NTT

2024.06.13

情報科学特別講義 I

生成モデルの基礎と応用

高橋 大志 (NTT)

自己紹介

- 名前　高橋 大志
- 経歴
  - 2009.4 – 2015.3　東京工業大学 (学士・修士)
  - 2020.4 – 2023.3　京都大学 (博士)
  - 2015.4 – 現在　NTT (研究所・ドコモ)
- 研究分野　生成モデル・異常検知

特に Variational Autoencoder (VAE)
というモデルの研究をしています

はじめに (10分)

生成AIについて

- 言語や画像、音声などを生成する人工知能 (AI) の総称

チャットボット

ChatGPT

Gemini

Claude

tsuzumi

画像生成

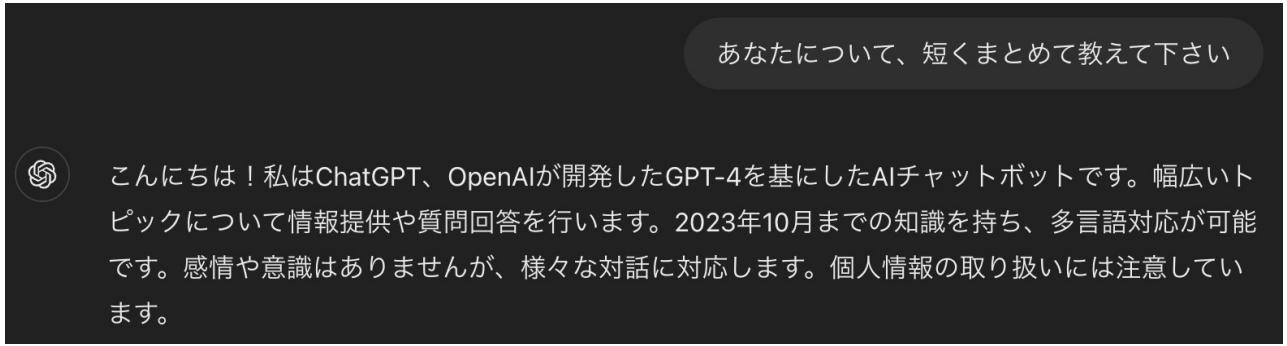
DALL-E[1]

Stable Diffusion[2]

1. https://ja.wikipedia.org/wiki/DALL-E
2. https://ja.wikipedia.org/wiki/Stable_Diffusion

ChatGPT (OpenAI)

- OpenAIが2022年11月に公開したAIチャットボット
  - 情報提供や質問回答、文章作成、対話、言語学習支援などが可能
  - GPT = Generative Pre-trained Transformer (後述)
  - 非常に高い性能から話題となり、ChatGPT公開後にOpenAIの評価額は290億ドルまで上がった[1] (2024年2月時点で推定800億ドル超え[2])



1. https://www.businessinsider.com/chatgpt-creator-openai-talks-for-tender-offer-at-29-billion-2023-1
2. https://forbesjapan.com/articles/detail/69210

DALL-E (OpenAI)

- OpenAIが2021年1月に公開した画像生成AI
  - ユーザが入力したテキストから画像を生成することが可能
  - 詳細は不明だがGPTベースとのこと (画像系AIはDiffusion (後述) が主流)

例: 桜が咲き誇る静かな日本庭園にある、アニメ風の静かな鯉の池

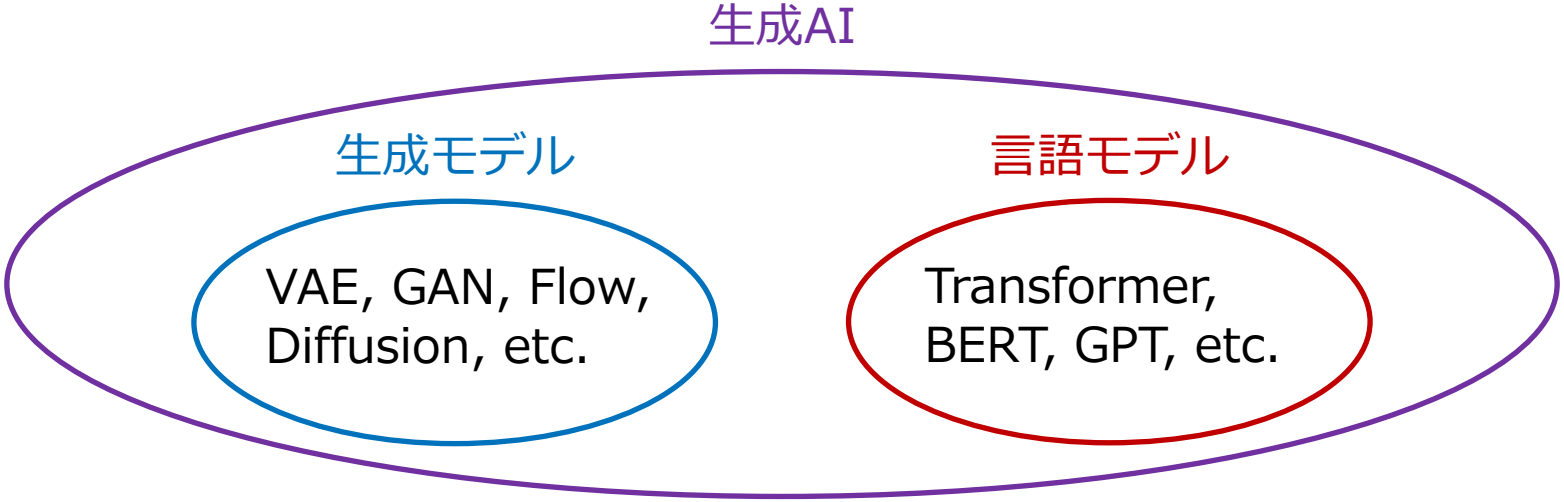
https://openai.com/index/dall-e-3/

本講義の狙い

- 生成AIは研究開発のスピードが非常に速く、キャッチアップが難しい
  - 2024年現在、数ヶ月ごとに大きな進化が起こっている
- その一方で、生成AIの基礎となっている部分は昔からほとんど変わっていない
  - 本講義で最初に説明する最尤推定は1912年頃から研究されている
- 本講義は、生成AIの基礎と応用を学び、将来的な生成AIの研究開発の一助となることを目指す
  - 最先端の生成AIの一歩手前までの内容を取り扱う

生成モデルと生成AI

- 本講義のタイトルに用いている生成モデルは機械学習用語
  - 現在は事実上、画像を生成するモデルを指すことが多い
  - 一方で、生成AIはテキスト・画像・音声などを生成するAI全般を指す
  - 本講義では、生成モデルと言語モデル (言語の生成モデル) を扱う



本講義の内容

- 本講義で取り扱うこと
  - 生成モデルの基礎 (最尤推定 / ガウス混合モデル)
  - 深層生成モデル (VAE / GAN / Flow / Diffusion) ← 主に画像
  - 言語モデル (Transformer / BERT / GPT) ← 主に言語
  - Hugging Face入門
  - 生成モデルの問題点
- 本講義で取り扱わないこと
  - 最適化の詳細について (主にEMアルゴリズムや確率的勾配法など)
  - 大規模言語モデル (主にTransformer以外の要素)

講義を始める前に

- 数式が多く出てきますが、すべて理解しようとしなくてOKです
  - 重要な数式は目立つようにします
- 講義中に質問する時間を多めに設けます
  - それ以外でも挙手や発声で止めていただいてOKです
- PCやスマホなどで資料を見ながらお聞きください
  - 3時間と長い講義なので、各自のペースでお聞きください

A. 生成モデルの基礎 (40分)

生成モデルについて

- データから学習された確率分布のこと
  - 例えば、コインの裏表の確率は1/2で、サイコロの目の確率は1/6
  - このようなデータが発生する「確率」を、データから学習したモデル
- 確率分布が分かっていれば、新たなデータを「生成」できる
  - 本講義では、データが従う確率分布を、生成モデルで学習する方法を学ぶ
- 本講義で扱う確率分布は下記の2つ:
  - データ x の確率分布 $p(x)$: コインやサイコロなどはこちら
  - 出力 y に対する条件付き確率分布 $p(y|x)$: ChatGPTなどはこちら
  - NOTE: データ x や出力 y はベクトルまたはスカラ (区別せずに表記)

確率分布について

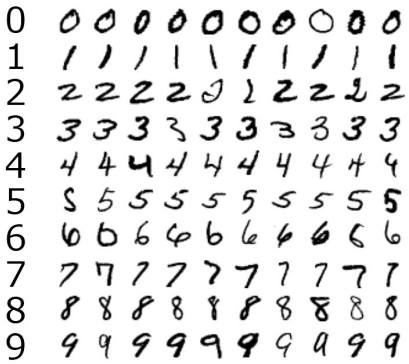
- データ x が発生する確率を表す関数
  - 例えば、コインの表裏を表す確率は $P(x) = \begin{cases} 0.5\ (x = \text{表}) \\ 0.5\ (x = \text{裏}) \end{cases}$
  - 離散値の場合は、$0 \leq P(x) \leq 1$ かつ $\sum_x P(x) = 1$ をみたす
- 連続値の場合は、確率密度関数 $p(x)$ を考える
  - データ x が a 以上 b 以下である確率は $P(a \leq x \leq b) = \int_a^b p(x)\,\mathrm{d}x$
  - $0 \leq p(x) < \infty$ かつ $\int_{-\infty}^{\infty} p(x)\,\mathrm{d}x = 1$ をみたす
  - NOTE: 確率分布 P と確率密度関数 p は区別せずに表記

生成モデルの仮定

データ x は真の確率分布 $p^*(x)$ に従う

数字一つがデータ x

$p^*_{\mathrm{MNIST}}(x)$



MNIST

$p^*_{\mathrm{CIFAR10}}(x)$

CIFAR10

画像一枚がデータ x

ITmedia: https://www.itmedia.co.jp/

最尤推定 (1)

- 前提: 真の分布 $p^*(x)$ は未知の関数だが、$p^*(x)$ に従うデータセット $\mathcal{D} = \{x_1, \ldots, x_N\}$ が利用可能
- 目的: データセット $\mathcal{D}$ を用いて、生成モデル $p_\theta(x)$ を真の分布 $p^*(x)$ に近づけるよう学習したい (θはパラメータ、後述)
- 方法: カルバック・ライブラー情報量という、2つの確率分布の差異を計る尺度を導入し、最小化することでモデルを学習する

$$\mathrm{KL}(p^*(x)||p_\theta(x)) = \int_{-\infty}^{\infty} p^*(x) \log \frac{p^*(x)}{p_\theta(x)} \mathrm{d}x$$

確率分布同士の距離のようなもの (対称性はない)

最尤推定 (2)

- カルバック・ライブラー情報量は下記のように変形できる

$$\mathrm{KL}(p^*(x)||p_\theta(x)) = \underbrace{\int_{-\infty}^{\infty} p^*(x)\log p^*(x)\mathrm{d}x}_{\text{定数 (生成モデルに関係ない値)}} - \int_{-\infty}^{\infty} p^*(x)\log p_\theta(x)\mathrm{d}x$$

- したがって、上記の最小化は下記で書き換えられる

$$\min_\theta \mathrm{KL}(p^*(x)||p_\theta(x)) = \max_\theta \int_{-\infty}^{\infty} p^*(x)\log p_\theta(x)\mathrm{d}x \simeq \max_\theta \frac{1}{N}\sum_{n=1}^{N}\log p_\theta(x_n)$$

定数を除き、最小化を最大化に

データセットを用いて近似

最尤推定 (3)

データセット $\mathcal{D} = \{x_1, \ldots, x_N\}$に対して、下記の対数尤度関数を最大化することで、生成モデル $p_\theta(x)$ を学習できる

$$\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(x_n)$$

- この学習方法を最尤推定と呼ぶ
  - 生成モデル以外の機械学習分野でも幅広く用いられている学習方法
  - 本講義で説明するモデルは一つ (GAN) を除いて全て最尤推定で学習する

例: コインの裏表の確率は？ (1)

- 表が1、裏が0として、 $\mathcal{D} = \{1, 0, 1, 0, 1, 1, 0\}$ からコインの裏表の確率を最尤推定したい
- 二値データをモデル化する時は、ベルヌーイ分布が用いられる

$$P_\theta(x = k) = \theta^k (1 - \theta)^{1-k}$$

- ここで、パラメータ θ は表が出る確率を表す
  - パラメータとは、確率分布を特徴付ける変数のことを呼ぶ
  - この場合は1次元の変数だが、後述する深層生成モデルでは、ニューラルネットワークの各層の重みやバイアスがパラメータとなる

例: コインの裏表の確率は？ (2)

- このモデルのパラメータ θ を最尤推定で学習する
  - 目的関数は下記の通り

$$\frac{1}{N}\sum_{n=1}^{N}\log P_\theta(x_n = k) = \frac{1}{N}\sum_{n=1}^{N}\{x_n \log\theta + (1-x_n)\log(1-\theta)\} = \mathcal{L}(\theta)$$

  - この場合は微分を用いて閉形式で最適な $\hat{\theta}$ が求められる

$$\frac{\partial\mathcal{L}(\hat{\theta})}{\partial\hat{\theta}} = \frac{1}{N}\sum_{n=1}^{N}\left\{\frac{x_n}{\hat{\theta}} - \frac{1-x_n}{1-\hat{\theta}}\right\} = 0 \Leftrightarrow \hat{\theta} = \boxed{\frac{1}{N}\sum_{n=1}^{N} x_n}$$

出た値の平均値になっている

例: コインの裏表の確率は？ (3)

$$\frac{1}{N}\sum_{n=1}^{N}\left\{\frac{x_n}{\hat{\theta}}-\frac{1-x_n}{1-\hat{\theta}}\right\}=0$$

$$\frac{1}{\hat{\theta}}\sum_{n=1}^{N}x_n=\frac{1}{1-\hat{\theta}}\sum_{n=1}^{N}(1-x_n)$$

$$\frac{1-\hat{\theta}}{\hat{\theta}}=\frac{N-\sum_{n=1}^{N}x_n}{\sum_{n=1}^{N}x_n}$$

$$\frac{1}{\hat{\theta}}=\frac{N}{\sum_{n=1}^{N}x_n}$$

$$\boxed{\hat{\theta}=\frac{1}{N}\sum_{n=1}^{N}x_n}$$

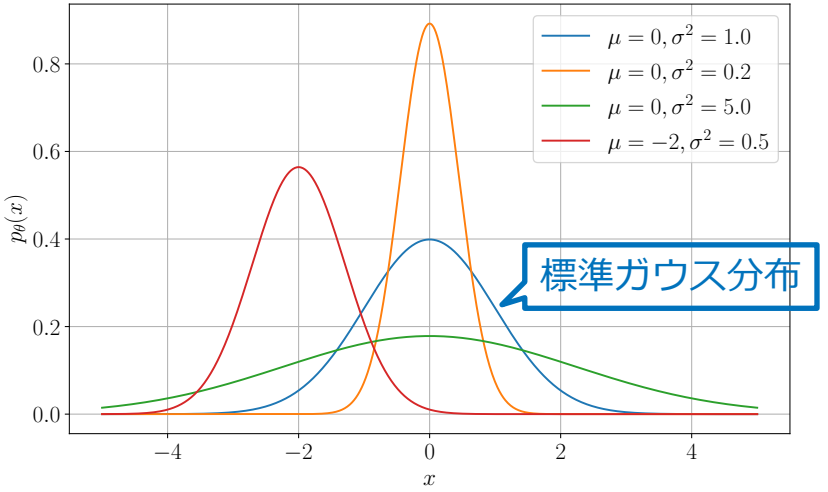
例: コインの裏表の確率は？ (4)

- $\mathcal{D} = \{1, 0, 1, 0, 1, 1, 0\}$ を用いて計算すると、$\hat{\theta} = 4/7$
  - コインの出た値の平均が最適なパラメータとなるため、直感と一致する
  - 偏りが一切ないコインであれば、データセットのサイズを大きくすれば、$\hat{\theta} = 1/2$ に収束する
  - 最尤推定で得られたパラメータを最尤推定量と呼ぶ
- このように、下記の手順を行うことで生成モデルは学習できる
  1. データセット $\mathcal{D} = \{x_1, \ldots, x_N\}$ に対して、適切なモデル $p_\theta(x)$ を選択する
  2. 対数尤度関数 $\frac{1}{N}\sum_{n=1}^{N} \log p_\theta(x)$ を計算する
  3. パラメータ θ に関する微分を用いて最適化する

ガウス分布 (1)

- データ x が連続値の場合は、ガウス分布やガウス混合モデルを用いることが多い
- 1次元のガウス分布は平均 μ と分散 σ^2 をパラメータにもち、下記で表される

$$\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



ガウス分布 (2)

- データセット $\mathcal{D} = \{x_1, \ldots, x_N\}$ が与えられた場合、ガウス分布の各パラメータの最尤推定量 $\hat{\mu}$ と $\hat{\sigma}^2$ は下記で計算できる
  - ベルヌーイ分布の場合と同様に、微分を用いて閉形式で求められる

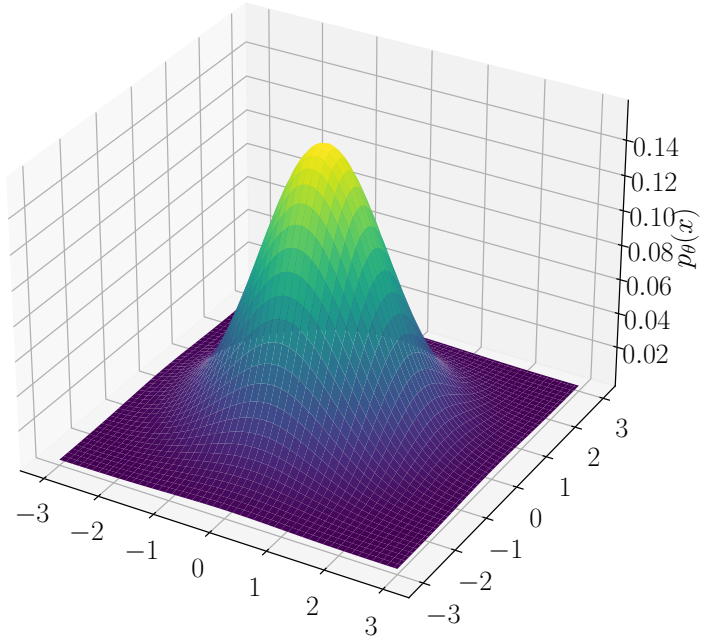
$$\hat{\mu} = \underbrace{\frac{1}{N}\sum_{n=1}^{N} x_n}_{\text{データの平均}}, \quad \hat{\sigma}^2 = \underbrace{\frac{1}{N}\sum_{n=1}^{N}(x_n - \hat{\mu})^2}_{\text{データの分散}}$$

- 様々な好ましい性質 (計算のしやすさ等) があるため、連続値データをモデル化する時に良く用いられる
  - 後述する深層生成モデルは全てガウス分布を用いる

ガウス分布 (3)

- D_x 次元のガウス分布は、平均 μ と共分散行列 $\sum$ をパラメータにもち、下記で表される

$$\mathcal{N}(x;\mu,\Sigma) = \frac{\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))}{\sqrt{(2\pi)^{D_x}|\Sigma|}}$$



ChatGPTを使ってプロットしてみよう

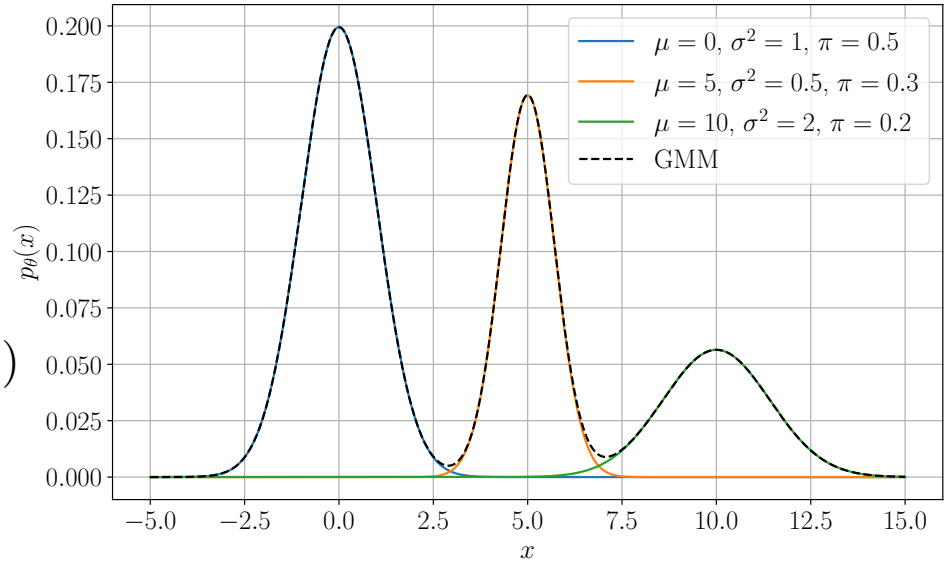
> 2次元のガウス分布を3Dプロットするコードを Pythonで書いてください

ガウス混合モデル (1)

- ガウス分布は単峰の分布のため、多峰の分布は表現できない
- そこで、ガウス分布を K 個混合したガウス混合モデルを考える
  - 積分して1になるように、重み $\pi_k \in [0,1]$ を係数としてかける

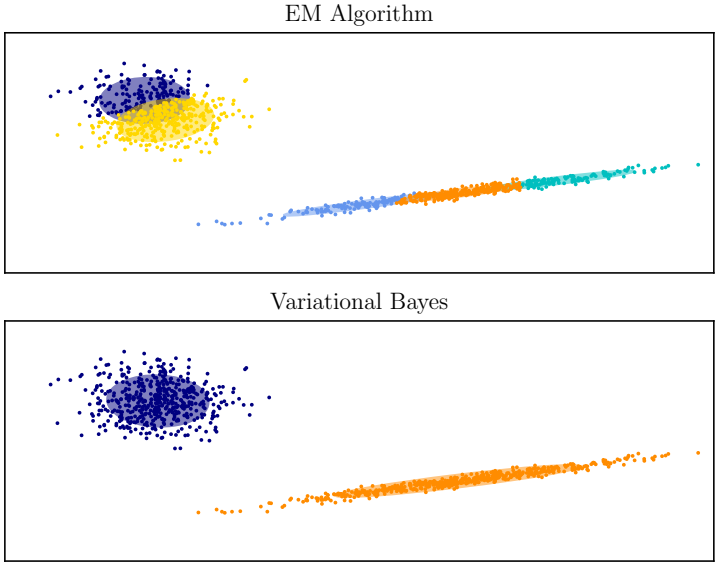
$$p_\theta(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \sigma_k^2)$$

$$\theta = (\pi_1, \ldots, \pi_K, \mu_1, \ldots, \mu_K, \sigma_1^2, \ldots, \sigma_K^2)$$



ガウス混合モデル (2)

- この場合のパラメータ θ は、閉形式で求めることができない
  - EMアルゴリズム [A1] や変分ベイズ [A2] を用いて最適化する (省略)
  - 混合数 K の決定が難しいため、変分ベイズを用いるのがおすすめ [A3]



$K = 5$ の場合

EMアルゴリズムの場合、クラスタを無理やり分割してしまう

変分ベイズの場合、適切なクラスタ数を選択することができる

まとめ: 生成モデルの基礎

- 従来の生成モデルは、データに対して適切なモデルを選択し、最尤推定を行う
  - 例えばコインのような二値データであれば、ベルヌーイ分布が最適
  - 連続値データならガウス分布やガウス混合モデルを用いることが多い
  - 理想的には、$K = \infty$ のガウス混合モデルで何でも表現できるはず
- では、より複雑なデータに対して適切なモデルはあるのか？
  - 画像のような複雑なデータになると、ガウス混合モデルでは対応できない
  - 深層学習の登場以降、生成モデルは飛躍的に発展し、画像のような複雑なデータのモデリングが可能になった
  - 以降では、深層学習を用いた生成モデルを紹介する

参考文献 A

1. "Expectation–maximization algorithm", https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm
2. Blei, David M., and Michael I. Jordan. "Variational inference for Dirichlet process mixtures." (2006): 121-143.
3. "2.1. Gaussian mixture models", https://scikit-learn.org/stable/modules/mixture.html

B. 深層生成モデル (70分)

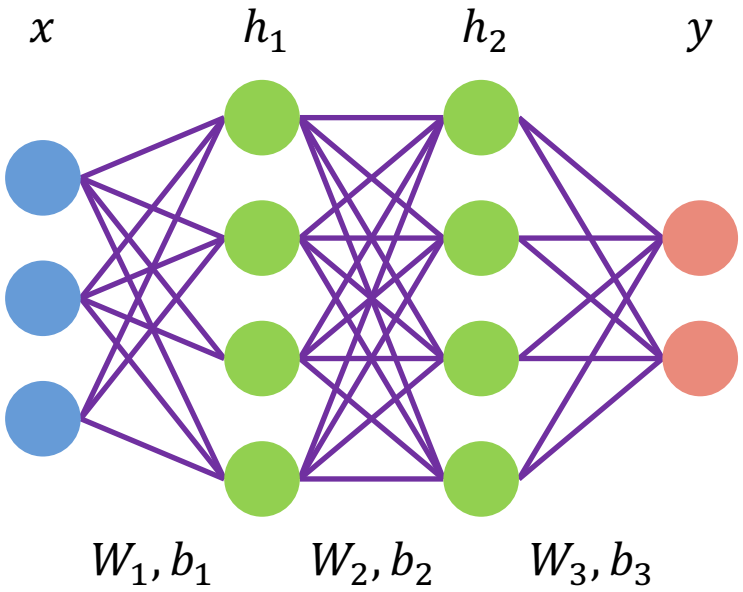
深層生成モデルについて

- 深層学習を用いて生成モデルを学習する手法の総称
- ガウス混合モデルなどの従来手法と比べ、画像のような複雑で高次元のデータの分布を学習することが可能となった
- 本講義ではまず深層学習とその最適化について説明し、続いて下記の4つのモデルを説明する
  - Variational Autoencoder (VAE) [B1]
  - Generative Adversarial Network (GAN) [B2]
  - Flow-based Model (Flow) [B3]
  - Diffusion Model (Diffusion) [B4]

Diffusionによる生成例 [B4]

深層学習について

- 任意の関数 $y = f_\theta(x)$ を近似するための手法
- 例えば、隠れ層が2層のニューラルネットワーク (NN) は下記



活性化関数 (シグモイド関数など)

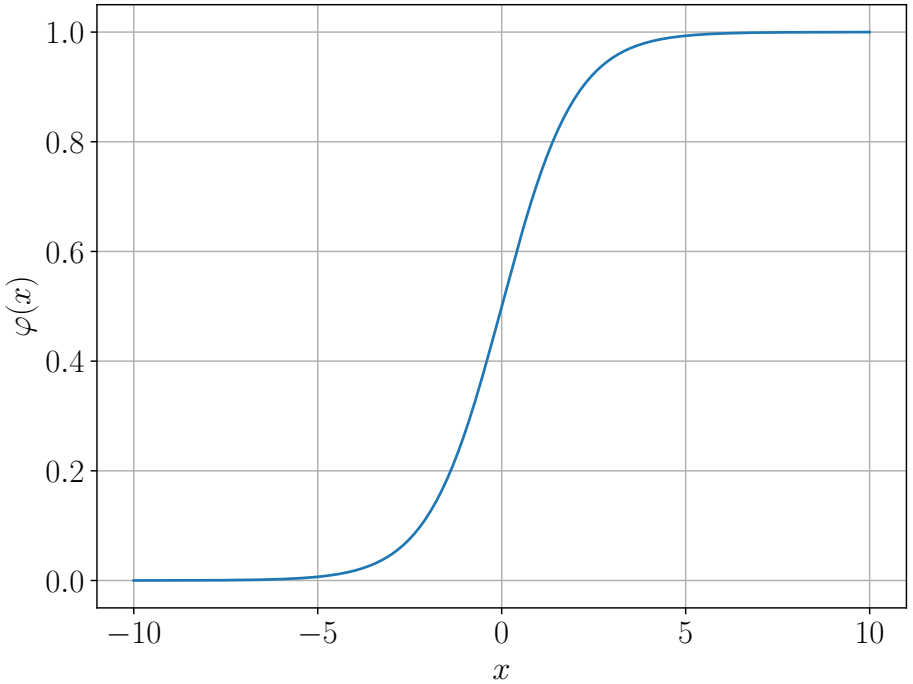
$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$y = W_3 h_2 + b_3$$

ここで、パラメータは下記

$$\theta = (W_1, W_2, W_3, b_1, b_2, b_3)$$

補足: シグモイド関数

[0,1] でバウンドされた非線形関数



確率的勾配法について

- NNは閉形式で最適なパラメータを求めることができないため、確率的勾配法 (stochastic gradient descent, SGD) を用いて最適なパラメータを求める
  1. データセット $\mathcal{D}$ からランダムにミニバッチ (部分集合) $\mathcal{B}$ を選ぶ
  2. ミニバッチ $\mathcal{B}$ に対し、損失関数 $\mathcal{L}(\theta;\mathcal{B})$ の値を計算する
  3. 損失関数の微分 $\nabla_\theta \mathcal{L}(\theta;\mathcal{B})$ を用いて、パラメータ θ を下記で更新する
  4. 上記を収束するまで繰り返す

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta;\mathcal{B})$$

更新率　例: 負の対数尤度関数

Variational Autoencoder (1)

- Variational Autoencoder (VAE) [B1] は、データ x の確率 $p_\theta(x)$ を、低次元の潜在変数 z を用いて下記で推定する

$$p_\theta(x) = \int p_\theta(x|z)p(z)\mathrm{d}z$$

デコーダ　事前分布

$p(x) = \int p(x|z)p(z)\mathrm{d}z$ この部分をモデル化

- x が連続値の時、デコーダと事前分布は下記で定義される

$$p_\theta(x|z) = \mathcal{N}(x; \mu_\theta(z), \sigma^2_\theta(z)), \quad p(z) = \mathcal{N}(z; 0, I)$$

ガウス分布の平均と分散を推定するNN　　標準ガウス分布

Variational Autoencoder (2)

- VAEは無限個のガウス混合モデルとみなせる
  - 事前分布 $p(z)$ からのサンプル $\tilde{z}_\ell$ ごとにガウス分布が決まる
  - 積分は $L \to \infty$ のため、無限個のガウス混合モデルとみなせる

$$p_\theta(x) \simeq \frac{1}{L}\sum_{\ell=1}^{L}\mathcal{N}(x;\mu_\theta(\tilde{z}_\ell),\sigma_\theta^2(\tilde{z}_\ell))$$

- この対数尤度を計算したいが、積分を含むため計算が難しい
  - 正確に計算するためには大量に $\tilde{z}_\ell$ をサンプルする必要があり、計算効率が非常に悪い

Variational Autoencoder (3)

- そこで、対数尤度関数の代わりに下記の変分下界を最大化する

$$\log p_\theta(x) = \log \int p_\theta(x|z)p(z)\mathrm{d}z$$

$$= \log \int q_\phi(z|x)\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}\mathrm{d}z$$

$$\geq \int q_\phi(z|x)\log\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}\mathrm{d}z$$

$$= \int q_\phi(z|x)\log p_\theta(x|z)\mathrm{d}z + \int q_\phi(z|x)\log\frac{p(z)}{q_\phi(z|x)}\mathrm{d}z$$

$$\simeq \frac{1}{L}\sum_{\ell=1}^{L}\log p_\theta(x|z_\ell) - \mathrm{KL}(q_\phi(z|x)||p(z))$$

$$\equiv \mathcal{L}_{\mathrm{VAE}}(x;\theta,\phi)$$

エンコーダ $q_\phi(z|x)$ による importance sampling

Jensenの不等式: 対数と和の交換

Reparameterization Trick (後述)

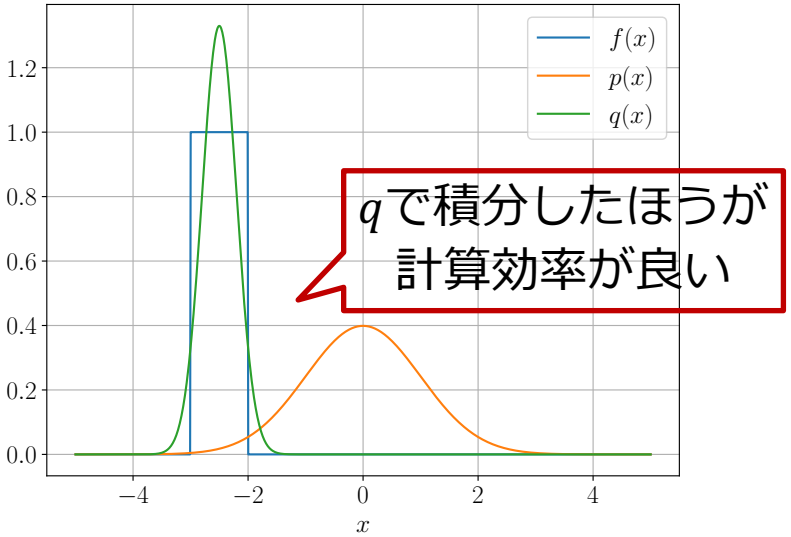
ガウス分布同士だと閉形式で計算可能

$z_\ell \sim q_\phi(z|x)$

補足: importance sampling

- 分布 $p(x)$ による関数 $f(x)$ の期待値を計算したい時、提案分布 $q(x)$ を用いることで、計算効率を上げる手法
  - 適切な $q(x)$ を選択することで、計算効率が大幅に向上する

$$
\begin{aligned}
\mathbb{E}_{p(x)}\left[f(x)\right] &= \int p(x) f(x) \mathrm{d}x \\
&= \int q(x) \left(\frac{p(x)}{q(x)} f(x) \right) \mathrm{d}x \\
&\simeq \frac{1}{L} \sum_{\ell=1}^{L} \frac{p(x_\ell)}{q(x_\ell)} f(x_\ell) \\
x_\ell &\sim q(x)
\end{aligned}
$$



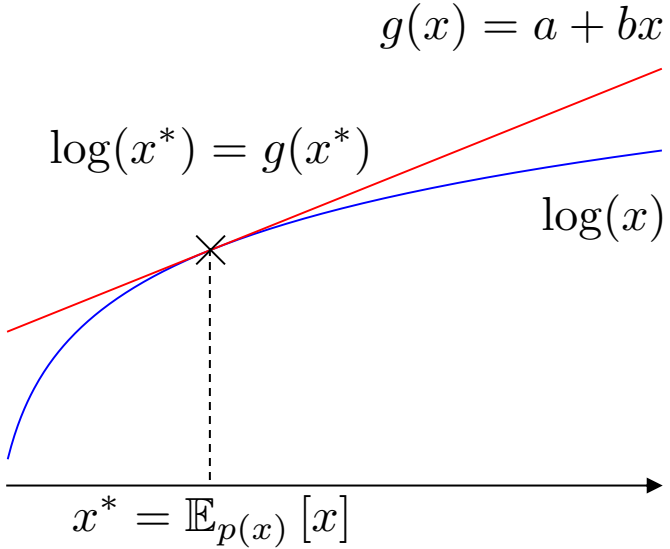
補足: Jensenの不等式

- 分布 $p(x)$ による $\log(x)$ の期待値は下記の上界を持つ:

$$\mathbb{E}_{p(x)}\left[\log(x)\right] \leq \log(\mathbb{E}_{p(x)}\left[x\right])$$

証明: $\log(x)$ の $x^* = \mathbb{E}_{p(x)}[x]$ における接線 $g(x) = a + bx$ を用いて下記が成立[1]

$$\begin{aligned}\mathbb{E}_{p(x)}\left[\log(x)\right] &\leq \mathbb{E}_{p(x)}\left[a + bx\right] \\ &= a + b\mathbb{E}_{p(x)}\left[x\right] \\ &= g(\mathbb{E}_{p(x)}\left[x\right]) \\ &= \log(\mathbb{E}_{p(x)}\left[x\right])\end{aligned}$$



1. https://x.com/daiti_m/status/1798336240858849432

Variational Autoencoder (4)

- エンコーダはガウス分布でモデル化される

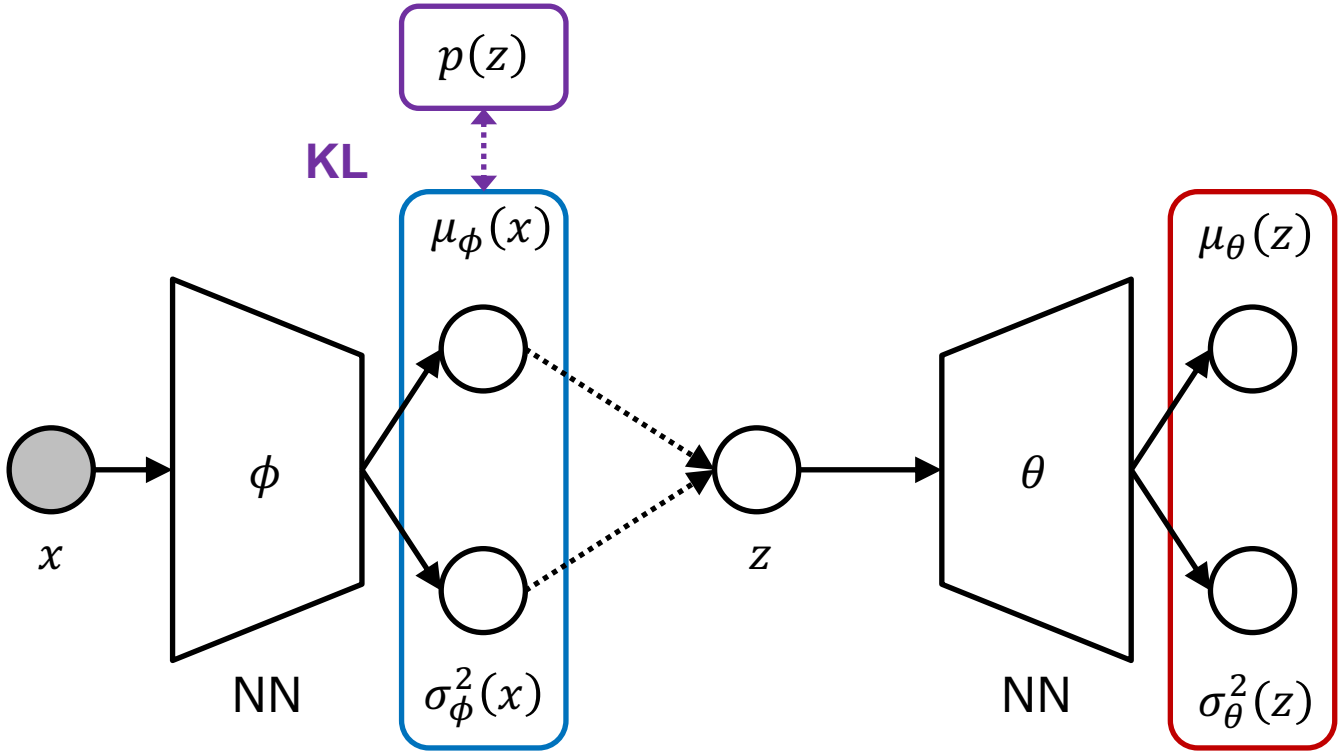
$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma^2_\phi(x))$$

- この時、エンコーダからのサンプルは下記で計算できる
  - Reparameterization Trickと呼ばれる手法 [B1]
  - エンコーダを勾配法で最適化するために必要

$$z_\ell = \mu_\phi(x) + \epsilon_\ell \odot \sigma^2_\phi(x)$$

$$\epsilon_\ell \sim \mathcal{N}(\epsilon; 0, I)$$

Variational Autoencoder (5)



$$q_\phi(z|x) = \mathcal{N}\left(z; \mu_\phi(x), \sigma^2_\phi(x)\right)$$

$$p_\theta(x|z) = \mathcal{N}\left(x; \mu_\theta(z), \sigma^2_\theta(z)\right)$$

Variational Autoencoder (6)

データセット $\mathcal{D} = \{x_1, \ldots, x_N\}$に対して、
下記の変分下界を最大化することでVAEを学習できる

$$\max_{\theta,\phi} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\mathrm{VAE}}(x_n; \theta, \phi)$$

ガウス分布同士だと
閉形式で計算可能

$$\mathcal{L}_{\mathrm{VAE}}(x; \theta, \phi) \equiv \frac{1}{L} \sum_{\ell=1}^{L} \log p_\theta(x|z_\ell) - \boxed{\mathrm{KL}(q_\phi(z|x)||p(z))} \leq \log p_\theta(x)$$

- 深層生成モデルは対数尤度の計算が難しい
- VAEのように、いかに効率的に対数尤度を計算するかが重要

Variational Autoencoder (7)

VAEで生成したMNIST (手書き数字) の画像 (2013年時点) [B1]

生成手順:

1. 事前分布 $p(z)$ から z_ℓ を生成
2. デコーダ $p_\theta(x|z_\ell)$ からデータを生成する

$$p_\theta(x|z_\ell) = \mathcal{N}(x; \mu_\theta(z_\ell), \sigma^2_\theta(z_\ell))$$

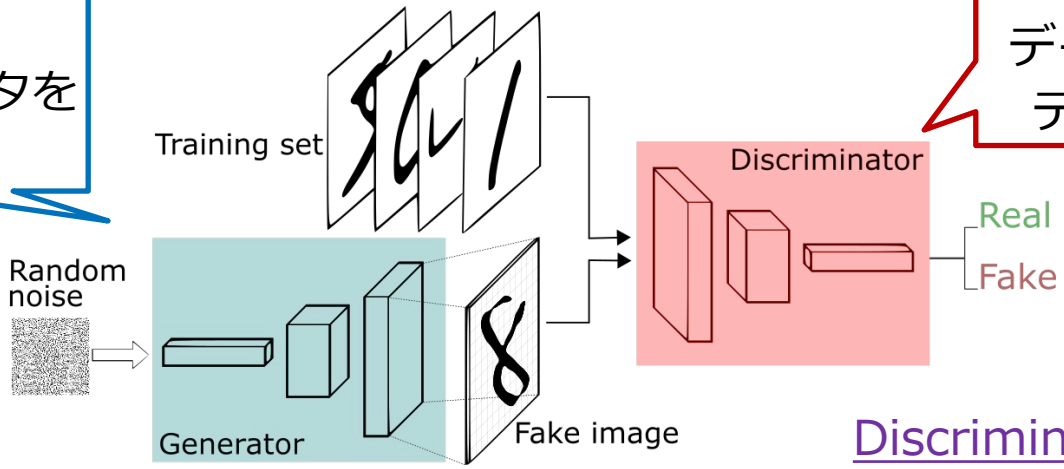
NOTE: VAEは学習しやすいが、他のモデルと比べて生成品質は低い

Generative Adversarial Network (1)

- Generative Adversarial Network (GAN) [B2] は、2つのニューラルネットワーク (Generator と Discriminator) を互いに競わせることで、生成モデルの学習を行う

Generator:
ノイズからデータを生成する

Discriminator:
データと生成されたデータを識別する



Discriminatorを騙すようにGeneratorを学習する

https://sthalles.github.io/intro-to-gans/

Generative Adversarial Network (2)

- 真の分布を $p^*(x)$、低次元の標準ガウス分布を $p(z)$ とする
- また、Generator を $G(z)$、Discriminator を $D(x)$ とする
- この時、G と D は下記の目的関数の最適化で学習できる
  - D に関する最大化と G に関する最小化を交互に行う

$$\min_G \max_D V(D,G) = \mathbb{E}_{p^*(x)}\left[\log D(x)\right] + \mathbb{E}_{p(z)}\left[\log(1 - D(G(z)))\right]$$

2クラス分類の目的関数

データ x と生成データ $G(z)$ を識別できるように学習

D を騙すように G を学習

Generative Adversarial Network (3)

- G が生成するデータの分布を $p_G(x)$ とした時、最適な $D(x)$ は下記で与えられる:

$$D^*(x) = \frac{p^*(x)}{p^*(x) + p_G(x)}$$

- この時、目的関数 $V(D^*, G)$ は、$p_G(x)$ と $p^*(x)$ のジェンセン・シャノン情報量となる

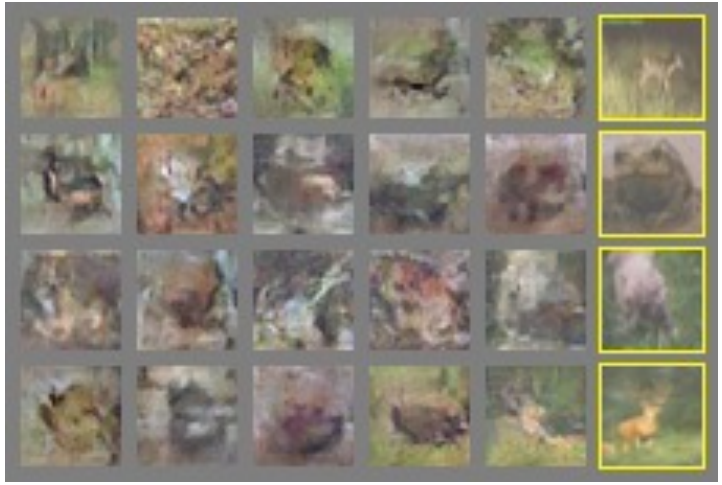
$p_G(x)$ を $p^*(x)$ に近づけるよう学習している

$$\mathrm{JS}(p^*(x)||p_G(x)) = \frac{1}{2}\left\{\mathrm{KL}\left(p^*(x)||m(x)\right) + \mathrm{KL}\left(p_G(x)||m(x)\right)\right\}$$

$$m(x) = \frac{p^*(x) + p_G(x)}{2}$$

Generative Adversarial Network (4)

GANで生成したMNISTとCIFAR10 (2014年時点) [B2]



NOTE:

- GANはデータの生成はできるが、データの確率は推定できない
- また、生成品質は高いが、学習が不安定という問題がある

Flow-based Model (1)

- VAEとGANにはそれぞれ下記のような問題がある
  - VAEは変分下界を計算するため、厳密な確率を計算できない
  - GANは生成品質は高いが、データの確率を計算できない
- そのため、品質が高く、厳密に確率を計算できる生成モデルの需要が高まっていた
- Flow-based Model [B3] は、データ x と同次元のガウス分布 $p(z)$ を用意し、全単射関数 $f_\theta(x)$ を用いて分布を推定する

$$z = f_\theta(x), \quad x = f_\theta^{-1}(z)$$

Flow-based Model (2)

- 全単射 $z = f_\theta(x)$ と $p(z)$ を用いて、確率 $p_\theta(x)$ は下記で書ける

$$\log p_\theta(x) = \log p(f_\theta(x)) + \log \underbrace{\left| \det \left(\frac{\partial f_\theta}{\partial x} \right) \right|}_{\text{ヤコビアン}}$$

ヤコビアン
(ヤコビ行列の行列式)

- ヤコビアンが計算できれば最尤推定が可能
  - 一般的な関数のヤコビアンを計算するのは難しい
  - ヤコビアンが計算しやすく、表現力の高い関数を考えるのがFlowの研究

Flow-based Model (3)

- 全単射の例として、下記のCoupling Layers [B3,B5] を考える

$$z_{1:d} = x_{1:d}$$

データ x を次元 d で２つに区切る

$$z_{d+1:D_x} = x_{d+1:D_x} \odot \exp(s_\theta(x_{1:d})) + t_\theta(x_{1:d})$$

$\mathbb{R}^d \to \mathbb{R}^d$ の任意NN

- この時、ヤコビ行列は下記で計算できる
  - 下三角行列となるので、行列式が簡単に計算できる

$$\begin{bmatrix} \exp(s_\theta(x_{1:d}))_1 & & 0 \\ & \ddots & \\ 0 & & \exp(s_\theta(x_{1:d}))_d \end{bmatrix}$$

$$\frac{\partial z}{\partial x} = \begin{bmatrix} \frac{\partial z_{1:d}}{\partial x_{1:d}} & \frac{\partial z_{1:d}}{\partial x_{d+1:D_x}} \\ \frac{\partial z_{d+1:D_x}}{\partial x_{1:d}} & \frac{\partial z_{d+1:D_x}}{\partial x_{d+1:D_x}} \end{bmatrix} = \begin{bmatrix} I_d & 0 \\ \frac{\partial z_{d+1:D_x}}{\partial x_{1:d}} & \mathrm{diag}(\exp(s_\theta(x_{1:d}))) \end{bmatrix}$$

この行列式の計算だけでOK

Flow-based Model (4)

- よって、$\log p_\theta(x)$ は下記で計算でき、最尤推定が可能

$$\begin{aligned}\log p_\theta(x) &= \log p(f_\theta(x)) + \log\left|\det\left(\mathrm{diag}(\exp(s_\theta(x_{1:d})))\right)\right| \\ &= \log p(f_\theta(x)) + \log(\exp(\mathrm{tr}(\mathrm{diag}(s_\theta(x_{1:d}))))) \\ &= \log p(f_\theta(x)) + \sum_j s_\theta(x_{1:d})_j\end{aligned}$$

$$\det(\exp(A)) = \exp(\mathrm{tr}(A))$$

- データ生成は、逆関数を用いてノイズ $z \sim p(z)$ から変換する

$$\begin{aligned}x_{1:d} &= z_{1:d} \\ x_{d+1:D_x} &= (z_{d+1:D_x} - t_\theta(x_{1:d})) \odot \exp(-s_\theta(x_{1:d}))\end{aligned}$$

Flow-based Model (5)

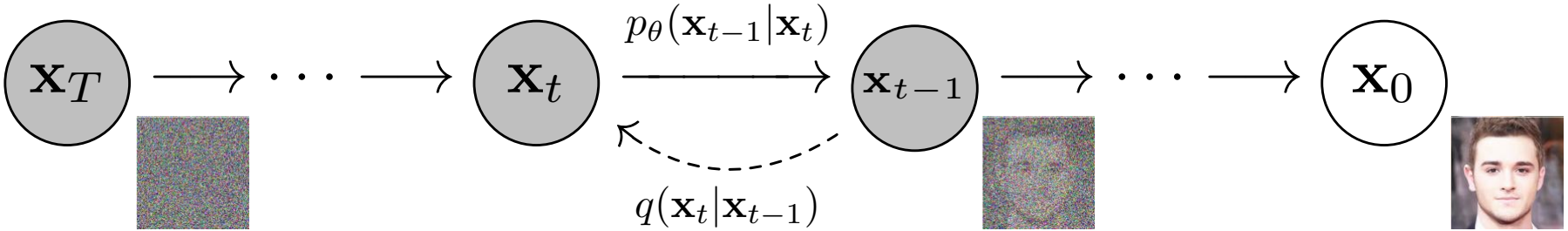
Real NVPで生成したCelebAとCIFAR10 (2017年時点) [B5]

NOTE:

- FlowはVAEよりも性能が高く、厳密に確率を求められるので注目されていた (GAN or Flow という流れだった)
- 欠点として、全単射1つだけでは表現能力が低く、多段にする必要があるため、計算量が膨大になることがあげられる

Diffusion Model (1)

- Diffusion Model [B4] は、T ステップかけて元のデータ x_0 にノイズを徐々に加え、ガウス分布に従うノイズに変換する手法
  - ノイズを加える順方向の過程を拡散 (Diffusion) と呼ぶ
- その逆変換を行うことで、ノイズから元のデータを復元できる
  - ノイズを除去する逆方向の過程をノイズ除去 (Denoising) と呼ぶ



Diffusion Model (2)

- データ x_0 の確率 $p_\theta(x_0)$ を下記で推定する

$$p_\theta(x_0) = \int p_\theta(x_{0:T}) \mathrm{d}x_{1:T}$$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

平均と分散を推定するNN

- また、拡散されたデータの確率を下記で推定する

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t I)$$

β_t: ハイパーパラメータ

Diffusion Model (3)

- この時、変分下界は下記で求められる

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T}) \mathrm{d}x_{1:T}$$

$$= \log \mathbb{E}_{q(x_{1:T}|x_0)} \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

$$\geq \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log p(x_T) + \sum_{t \geq 1} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right]$$

$$\equiv \mathcal{L}_{\mathrm{DDPM}}(x_0; \theta)$$

$q(x_{1:T}|x_0)$ による importance sampling

Jensenの不等式: 対数と和 (期待値) の交換

Diffusion Model (4)

- この変分下界を式変形すると、下記のように簡略化できる
  - 各時刻 t のデータ x_t に付加されるノイズをNN ϵ_θ で推定できるよう学習

$$-\mathcal{L}_{\mathrm{DDPM}}(x_0;\theta) \propto \mathbb{E}_{\mathcal{U}(t),p(\epsilon)}\left[\left\|\epsilon-\epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon,t\right)\right\|^2\right]$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1,\dots,T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},t)\right\|^2$$
6: **until** converged

$$\alpha_t = 1-\beta_t$$

$$\bar{\alpha}_t = \prod_{s=1}^{t}\alpha_s$$

Diffusion Model (5)

- データ x_0 の生成は下記の手順で行われる

Algorithm 2 Sampling

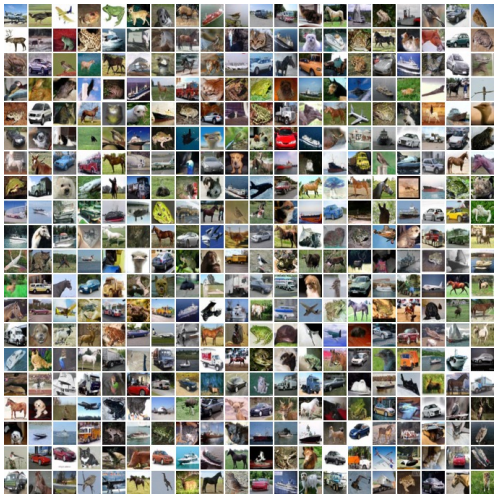
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

ガウス分布からノイズ x_T を生成

T ステップかけてノイズを除去

Diffusion Model (6)

Diffusion Modelで生成したCelebAとCIFAR10 (2020年時点) [B4]



NOTE:
- Diffusionは非常に生成品質が高く、2024年時点でデファクトスタンダード
- 一方で、ステップサイズ T を大きくする必要があり、計算量が大きい

まとめ: 深層生成モデル

- 深層生成モデルは、ガウス分布に従うノイズから、どのような方法でデータを生成するかに焦点を当てている
  - VAE: ノイズごとに条件付き分布を推定し、データを生成
  - GAN: ノイズからデータを直接生成
  - Flow: ノイズとデータの全単射関数を用いて、データを生成
  - Diffusion: ノイズを徐々に除去することで、データを生成
- それぞれのモデルごとに最尤推定する方法を工夫している
  - 変分下界を導出、分類器を騙すように学習、全単射で厳密に計算、など
- 現在主流のStable Diffusion [B6] は VAE + Diffusion

補足: 条件付き分布への拡張

- 条件 c が与えられたもとでのデータ x の分布 $p_\theta(x|c)$ を、条件付き分布と呼ぶ
  - 今まで説明した深層生成モデルは全て条件付き分布に拡張可能
  - 条件 c として、データが属するクラス (手書き数字の生成なら「1」など) や、データを表現するテキスト (例えば「桜が咲き誇る静かな日本庭園にある、アニメ風の静かな鯉の池」) などが考えられる
  - CLIP [B7] と呼ばれるテキスト埋め込みモデルを用いることで、画像生成に適した条件 c が得られる
- 大規模な画像データセットでCLIPを用いて条件付き分布を学習することで、Stable Diffusionのような画像生成が可能となる

参考文献 B

1. Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).
2. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).
3. Dinh, Laurent, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation." arXiv preprint arXiv:1410.8516 (2014).
4. Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." Advances in neural information processing systems 33 (2020): 6840-6851.

参考文献 B

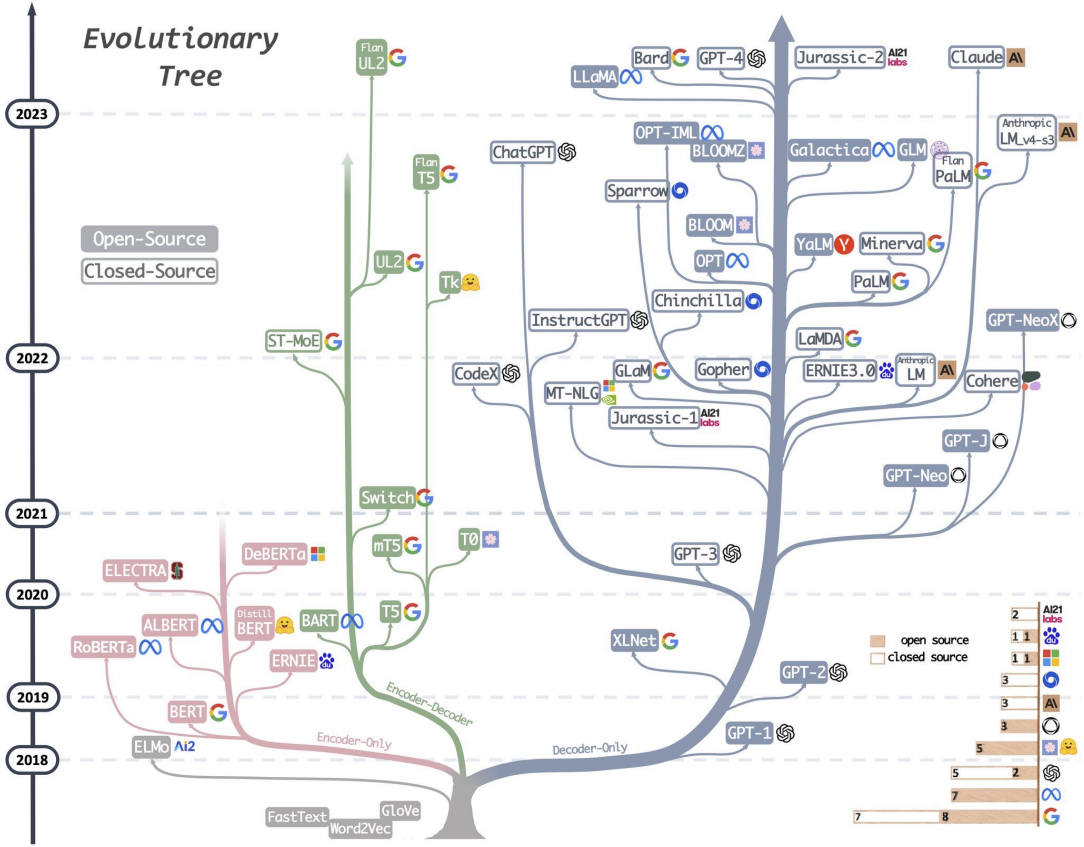
5. Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp." arXiv preprint arXiv:1605.08803 (2016).
6. Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.
7. Radford, Alec, et al. "Learning transferable visual models from natural language supervision." International conference on machine learning. PMLR, 2021.

C. 言語モデル (40分)

言語モデルについて

- 入力 x に対する出力 y の確率 $P_\theta(y|x)$ を推定するモデル
  - 言語は画像と性質が異なるため、深層生成モデルとは異なる発展を遂げた
- 2017年にTransformer [C1] が登場し、飛躍的に進化した
  - 言語モデルの事前学習の効果の確認 [C2] (2018)
    - › 大規模かつ多様なデータセットで言語モデルを事前学習すれば、少量のラベル付きデータによるFine-Tuningで高精度を達成できる
  - 事前学習済のTransformer (BERT/GPT) の登場 (2018)
    - › BERT [C3] はTransformerのエンコーダ部分を事前学習したモデル
    - › GPT [C4] はTransformerのデコーダ部分を事前学習したモデル
  - 以降様々なモデルが登場

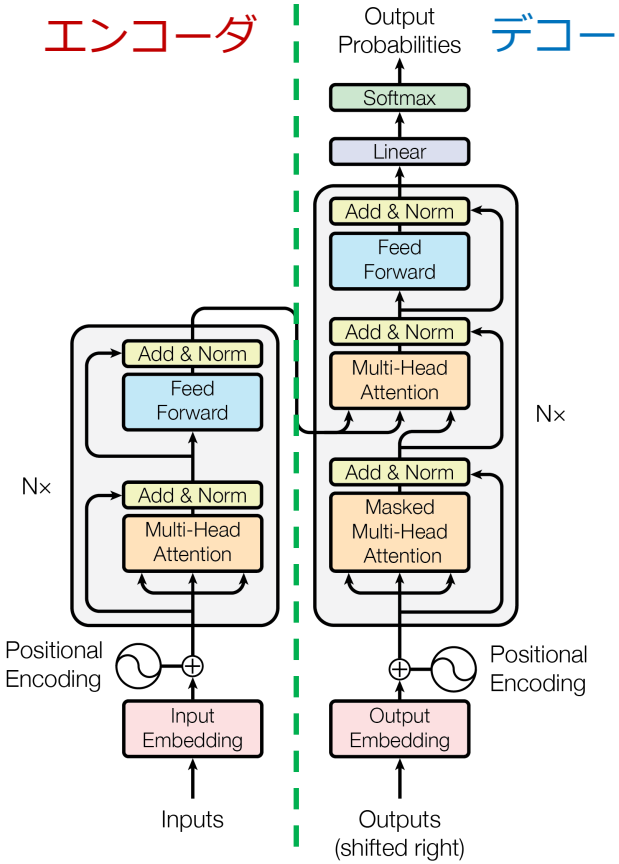
言語モデルの進化 (2017 - 2023)



https://x.com/DrJimFan/status/1651968203701231616

Transformer (1)

- 条件付き分布 $P_\theta(y|x)$ を出力するためのニューラルネットワークの構造
  1. 文章 x, y をベクトルに変換する (それぞれ e_x, e_y とする)
  2. ベクトル e_x をエンコーダに入力
  3. エンコーダの出力とベクトル e_y をデコーダに入力
  4. 条件付き確率 $P_\theta(y|x)$ を計算
- エンコーダとデコーダは、Attentionという共通の構造を用いている



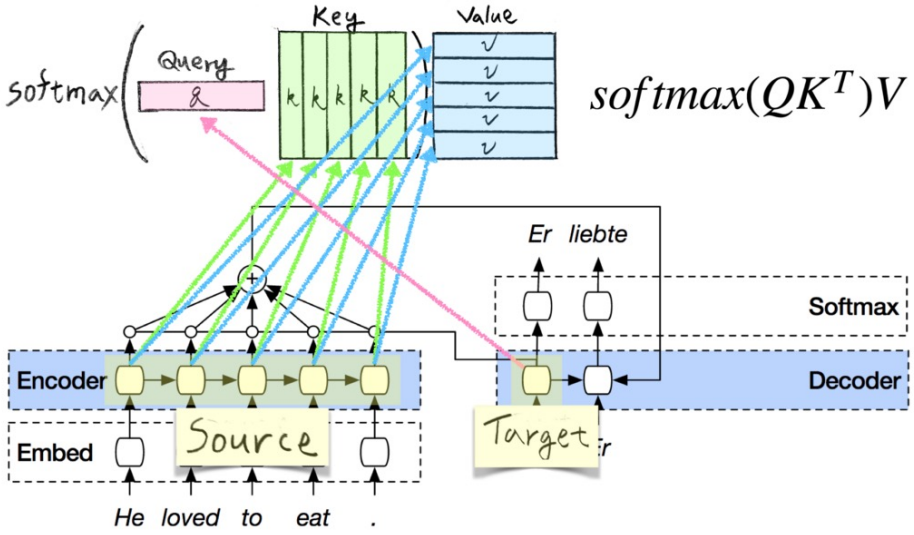
Transformer (2)

- Attentionはクエリ Q、キー K、バリュー V から下記を計算

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

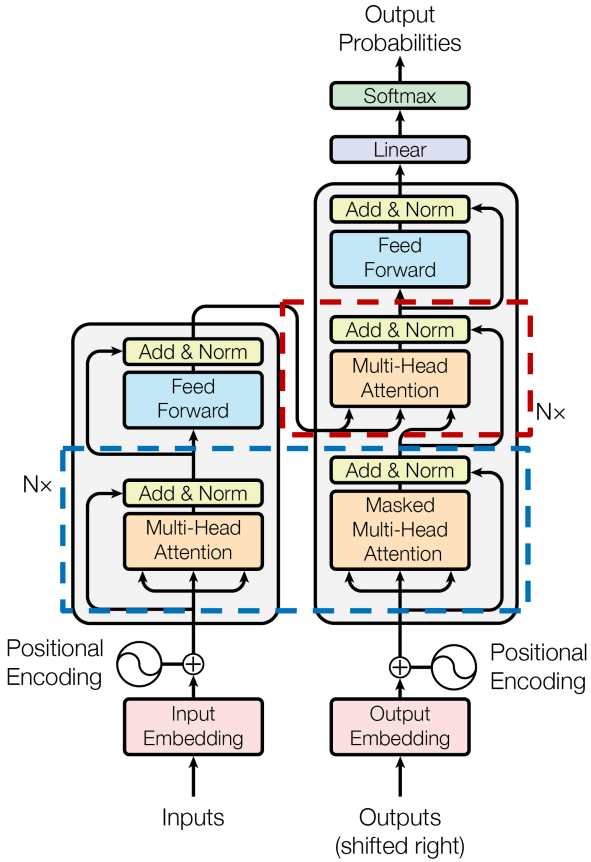
Attentionは辞書オブジェクト [C5]

- Q と K の類似度を内積で計算し、重みとして V に掛けている
- Q と K が似ている部分は V がそのまま出てくる
- 逆に似ていない部分は 0 になる
- これは辞書と同じ機能を持つ

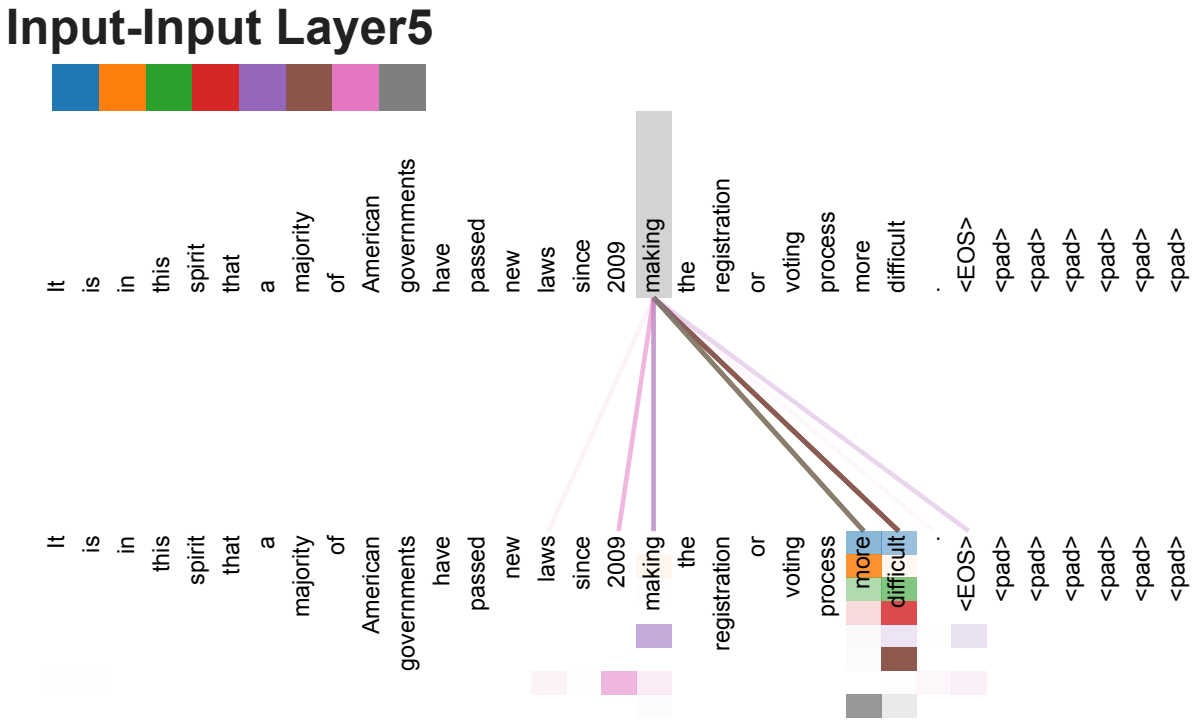


Transformer (3)

- Self-Attention は $Q = K = V$ のAttention
  - 入力内の各要素が他の全ての要素に対してどの程度関連しているかを計算
  - エンコーダとデコーダの初段はSelf-Attention
- Cross-Attention は $K = V$ のAttention
  - Q が K, V とどの程度関連しているかを計算
  - デコーダの二段目以降は Q がデコーダへの入力、K, V がエンコーダの出力のCross-Attention
- Attention を複数のモデルで計算することでさらに性能が上がる (Multi-Head)



Transformer (4)



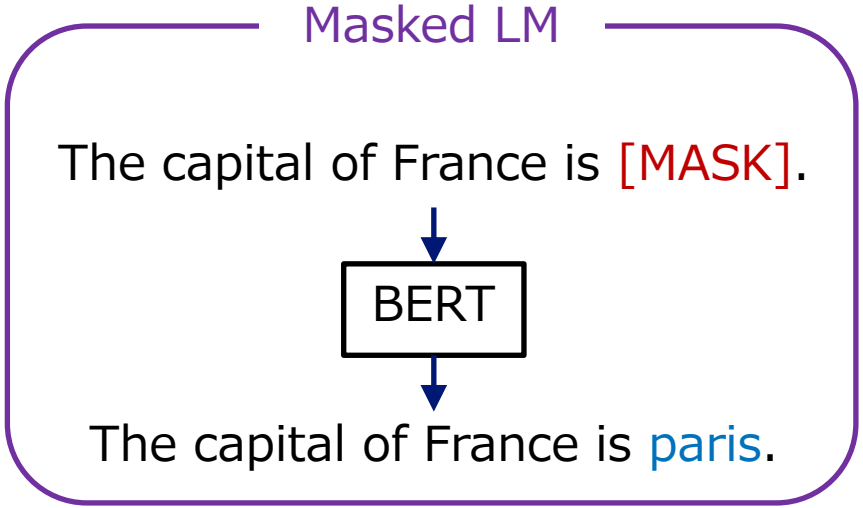
Self-Attentionの例: makingとmore difficultが関連していると判断 [B1]

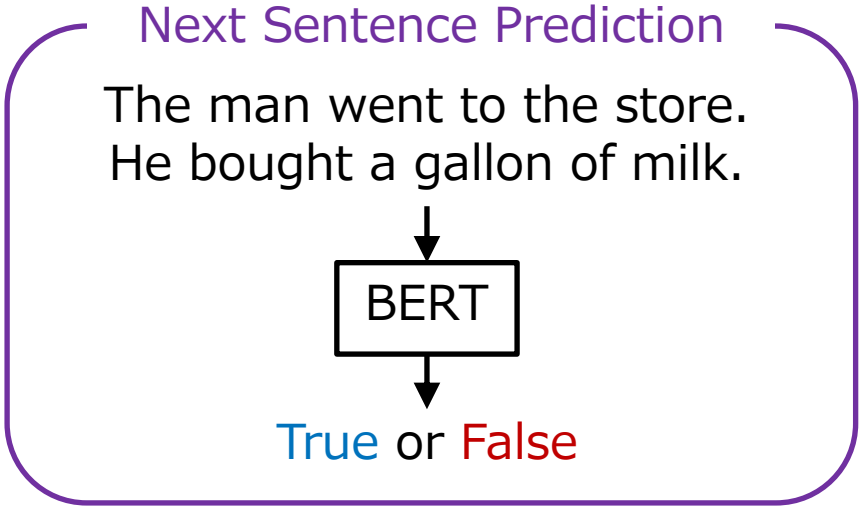
Transformer (5)

- Attentionは従来の言語モデルと比べて、下記の利点がある
  - 依存関係の捉えやすさ: 文字列内の遠く離れた部分の依存関係を効果的に捉えることができる
  - 並列処理のしやすさ: 文字列全体に対して同時に計算できるため、並列に処理が可能であり、従来の言語モデルより高速
- Transformerは2017年の登場時、機械翻訳で当時の最高性能を達成し、言語モデルの基盤となった
  - その後、事前学習の重要性 [C2] や、事前学習済のエンコーダ (BERT) [C3] とデコーダ (GPT) [C4] が登場し、大規模言語モデルへと発展
  - Transformerの詳細は [C5,C6,C7] を参照

BERT: Bidirectional Transformer

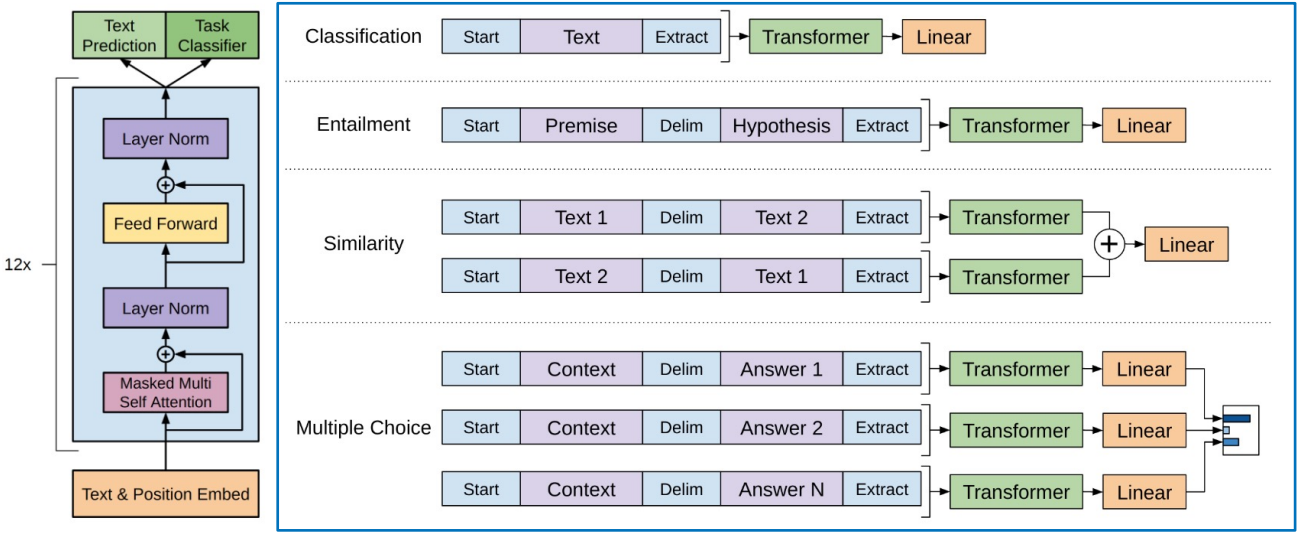
- BERT [C3] は下記の2タスクで事前学習されたエンコーダ
  - Masked LM: マスクされた文章から元の文章を推定する
  - Next Sentence Prediction: 2つの文章が与えられた時、一方の文章が、もう一方の次の文章になっているかどうかを判定する





Generative Pretrained Transformer

- GPT [C4] は事前学習されたデコーダで、大規模言語モデルの基礎となっているモデル (当時はFine-Tuningが前提)
  - GPT-2 [C8] や GPT-3 [C9] を経て、現在は GPT-4o (論文非公開)



これらのタスクでFine-Tuning

まとめ: 言語モデル

- 言語モデルは言語の生成モデル
  - 画像と言語は特徴が異なるため、深層生成モデルとは異なる構造を持つ
- Transformerは、言語の条件付き確率を推定するためのNN
  - Attentionという辞書に似た構造を用いることで、文章内の離れた位置の依存関係を捉えることが可能となり、また並列計算が可能に
  - エンコーダを事前学習したBERTやデコーダを事前学習したGPTは、様々なタスクで高い性能を達成
  - 特にGPTは現在の大規模言語モデルにも採用されている
- 大規模言語モデルについては [C10,C11] を参照
  - 発展が非常に速い分野なので、最新の資料を読むと良いと思います

C+. Hugging Face入門

背景: 簡単に言語モデルを扱いたい

NTT

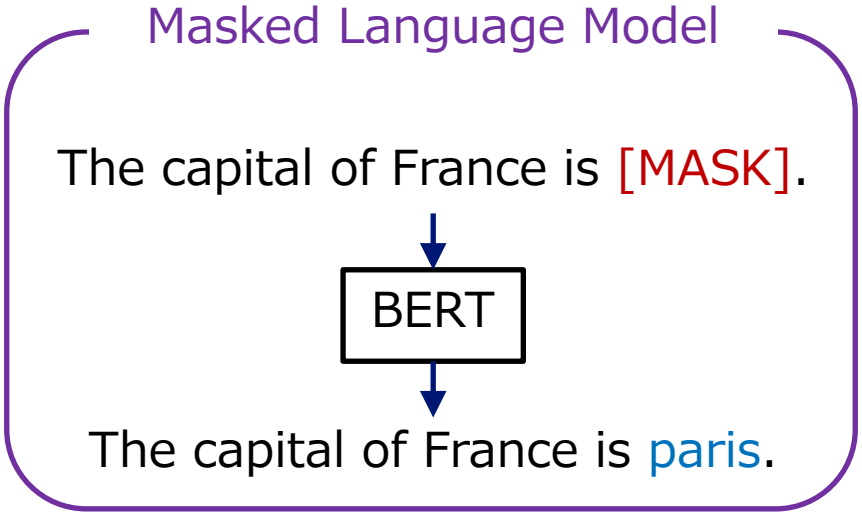
- 従来の言語モデルの利用手順は以下の通り複雑だった
  1. PyTorch/TensorFlowでベースとなるモデルを実装
  2. 配布されている事前学習済モデルの重みを読み込む
  3. 入力の前処理やモデルの出力の後処理を実装
  4. DataLoaderやLoss、Optimizerを実装
- 🤗 Hugging Face はこれらの作業を簡略化するためのシステム
  - 多くのモデルをカバーした、統一されたインターフェイスを提供
  - 主要なフレームワーク (PyTorch/TensorFlow/Jax) のサポート
  - 事前学習済モデルが集約されているモデルハブの提供

Hugging Face について

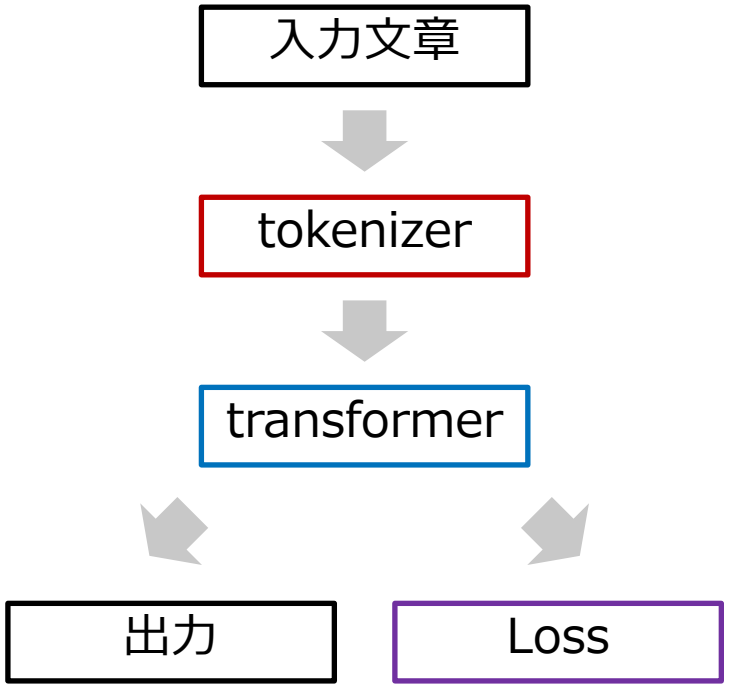
- 🤗 Hugging Faceはライブラリ/エコシステムの総称 [C12]
- Transformer系のライブラリ (インターフェイス) は下記の通り
  - 🤗 Transformers/🤗 Tokenizers (Transformer全般) ← 本日紹介
  - 🤗 Datasets (NLP系のデータセット)
  - 🤗 Evaluates (NLP系の評価指標)
  - 🤗 Accelerate (Transformer高速化)
- また、事前学習済のモデルを共有するモデルハブや、データセットを共有するデータセットハブなどのエコシステムも提供
- 今回のサンプルコードは [C13] で利用可能

最初の一歩: BERTによる文書校正

- BERTを用いて、「私は野菜が空きだ」という入力文章を、「私は野菜が好きだ」という文章に修正する
  - BERTはMasked Language Modelという、マスクされたトークンから元のトークンを推定するタスクで事前学習されており、文書校正は得意



処理の流れ



例: 私は野菜が空きだ

形態素解析 + ベクトル化

Embedding + Encoder or/and Decoder
今回はBERTなのでEncoderのみ

例: 私は野菜が好きだ
ラベルが与えられている場合はLossも出るため、学習に利用可能

サンプルコード

shell

```
$ pip install transformers torch ipadic fugashi
```

python

```python
from transformers import BertJapaneseTokenizer, BertLMHeadModel
model_id = "cl-tohoku/bert-base-japanese-whole-word-masking"
tokenizer = BertJapaneseTokenizer.from_pretrained(model_id)
model = BertLMHeadModel.from_pretrained(model_id)

wrong_text = "私は野菜が空きだ"
wrong_tokens = tokenizer(wrong_text, return_tensors="pt")
logits = model(**wrong_tokens).logits
correct = tokenizer.decode(logits[0][1:-1].argmax(axis=1))
print(correct)
```

shell

```
私 は 野菜 が 好き だ
```

コード解説 (1)

shell

```
$ pip install transformers torch ipadic fugashi
```

必要ライブラリのインストール

transformers	🤗 Hugging Faceのライブラリ
torch	🤗 transformersのbackend
fugashi	日本語形態素解析器 (MeCabのラッパー)
ipadic	MeCabに必要な辞書

コード解説 (2)

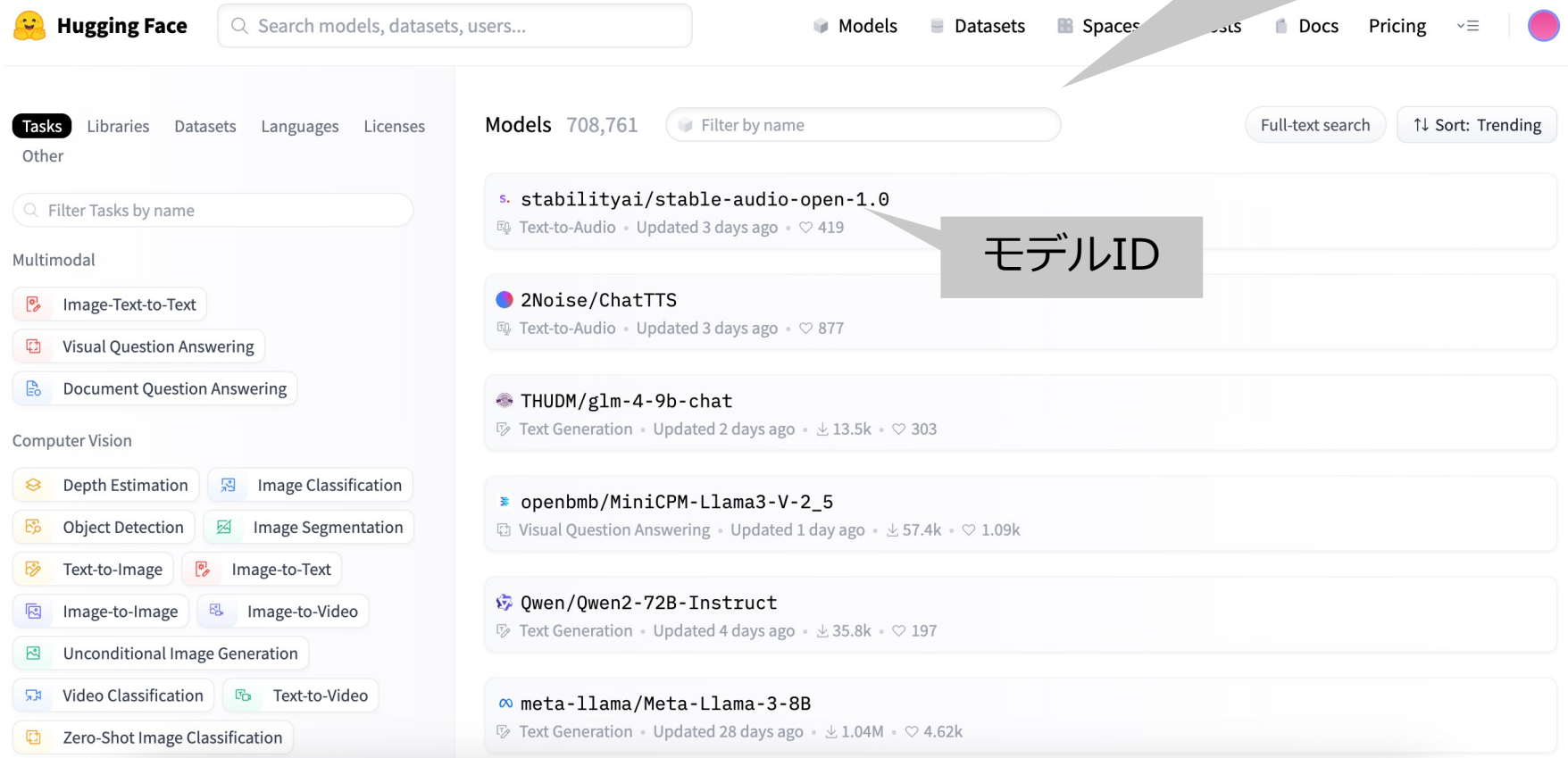
```python
from transformers import BertJapaneseTokenizer, BertLMHeadModel
model_id = "cl-tohoku/bert-base-japanese-whole-word-masking"
tokenizer = BertJapaneseTokenizer.from_pretrained(model_id)
model = BertLMHeadModel.from_pretrained(model_id)
```

事前学習済モデルの読み込み

BertJapaneseTokenizer	BERTの日本語用Tokenizer
BertLMHeadModel	BERTの言語モデル
model_id	モデルハブに登録されているモデルID
from_pretrained	事前学習済のtokenizer/modelの読み込み

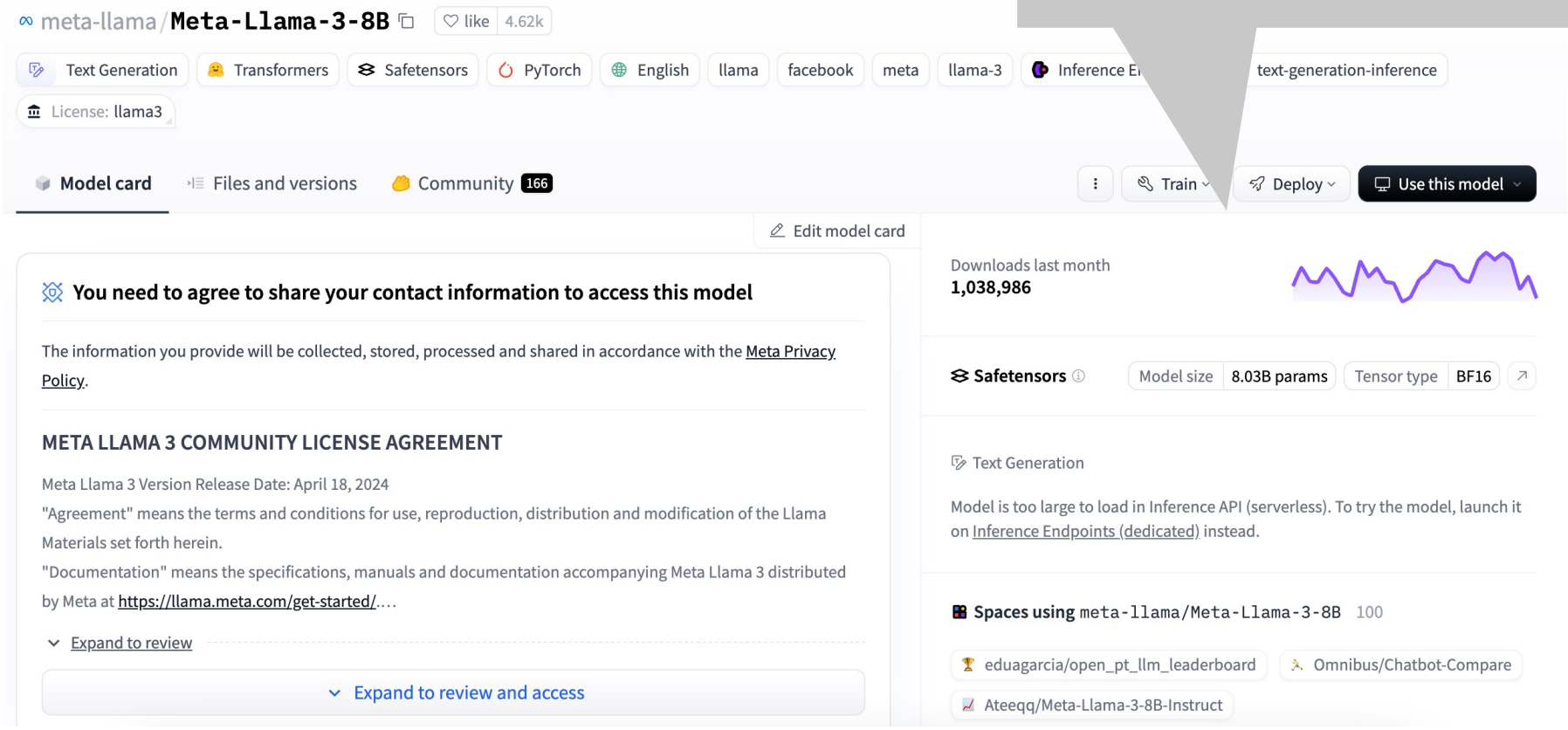
モデルハブ (1)

多くのモデルが集まっており、柔軟な検索が可能



モデルハブ (2)

アーキテクチャや利用した
データセットが一目でわかる



コード解説 (3)

python

```python
wrong_text = "私は野菜が空きだ"
wrong_tokens = tokenizer(wrong_text, return_tensors="pt")
logits = model(**wrong_tokens).logits
correct = tokenizer.decode(logits[0][1:-1].argmax(axis=1))
print(correct)
```

文書校正

2行目	入力文章のトークン化 (=形態素解析+ベクトル化)
3行目	BERTに入力し、出力 (logits) を取得
4行目	出力を文章にデコード

モデルの学習

- すべてのモデルは nn.Module の子クラスなため、PyTorchのOptimizerを使って最適化可能
- model.forward にlabelを与えればlossを計算してくれる

python

```python
label_text = "私は野菜が好きだ"
label_tokens = tokenizer(label_text, return_tensors="pt")
loss = model(**wrong_tokens, labels=label_tokens.input_ids).loss
print(loss)
```

shell

```
tensor(15.4039, grad_fn=<NllLossBackward0>)
```

まとめ: Hugging Face入門

- まとめ
  1. モデルIDを指定し、tokenizerとmodelの事前学習済モデルを読み込む
  2. 入力文章をトークン化し、ラベルとともにモデルに入力
  3. 得られたLossを用いてOptimizerで最適化
- 今回は文書校正を例にしたが、他のタスクにも応用可能
  - テキスト分類/固有表現認識/質問応答/要約/翻訳/テキスト生成/etc…
  - モデルは `nn.Module` の子クラスのため、簡単に手を加えられる
- オープンなLLMもHugging Faceで公開されている
  - 例えばmetaが公開しているllama [C14] など

参考文献C

1. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
2. Howard, Jeremy, and Sebastian Ruder. "Universal language model fine-tuning for text classification." arXiv preprint arXiv:1801.06146 (2018).
3. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
4. Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).

参考文献C

5. "論文解説 Attention Is All You Need (Transformer)", https://deeplearning.hatenablog.com/entry/transformer
6. "Transformer: A Novel Neural Network Architecture for Language Understanding", https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/
7. "The Illustrated Transformer", http://jalammar.github.io/illustrated-transformer/
8. Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI blog 1.8 (2019): 9.

参考文献C

9. Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.
10. "大規模言語モデル", https://speakerdeck.com/chokkan/llm
11. "大規模言語モデルの開発”, https://speakerdeck.com/chokkan/jsai2024-tutorial-llm
12. "Hugging Face", https://huggingface.co/
13. https://gist.github.com/takahashihiroshi/ac26aae8834f4c89d4ae595c53a466dc
14. "Meta Llama", https://huggingface.co/meta-llama

D. 生成モデルの問題点 (10分)

生成AIの不適切な利用

- 生成AIの不適切な利用[1,2]が相次ぎ、「規制を強化すべき」との声が強くなっている

「生成AI」偽情報と規制 “規制強化すべき”61% NHK世論調査

2024年5月3日 21時58分

人工知能「生成AI」について、日本では規制する法律はありませんがインターネット上で偽の動画や画像が問題になるケースが増えています。これについてNHKの憲法に関する世論調査でどう対応すべきか聞いたところ「規制を強化すべき」が61%、「今のままでよい」が8%でした。

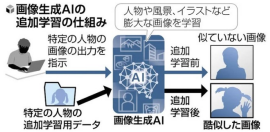
実在児童に酷似した性的画像、生成ＡＩで作成か…ネット上で追加学習用データを売買

2024/06/02 05:00 生成ＡＩ

この記事をスクラップする

画像生成ＡＩ（人工知能）で、実在の児童に酷似した画像を生成できる追加学習用データが、ネット上で売買されていることがわかった。過去に日本で活動した児童タレントの追加データもあり、追加データで作ったとみられる児童タレントに酷似した性的画像が別のサイトで販売されていた。日本の児童買春・児童ポルノ禁止法で規制することは困難とみられ、専門家は法整備の必要性を訴えている。

▸米半導体大手マイクロン、相模原市の研究開発拠点を拡充へ…生成ＡＩや自動運転向けへ投資



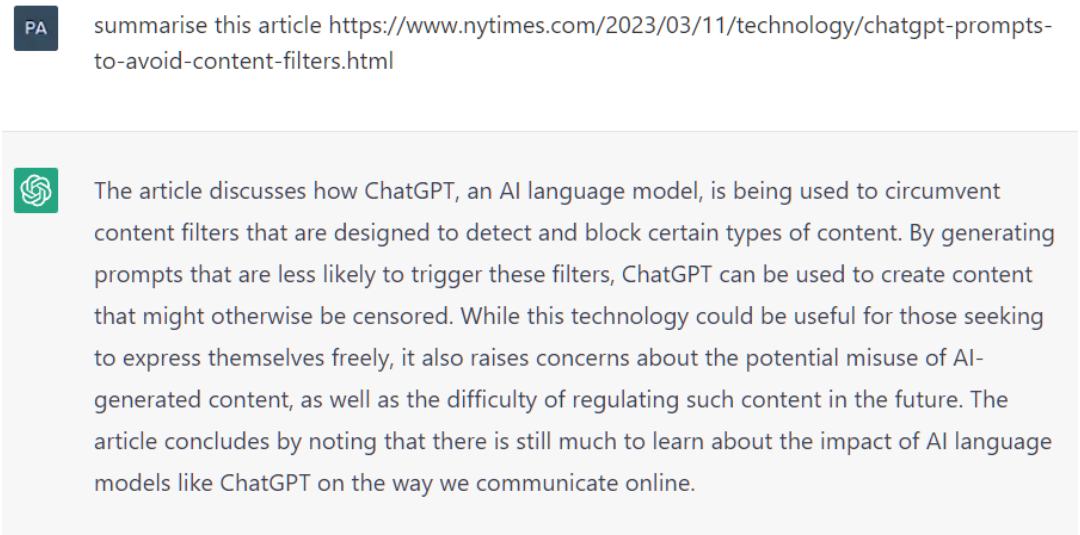
画像生成ＡＩは、膨大な画像データを学習し、文章で指示を受けただけで、写真のような精巧な画像を生成する。特定の人物名を入力するだけでは酷似した画像を生成することは難しいが、一部の画像生成ＡＩは、その人物の画像数十点を基に作った追加データを学習させれば、酷似した画像を生成できる仕組みになっている。追加データは「ＬｏＲＡファイル」などと呼ばれ、こうした手法は「ファインチューニング（微調整）」と呼ばれる。

1. https://www3.nhk.or.jp/news/html/20240503/k10014439661000.html
2. https://www.yomiuri.co.jp/national/20240601-OYT1T50224/

生成AIは誤った回答を生成する

- 生成AIは誤った回答を生成することがあり、ハルシネーションと呼ばれている [D1]

「ニューヨーク・タイムズ」の実在しない記事を要約するChatGPT[1]



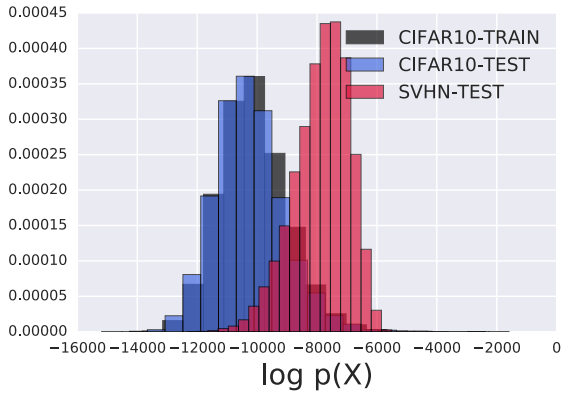
https://upload.wikimedia.org/wikipedia/commons/3/3a/ChatGPT_hallucination.png

生成AIは正しく学習されていない

- 深層生成モデルは、学習データに含まれない「分布外データ」に高い確率を割り当ててしまうことがある [D2]
  - CIFAR10で学習したVAEは、SVHNに高い確率を割り当てる
  - つまり、分布外データを生成する可能性がある

CIFAR10

SVHN



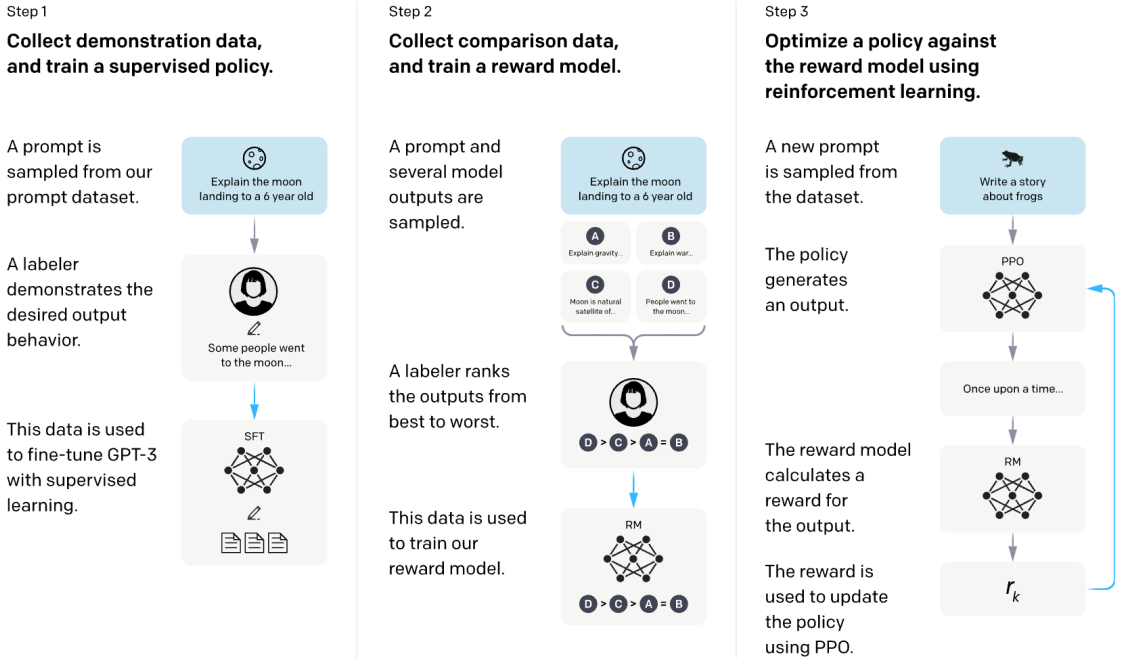
ITmedia: https://www.itmedia.co.jp/

生成AIは人間なしでは不完全

- 生成AIは誤った回答を生成するし、正しく学習されていない
- そもそも生成AIはデータから学習しているだけであって、人間にとって「正しい」かどうかを考えていない
- 人間にとって「正しい」モデルにするためには、人間からのフィードバックが重要
- 最後に、人間からのフィードバックを用いて生成AIを改善する論文を2本紹介する
  1. Reinforcement Learning from Human Feedback (RLHF, 2022) [D3]
  2. Selective Amnesia (2024) [D4]

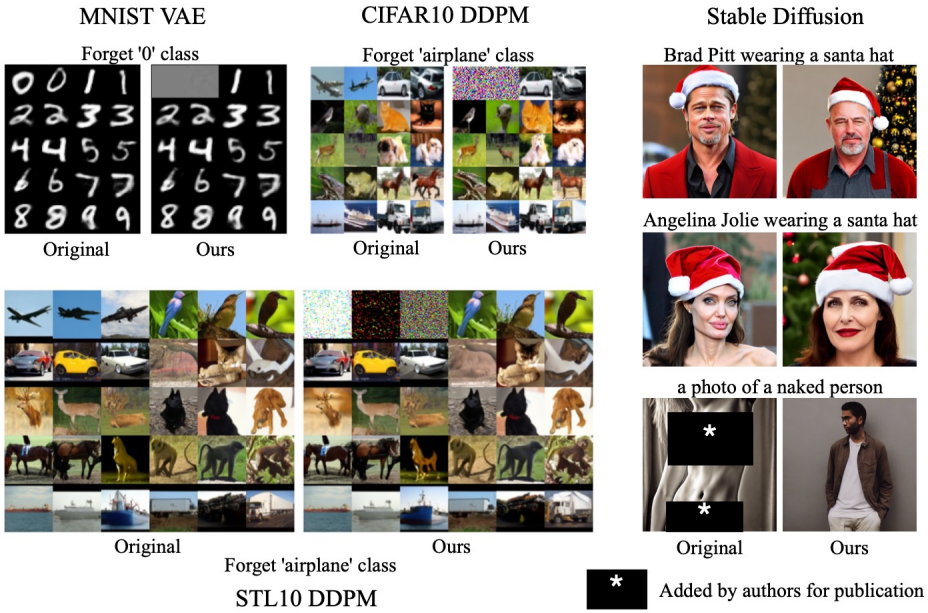
RLHF (2022)

- 言語モデルの出力を人間が良い順にランク付けし、それに従い言語モデルの出力を改善する手法 (強化学習を用いる) [D3]



Selective Amnesia (2024)

- 深層生成モデルから忘れさせたい概念を忘却させる手法
  - 忘れさせたい概念に対する対数尤度を最小化することで実現



参考文献D

1. Ji, Ziwei, et al. "Survey of hallucination in natural language generation." ACM Computing Surveys 55.12 (2023): 1-38.
2. Nalisnick, Eric, et al. "Do deep generative models know what they don't know?." arXiv preprint arXiv:1810.09136 (2018).
3. Ouyang, Long, et al. "Training language models to follow instructions with human feedback." Advances in neural information processing systems 35 (2022): 27730-27744.
4. Heng, Alvin, and Harold Soh. "Selective amnesia: A continual learning approach to forgetting in deep generative models." Advances in Neural Information Processing Systems 36 (2024).

まとめ・質疑応答 (10分)

まとめ

- 生成モデルに関する下記のトピックについて学習した
  - 生成モデルの基礎 / 深層生成モデル / 言語モデル / 生成モデルの問題点
  - 現在の生成AIは非常に複雑になっているので、わからなくなったときは、簡単な例でイメージしよう (例えばコインでの最尤推定など)
- 生成AIと生成モデルのギャップはモデルサイズとデータの量
  - 本講義で説明した基礎的なことを理解したら、最新の論文を読んでみよう
  - 大きなモデル・大きなデータだからこその難しさがある
- 生成AIの発展には、人間からのフィードバックが重要
  - 今後は生成AIが適切な回答を行えるようにする、特定の概念を忘れるなどの研究が重要になってくると予想