TensorFlow中級コース

GANによる画像生成AI開発に挑戦

eEducation Labs
井上 博樹
hinoue@learningdesign.jp

GANで生成した画像サンプル

黄色で囲まれた部分が

GANによる合成画像

Generative Adversarial Nets
2014, Ian Goodfellowら
https://arxiv.org/pdf/1406.2661.pdf

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

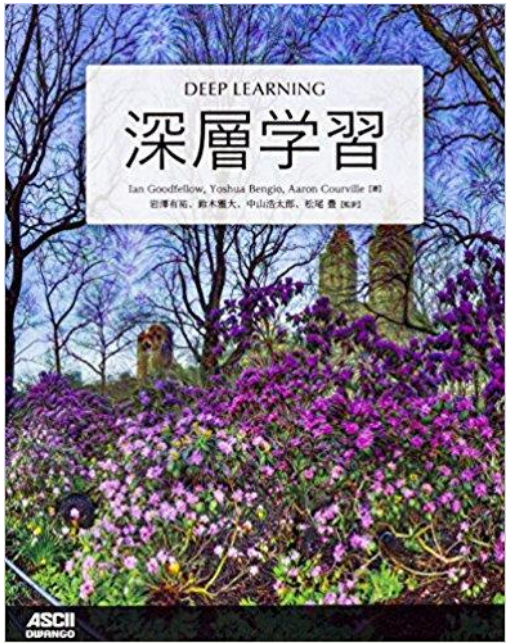
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Ian Goodfellow（GAN考案者）



@goodfellow_ian

https://arxiv.org/pdf/1406.2661.pdf

GANの仕組み

本物
x

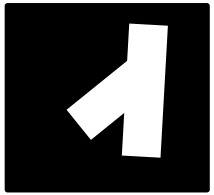
ランダムノイズ
z

Generator

Discriminator

Real（本物）

Fake（偽物）



生成・学習に多層NNを使用

CNNを使うと「DCGAN」

Generator

z（100）

h1(128)

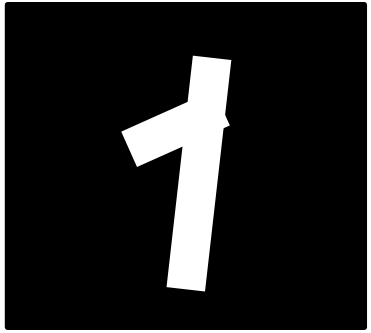
logits(784)

out(784)

ランダムノイズ

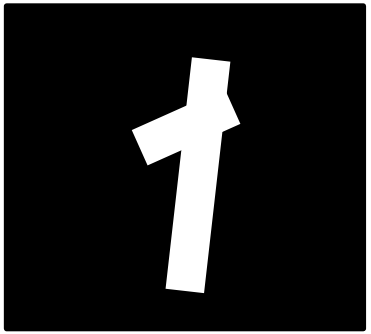
Leaky RELU

tanh



自動生成画像

Discriminator



判定対象
（偽物）
（本物）

x（784）

h1(128)

Leaky
RELU

logits

sigmoid

out(0~1)

真偽の確率

偽物 --> 0へ
本物 --> 1へ

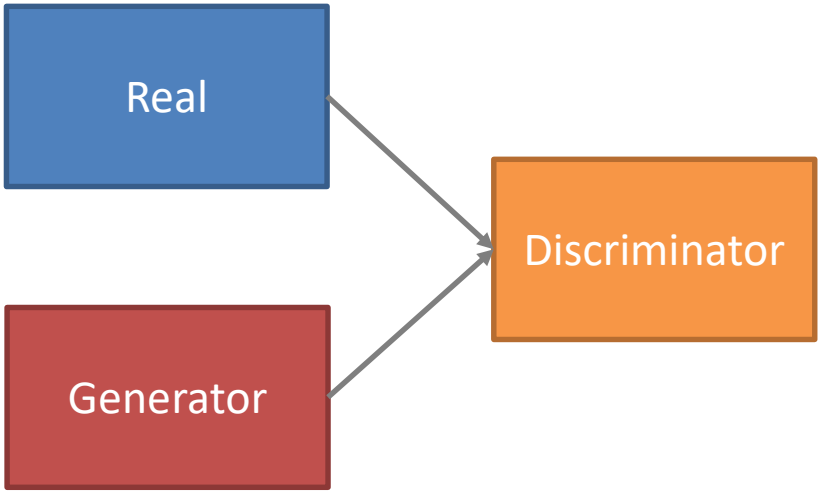
プログラムの流れ

1. パッケージのインポート
2. データのダウンロード

3. インプットデータの定義
4. ジェネレータを定義
5. ディスクリミネーターを定義
6. ハイパーパラメーターの初期化
7. 計算グラフの定義
    1. ロスの定義
    2. オプティマイザーの定義

8. トレーニング

9. 損失の評価
10. データの確認



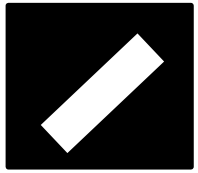
Generator

z（100）

h1(128)

logits(784)

out(784)

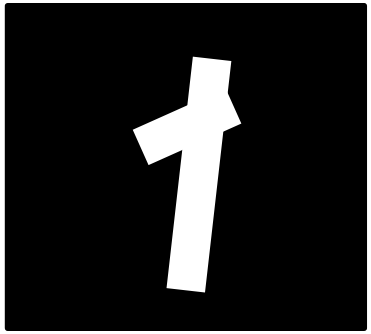


ランダムノイズ

ー1~1 一様分布

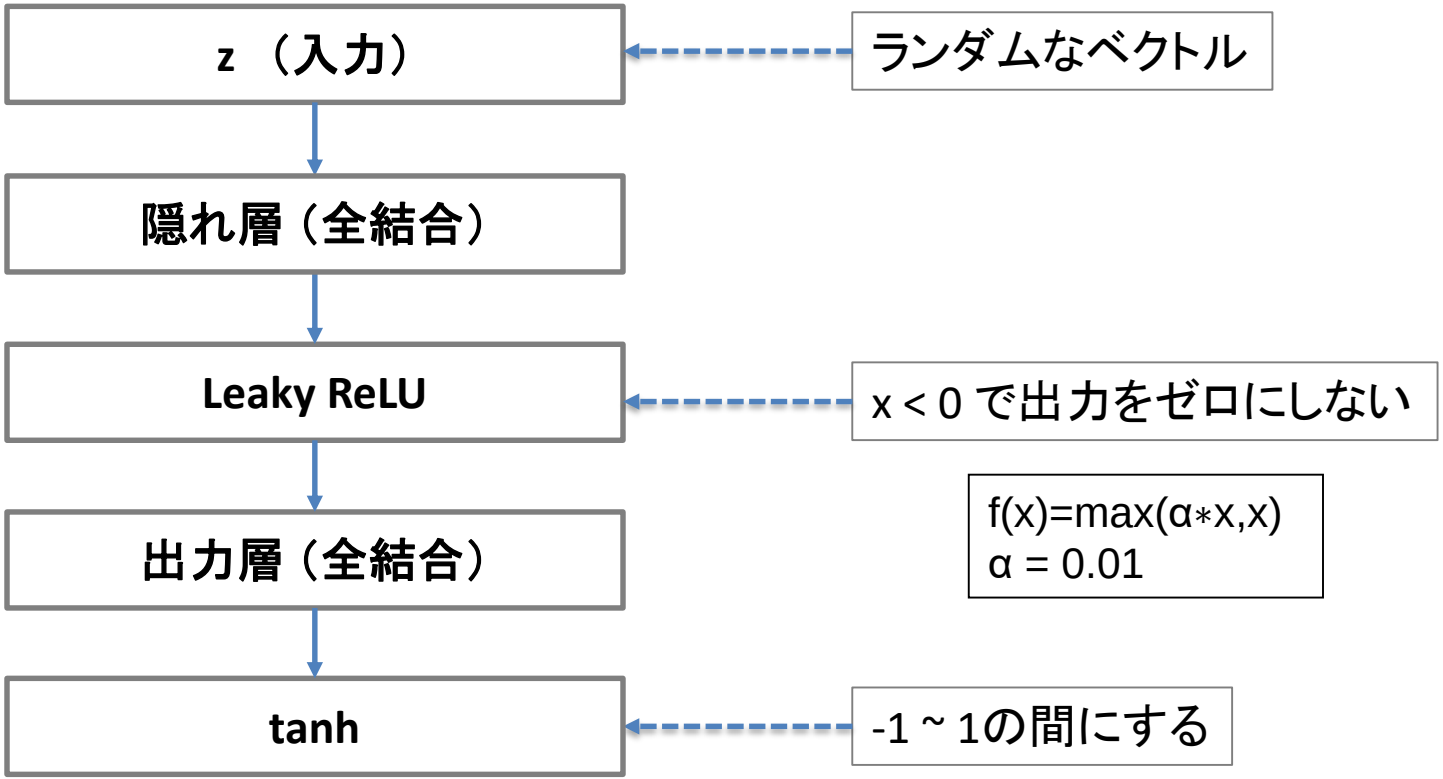
Leaky RELU

tanh



自動生成画像

Generator


z （入力）
ランダムなベクトル
隠れ層（全結合）
Leaky ReLU
x < 0 で出力をゼロにしない
f(x)=max(α∗x,x)
α = 0.01
出力層（全結合）
tanh
-1 ~ 1の間にする

tf.variable_scope

- reuseオプション
  - False: 関数が呼び出されるたびに値をリセット
  - reuse: 前回の値を保持する

tf.layers

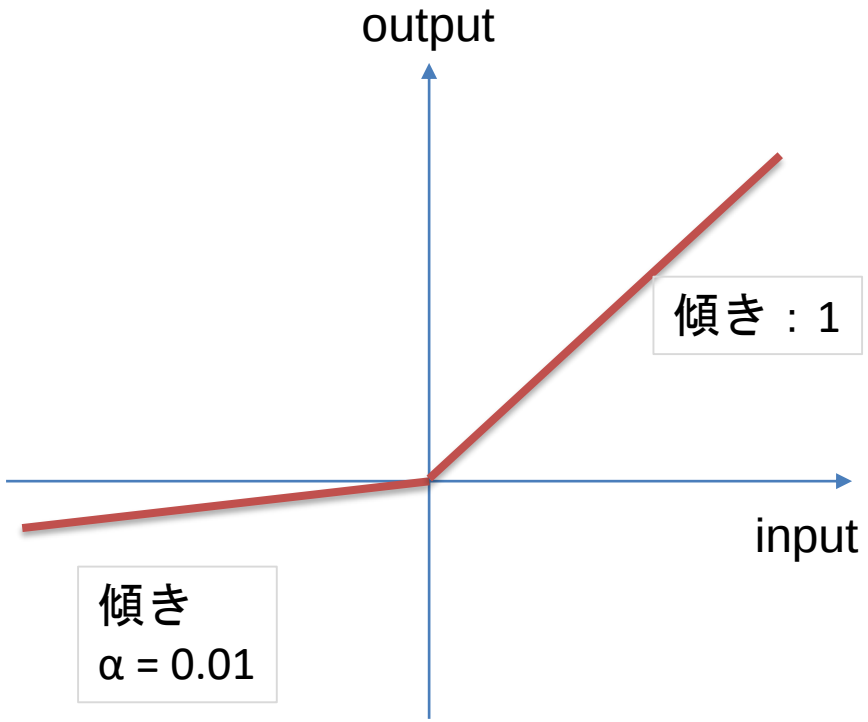
- 多層ニューラルネットワークを定義するライブラリ
- 全結合層
- 畳込み層

Leaky RELU

f(x)=max(α∗x,x)
α = 0.01

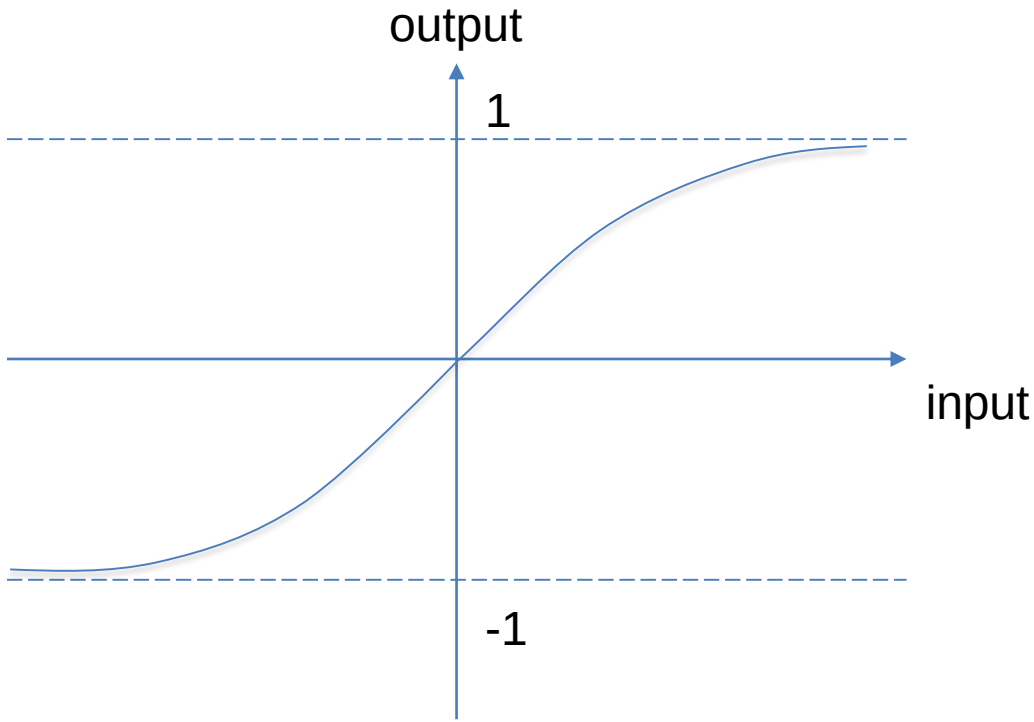
RELUなら

α = 0

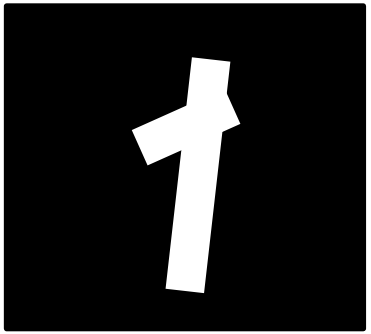


tanh（ハイパボリックタンジェント）

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



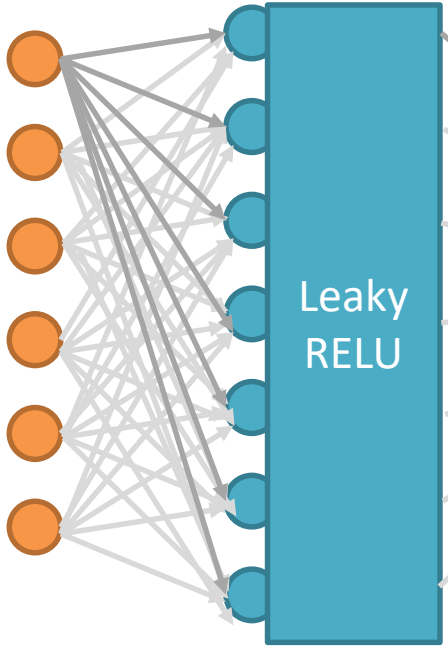
Discriminator



判定対象
（偽物）
（本物）

x（784）

h1(128)



logits

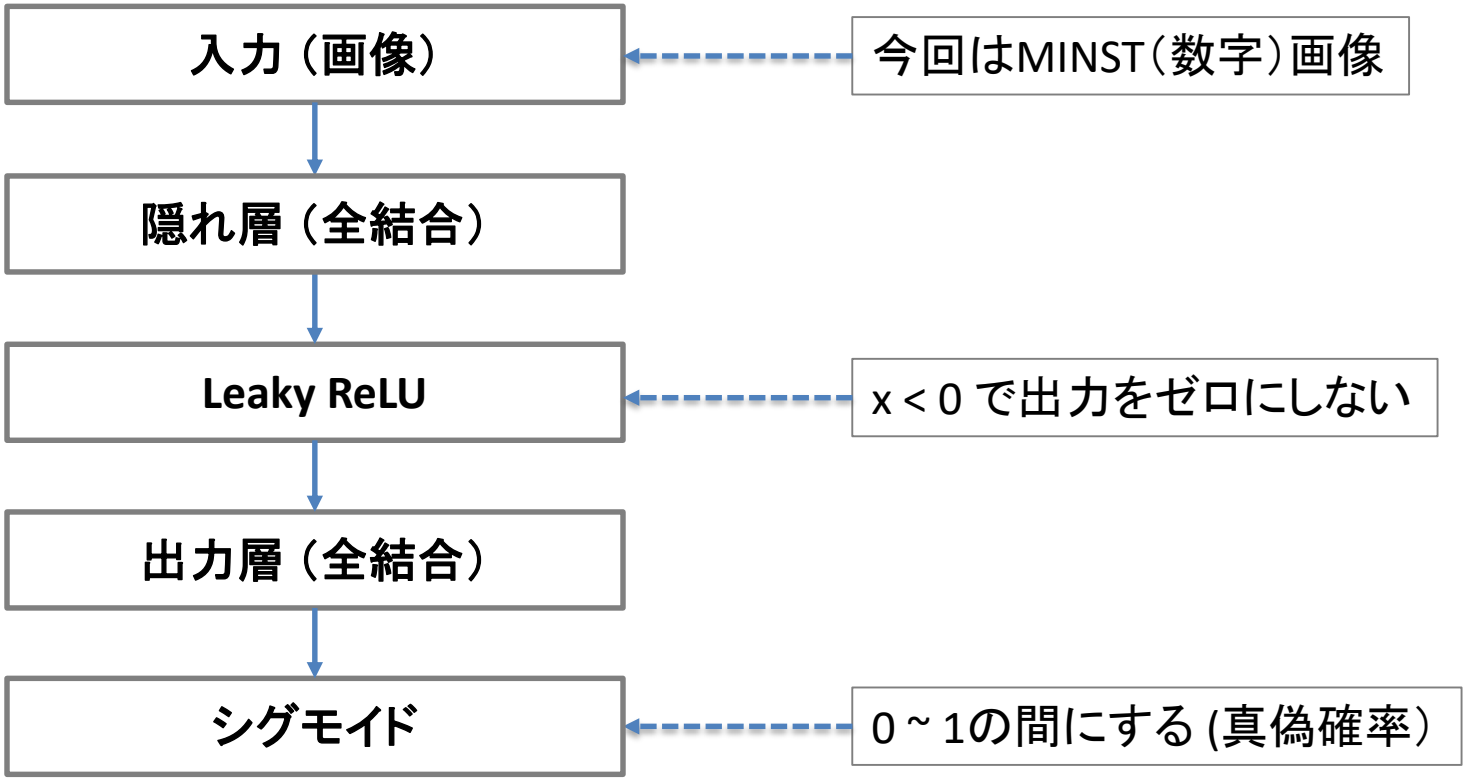
sigmoid

out(0~1)

真偽の確率

偽物 --> 0
本物 --> 1

Discriminator



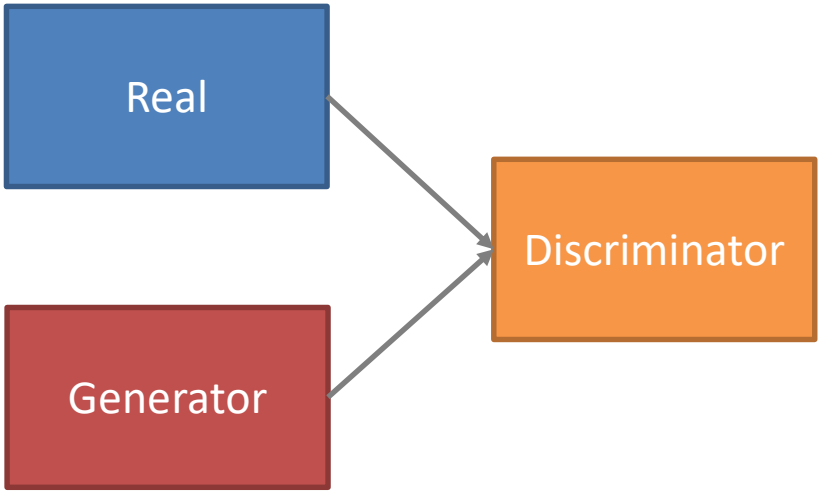
プログラムの流れ

1. パッケージのインポート
2. データのダウンロード

3. インプットデータの定義
4. ジェネレータを定義
5. ディスクリミネーターを定義
6. ハイパーパラメーターの初期化
7. 計算グラフの定義
    1. ロスの定義
    2. オプティマイザーの定義

8. トレーニング

9. 損失の評価
10. データの確認



GANの仕組み

本物
x

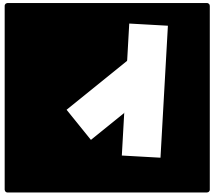
ランダムノイズ
z

Generator

Discriminator

Real（本物）

Fake（偽物）



生成・学習に多層NNを使用

CNNを使うと「DCGAN」

GANの構築

1. **モデルの定義**

    1. **登場する変数を定義する（D, G）**

2. 損失関数（Loss）を定義する

3. 最適化手法を定義する

モデル（式）の定義

1. 入力変数を作る（input_real, input_z）

2. ジェネレータ出力を作る（g_model）

3. ディスクリミネーターを作る
   - リアル（d_model_real, d_logits_real）
   - フェイク（d_model_fake, d_logits_fake）

GANの構築

1. モデルの定義
    1. 登場する変数を定義する（D, G）
2. **損失関数（Loss）を定義する**
3. 最適化手法を定義する

損失関数の定義

- クロスエントロピー（最小化したい）
  - d_loss_real: 1（-smooth）との誤差
  - d_loss_fake：ゼロとの誤差
  - d_loss = d_loss_real + d_loss_fake

  - g_loss：　正解1との誤差

GANの構築

1. モデルの定義
    1. 登場する変数を定義する（D, G）
2. 損失関数（Loss）を定義する
3. **最適化手法を定義する**

最適化

1. learning_rate：学習率

2. 最適化対象の取り出し：　trainable_variables
   - NNのセル間結合の重みやバイアスなどのパラメーター

3. 最適化処理の定義

   - variable_scope（名前で関連付け）でD/G識別

   - d_train, g_trainを最適化

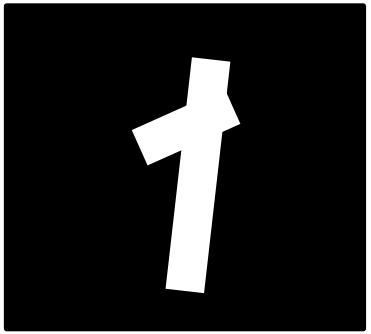
Discriminator

weight・bias

x（784）

h1(128)

h1 = **w** * x + **b**



判定対象
（偽物）
（本物）

Leaky
RELU

logits

out(0~1)

sigmoid

真偽の確率

偽物 --> 0
本物 --> 1

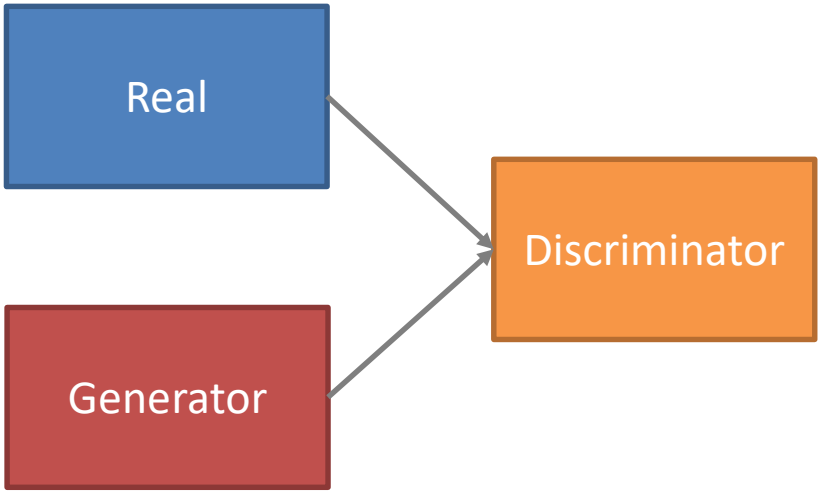
プログラムの流れ

1. パッケージのインポート
2. データのダウンロード

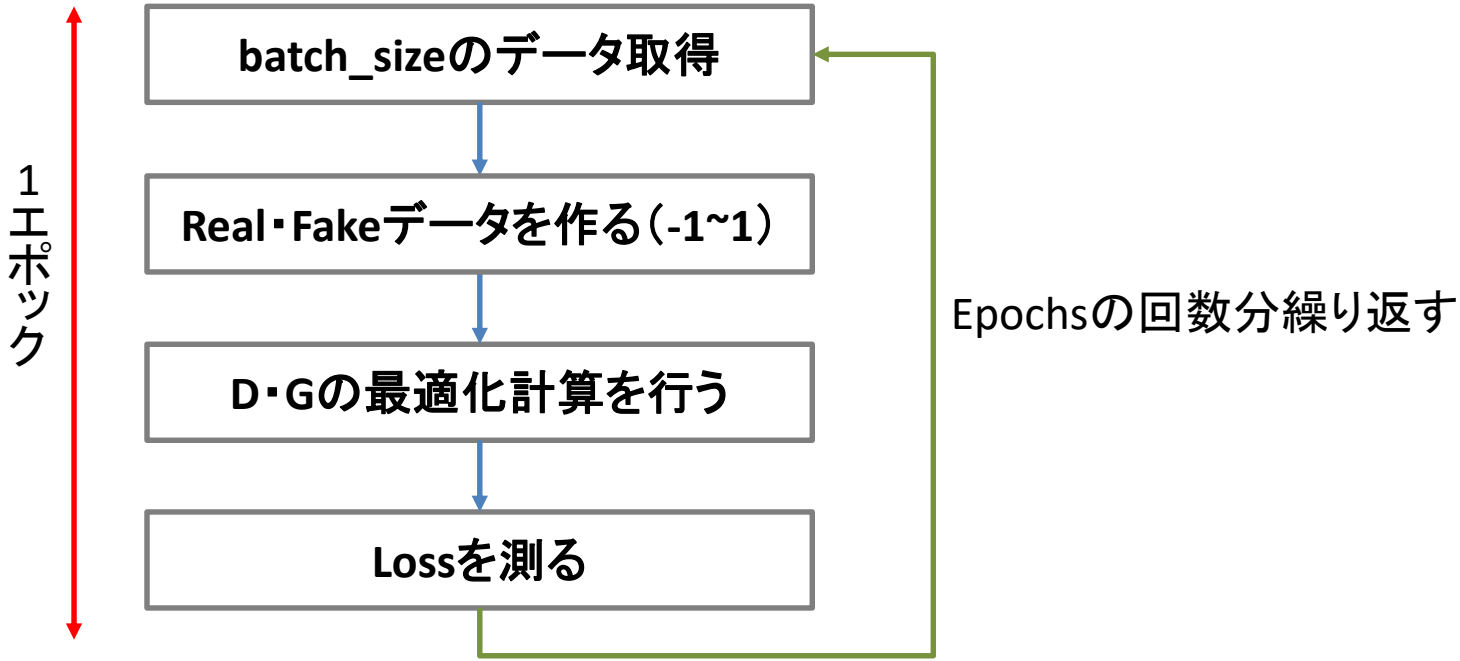
3. インプットデータの定義
4. ジェネレータを定義
5. ディスクリミネーターを定義
6. ハイパーパラメーターの初期化
7. 計算グラフの定義
    1. ロスの定義
    2. オプティマイザーの定義

8. トレーニング

9. 損失の評価
10. データの確認



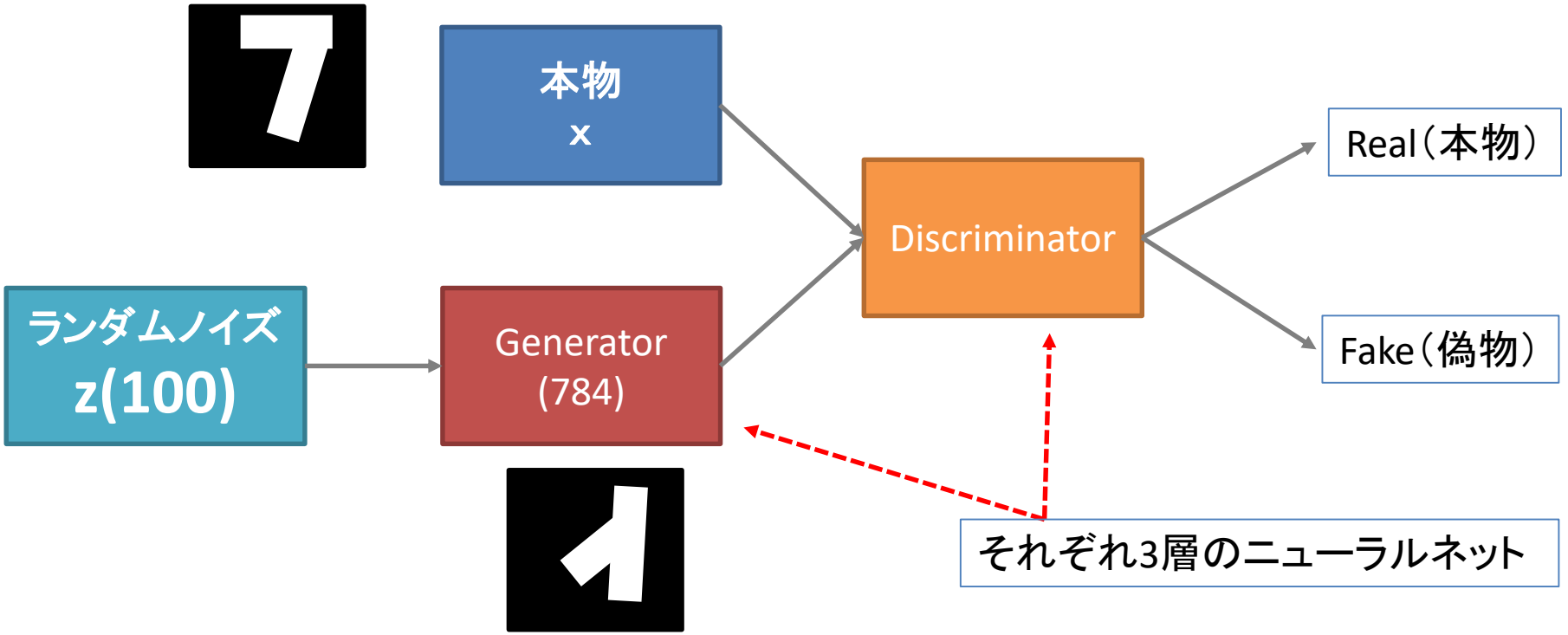
トレーニングの流れ


1エポック
batch_sizeのデータ取得
Real・Fakeデータを作る（-1~1）
D・Gの最適化計算を行う
Lossを測る
Epochsの回数分繰り返す

MNISTデータの確認

- batchを取り出す
  - batch = mnist.train.next_batch(batch_size)
- 表示してみる（batch[0][0], batch[1][0]）
  - データとラベル

セクションのWrap-up



TensorFlowの使い方

1. モデル(計算式）を定義しておく
   - 各式は連携している（計算グラフ）
   - データの生成、ニューラルネットワーク出力、パラメータ更新アルゴリズム
2. プレイスホルダー（変数を収容）
3. トレーニングセッションを実行
   1. feed_dictから計算グラフにデータを投入
   2. エポック数分トレーニングを繰り返す
      1. Generatorは本物を目指す
      2. Discriminatorは、真偽判定の精度を上げたい
4. モデルを使用して生成を行う
   1. セッションでGeneratorを実行する

次のセクション

- 畳み込みNNでD・Gをつくる
- Street View Housing Data Set