random Bit Generator, DRBG)**

: 비밀정보인 초기값 seed와 다른 입력들을 결정론적 알고리즘에 적용하여 의사난수 생성

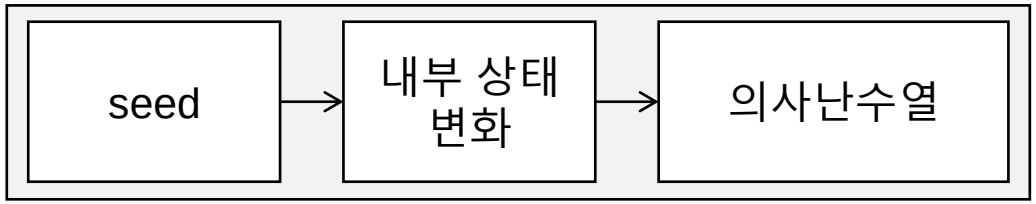
→ 특정 입력이나 조건에 따라 무작위로 선택된 것처럼 보이는 난수 생성

entropy source

- 물리 현상으로부터 정보를 수집
    1. cpu 온도
    2. 시간
    3. 마우스 위치 정보
    4. 키보드 입력 주기
    5. 네트워크 인터럽트
    6. 이미지나 오디오 등

TRNG vs PRNG

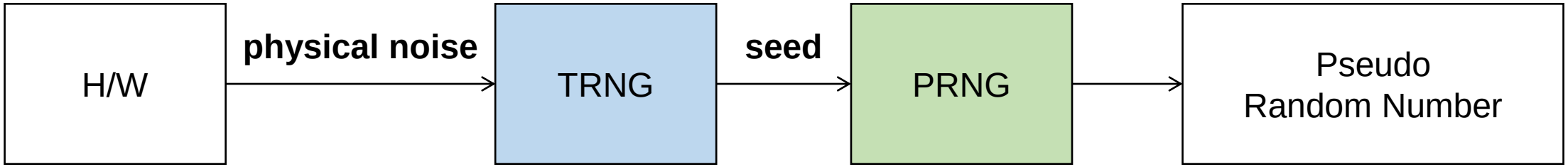
True Random Number	Pseudo Random Number
무규칙성	규칙성
비결정적	결정적
엔트로피 높음	엔트로피 낮음
느림	빠름
물리적 잡음 기반 난수 생성	알고리즘과 초기값(seed)기반 의사난수 생성
생성된 난수값이 편향될 가능성 존재 (entropy저하) → 편향된 성질을 보정하여 높은 엔트로피를 갖도록 수정 필요	결정론적 알고리즘만으로는 좋은 난수를 얻을 수 없음 → DBRG에 적은 entropy의 잡음을 입력하여 의사난수 생성



Cryptographically Secure RNG

- 암호학적 난수발생기 (CSRNG)

  TRNG의 출력을 PRNG의 seed로 사용하여 난수 생성

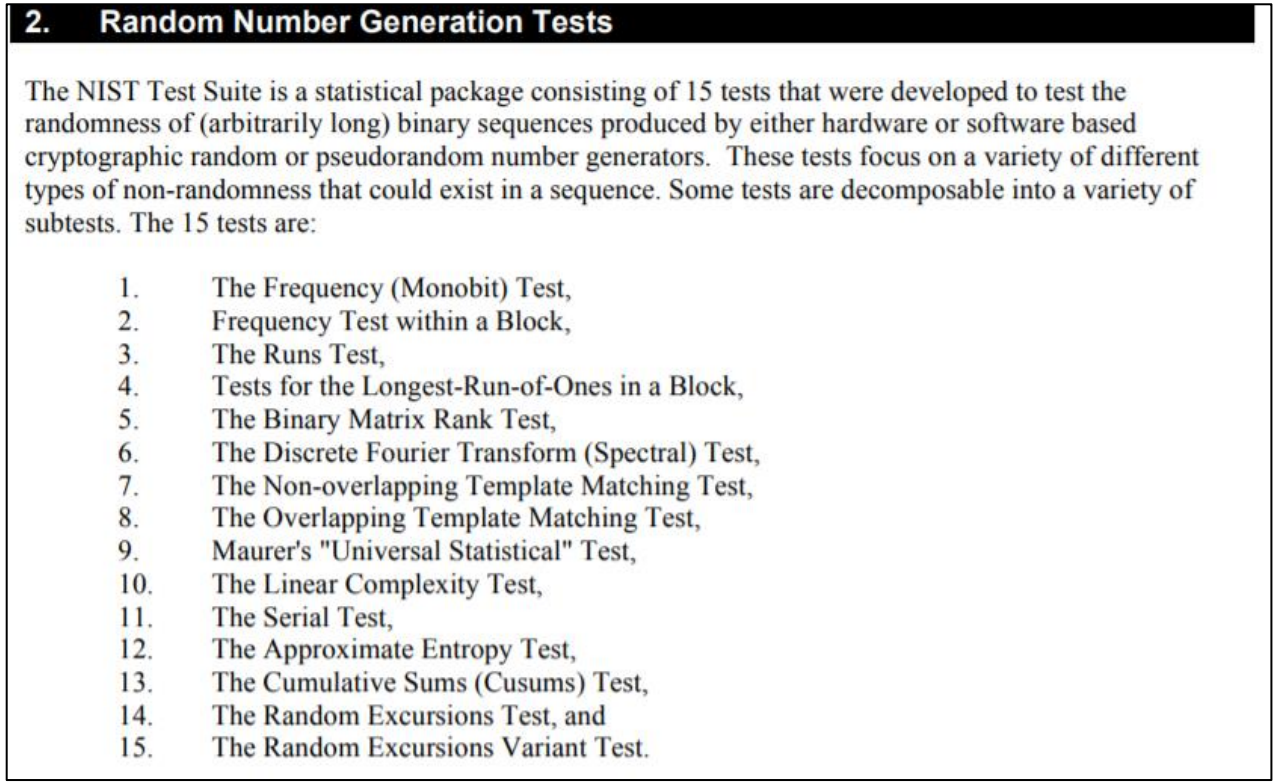


RNG TEST

- 난수발생기의 안전성은 출력생성에 사용되는 내부상태나 entropy, seed의 비밀성에 의존
- 따라서
  1. 수집되는 **엔트로피의 건전성 평가**
  2. 출력난수의 **통계적 랜덤성 평가**
- **KCMVP의 임의의 입력값에 대한 검사(KAT)**

  임의의 입력 정보(엔트로피, 논스, 추가 입력 등)에 대해 난수값을 올바르게 생성하는지에 대한 검사
- **NIST test suite 사용한 통계적 랜덤성 검사**

  오른쪽 사진과 같이 15개의 다양한 항목 존재



2. Random Number Generation Tests

The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The 15 tests are:

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test.

NIST test suite

공격

- 초기 내부 상태 및 향후 변화하는 내부 상태들은 난수발생기의 입력에 대해 결정적인 값

  1. **높은 entropy를 갖는 seed 필요** (결정적 알고리즘에서도 예측이 어렵도록)
  2. 내부 상태를 구성하는 변수들은 짧으면 안됨 그에 맞는 길이 필요

  ex) 128비트 길이의 키를 생성 시, 내부 상태 변수들도 128비트 이상으로 구성

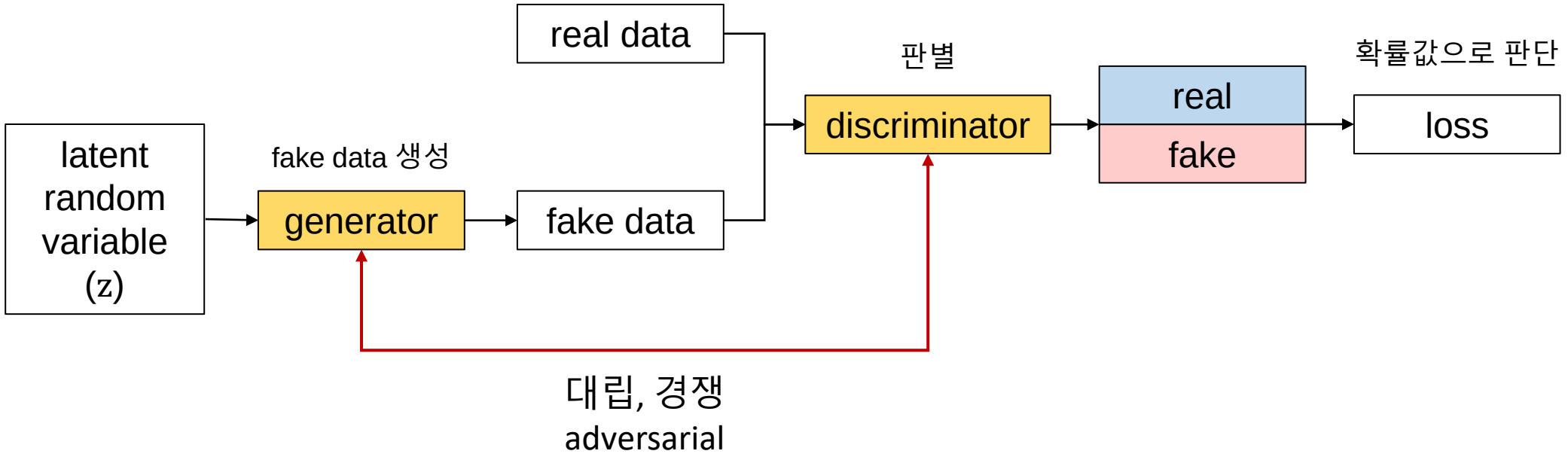
- if 잡음원 예측 및 부분적 제어 가능 → seed & 내부 상태 파악이 쉬워짐

  다양하고 예측하기 어려운 잡음원 사용 필요성 존재

- 내부 상태가 알려질 경우 예측/역예측 공격 가능
  - 새로운 난수 생성 위해 저장된 내부 상태, 변수가 알려지면 이전 또는 이후의 출력을 알아낼 수 있음
  - 이를 막기 위해 내부 상태에 **새로운 entropy 추가** (추가입력 통해 상태 갱신)

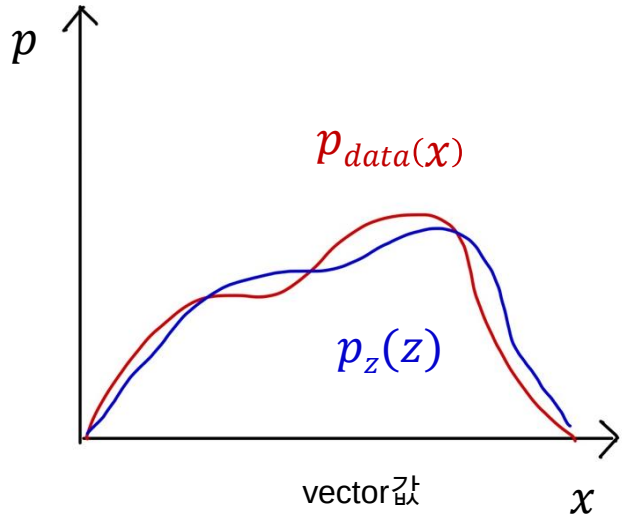
Generative Adversarial Network

Generative Adversarial Network



1. generator는 discriminator를 속이기 위해 진짜같은 가짜를 생성
2. discriminator는 generator의 가짜 출력을 판별하기 위해 학습

Generative Adversarial Network



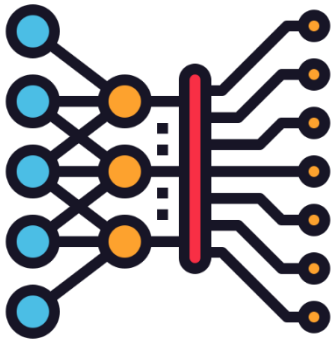
- data의 특징을 나타내는 vector값의 분포 (p)
- real data의 분포($p_{data}(x)$)와 fake data의 분포($p_z(z)$)를 학습을 통해 비슷하게 만드는 것이 목적
  → label을 통한 분류가 아닌 training data의 분포를 학습

➢ 확률 분포가 정확히 일치하면 real data와 fake data를 구분할 수 없음

GAN based RNG
on Embedded system

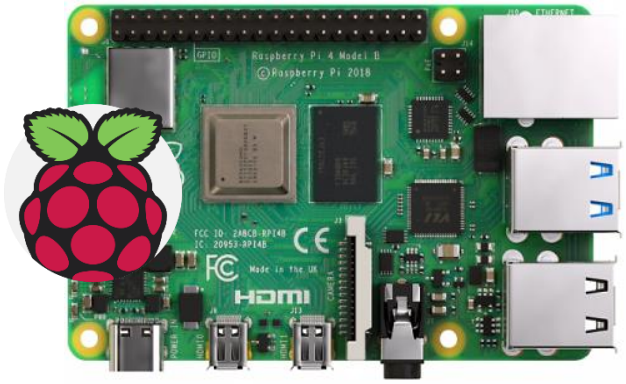
GAN based RNG on Embedded system

- raspberry pi의 물리적 잡음원으로부터 entropy 수집하여 학습데이터 생성
- 해당 학습데이터의 확률분포를 학습하여 실제 난수와 비슷한 수준의 난수성을 갖는 의사난수 생성
- 임베디드 상에서의 난수 생성기



Generative Adversarial Network

TensorFlow Lite



Random Number Generator on Embedded system

embedded system & deep learning

TensorFlow Lite

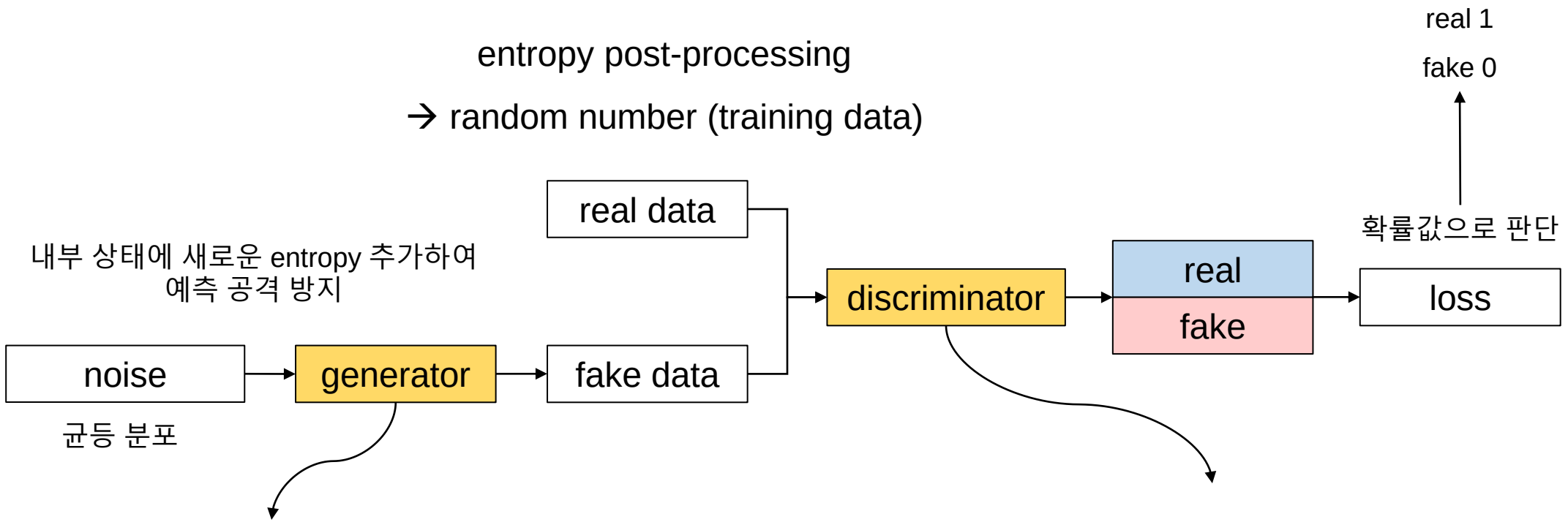
TensorFlow Lite

- mobile & IoT device에 딥러닝 모델 배포
- android, raspberrypi 등 지원

Edge TPU

- Edge TPU compiler 제공 → .tflite compile 지원
- raspberrypi 지원 → 임베디드 상에서 딥러닝 가능

system configuration



*여기서는 매 학습마다 D의 결과에 따라 G가 생성되는 게 바뀌는데
training data말고 해당 noise의 엔트로피가 중요한 부분인지... 잘모르겠습니다..

model (generator)

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 128)               384
_________________________________________________________________
leaky_re_lu_45 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
leaky_re_lu_46 (LeakyReLU)   (None, 64)                0
_________________________________________________________________
dense_37 (Dense)             (None, 32)                2080
_________________________________________________________________
leaky_re_lu_47 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_38 (Dense)             (None, 32)                1056
_________________________________________________________________
leaky_re_lu_48 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_39 (Dense)             (None, 8)                 264
=================================================================
Total params: 12,040
Trainable params: 12,040
Non-trainable params: 0
```

generator

```python
def make_generator_model():
  model = tf.keras.Sequential()
  #1
  model.add(layers.Dense(128, input_dim=2))
  model.add(layers.LeakyReLU())
  #2
  model.add(layers.Dense(64))
  model.add(layers.LeakyReLU())
  #3
  model.add(layers.Dense(32))
  model.add(layers.LeakyReLU())
  #4
  model.add(layers.Dense(32))
  model.add(layers.LeakyReLU())
  #5
  model.add(layers.Dense(8, activation=custom_activation))

  return model
```

```python
def custom_activation(x):
    return tf.keras.backend.round(abs(x*(2**16-1)))%65536
```

- 0~65535 사이의 수를 생성하기 위한 activation
- LeakyReLu activation은 음의 값을 가지기 때문에
  출력 벡터에 절댓값 씌워 반올림하여 가까운 정수로 보냄

model (discriminator)

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_20 (Conv1D)           (None, 7, 4)              12
_________________________________________________________________
leaky_re_lu_49 (LeakyReLU)   (None, 7, 4)              0
_________________________________________________________________
conv1d_21 (Conv1D)           (None, 6, 4)              36
_________________________________________________________________
leaky_re_lu_50 (LeakyReLU)   (None, 6, 4)              0
_________________________________________________________________
conv1d_22 (Conv1D)           (None, 5, 4)              36
_________________________________________________________________
leaky_re_lu_51 (LeakyReLU)   (None, 5, 4)              0
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 4, 4)              36
_________________________________________________________________
leaky_re_lu_52 (LeakyReLU)   (None, 4, 4)              0
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 3, 4)              0
_________________________________________________________________
flatten_5 (Flatten)          (None, 12)                0
_________________________________________________________________
dense_40 (Dense)             (None, 4)                 52
_________________________________________________________________
leaky_re_lu_53 (LeakyReLU)   (None, 4)                 0
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 5
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
```

discriminator

```python
def discriminator_model():
    model = tf.keras.Sequential()
    #1
    model.add(layers.Conv1D(4,2, input_shape=(batch_size,1)))
    model.add(layers.LeakyReLU())
    #2
    model.add(layers.Conv1D(4,2))
    model.add(layers.LeakyReLU())
    #3
    model.add(layers.Conv1D(4,2))
    model.add(layers.LeakyReLU())
    #4
    model.add(layers.Conv1D(4,2))
    model.add(layers.LeakyReLU())

    #MaxPool
    model.add(layers.MaxPooling1D(pool_size=2, strides=1))
    model.add(layers.Flatten())
    #1
    model.add(layers.Dense(4))
    model.add(layers.LeakyReLU())

    #2
    model.add(layers.Dense(1, activation="sigmoid"))
    return model
```

- 0~65535 (16bits) 사이의 값

  128/16 = 8 (N=128)
- 한번에 128 bit씩(8개씩) 생성

0	1	2	3	4	5	6	7
8	9	10	…				
…						…	

- 확률값(0~1)으로 나타내기 위해 sigmoid 함수 사용
- output의 형태는 fake = 0, real = 1

model (GAN)

- generator의 출력을 discriminator의 입력으로 사용

  → 두 model을 이어서 사용하여 번갈아 가며 학습

* g = generator, d = discriminator

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_19 (Sequential)   (None, 8)                 12040
_________________________________________________________________
reshape_9 (Reshape)          (None, 8, 1)              0
_________________________________________________________________
sequential_20 (Sequential)   (None, 1)                 177
=================================================================
Total params: 12,217
Trainable params: 12,217
Non-trainable params: 0
```

GAN (generator + discriminator)

```
def build_model(g, d):
    model = tf.keras.Sequential()
    model.add(g)
    model.add(layers.Reshape((batch_size,1)))
    d.trainable = False
    model.add(d)
    return model
```

generator의 출력이 2차원인데 discriminator의 입력이 3차원이라 reshape

discriminator가 훈련되지 않도록 가중치 고정

compile

- gan 모델 빌드 후 compile

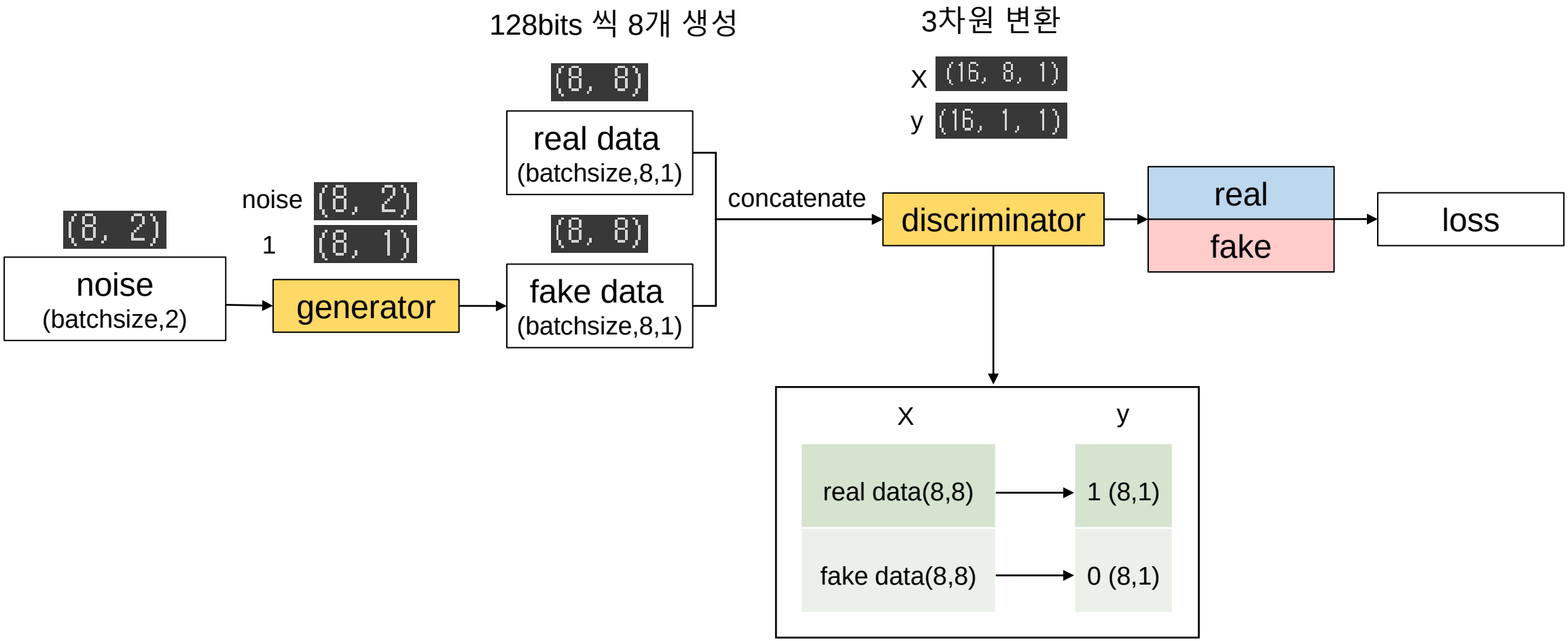
```
gan = build_model(generator,discriminator)

generator.compile(loss='binary_crossentropy', optimizer= "Adam")
gan.compile(loss='binary_crossentropy', optimizer="Adam")
discriminator.trainable = True
discriminator.compile(loss='binary_crossentropy', optimizer="Adam")
```

loss		optimizer	
binary_crossentropy	0,1로 나눌 때 (fake,real)	Adam	현재 가장 많이 사용되는 최적화함수 (성능 굿)

진짜로 판단되면 1, 가짜로 판단되면 0

진짜인지 가짜인지 헷갈리면 0.5에 가까운 값

training


128bits 씩 8개 생성
3차원 변환
(8, 8)
real data
(batchsize,8,1)
X (16, 8, 1)
y (16, 1, 1)
noise (8, 2)
1 (8, 1)
(8, 2)
noise
(batchsize,2)
generator
(8, 8)
fake data
(batchsize,8,1)
concatenate
discriminator
real
fake
loss
X
y
real data(8,8)
1 (8,1)
fake data(8,8)
0 (8,1)

training

- EPOCHS, batch_size 등 설정 후 학습

train()

```
EPOCHS = 100000
M = 32
batch_size = 8
train_per_epoch = M // batch_size
```

EPOCHS = 전체 학습 횟수

M = 총 생성할 개수

batch_size = 난수열 하나에 들어갈 개수

train_per_epoch = 1 epoch당 학습 횟수

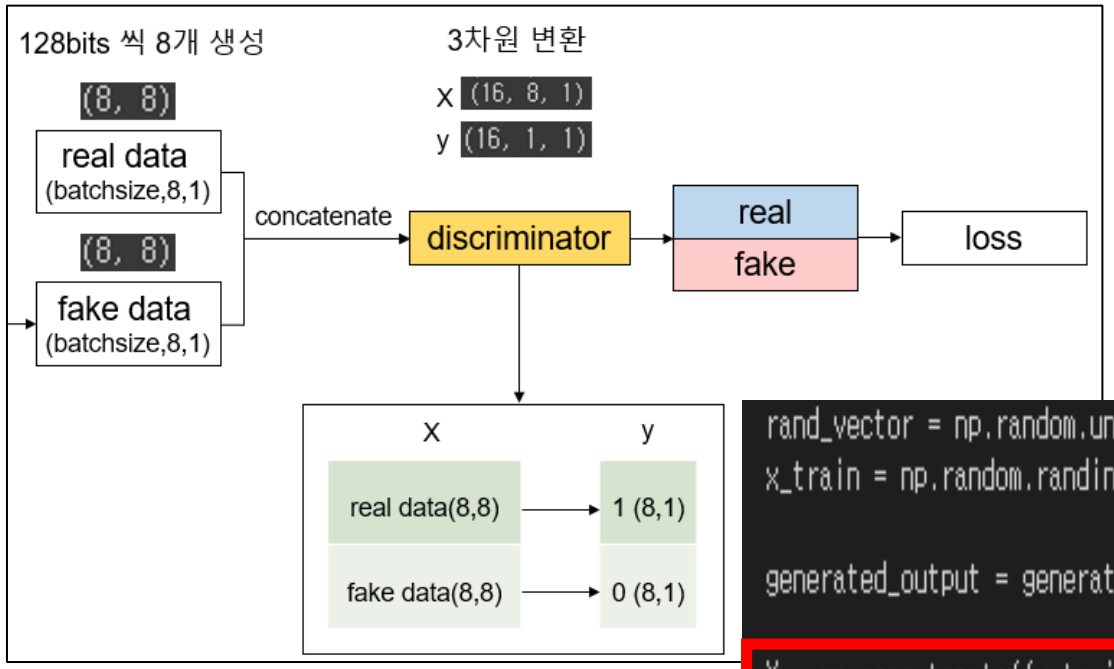
- batch_size로 나누어 학습

→ GAN모델은 D가 학습할 때마다

G의 출력이 변해서 batch 단위로 학습

```
def train():
    for epoch in range(EPOCHS):
        print("Epoch {}".format(epoch+1))

        # D
        rand_vector = np.random.uniform(0, 1, (batch_size, 2))
        x_train = np.random.randint(0, 65536, (batch_size, batch_size))

        generated_output = generator.predict(rand_vector, verbose=2)

        X = np.concatenate((x_train, generated_output))
        y = np.ones([2*batch_size, 1])
        y[batch_size:, :] = 0

        X = tf.reshape(X,(2*batch_size,batch_size,1))
        y = tf.reshape(y,(2*batch_size, 1, 1))


        d_loss = discriminator.train_on_batch(X, y)

        # G
        rand_vector = np.random.uniform(0, 1, (batch_size, 2))

        discriminator.trainable = False

        l = np.ones((batch_size, 1))
        g_loss = gan.train_on_batch(rand_vector,l)

        discriminator.trainable = True
```

training (discriminator)




```
rand_vector = np.random.uniform(0, 1, (batch_size, 2))
x_train = np.random.randint(0, 65536, (batch_size, batch_size))

generated_output = generator.predict(rand_vector, verbose=2)

X = np.concatenate((x_train, generated_output))
y = np.ones([2*batch_size, 1])
y[batch_size:, :] = 0

X = tf.reshape(X,(2*batch_size,batch_size,1))
y = tf.reshape(y,(2*batch_size, 1, 1))


d_loss = discriminator.train_on_batch(X, y)
```

진짜 가짜를 판별하는 능력을 훈련

1. 각 epoch마다 data를 batch size로 나누어 학습
2. G의 출력과 real data를 입력으로 사용

균등하게 분포한 noise 추출(G의 입력)

물리적 잡음으로부터 추출할 계획..!

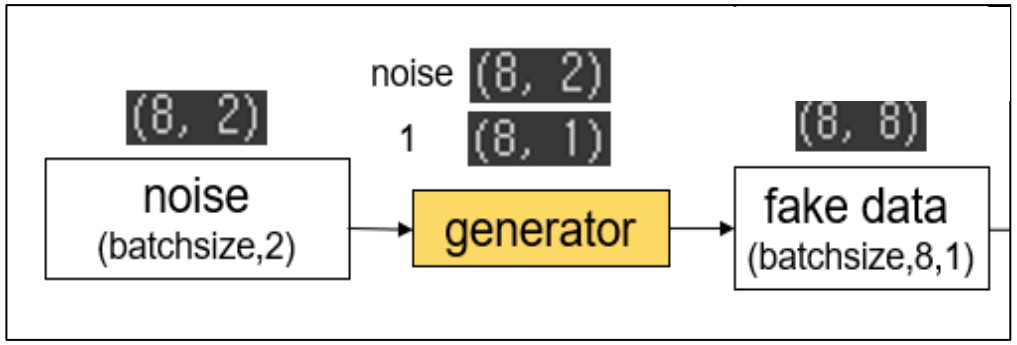
D학습을 위해 noise로부터 fake data 생성

두 데이터 연접
→ shape 동일

D의 입력에 맞게 reshape

training data를 batch_size로 나누어 학습

training (generator)



진짜 난수같은 수열을 생성

1. generator와 discriminator를 결합한 gan모델을 통해 학습
2. gan 모델은 이미 real or fake의 답을 알고 있기 때문에 D의 답(틀린 것, 맞은 것)을 보고 학습해나감

```
rand_vector = np.random.uniform(0, 1, (batch_size, 2))

discriminator.trainable = False

l = np.ones((batch_size, 1))
g_loss = gan.train_on_batch(rand_vector,l)

discriminator.trainable = True
```

균등하게 분포한 noise 추출

discriminator는 학습되면 안되기 때문에 False로 설정

batch_size로 나누어 학습

discriminator 학습 가능하도록 변경

generated random number example

```
[Generated Random Number]
[[ 2602.  3446.    67.  2795. 11873.  4994.  1871.  6708.]
 [ 1672.  2236.   163.  1884.  8402.  3607.  1171.  4883.]
 [ 1448.  1807.   300.   618.  6668.  2565.   548.  3982.]
 [  650.   845.   178.    29.  2945.  1008.   114.  1762.]
 [ 2901.  3955.   877.  2368.  9710.  3392.  2053.  4570.]
 [ 2857.  6850.  2041.  6884. 14886.  5332.  4881.  6356.]
 [  685.   909.    30.   729.  3079.  1289.   496.  1730.]
 [ 1603.  2018.   364.   536.  7371.  2762.   529.  4399.]]
```

```
[Generated Random Number]
[[7760.  486.  821. 4174. 3060. 3438. 1600. 3494.]
 [8920. 1976.  376. 1325.  833. 1994.  907. 5132.]
 [9341. 2122. 1294. 3461. 3424. 3875. 1811. 4622.]
 [9256.  586. 1087. 4763. 3828. 4191. 2410. 4043.]
 [8120.  491.  863. 4382. 3124. 3627. 1667. 3638.]
 [7441. 1780. 1014. 2590. 2535. 2954. 1259. 3789.]
 [8569.  539.  906. 4607. 3387. 3793. 1767. 3860.]
 [6836. 1279.  591.  279.  245. 1293. 1897. 3522.]]
```

생성되는 값들이 학습을 반복하며 달라짐

*근데 뭔가 헷갈려서 계속 수정하다보니 진짜 난수인지는 아직 확인하지 못했습니다..ㅠㅠ

향후 계획

- **해당 모델을 통해 실제 난수가 생성되는지 난수성 검사 필요**

  → NIST TEST SUITE 적용

- **seed값 추출 방안 연구**

  → cpu 온도, 이미지 등 앞에서 살펴본 엔트로피 소스로부터 추출

- **GAN 모델 자체의 성능 향상**

  → activation, loss, layer 수 등의 hyperparameter 조정

- **GAN에 관련된 최신 연구 참고**

  → 현재 DCGAN 사용

  → 모델 종류도 다양하고, 응용분야가 많은 것 같아 조사 필요

Q & A