近年の深層学習について

田中章詞　(RIKEN AIP/iTHEMS)

@DLAP2023 6/1, 2023

自己紹介

■ 所属

- 理研 AIP/iTHEMS
  └─── 上級研究員

■ 研究

- 機械学習 (理論/応用)
- 数理物理 (素粒子理論)

AkinoriTanaka-phys

学術変革領域研究（A） 2022〜2026年度

B01班： 深層学習の数理と応用 代表

- メンバー
  - 唐木田 亮さん (産総研)
  - 瀧 雅人さん (立教大学)
- 目的
  - 深層学習の研究

1. 導入

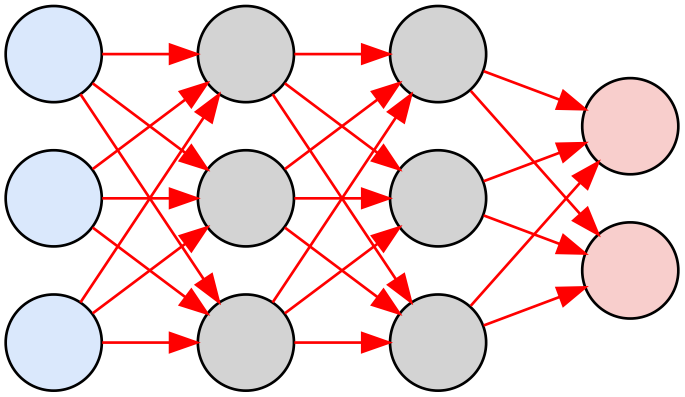
1. 導入

■ 1-1. 深層学習とは

教師あり機械学習

$f_\theta : X \to Y$ を導入 → うまい θ を取ることで実現しようとする

$f_\theta : X \to Y$ に **深層ニューラルネット** を使う手法 ＝ 教師ありの場合の **深層学習**



1. 導入 ↩

■ 1-2. 物理学との関連

深層学習の 物理学への 応用

物理学の問題に対する「汎用的な」手法となりつつある

- 相転移検出 [Carrasquilla & Melko, arXiv:1605.01735]
- MCMCの高速化 [Liu et al., arXiv:1610.03137]，[Kanwar & Shanahan, arXiv:1904.12072]
- 量子状態探索 [Carleo & Troyer, arXiv:1606.02318]

⇕ 相互の発展が見込まれる

物理学の 深層学習への 応用

物理学と関連した数理的手法が近年、盛んに取り込まれている

- Physics-Informed NN [Greydanus, Dzamba, & Yosinski, arXiv:1906.01563]
- ニューラルネット平均場解析 [Poole et al., arXiv:1606.05340]
- 拡散モデル [Sohl-Dickstein et al., arXiv:1503.03585]

2. 深層学習 (2019~)

2. 深層学習 (2019~)

■ 2-1. Transformerの登場

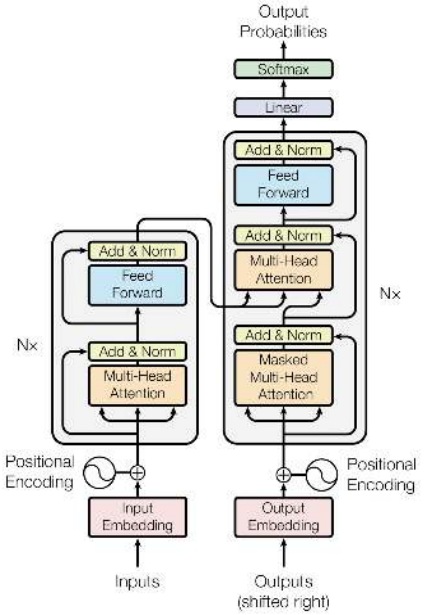
深層学習といえばお決まりだった、ネットワークアーキテクチャが軒並み Transformer なるものに置き換わってきている：

時系列データ（言語など）

年	その時の最先端モデル
~2017	リカレントNN (LSTMなど)
2017-	Transformer [Vaswani et al., arXiv:1706.03762]

画像データ

年	その時の最先端モデル
~2020	畳み込みNN
2020-	Vision Transformer [Dosovitskiy et al., arXiv:2010.11929]



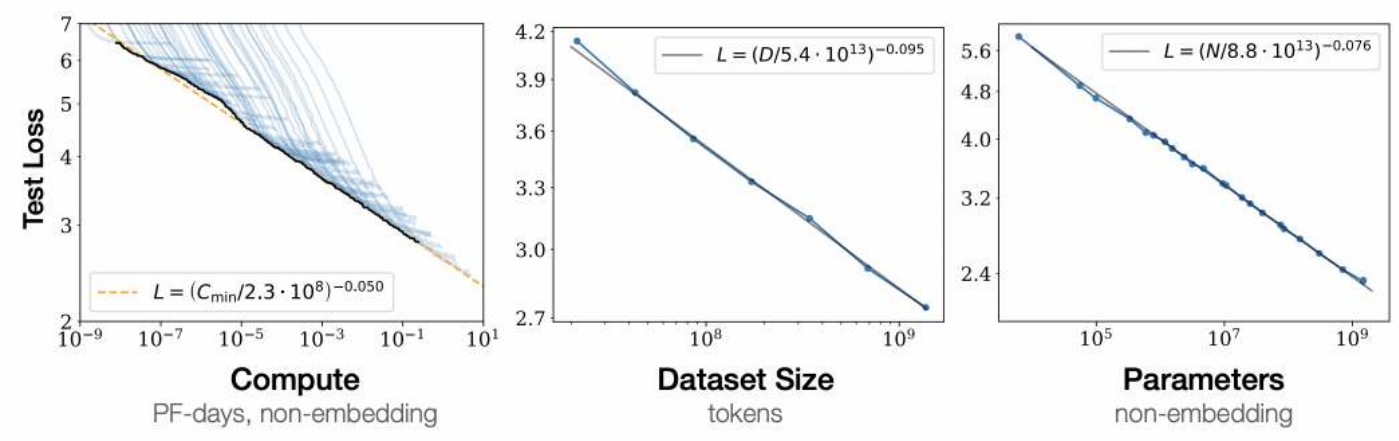
[Vaswani et al., arXiv:1706.03762] より抜粋

深層NNは ただ深くするだけで良いというものではない：ResNet [He et al., arXiv:1512.03385] の最初の図など参照 ので、何か本質的な改良のように思われる

2. 深層学習 (2019~)

■ 2-1. Transformerの登場

Transformerのスケーリング則



[Kaplan et al., arXiv:2001.08361] より抜粋

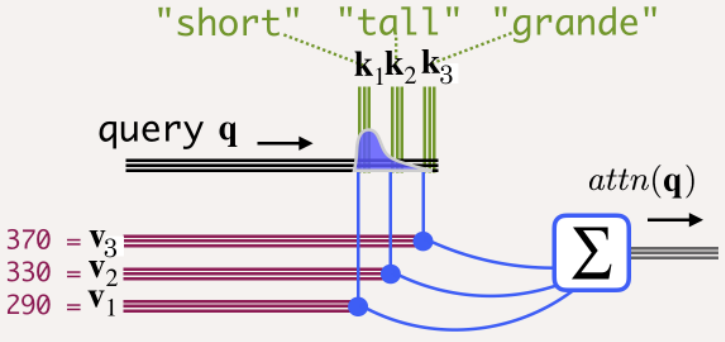
計算機, データサイズ, パラメータ数をスケール = 好きなだけテスト精度を上げられるということに...

DeepL翻訳 や ChatGPT などは Transformer に基づいていると思われる。

2. 深層学習 (2019~)

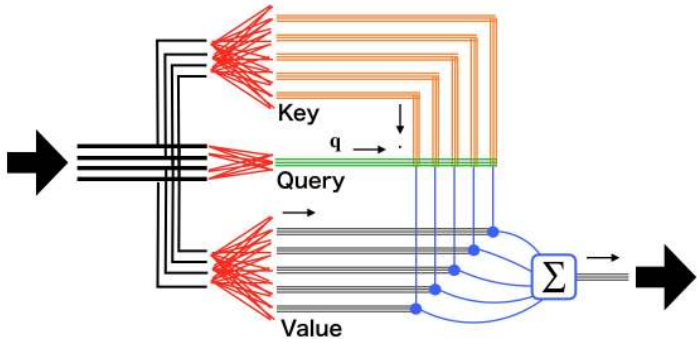
■ 2-1. Transformerの登場

Attention機構



クエリ（要求）に対して、類似度の大きい鍵を探し、対応する値を渡す

Self-attention: 自己注意



画像認識では Self attention ではないものを使っても良い：

- MLP-Mixer [Tolstikhin et al., arXiv:2105.01601]
- RaftMLP [Tatsunami & TAKI, arXiv:2108.04384]

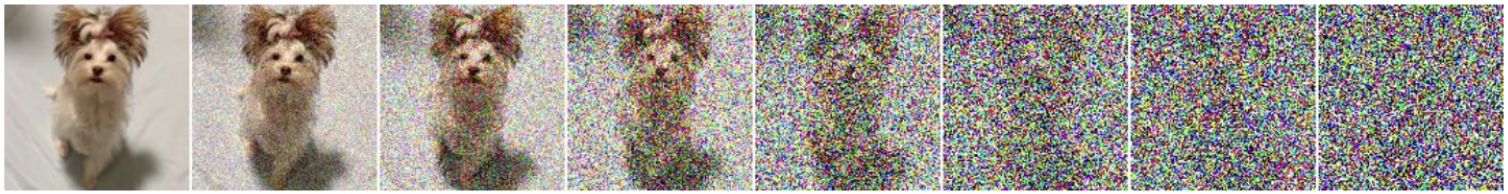
2. 深層学習 (2019~)

■ 2-2. 拡散モデルの登場

（画像）生成モデルでも大きな変革が起こる：

年	その時の最先端モデル
2013	VAE [Kingma & Welling, arXiv:1312.6114]
2014-2019	GAN [Goodfellow et al., arXiv:1406.2661]
2019-現在	拡散モデル [Song & Ermon, arXiv:1907.05600]

画像 → ノイズ の向きに変更を加えたのち



[Song et al., arXiv:2011.13456] より抜粋

画像 ← ノイズ のプロセスを学習させる。

2. 深層学習 (2019~)

■ 2-2. 拡散モデルの登場

様々なモデルがすでに数多くリリース：

- Stable diffusion （公開モデルへのリンク付き） [Rombach et al., arXiv:2112.10752]

- お絵描きばりぐっどくん https://page.line.me/877ieiqs

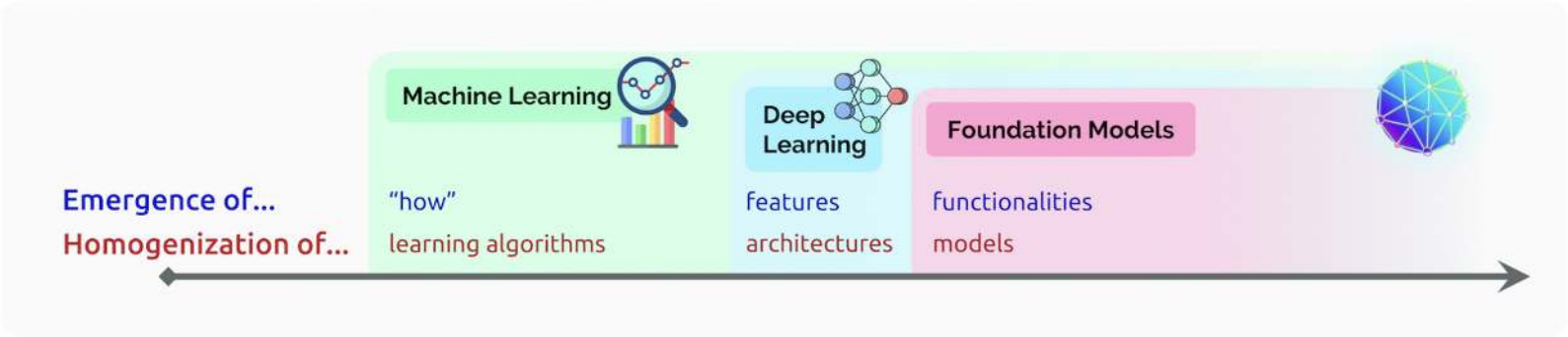
- Midjourney https://www.midjourney.com/home/?callbackUrl=%2Fapp%2F

2. 深層学習 (2019~)

■ 2-3. "基盤モデル"?

訓練済みの 大規模言語モデル (LLM) や 拡散モデル (Stable diffusion など)
➡ これらを別の様々なタスクのために 再学習（転移学習） させることができる

- 再利用できる公開モデルは： https://huggingface.co/ から呼び出すことができる
- ファインチューニング / 知識の蒸留 [Hinton, Vinyals, & Dean, arXiv:1503.02531]



[Bommasani et al., arXiv:2108.07258] より抜粋

1. 機械学習 = 学習アルゴリズムの均一化
2. 深層学習 = アーキテクチャの均一化
3. 基盤モデル = モデルの均一化

} ※ 注意：この方向性以外もある

3. 過剰パラメータの数理

3. 過剰パラメータの数理

■ 3-1. 過剰パラメータの統計学

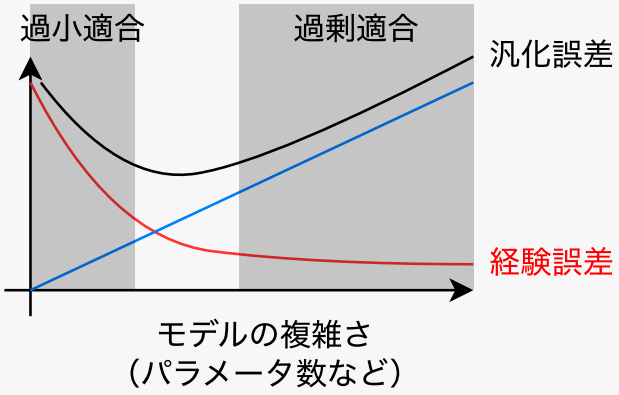
モチベーション

深層学習 ≠ 魔法の杖 であり、上手くいく背後には必ず理由があるはず

深層学習＝過剰パラメータによる統計学？

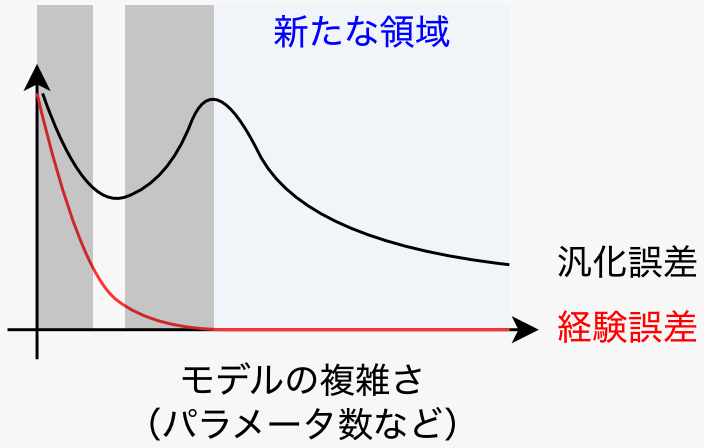
過剰パラメータ ≠ 過剰適合

深層学習*以前*



- PAC学習理論
- 情報量規準

深層学習*以後*



- 二重（多重）降下現象の理論
- over-parametrization の統計学

3. 過剰パラメータの数理 ↩

■ 3-1. 過剰パラメータの統計学

モチベーション

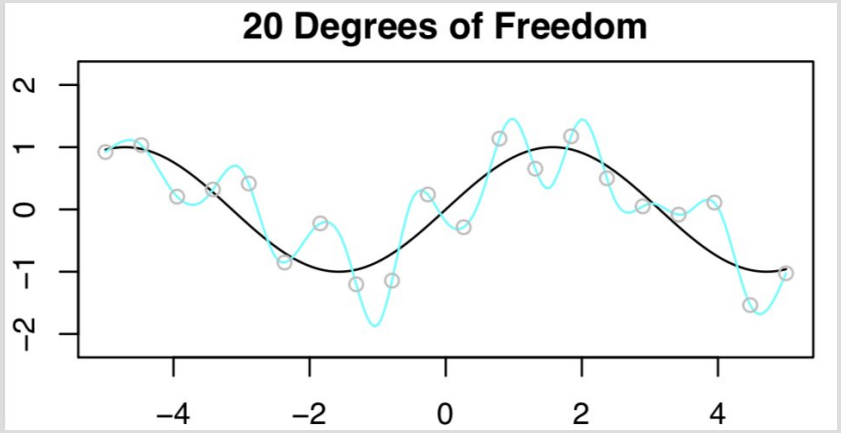
深層学習 ≠ 魔法の杖 であり、上手くいく背後には必ず理由があるはず

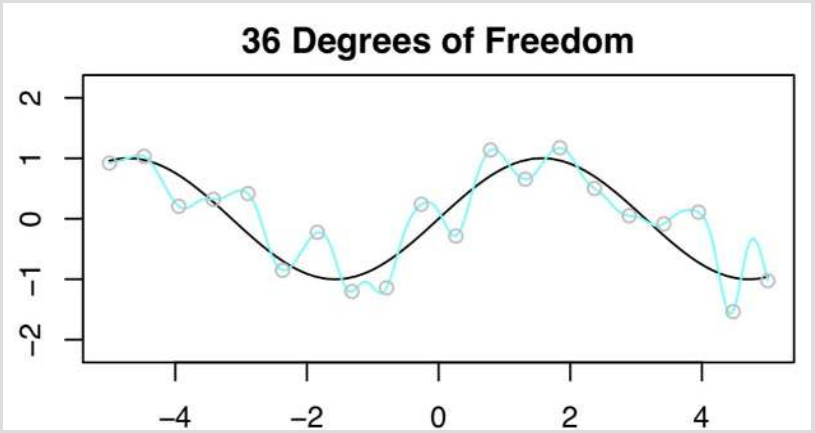
深層学習＝過剰パラメータによる統計学？

過剰パラメータ ≠ 過剰適合

二重降下が起こる例：

double descent daniela 検索 https://twitter.com/daniela_witten/status/1292293102103748609 より抜粋





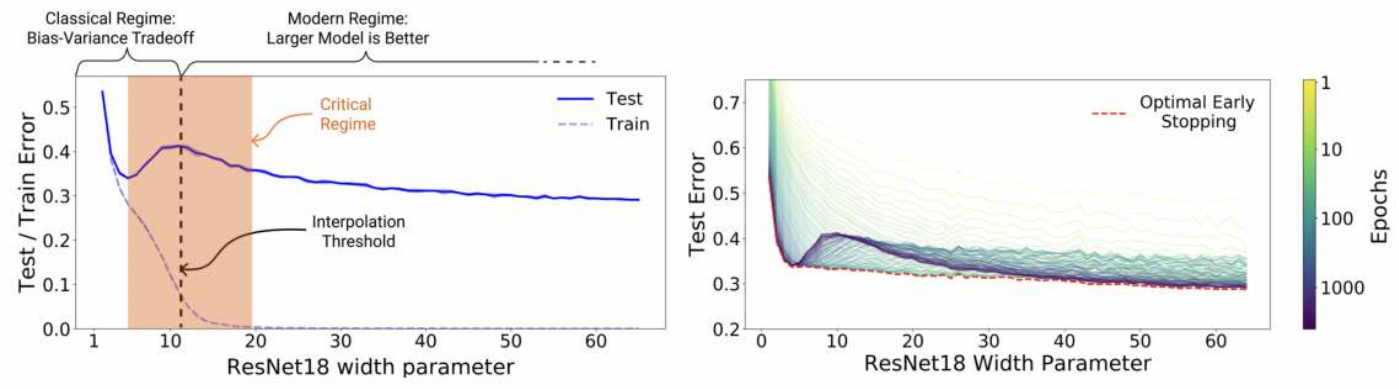
3. 過剰パラメータの数理 ↩

■ 3-1. 過剰パラメータの統計学

モチベーション

深層学習 ≠ 魔法の杖 であり、上手くいく背後には必ず理由があるはず

深層ニューラルネットでも二重降下が起きる



[Nakkiran et al., arXiv:1912.02292] より抜粋

ランダム行列による解析：[Hastie et al., arXiv:1903.08560]

汎化誤差解析による理論：[Nakada & Imaizumi, arXiv:2103.00500]

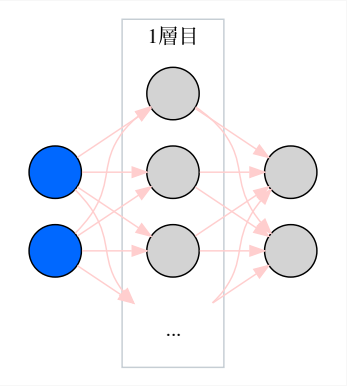
3. 過剰パラメータの数理

■ 3-2. 無限幅ニューラルネットの理論

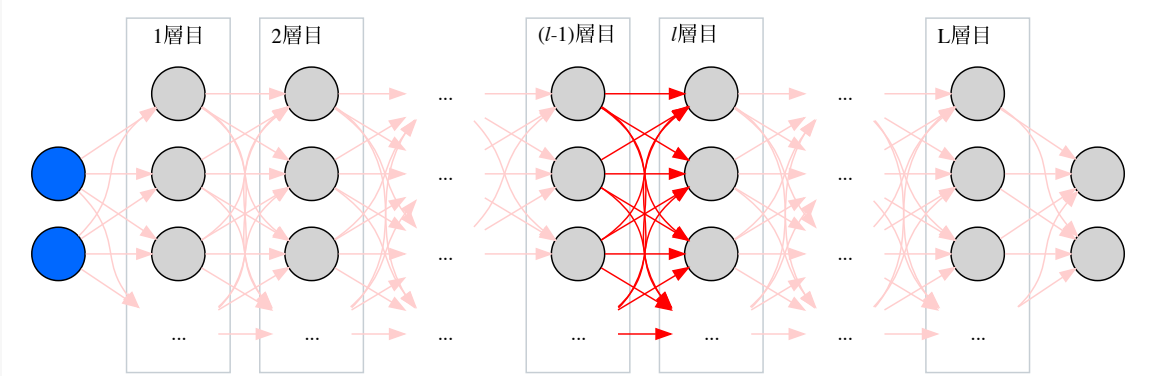
理論的に単純な過剰パラメータ極限

深層ニューラルネットの幅を無限に飛ばす解析

1層



多層



- 普遍性定理

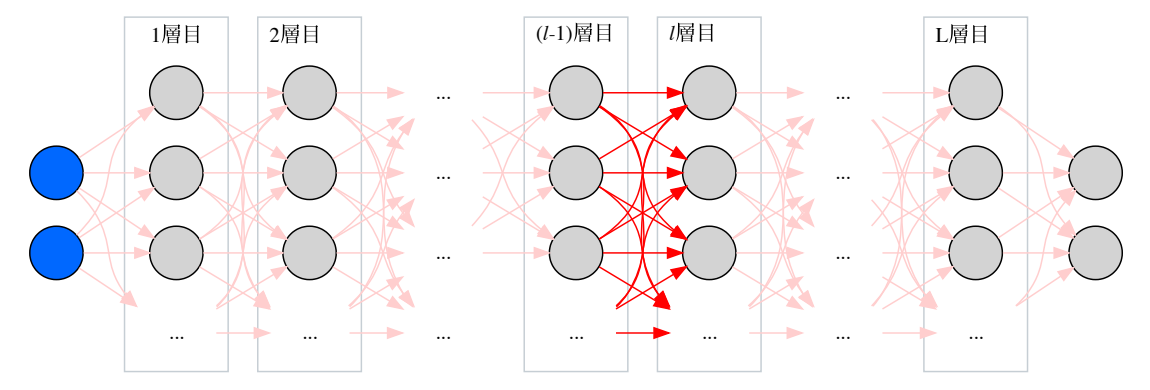
  任意の関数は幅無限大で表現できる

- ランダムNNの平均場解析 初期NNの振る舞い
- ニューラル接核理論 (NTK) NN訓練の振る舞い

3. 過剰パラメータの数理

■ 3-2. 無限幅ニューラルネットの理論

無限幅NNは Gaussian Process になる [Poole et al., arXiv:1606.05340]，[Lee et al., arXiv:1711.00165]



重み行列の共分散 についての仮定	バイアスの分散 に関する仮定
$\langle W_{\bullet j} W_{\bullet k} \rangle = \frac{\sigma_w^2}{N} \delta_{jk}$	$\langle b^2 \rangle = \sigma_b^2$

だとして、l層目 の入力に**中心極限定理**を適用する（幅無限大 $N \to \infty$）

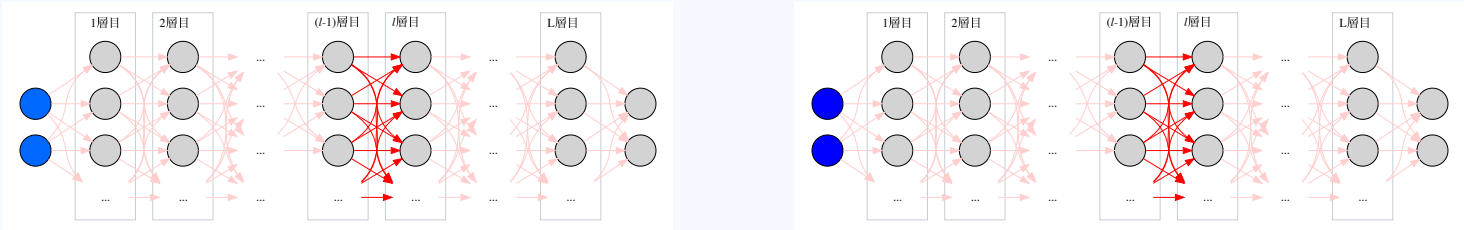
$$h^l = \sum_{j=1}^{N} W_{\bullet j} \phi(h_j^{(l-1)}) + b \sim \mathcal{N}(0, \sigma_w^2 \langle \phi(h^{(l-1)})^2 \rangle + \sigma_b^2)$$

3. 過剰パラメータの数理

■ 3-2. 無限幅ニューラルネットの理論

無限幅NNは Gaussian Process になる [Poole et al., arXiv:1606.05340], [Lee et al., arXiv:1711.00165]

分類の時：異なる入力 $\mathbf{x}, \mathbf{y}$ を入れた時の振舞いが知りたい



$(h^l[\mathbf{x}], h^l[\mathbf{y}])$ の "二点関数"

$$q^l_{\mathbf{xy}} = \langle h^l[\mathbf{x}] h^l[\mathbf{y}] \rangle$$

幅無限大 $N \to \infty$ で、やはりGaussianになる

無限幅(Gaussian = free theory) からのずれをFeynmann図で補正:

[Halverson, Maiti, & Stoner, arXiv:2008.08601]

3. 過剰パラメータの数理

■ 3-2. 無限幅ニューラルネットの理論

"二点関数" の相転移 [Poole et al., arXiv:1606.05340]

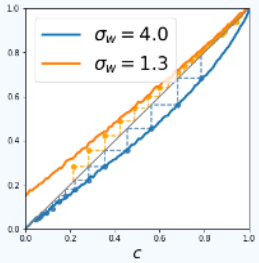
層ごとに再起式に従う：

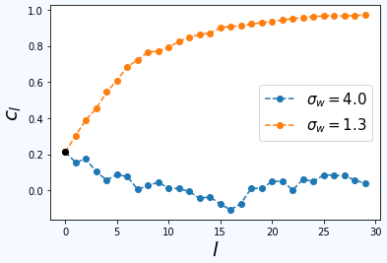
$$q^l_{\mathbf{xy}} \approx \mathcal{C}(q^{(l-1)}_{\mathbf{xy}})$$

$\mathcal{C}$ と傾き1の直線

実際のNNの二点相関

$\phi = \tanh$

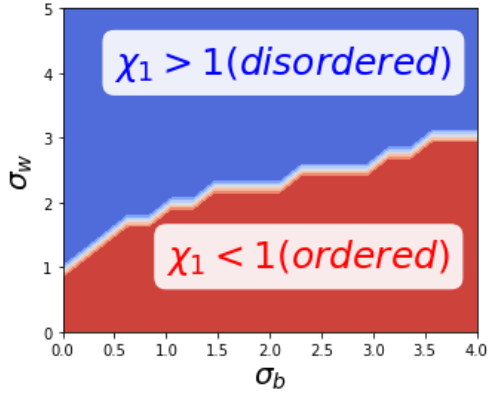




"相転移" (無秩序/秩序)

秩序パラメータ

$$\chi_1 = \left.\frac{\partial \mathcal{C}}{\partial c}\right|_{c=1}$$
$$= \sigma_w^2 \int dz \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} \left(\phi'(\sqrt{q^\infty} z)\right)^2$$



3. 過剰パラメータの数理 ↩

■ 3-2. 無限幅ニューラルネットの理論

- Backpropagation の 平均場解析 [Schoenholz et al., arXiv:1611.01232]
  - 秩序変数 χ_1 = Backpropagationの分散係数
    $\rightarrow \chi_1 \approx 1$:"カオスの縁" で勾配が消失/発散しない
- NNのFisher情報行列 $F(\theta)$ の固有値 [KARAKIDA, Akaho, & Amari, arXiv:1806.01316]
  - 勾配更新 $\theta_{t+1} = \theta_t - \eta \nabla_\theta E(\theta)$ の収束（必要）条件
- Neural Tangent Kernel [Jacot, Gabriel, & Hongler, arXiv:1806.07572]
  - 同じ極限で訓練プロセスを議論、大域収束が証明できる
- Feature Learning 極限 [Yang & Hu, arXiv:2011.14522]
  - より訓練性の良い無限幅極限

４. 拡散モデル

4. 拡散モデル

■ 4-1. 生成モデルとは

与えられたデータから、それに似たデータを作り出すモデル

例： 文章生成
(http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

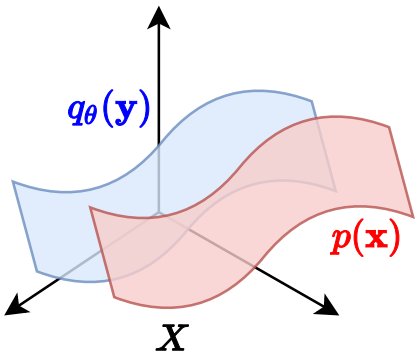
データ		生成
LARTIUSIs: O,'tis Marcius! Let's fetch him off, or make remain alike.	→	AKINORI: Came, bloody decrees, By city, not her brows, he is bloody woe!

例: 画像生成
(https://www.cs.toronto.edu/~kriz/cifar.html)

データ → 生成

■ 一つの考え方

- データ $\mathbf{x}$ は X上 確率分布 $p(\mathbf{x})$ に従う
- 生成データ $\mathbf{y}$ は X上 モデル $q_\theta(\mathbf{y})$ に従う

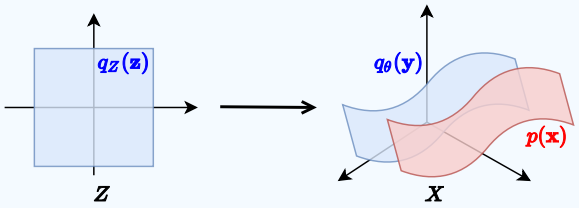


4. 拡散モデル ↩

■ 4-1. 生成モデルとは

深層生成モデル = 深層学習を用いた生成モデル

潜在空間モデル



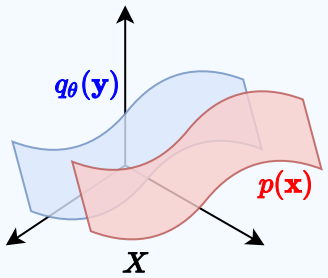
Z を別途用意、$Z \to X$ を推定

- Variational AutoEncoder, VAE

  [Kingma & Welling, arXiv:1312.6114]
- Generative Adversarial Net, GAN

  [Goodfellow et al., arXiv:1406.2661]

X上での直接モデル



X 上の確率を直接推定しにかかる

- Autoregressive model
- Flow-based model

  [Dinh, Krueger, & Y. Bengio, arXiv:1410.8516] ,

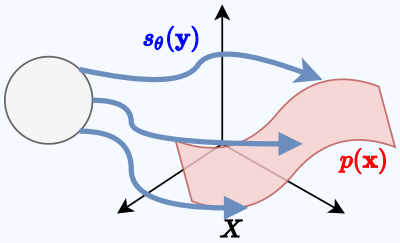
  [Dinh, Sohl-Dickstein, & S. Bengio, arXiv:1605.08803]

4. 拡散モデル ↩

■ 4-1. 生成モデルとは

深層生成モデル = 深層学習を用いた生成モデル

拡散モデル



X 上の ベクトル場 を推定する

- Noise conditioned score network, NCSN
  [Song & Ermon, arXiv:1907.05600]
- Denoising diffusion prob model, DDPM
  [Ho, Jain & Abbeel, arXiv:2006.11239]

■ 良い点

高クオリティなデータ生成が可能
（画像ではかなりの成功）

■ 悪い点

モデルからの標本抽出 = ベクトル場の
（確率）積分 → 推論が遅い

4. 拡散モデル ↩

■ 4-2. 拡散モデル lightning review

拡散（ノイズ）の逆プロセス（デノイズ）をモデル化した系列

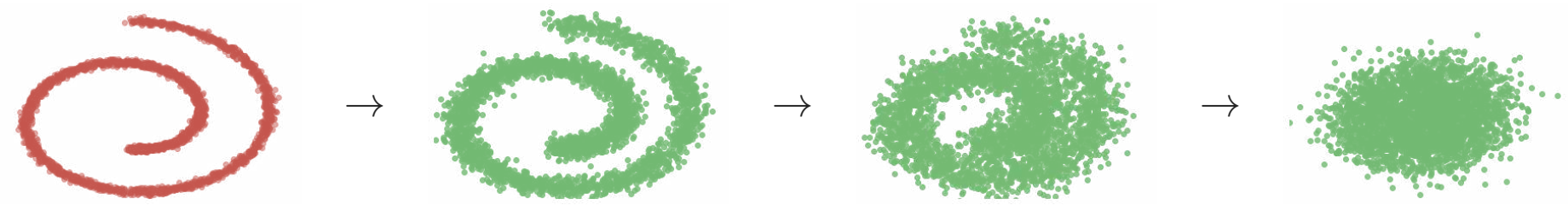
論文	内容
Deep Unsupervised Learning using Nonequilibrium Thermodynamics [Sohl-Dickstein et al., arXiv:1503.03585]	「非平衡熱力学」に着想を得た始祖的な論文
Denoising Diffusion Probabilistic Models [Ho, Jain, & Abbeel, arXiv:2006.11239]	上の論文を更に進化させたもので、拡散モデルブームの火付け役
Diffusion Models Beat GANs on Image Synthesis [Dhariwal & Nichol, arXiv:2105.05233]	画像生成で GAN を凌駕することを示した ブームの火付け役 2

拡散（ノイズ）入りデータ分布の スコア（フォース）の推定

論文	内容
Generative Modeling by Estimating Gradients of the Data Distribution [Song & Ermon, arXiv:1907.05600]	拡散モデル＝スコア推定による最初の解釈
Score-Based Generative Modeling through Stochastic Differential Equations [Song et al., arXiv:2011.13456]	確率微分方程式(SDE)による統一的な枠組み

4. 拡散モデル ↩

■ 4-2. 拡散モデル lightning review



$$\text{SDE:} \quad d\mathbf{x}(t) = \underbrace{\mathbf{f}\big(\mathbf{x}(t), t\big)}_{\text{ドリフト}} dt + \underbrace{g(t)}_{\text{ノイズ強度}} \quad d\mathbf{w}(t)$$

時刻 t の分布が従う方程式：Fokker-Planck方程式

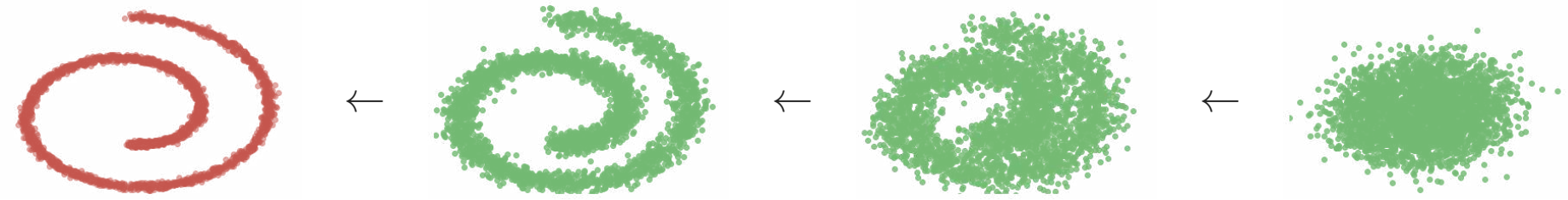
$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla_{\mathbf{x}} \cdot \Big(\underbrace{\mathbf{v}(\mathbf{x}, t)}_{\mathbf{f}(\mathbf{x},t) - \frac{g(t)^2}{2} \nabla_{\mathbf{x}} \log p_t(\mathbf{x})} \quad p_t(\mathbf{x}) \Big)$$

時間を逆転させたFP方程式に対応するダイナミクス がわかれば

ノイズ → データ生成？

4. 拡散モデル ↩

■ 4-2. 拡散モデル lightning review



逆向きSDE: $$d\mathbf{x}(\tau) = -\Big(\mathbf{f}(\mathbf{x}(\tau), t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}(\tau))\Big) d\tau + g(t) d\bar{\mathbf{w}}(\tau)$$

確率フロー: $$d\mathbf{x}(\tau) = -\left(\mathbf{f}(\mathbf{x}(\tau), t) - \frac{g(t)^2}{2} \nabla_{\mathbf{x}} \log p_t(\mathbf{x}(\tau))\right) d\tau$$

$\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$? → ニューラルネット $s_\theta(\mathbf{x}, t)$ で近似する

データだけから訓練できる

スコアマッチング: [Hyvärinen, 2005], デノイジング スコアマッチング: [Vincent, 2011]

詳細は https://hohno0223.github.io/comp_phys_spring_school2023/ の「生成模型入門」参照

4. 拡散モデル ↩

■ 4-3. 拡散モデルにおける様々な疑問

- 画像生成でGANを凌駕した性能を示す [Dhariwal & Nichol, arXiv:2105.05233] のはなぜ？
    - 訓練の安定性のため？
    - 拡散/逆拡散のパラダイム（≒ 物理？）が本質的なのか？
- 画像生成に時間がかかる？
    - 生成時間/クオリティの間にトレードオフがあるように見える

[Lipman et al., arXiv:2210.02747] より抜粋

- 最適輸送(OT)との関連？ 2-Wasserstein 勾配流？
    - 画像生成でOTはよく使われてきている
        - (V)AE [Tolstikhin et al. arXiv:1711.01558]
        - GAN [Arjovsky, Chintala, & Bottou, arXiv:1701.07875]，[TANAKA, arXiv:1910.06832]

付録：最近の深層学習の実装面について

深層学習ライブラリ ↩

■ 2大 コア ライブラリ： TensorFlow / PyTorch

TensorFlow https://www.tensorflow.org/

- チュートリアルが非常に充実： https://www.tensorflow.org/tutorials?hl=ja
- 色々なインターフェイスで動かすことが可能
    - TensorFlow.js (JavaScript) / TensorFlow Lite (Raspi)
- Google 研究メインは 後述の JAX と Flax/Haiku に移行か?
    - https://twitter.com/mattlynley/status/1538185558953893888

PyTorch https://pytorch.org/

- バッチ処理だけなら、お手軽に シングルGPU → マルチGPU
    - https://aru47.hatenablog.com/entry/2020/11/06/225052
- 用途別にライブラリが分かれている
    - コアライブラリ：torch https://pytorch.org/docs/stable/index.html
    - データ処理用：torchvision / torchaudio / ...
- 研究での使用率高、後述の HaggingFace のデフォルト
    - https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

深層学習ライブラリ ↩

■ 第3勢力：JAX + α

JAX https://jax.readthedocs.io/

> 自動微分 + jitコンパイル(CPU/GPU/TPU) 付きの numpy と思えば良い
>
> - Google が開発、比較的若いプロジェクト
> - 関数型言語の設計思想
> - 標準で並列計算
> - ランダム変数 の取り扱いが独特（seed や遷移を明示し、再現性高）
>
> ニューラルネットは入っていない。後述

適当な関数を定義 → 導関数は `jax.grad()` で計算可能

- それだけなら他のライブラリ（sympyなど）でも出来るが、jit コンパイルで GPU 等に最適化できるのが良い
- 変数の取り扱いを tree 構造で定義できるので、複雑な関数実装を整理しやすいかもしれない

深層学習ライブラリ ↩

■ 第3勢力：JAX + α

JAX だけで 例えば線形回帰を SGD で解く：

Python

```python
def loss(params, xs, ys):
  w, b = params
  os = w * xs + b
  return jnp.mean((os - ys)**2)

# training
params = jnp.array([1., 1.])
while not convergent:
  xs, ys = data_generator.sample()
  params -= 0.1*jax.grad(loss)(params, xs, ys)
```

これくらいなら簡単、複雑な深層ニューラルネットは？→ 他のライブラリ

深層学習ライブラリ ↩

■ 第3勢力：JAX + α

Flax https://flax.readthedocs.io/

> Google Research が開発

Haiku https://dm-haiku.readthedocs.io/

> DeepMind が開発

両方共にモデルは 純粋な関数 (訓練パラメータを含まない) として実装

Python

```python
model = ...  # ここの構成法は両者で少し違う
params = model.init(...)                              # この値が更新される
while not convergent:
  xs, ys = data_generator.sample()
  params -= 0.1*jax.grad(loss)(params, xs, ys)  # 更新 (jax.jitをかけると良い)
#test
xs, ys = data_generator.sample()
print(model.apply(params, xs) == ys)        # 出力は model.apply(params, x)
```

optax https://optax.readthedocs.io/ から Adam なども使える。（Flax は `TrainState` というAPIで model/params/optimizer をまとめて管理できる）

深層学習ライブラリ ↩

■ その他の深層学習ライブラリ

- MXNet (ワシントン大/カーネギーメロン大 → AWS?) https://mxnet.apache.org/
- CNTK (Microsoft) https://learn.microsoft.com/en-us/cognitive-toolkit/
- ...

Python以外もあります。

- TensorFlow (C++) https://www.tensorflow.org/api_docs/cc
- PyTorch (C++) https://pytorch.org/cppdocs/
- Flux (Julia lang) https://fluxml.ai/Flux.jl/stable/
- Mathematica

  https://reference.wolfram.com/language/tutorial/NeuralNetworksIntroduction.html.ja?source=footer

全部は調べきれていません。他にも沢山あると思います。

深層学習ライブラリ ↩

■ 深層強化学習ライブラリ

深層強化学習もかなりお手軽にできるようになっています

環境のライブラリ：

- シングルエージェント：OpenAI Gym → Gymnasium https://gymnasium.farama.org/
- マルチエージェント： PettingZoo https://pettingzoo.farama.org/ 他

Atariゲームに関しては、おそらく著作権の関係からROMのライセンスを承諾する形でのみダウンロード可能に変更

深層強化学習のライブラリ：

- TensorFlowベース：TF-Agents https://www.tensorflow.org/agents?hl=ja 他
- PyTorchベース：PFRL http://pfrl.readthedocs.io/ 他
- JAXベース：RLax https://rlax.readthedocs.io/ 他
    - 個人的には coax https://coax.readthedocs.io/ が良い気がします

基板モデル使用のためのライブラリ ↩

■ **HuggingFace** https://huggingface.co/

アメリカのAIベンチャー企業： https://ja.wikipedia.org/wiki/Hugging_Face

GitHubのようなプラットフォーム：https://huggingface.co/docs/hub/

■ **訓練済みモデルを使うライブラリ**

共に、自前データ＋層の付け足し 追加訓練 も可能

- **Transformers** https://huggingface.co/docs/transformers/

> 名前の通り、Transformer系のモデルを扱うことができる

- **Diffusers** https://huggingface.co/docs/diffusers/

> 名前の通り、拡散モデル系のモデルを扱うことができる

TenforFlow / PyTorch / Flax をサポート。

基板モデル使用のためのライブラリ ↩

Transformers https://huggingface.co/docs/transformers/

基本構造

- `pipeline` ・・・取りまとめインスタンス
  - `tokenizer` ・・・文字に番号を振るルール
  - `model` ・・・transformer モデル
  - その他 事後処理など

例：英和翻訳 (checkpoint作成者の方： https://twitter.com/voleneko)

Python

```python
from transformers import pipeline

translator = pipeline('translation',
        model='staka/fugumt-en-ja')
translator("I'm now talking about how
to use the Transformers library in
python.")
```

output

```
[{'translation_text': '私は今、
PythonでTransformersライブラリを使用
する方法について話しているところで
す。'}]
```

Python

```python
dir_t = vars(translator)
for key in dir_t.keys():
  print(key, ": ",
type(dir_t[key]))
```

output

```
task :  <class 'str'>
model :  <class '....'>
tokenizer :  <class '....'>
device :  <class
'torch.device'>
...
```

基板モデル使用のためのライブラリ ↩

Transformers https://huggingface.co/docs/transformers/

基本構造

- `pipeline` ・・・取りまとめインスタンス
  - `tokenizer` ・・・文字に番号を振るルール
  - `model` ・・・transformer モデル
  - その他 事後処理など

例：画像分類

https://huggingface.co/datasets/huggingface/documentation-images/resolve/main/pipeline-cat-chonk.jpeg

Python

```python
from transformers import pipeline

classifier = pipeline(
    model="google/vit-base-patch16-224")
classifier(images="xxx.jpeg")[0]
```

output

```
{'score': 0.44030264019966125,
'label': 'lynx, catamount'} # 山猫
```

Python

```python
dir_c = vars(classifier)
for key in dir_c.keys():
  print(key, ": ",
type(dir_c[key]))
```

output

```
task :  <class 'str'>
model :  <class '...'>
image_processor :  <class
'...'>
device :  <class
'torch.device'>
...
```

基板モデル使用のためのライブラリ ↩

Diffusers https://huggingface.co/docs/diffusers/

基本構造

- `pipeline` ・・・取りまとめインスタンス
  - `model` ・・・スコア推定モデル
  - `scheduler` ・・・ノイズ付与/除去のスケジューラー

例：ネコ画像生成器

Python

```python
from diffusers import DDPMPipeline

ddpm = DDPMPipeline.from_pretrained(
                    "google/ddpm-cat-256").to("cuda")
images = ddpm(num_inference_steps=50,
                batch_size=3).images
```



Python

```python
dir_d = vars(ddpm)
for key in dir_d.keys():
  print(key, ": ",
type(dir_d[key]))
```

output

```
_internal_dict :  <class
'...'>
unet :  <class '...'>
scheduler :  <class '...'>
```

unet が model に対応

セキュリティリスクについて ↩

TensorFlow: モデルそのものがプログラムにできる

- 参考：https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md
    - kerasの ラムダ層 によるもの？ https://keras.io/api/layers/core_layers/lambda/

PyTorch： ここでの話に限らず、モデルのロードで pickle を使うが、これには任意コード実行の脆弱性がある：

- https://pytorch.org/docs/stable/generated/torch.load.html の WARNING
- 参考：https://huggingface.co/docs/hub/security-pickle

TF/Flax は大丈夫らしいので、HaggingFaceの場合、そっちのデータがある場合はそれを流用できる。後述。

Flax/Haiku の情報は調べられていないです…
いずれにしても、事前学習モデルを使う場合は注意が必要。

セキュリティリスクについて ↩

■ HaggingFace Hub 上での確認

`pickle` に関しては、前ページのHaggingfaceの注意ページによると：

```
...you can only reference functions and classes from the top level module; you cannot embed them in the pickle file itself. (トップレベルモジュールからの関数/クラス参照しかできず、pickleファイル 内に関数を埋め込めない)
```

→ 何が pickle で呼ばれているか見れば、ある程度の危険度がわかる。

- OSのシステムコール系
  - `os` モジュールなど
- 組み込み関数でシステムに触れる系
  - `eval`, `exec` など： https://qiita.com/tkmz/items/717d524083b71a4af75f

Hub のリポジトリの "**Files and versions**" で、pickle ファイルがある場合は、その横の pickle をクリックすると、何がインポートされているか見れる。
（そして怪しい場合は赤色で警告が出ている）

- モデルが `.safetensors` という拡張子の場合は更に安全とのこと

セキュリティリスクについて

■ PyTorchモデルをどうしても使いたくない場合

安全な PyTorchモデルが大半なはず… でも万が一の場合があるので、という方は、

- `from_tf=True` あるいは `from_flax=True` とする(HaggingFace Hub に TensorFlow/Flaxのモデル重みがある場合のみ有効)

Python

```python
from transformers import AutoModelForTokenClassification

model = AutoModelForTokenClassification.from_pretrained("distilbert-base-uncased", from_tf=True)
```

output

```
All TF 2.0 model weights were used when initializing DistilBertForTokenClassification.

All the weights of DistilBertForTokenClassification were initialized from the TF 2.0 model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use DistilBertForTokenClassification for predictions without further training.
```

TensorFlow/Flaxモデルがない場合は、エラーが出ます。

セキュリティリスクについて ↩

■ というわけで

訓練済みモデルを使うのがこれから重要になってくると思われますが、その使用には注意してください。`sudo` コマンドを使う時の気持ちになると良いかも：

With great power comes great responsibility

■ とはいうものの

原義の（正義の？）ハッカーの方々はクラッキング（世間的にいうハッキング※）はやらない と思います：参考と以下の引用： https://cruel.org/freeware/hacker.html

> (順不同) **ハッカーは問題を解決し、物事を築きます。そして自由と自発的な助け合いを信条としています。** ... 実は**どの科学や芸術分野でも、一番高い水準ではこのハッカー精神が見られます。**... ハッカーを声高に名乗る別の集団が存在しますが、彼らはハッカーではありません。これはコンピュータに侵入したり、電話のただがけしたりする人々（主に男のガキ）です。**本物のハッカーはこの連中を「クラッカー（cracker）」と呼び、一切関わりを持ちたくないと思っています。**...

※ 元来、ハッキングに悪い意味はありません：https://ja.wikipedia.org/wiki/ハッキング

GPU の使用

ここまで紹介した全てのライブラリで使用可能。自前で用意する場合は注意：

1. NVIDIA ドライバーをダウンロード、 `nvidia-smi` が通ることを確認
2. ドライバーとCUDAのバージョン対応を確認↓しCUDAを入れる
   - https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html (の CUDA Driver)
3. CUDAバージョンに適した TensorFlow/PyTorch/JAX をダウンロード
   - うまくGPUを認識できている場合は、`pip` でのダウンロードが簡単

dockerイメージがある場合 2,3 は結合可能。GPU使用の際は

```
docker run --gpus 枚数 ...
```

のオプションをつけてコンテナを開く。(docker >=19.03)

- JAX の dockerイメージは 公式のものがまだない（開発中？）ですが、TensorFlowのイメージをベースにするとうまくいきました。

`Python` 以外での使用に関しては僕はわかりません。すいません。

ご静聴ありがとうございました

- 1. 導入
    - 1-1. 深層学習とは
    - 1-2. 物理学との関連
- 2. 深層学習 2019~
    - 2-1. Transformerの登場
    - 2-2. 拡散モデルの登場
    - 2-3. "基盤モデル"
- 3. 過剰パラメータの数理
    - 3-1. 過剰パラメータの統計学
    - 3-2. 無限幅ニューラルネットの理論
- ４. 拡散モデル
    - 4-1. 生成モデルとは
    - 4-2. 拡散モデル lightning review
    - 4-3. 拡散モデルにおける様々な疑問

- 深層学習ライブラリ
- 基板モデル使用のためのライブラリ
- セキュリティリスクについて
- GPU の使用