案

GithubにｱｯﾌﾟしてW5完了

Schedule案0

TPUｺｰﾃﾞｨﾝｸﾞやる。

Schedule案1

KaggleNotebookをまるごとColabに引っ越し

→Colabから出力、ipynb保存

→Gitにｱｯﾌﾟﾛｰﾄﾞ

Schedule案2

4/14-18 DMPFinal 発展DL7-6読み込み、理解、”Transformer”実装

【KaggleGPU開放待ち】

4/19;9am~ IntroDLW5”CycleGAN” 最新ｺｰﾄﾞで出力、ipynbﾀﾞｳﾝﾛｰﾄﾞ、Gitｱｯﾌﾟﾛｰﾄﾞ、Coursera提出

4/20～25 IntroDLFinal”Sagat”実装～

DMProject Final候補

発展DL7-6Transformer実装＋Slurmで分散訓練

＋FSDB（精度比較）

＋Flash Attention（精度比較）

G-chanメモ

|  |  |  |
| --- | --- | --- |
| FSDP | モデルシャード化 | 大規模モデルの分散に必須 |

|  |  |  |
| --- | --- | --- |
| Flash Attention | メモリ節約＋計算効率化 | 長いシーケンスのとき超効果的 |

|  |  |  |
| --- | --- | --- |
| Slurm + NCCL | 分散通信高速化 | クラスタ環境で鉄板 |

参考：FSDP による分散型トレーニング

https://zenn.dev/intel\_developer/articles/961037e2530a14

（G-chanお薦め順）

・Transformer実装

・BERT実装

・OpenPose実装

・Transformerによる画像ｱﾉﾃｰｼｮﾝ実装（著者はｸﾗｳﾄﾞ有料枠を利用→Slurmで）

IntroDL W5: cycleGAN

Version7 = TPU code試行中

Version6 = GPU code完成版

新規ﾉｰﾄﾌﾞｯｸで作り直し

（某からｺﾋﾟｰしたﾉｰﾄﾌﾞｯｸ、と明記されるのはﾎﾟｰﾄﾌｫﾘｵとして要改善）

案：ついでにSlurmを追加することで独自性だす＋DMP FinalでのSlurmの予行練習

Monet Painting Dataset

<https://www.kaggle.com/competitions/gan-getting-started>

IntroDL Final: SAGAN

・写経：完

・理解埋め込み

・FineTuning（前後値でテスト）

・分析結果のAnalysis＆Conclusion記述

・英文化

・Githubｱｯﾌﾟ

・プレゼン録画

・class registration

・Courseraｱｯﾌﾟﾛｰﾄﾞ

図表と同じように私たちのGeneratorも5層の畳み込み層で構成されます。

図表と違うのは、4層目および5層目のConv層にはAttentionが組み込まれている点、入力する乱数ﾍﾞｸﾄﾙは100次元ではなく20次元、最終層は3次元ではなく1次元になります。次元の調整は我々の取り組みがｸﾞﾚｰｽｹｰﾙの数字画像生成であるためです。

# Introduction

このNotebookでは、MSDS Introduction to Deep Learning courseのFinal projectとしてSAGANの実装を行います。

同じcourseの別のprojectとして、cycleGANの課題があり、私はそのprojectの実装をとても楽しみました。

それなので、今回はcycleGANで学んだGANの実装に加えて、今日の多くのMLｱﾌﾟﾘｹｰｼｮﾝに応用されているAttention mechanismを取り込んだSelf Attention GANに取り組むことにしました。

ジェネレータは乱数ﾍﾞｸﾄﾙを画像の種として、転置畳み込みを複数回繰り返すことで期待する画像サイズの画像を生成します。

このDCGANのGenerator構造はその後のGANの標準的なデザインのﾍﾞｰｽとなっており、SAGANやStyleGANもこの流れを踏襲しています。

Self Attention GANはAttention mechanismのみならず、pointwise convolutionやspectral normalizationなど重要かつ興味深いdeep learning関連技術が実装されています。

それらの技術を理解するのは決して簡単ではありませんが、実装を通してしっかり身につけたいと思います。

The goal of the notebook is to generate MNIST-like mimicked-hand-written number from SAGAN.

In order to reduce computation, I will only use '7' and '8' labeled data from the MNIST.

図表はSelf Attention GANの論文から抜粋しました。

図表中のf(x)をkey, g(x)をquery, h(x)をvalueと置き換えれば、これはtransformerやMegatron-LMなどの論文に見られるAttention mechanismそのものです。

Transfomerで利用されるSelf AttentionのScaled dot productの式 Attention(Q,K,V) = softmax(Qk^T/sqrt(D))Vがそのまま図表で表されていることがわかります。

Self AttentionとCross Attentionの違い：

TransformerにはそのｱｰｷﾃｸﾁｬにSelf AttentionとCross Attentionが含まれます。

ﾓﾃﾞﾙによってはSelf Attentionのみ（BERTやSAGANなど）を使う場合もありますが、この二つの違いを整理します。

Self Attention：

Self Attentionは、入力ﾃﾞｰﾀ内で自分自身の異なる部分同士を比較して関係を捉える仕組みです。

画像の場合、あるﾋﾟｸｾﾙ（またはﾊﾟｯﾁ）が、同じ画像内の他のﾋﾟｸｾﾙとどれくらい関連があるかを計算します。

このように「同じﾃﾞｰﾀ内での自己参照」が行われるため、self attentionと呼ばれます。

Cross Attention：

Cross Attentionは、異なるデータ同士を比較する仕組みです。

例えば、TransformerではEncoderが抽出した特徴をDecoderが参照する際にCross Attentionが使われます。

具体的には、Self Attentionが抽出した特徴（これはEncoderの処理です）と、ﾓﾃﾞﾙ自身の過去の出力ﾃﾞｰﾀに基づいて計算（これはDecoderの処理です）して予測・出力を行います。

なお、Self AttentionもCross Attentionも、最終的にはQueryとKeyの内積を取る点は共通しています。

つまり

・Self Attentionは「同じﾃﾞｰﾀ内での自己参照」

・Cross Attentionは「異なるデータ同士の比較参照」

という関係です。

W5 GAN道しるべ

Competition page:

<https://www.kaggle.com/competitions/gan-getting-started>

kaggle kernels output yu1111/monet-cyclegan -p /path/to/dest

**GAN Implementation tips**

* Scale the image pixel value between -1 and 1. Use tanh as the output layer for the generator.
* Experiment sampling ***z*** with Gaussian distributions.
* Batch normalization often stabilizes training.
* Use PixelShuffle and transpose convolution for upsampling.
* Avoid max pooling for downsampling. Use convolution stride.
* Adam optimizer usually works better than other methods.
* Add noise to the real and generated images before feeding them into the discriminator.

<https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b>

MNISTのGANｽｸﾗｯﾁ実装

https://qiita.com/mgmk2/items/4847e01501a3d56a3094

**ｻﾗﾐ：ペア画像必要なし！教師なし学習で光学画像とSAR画像を相互変換する**

<https://sorabatake.jp/15701/>

この資料↓超読み易い！写経おすすめ

Keras-GAN-cycleGAN

<https://github.com/eriklindernoren/Keras-GAN/blob/master/cyclegan/cyclegan.py>

Keras-GAN-pix2pix

<https://github.com/eriklindernoren/Keras-GAN/blob/master/pix2pix/pix2pix.py>

W5PeerReview；

**I’m Something of a Painter Myself**

<https://www.kaggle.com/competitions/gan-getting-started>

参照；

Pytorch code with bronze medal：

<https://www.kaggle.com/code/chuyangzhang/i-m-something-of-a-painter-myself-cyclegan>

Monet CycleGAN Tutorial by Amy Jang

<https://www.kaggle.com/code/amyjang/monet-cyclegan-tutorial>

実装練習；

DCGAN implementation by Keras

<https://keras.io/examples/generative/dcgan_overriding_train_step/>

<https://www.tensorflow.org/tutorials/generative/dcgan>

Using tf.GradientTape

<https://www.tensorflow.org/guide/autodiff>

# Introduction

Welcome!

このノートブックでは、偉大な巨匠モネの絵画を写真調に、また風景写真をモネの絵画調に変換するGenerative Adversarial Network, cycleGANの仕組みをｺｰﾄﾞを通して探索します。

また私自身、TensorFlowでdeep learningの本格的なｺｰﾄﾞに取り組むのは初めてです。そのためこのNotebookでは、GANの構造の探索に留まらず、TensorFlowにおけるDeep Learningのｺｰﾃﾞｨﾝｸﾞの構造を並行して探索します。

TensorFlowに慣れている方には、既知の情報がしつこく記述されて恐縮ですが、TensorFlowに慣れていない方には有用な情報が少しでもお役に立てば幸いです。

# Discriminator

The discriminator is a neural network trained to distinguish between real images and from the dataset and fake images generated by the generator.

# The patchGAN

patchGANは、画像全体ではなく、ﾛｰｶﾙな小さな領域ごとに判定を行います。例えば、256x256の画像に対して全範囲にfakeかrealかの判定を行うのではなく、例えば70x70の小さなﾊﾟｯﾁ単位で判断を行います。ﾊﾟｯﾁとは受容野のことです。この局所的な判定により高周波数成分（ｴｯｼﾞやﾃｷｽﾁｬなど細部の特徴）を捉えることができ、細部のﾘｱﾘﾃｨを向上させる効果があります。最終的に全ﾊﾟｯﾁの平均を計算して判定に利用します。この手法をpatchGANと呼びます。

# Three times downsampling

今回のdiscriminatorは3回のdownsamplingによって次元削減した特徴ﾏｯﾌﾟに調整を加えて、最終的に30x30のｽｺｱﾏｯﾌﾟを出力します。

ここで、30x30はｽｺｱﾏｯﾌﾟであり、ﾊﾟｯﾁｻｲｽﾞではないことにご注意ください。ﾊﾟｯﾁとは、30x30のｽｺｱﾏｯﾌﾟの「各ｾﾙ」が見ている、元の画像のﾋﾟｸｾﾙ領域（受容野）です。

今回のﾊﾟｯﾁｻｲｽﾞは70x70で、この限定された範囲からﾀﾞｳﾝｻﾝﾌﾟﾘﾝｸﾞ（通常はｽﾄﾗｲﾄﾞ付き畳み込み）によって特徴を抽出し、最終的に1つのｽｺｱ（確率）が付与されます。

30x30のｽｺｱﾏｯﾌﾟとは、個々のｾﾙに、そのｾﾙの受容野に対するｽｺｱが割り当てられたものです。

また、画像ｻｲｽﾞがより大きい場合や小さい場合は、適切なスコアﾏｯﾌﾟを出力するためdownsamplingの回数を調整します。例えば、より大きな画像ならﾀﾞｳﾝｻﾝﾌﾟﾘﾝｸﾞの回数を増やし、小さな画像なら減らします。

# Identity Mapping (Identity loss)

ここでもう一つの損失であるidentity lossを追加します。この損失は、生成画像と元の画像の誤差を測定することで、Generatorが画像の色調を維持することを学習します。

identity lossが無い場合、Generatorは元の画像に似たものを生成できたとしても、空が赤かったり、海が黄色だったりと、色調を上手く再現できないことがあります。

これを防ぐためにidentity lossが有効です。

identity lossは次の式で表されます：

$$

\mathcal{L}\_{identity}(G, F) =

\mathbb{E}\_{x \sim P\_{\text{data}}(x)}

\left[ ||F(x) - x||\_1 \right] +

\mathbb{E}\_{y \sim P\_{\text{data}}(y)}

\left[ ||G(y) - y||\_1 \right]

$$

LAMBDA = 12

Epoch 5/5

300/300 [==============================] - 79s 250ms/step - monet\_gen\_loss: 3.8285 - photo\_gen\_loss: 4.0929 - monet\_disc\_loss: 0.6338 - photo\_disc\_loss: 0.5853

LAMBDA = 10

Epoch 5/5

300/300 [==============================] - 145s 475ms/step - monet\_gen\_loss: 3.2847 - photo\_gen\_loss: 3.3653 - monet\_disc\_loss: 0.6294 - photo\_disc\_loss: 0.6132

LAMBDA = 8

Epoch 5/5

300/300 [==============================] - 74s 242ms/step - monet\_gen\_loss: 2.8941 - photo\_gen\_loss: 2.9720 - monet\_disc\_loss: 0.6323 - photo\_disc\_loss: 0.6246