가치창출대학
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

PIAI
인공지능연구원
POSTECH Institute of Artificial Intelligence

Keras Additional Functions

GyeongYeong Kim

ML/DL Lab 4rd Day

PIAI Research Department

가치창출대학
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Keras Additional Functions
  - LSTM
  - GAN
- Pytorch Additional Functions
  - GAN

LSTM

PIAI Research Department


```
def lstm(n_in, n_out):
    # Coding Time
    model = Sequential()
    model.add(LSTM(30, input_shape=(28,28)))
    model.add(Dense(n_out, activation='softmax'))
    return model

model = lstm(n_in, n_out)
```

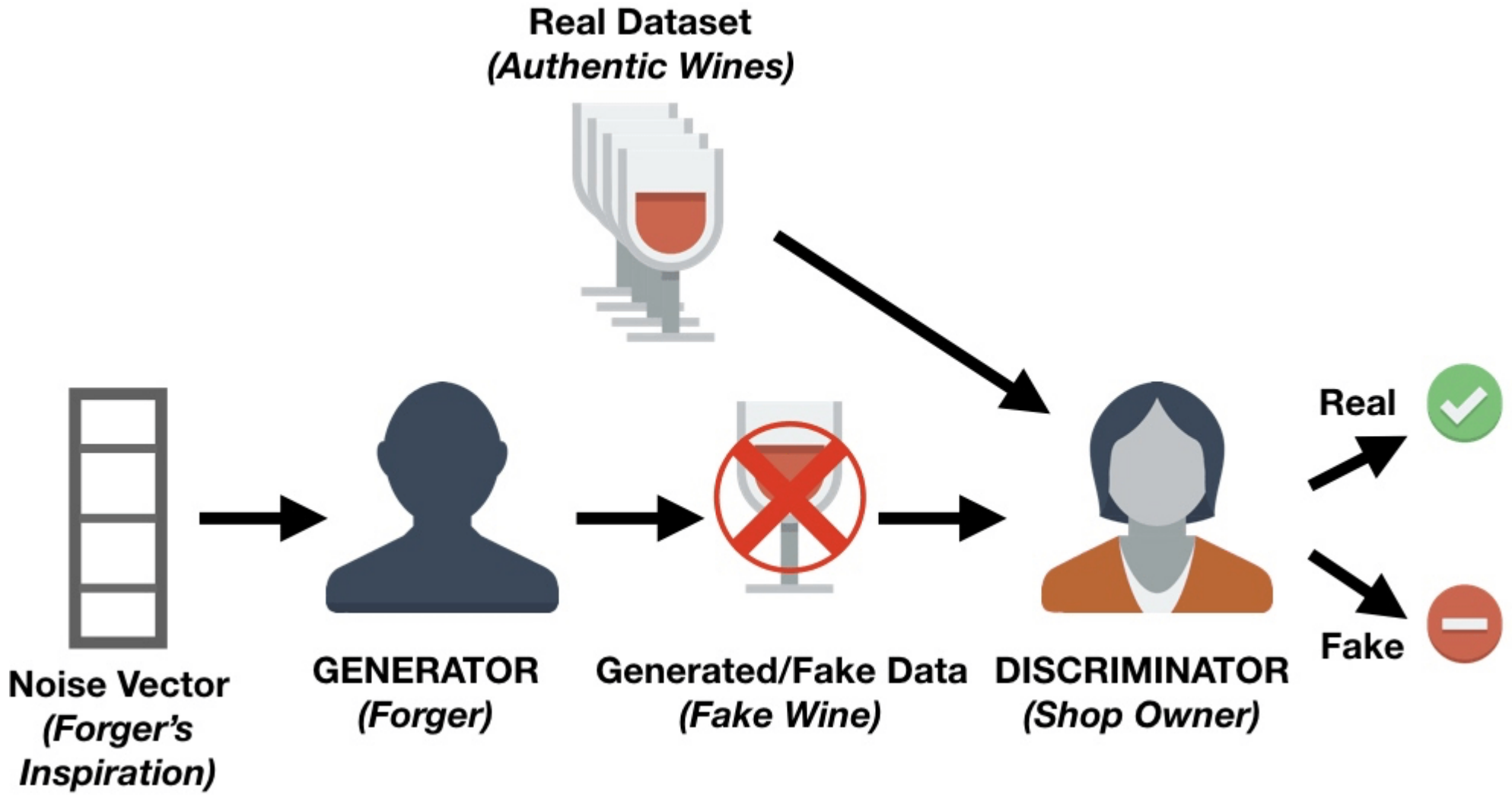
종류

- **RNN (MinimalRNNCell)**
- **SimpleRNN (SimpleRNNCell)**
- **GRU (GRUCell) (CuDNNGRU)**
- **LSTM (LSTMCell) (CuDNNLSTM)**
- **ConvLSTM2D**
- **Bidirectional**

(https://keras.io/ko/layers/recurrent/)

GAN

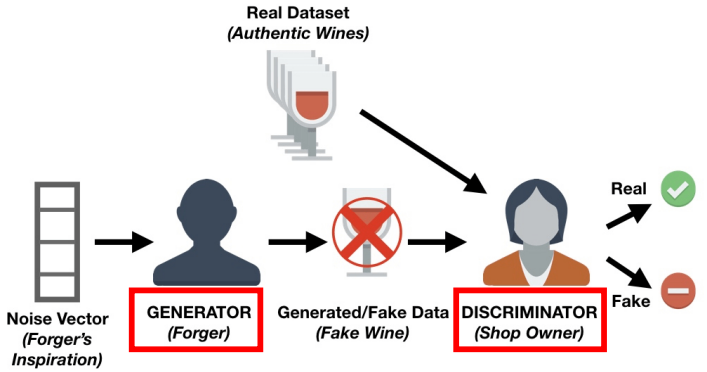
PIAI Research Department



GAN

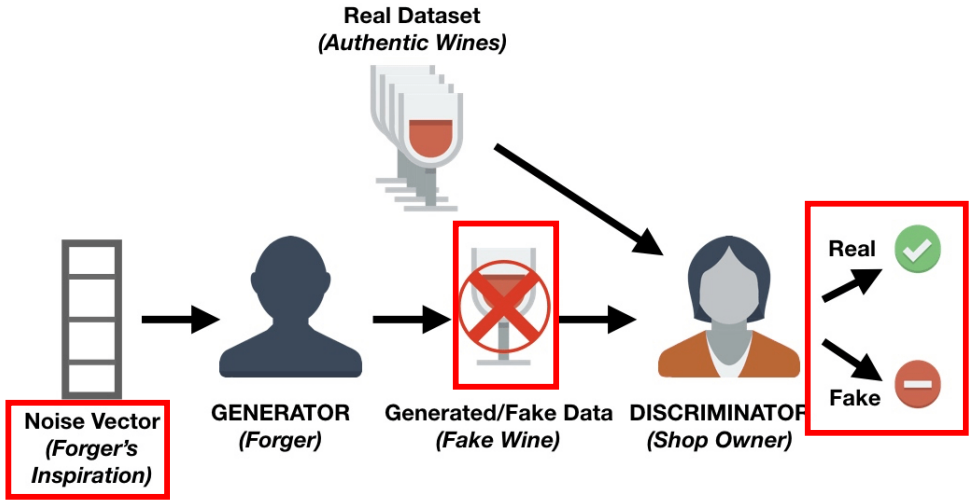
PIAI Research Department

```python
def get_generator(optimizer):
    generator = Sequential()
    generator.add(Dense(256, input_dim=random_dim, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(784, activation='tanh'))
    generator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return generator


def get_discriminator(optimizer):
    discriminator = Sequential()
    discriminator.add(Dense(1024, input_dim=784, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(256))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(1, activation='sigmoid'))
    discriminator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return discriminator
```



GAN

PIAI Research Department




```
def get_gan_network(discriminator, random_dim, generator, optimizer):

    discriminator.trainable = False # Generator와 Discriminator를 동시에 학습시 trainable을 False로 설정

    gan_input = Input(shape=(random_dim,)) # gan_input : 노이즈(100 차원)

    x = generator(gan_input) # X:이미지

    gan_output = discriminator(x) # gan_output : 이미지가 진짜인지 아닌지에 대한 확률

    gan = Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='binary_crossentropy', optimizer=optimizer)
    return gan
```

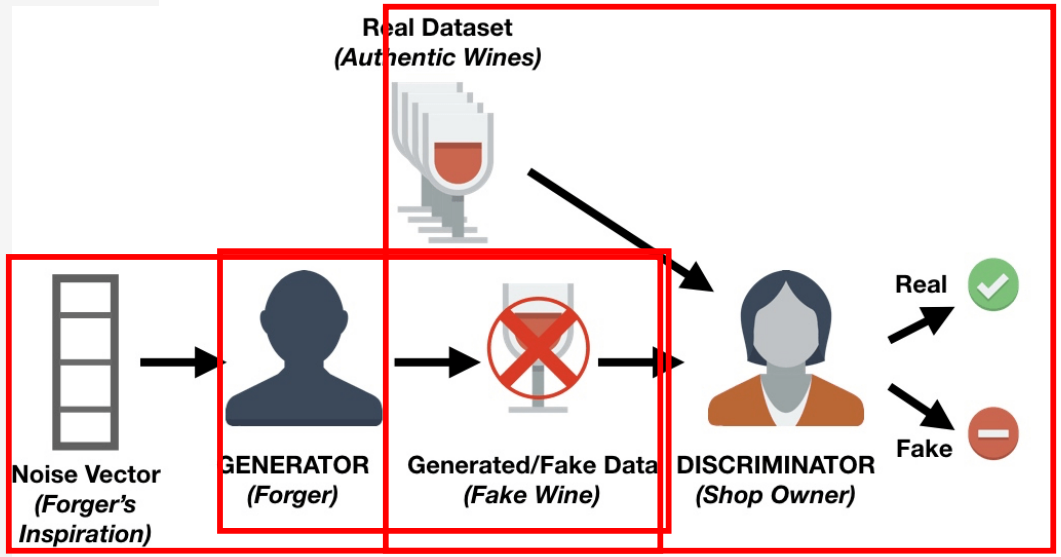
GAN

PIAI Research Department

```python
# Generator를 통해 MNIST 이미지를 생성
generated_images = generator.predict(noise, verbose =0)
X = np.concatenate([image_batch, generated_images])
```

```python
# Discriminator 학습
y_dis = np.zeros(2*batch_size)
y_dis[:batch_size] = 0.9
discriminator.trainable = True
discriminator.train_on_batch(X, y_dis)
```

```python
# Generator 학습
noise = np.random.normal(0, 1, size=[batch_size, random_dim])
y_gen = np.ones(batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, y_gen)
```



GAN

PIAI Research Department

1 Epoch

20 Epoch

GAN

PIAI Research Department

100 Epoch

400 Epoch

GAN – Another Data Augmentation?!

PIAI Research Department

Ground Truth

400 Epoch

Code Running

(student)4. Keras CNN – Additional Functions 2

가치창출대학
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

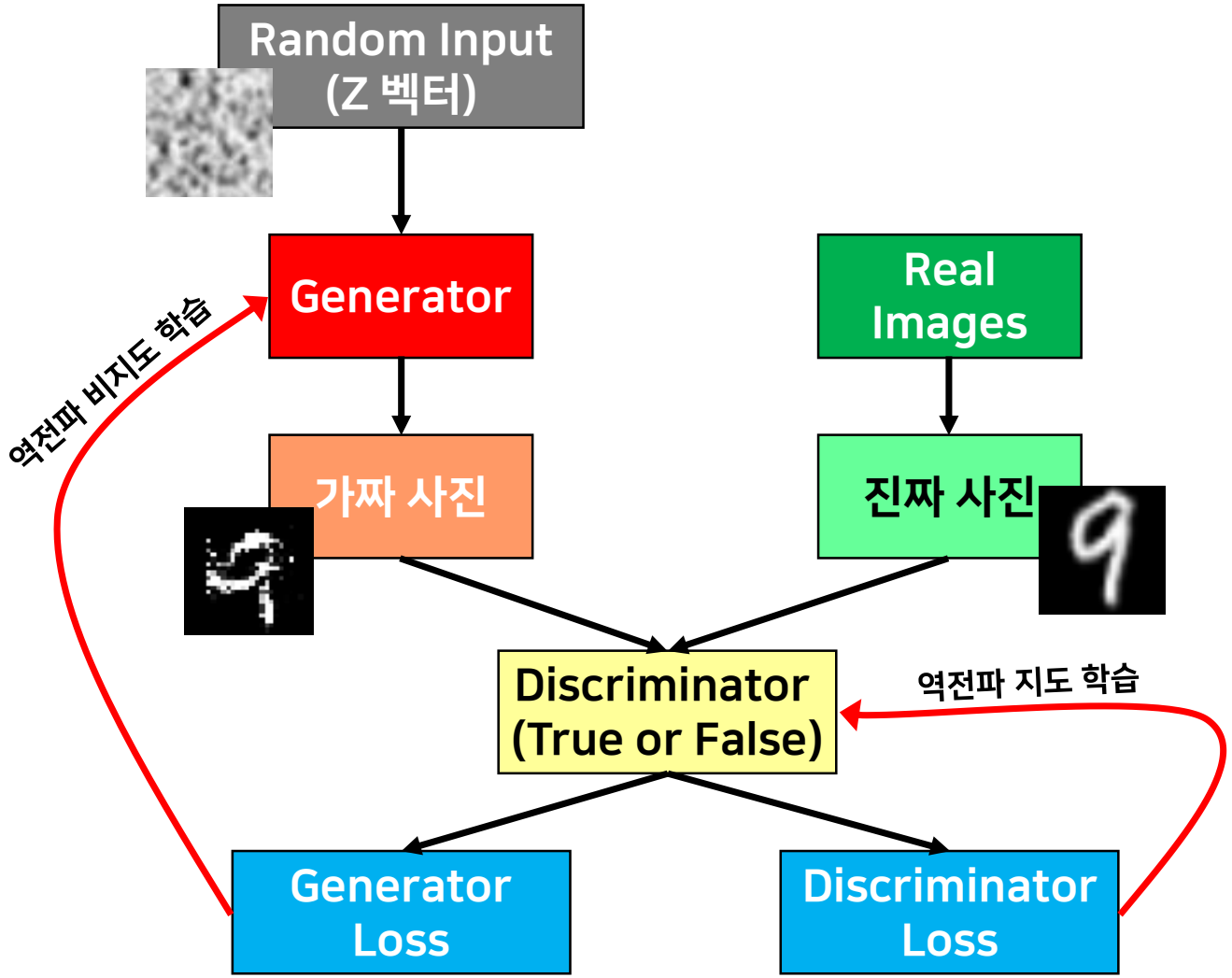
PIAI
인공지능연구원
POSTECH Institute of Artificial Intelligence

PyTorch Additional Functions

I-Eon, Na

GAN – with MNIST

PIAI Research Department



- GAN(Generative Adversarial Networks)
- 생성적 적대 신경망이라는 의미
- 비지도 학습에 사용되는 머신러닝
- 기존에 없는 새로운 데이터를 생성
- Generator와 Discriminator가 서로 경쟁하며 발전

GAN – with MNIST

PIAI Research Department

- **Generator 아키텍처**

-1 ~ 1 사이로 정규화

```
# MNIST Dataset
transform = transforms.Compose([ToTensor(), Normalize(mean=(0.5,), std=(0.5,))])
```

```
# 진짜같은 가짜 생성기
class Generator(nn.Module):
    def __init__(self, g_input_dim, g_output_dim):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(g_input_dim, 256)
        self.fc2 = nn.Linear(self.fc1.out_features, self.fc1.out_features*2)
        self.fc3 = nn.Linear(self.fc2.out_features, self.fc2.out_features*2)
        self.fc4 = nn.Linear(self.fc3.out_features, g_output_dim)

    # forward method
    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        return torch.tanh(self.fc4(x))
```

- 가짜 데이터를 생성하는 Generator 아키텍처 설계
- 입력으로 난수 입력
- 출력으로 28 * 28 이미지 생성
- -1 ~ 1 사이의 값으로 출력하기 위하여 Tanh Activation Function 사용

GAN – with MNIST

- **Generator 학습 알고리즘**

```
z_dim = 100
mnist_dim = train_dataset.train_data.size(1) * train_dataset.train_data.size(2)

G = Generator(g_input_dim = z_dim, g_output_dim = mnist_dim).to(device)
D = Discriminator(mnist_dim).to(device)
```

```
def G_train(x):

    G.zero_grad()

    z = torch.randn(batch_size, z_dim).to(device)
    y = torch.ones(batch_size, 1).to(device)

    G_output = G(z)
    D_output = D(G_output)
    G_loss = criterion(D_output, y)

    G_loss.backward()
    G_optimizer.step()

    return G_loss.data.item()
```

Binary Cross Entropy Loss

- G를 Generator 모델로 선언
- 난수로 가짜 이미지 생성
- 가짜 이미지를 Discriminator 투입
- Discriminator의 예측 값과 1 사이의 loss 산정
- 계산된 loss로 Generator 학습
- 결국 진짜에 가까운 이미지를 만들도록 학습됨

GAN – with MNIST

- **Discriminator 아키텍처**

```python
class Discriminator(nn.Module):
    def __init__(self, d_input_dim):
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(d_input_dim, 1024)
        self.fc2 = nn.Linear(self.fc1.out_features, self.fc1.out_features//2)
        self.fc3 = nn.Linear(self.fc2.out_features, self.fc2.out_features//2)
        self.fc4 = nn.Linear(self.fc3.out_features, 1)

    # forward method
    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        return torch.sigmoid(self.fc4(x))
```

진짜일 확률값을 출력

- 진짜와 가짜를 판별하는 Discriminator 아키텍처 설계
- 입력 28×28 이미지
- 출력으로 1 or 0 (진짜 or 가짜)

GAN – with MNIST

PIAI Research Department

- **Discriminator 학습 알고리즘**

```
def D_train(x):

    D.zero_grad()

    x_real = x.view(-1, mnist_dim)
    y_real = torch.ones(batch_size, 1)

    x_real, y_real = x_real.to(device), y_real.to(device)

    D_output = D(x_real)
    D_real_loss = criterion(D_output, y_real)
    D_real_score = D_output

    z = torch.randn(batch_size, z_dim).to(device)
    x_fake, y_fake = G(z), torch.zeros(batch_size, 1).to(device)

    D_output = D(x_fake)
    D_fake_loss = criterion(D_output, y_fake)

    D_loss = D_real_loss + D_fake_loss
    D_loss.backward()
    D_optimizer.step()

    return  D_loss.data.item()
```

- 실제 데이터셋 로드
- 진짜를 구별하도록 Loss 산정
- G로 가짜 데이터 생성
- 가짜를 구별하도록 Loss 산정
- 진짜와 가짜를 모두 구별하도록 학습

GAN – with MNIST

PIAI Research Department

- **GAN (G + D) 학습 알고리즘**

```
for epoch in range(1, n_epoch+1):
    D_losses, G_losses = [], []
    for batch_idx, (x, _) in enumerate(train_loader):
        D_losses.append(D_train(x))
        G_losses.append(G_train(x))
```

- G와 D를 번갈아 가며 학습

GAN – with MNIST

PIAI Research Department

1 epochs　　100 epochs　　200 epochs

- MNIST 데이터를 학습하여 세상에 없던 손 글씨를 생성
- MNIST 데이터 증강이 가능할지도..?

n epochs

출처: https://debuggercafe.com/generating-mnist-digit-images-using-vanilla-gan-with-pytorch/

Code Running

(student)4. Pytorch CNN – Additional Functions 2