Variational Autoencoders and Generative Adversarial Networks

20.9 Back-Propagation through Random Operations
20.10 Directed Generative Nets
　20.10.2 Differentiable(微分可能) Generator Networks
　20.10.3 Variational(変分) Autoencoders
　20.10.4 Generative Adversarial Networks

楊　森（Sen Yang）2019/08/31

参考：

[2014, Kingma et al.] Auto-Encoding Variational Bayes (VAE)
[2014, Goodfellow et al.] Generative Adversarial Nets (GAN) (https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf)
[2016, Radford et al.] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (DCGAN)
[2016, Goodfellow] NIPS 2016 Tutorial: Generative Adversarial Networks

Back-Propagation through Random Operations
（連続の場合、reparametrization trick）

背景：伝統的なNeural Networkはインプットxに**決定的な(deterministic)**変換をする。生成モデルを開発する時、**確率的(stochastic)**な変換もできるようにNeural Networkを拡張したい。一つわかりやすい方法は、簡単な確率分布（一様分布、ガウス分布）から**サンプリング**した別のインプットzを入れる。

つまり、関数だけではなく、分布も含まれる場合、どうやってback-propagation?

例：正規分布からyをサンプリングする場合の微分（back-propagation用）：

yは関数から生成されるではなく、サンプリングされるので、μ,σに対して微分を計算するのが直観に反する

$$y \sim \mathcal{N}(\mu, \sigma^2) \quad \Rightarrow \quad \begin{aligned} z &\sim \mathcal{N}(z; 0, 1) \\ y &= \mu + \sigma z \end{aligned}$$

こう変換すると、yのμ,σに対する微分を計算するようになる。勿論zの分布がμ,σと関係ないのが必須。

一般化すると、どのような分布でもback-propagation（学習）できるようになれる

$$y \sim p(y \,|\, \omega) \quad \Rightarrow \quad y = f(z; \omega)$$

ωはモデルパラメータθとインプットx両方含む、**zはランダム源**。勿論ωはzの関数ではなく、zはωの関数ではないのが必須。この方法は**reparametrization trick**、stochastic back-propagation、perturbation analysisと言う。

VAEが使う重要な手法の一つ

Back-Propagation through Discrete Random Operations （離散の場合、REINFORCE）

背景：yが離散の場合、f(z,ω)が階段関数(step function)にならないといけない。階段関数だと微分は未定義か0なので、back-propagationできない。

解決方法：コスト関数 $J(f(z;\omega))$ がstep関数だが、コスト関数の期待値 $\mathbb{E}_{z\sim p(z)}J(f(z;\omega))$ は滑らかな関数なので、gradient descentできる。

コスト期待値のωに対する微分の計算方法：

$$\mathbb{E}_z[J(y)] = \sum_y J(y)p(y)$$

$$\frac{\partial \mathbb{E}_z[J(y)]}{\partial \omega} = \sum_y J(y)\frac{\partial p(y)}{\partial \omega}$$

$$= \sum_y J(y)p(y)\frac{\partial logp(y)}{\partial \omega}$$

$$\approx \frac{1}{m}\sum_{y^{(i)}\sim p(y),i=1}^{m} J(y^{(i)})\frac{\partial logp(y^{(i)})}{\partial \omega}$$

unbiased Monte Carla estimator

このMonte Carla推定器のvarianceはまだ高いので、またvariance reduction方法で大きくvarianceを下げる処理も必要。略。（p681-682）

Differentiable（微分可能） Generator Networks

意味：微分可能関数$g(z;\theta^{(g)})$で潜在変数zをサンプルxもしくはサンプルxでの分布$p(x)$に変換するモデル。$g(z;\theta^{(g)})$は一般的にNeural Networkで表現される。

3種類：①variational autoencoders: <inference net, generator net>ペア
②generative adversarial networks: <discriminator net, generator net>ペア
③孤立的にgenerator netをトレーニングする技術

①zをxに変換する場合：（サンプル）

$$x = g(z)$$

$$p_z(z) \xrightarrow{g(z;\theta^{(g)})} p_x(x)$$

生成器ネットワークはzの分布をxの分布に変換する

g(z)が可逆(invertible)、可微分、連続の場合：

$$p_z(z) = p_x(g(z))\,|det(\frac{\partial g}{\partial z})| \Rightarrow p_x(x) = \frac{p_z(g^{-1}(x))}{|det(\frac{\partial g}{\partial z})|}$$

この式を評価（実装?）するのが難しいので、直接にlog p(x)を最適化するではなく、間接的にgを学習する

②zをxの条件付き分布に変換する場合：（分布のパラメータ）

例えばx離散の場合 $p(x_i = 1 | z) = g(z)_i$

p(x|z)の周辺分布でp(x)を計算する： $p(x) = \mathbb{E}_z p(x|z)$

②の場合連続＆離散データを生成できる。①の場合連続データのみ生成できる。

微分可能生成器ネットワーク方法のモチベーションは、gradient descentを使う微分可能feedforwardネットワークの分類での成功。

Variational Autoencoders①VAEのLoss関数

参考：[2014, Kingma et al.] Auto-Encoding Variational Bayes

やり方：Chapter 19近似推定で紹介したELBOを最大化する。

$$\mathscr{L}(q) = \mathbb{E}_{z \sim q(z|x)} log p_{model}(z, x) + \mathscr{H}(q(z|x)) \ldots \mathscr{L}^A$$

p(z|x)ではない。p(z|x)は解けない。

zがサンプリングされるので、q分布のパラメータに対してback-propagationできるように、zを関数に変換する

$$= \mathbb{E}_{z \sim q(z|x)} log p_{model}(x|z) - D_{KL}(q(z|x) \| p_{model}(z)) \ldots \mathscr{L}^B$$

$$\leq log p_{model}(x)$$

εはランダム源

①reparametrization trick： $z \sim q_\phi(z|x)$ → $\epsilon \sim p(\epsilon)$, $z = g_\phi(\epsilon, x)$

φはq分布のパラメータ。例えば正規分布の場合、φはμ、σ

②Monte Carlaサンプリングでzでの期待値を近似：

$$\mathscr{L}^A(\theta, \phi; x^{(i)}) = \frac{1}{L} \sum_{l=1}^{L} log p_\theta(x^{(i)}, z^{(i,l)}) - log q_\phi(z^{(i,l)} | x^{(i)})$$

$$\mathscr{L}^B(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)}) \| p_\theta(z)) + \frac{1}{L} \sum_{l=1}^{L} log p_\theta(x^{(i)} | z^{(i,l)})$$

With: $z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$

$\epsilon^{(l)} \sim p(\epsilon)$

KL Divergenceはよく分析的に積分できるので（論文のAppendix Bにガウス分布の場合の数学証明がある、正規分布もVAE実現の選んでる分布）、Monte Carlaサンプリングする必要がなくなって、実装しやすくなる。VAEの実現はBバージョンのELBOを使う

Variational Autoencoders② $\mathscr{L}^B$ とELBOが等価の証明

ELBO定義

$$\mathscr{L} = logp(x) - D_{KL}(q(z|x)\|p(z|x))$$

[2014, Kingma]の式(1)。p(z|x)は解けないので、zの事前確率分布p(z)を利用する式に変換

$$= logp(x) - \mathbb{E}_{z\sim q}log\frac{q(z|x)}{p(z|x)} = logp(x) - \mathbb{E}_{z\sim q}log\frac{q(z|x)}{\frac{p(z,x)}{p(x)}}$$

$$= logp(x) - \mathbb{E}_{z\sim q}log\frac{q(z|x)p(x)}{p(z,x)} = logp(x) - \mathbb{E}_{z\sim q}log\frac{q(z|x)p(x)}{p(z)p(x|z)}$$

$$= -\mathbb{E}_{z\sim q}log\frac{q(z|x)}{p(z)p(x|z)} = \mathbb{E}_{z\sim q}logp(x|z) - \mathbb{E}_{z\sim q}log\frac{q(z|x)}{p(z)}$$

$$= \mathbb{E}_{z\sim q}logp(x|z) - D_{KL}(q(z|x)\|p(z))$$

[2014, Kingma]の式(3)。実装しやすい。

伝統的なAutoencoderの再建(reconstruction) log-likelihood。

regularizerと見れる。q(z|x)とp(z)がお互いに近づく。

Variational Autoencoders③VAEの実現(ガウス分布の場合)

①KL Divergenceの計算（[2014, Kingma] Appendix B）

$$\int q_\phi(z) log p_\theta(z) dz = \int \mathcal{N}(z;\mu,\sigma^2) log \mathcal{N}(z;0,I) dz = -\frac{J}{2} log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(\mu_j^2+\sigma_j^2)$$

$$\int q_\phi(z) log q_\phi(z) dz = \int \mathcal{N}(z;\mu,\sigma^2) log \mathcal{N}(z;\mu,\sigma^2) dz = -\frac{J}{2} log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(1+log\sigma_j^2)$$

$$-D_{KL}(q_\phi(z)\|p_\theta(z)) = \int q_\phi(z)(log p_\theta(z) - log q_\phi(z)) dz = \frac{1}{2}\sum_{j=1}^{J}(1+log\sigma_j^2-\mu_j^2-\sigma_j^2)$$

②variational lower boundに代入する

$$\mathcal{L}^B(\theta,\phi;x^{(i)}) = -D_{KL}(q_\phi(z\,|\,x^{(i)})\|p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} log p_\theta(x^{(i)}\,|\,z^{(i,l)})$$

$$= \frac{1}{2}\sum_{j=1}^{J}(1+log(\sigma_j^{(i)})^2-(\mu_j^{(i)})^2-(\sigma_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L} log p_\theta(x^{(i)}\,|\,z^{(i,l)})$$

VAEソースコードにこの式が出る。LossはKL lossとCross-entropy lossの2つ部分から構成される

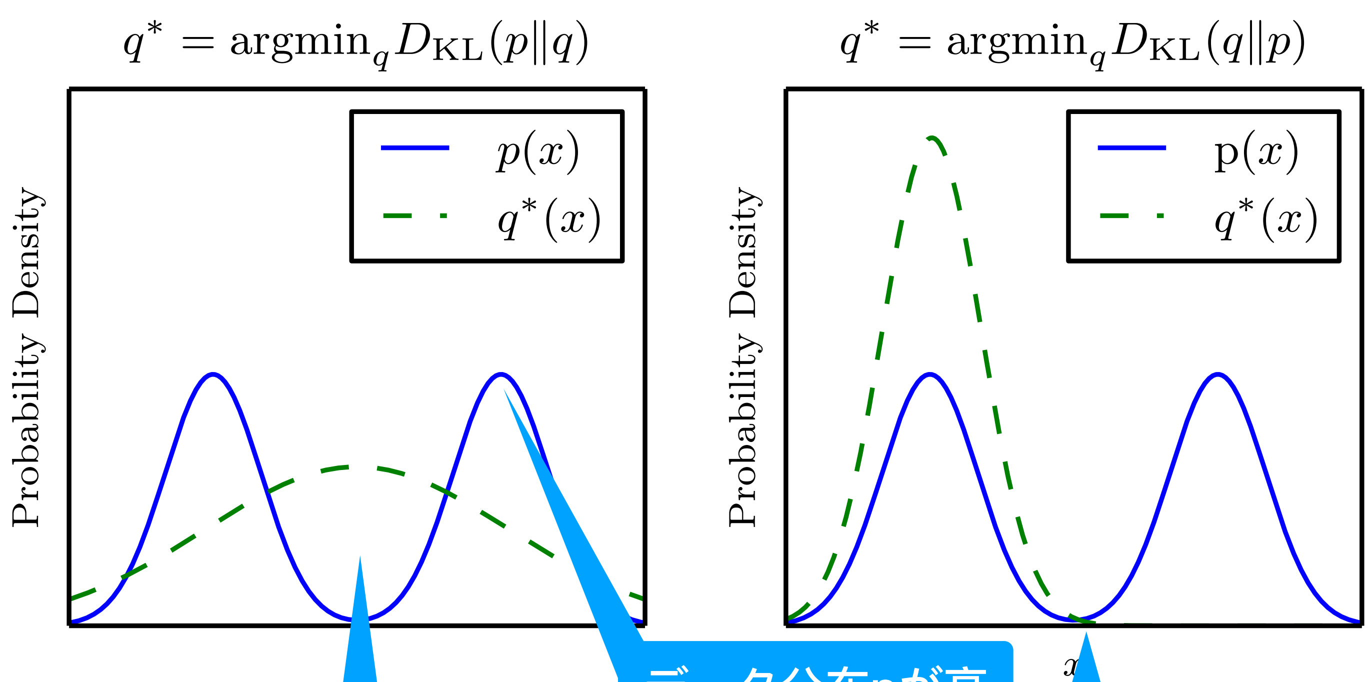
Where: $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$

$\epsilon^{(l)} \sim \mathcal{N}(0,I)$

$\mu^{(i)}, \sigma^{(i)}$ はInference networkのアウトプット。つまり$x^{(i)}$をencodeした結果。

Variational Autoencoders④VAEのプロパティ

①Variational Autoencodersから生成されたサンプルは若干ぼやけて(blurry)見える



この辺のポイント（ぼやける画像が含まれる）はトレーニングデータに出てないけど、確率が高い。

データ分布pが高いところに必ずqも高くする。qが分母だ。Maximum Likelihood学習

データ分布pが低いところに必ずqも低くする。qが分子だ

可能な理由：VAEは$D_{KL}(p_{data}\|p_{model})$を最小化する。つまり左側の場合、モデルはトレーニングデータに出てるポイントに高い確率を割り当てるが、他の出てないポイントにも高い確率を割り当てる。注１

なぜ$D_{KL}(p_{model}\|p_{data})$を使わない？計算量の考え。D(q||p)の場合q分布での期待値を計算すればOK。D(p||q)の場合p分布での期待値（p事後分布が解けない）を計算しないといけない。Chapter 19.4 p630

②素晴らしいmanifold(多様体)学習アルゴリズム。Autoencoders勉強会紹介済み

③他のVAEモデル、例えばDRAW(deep recurrent attention writer)モデル。recurrent encoderとrecurrent decoderを使う。

注1：Goodfellow氏のGAN tutorialによると、KL Divergenceの方向は原因ではないはず。Maximum Likelihoodで学習したGANモデルもsharp sampleを生成できるから。(3.2.5 Is the choice of divergence a distinguishing feature of GANs?を参考してください)

Variational Autoencoders⑤Variational方法の要注意点

①参考：[2016, Goodfellow] NIPS Tutorial: Generative Adversarial Networks

もし $q(z|x)$ と$p(z)$が弱ければ、最適化完璧でも、トレーニングデータが無限でも、ELBOと本当のlog尤度の差のせいで、p_{model} は p_{data} ではなく、他のことを学習しちゃう。The main drawback of variational methods is that, when too weak of an approximate posterior distribution or too weak of a prior distribution is used, even with a perfect optimization algorithm and infinite training data, the gap between L and the true likelihood can result in p_model learning something other than the true p_data. (2.3.2 Explicit models requiring approximation, Variational approximations)

②参考：19.4.4 Interactions between Learning and Inference, p642

学習が成功したか（変分近似から学習への害）を評価するために、$\theta^* = max_\theta logp(v;\theta)$ の時のELBOとlog尤度の差を計算するのが必要です。

残念ながら、θ*は分からない。これはまさに学習目標だ。だから、左の事象が本当に発生しているかも分からない。

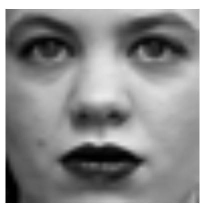
なぜなら、$\mathscr{L}(v,\theta,q) \approx logp(v;\theta)$ と $logp(v;\theta) \ll logp(v;\theta^*)$ が同時に満たす可能性がある。もし$max_q \mathscr{L}(v,\theta^*,q) \ll logp(v;\theta^*)$ だったら、θ*の時の事後確率分布p(h|v)がq(h|v)にとっては複雑過ぎて、そうすると、学習プロセスは絶対θ*に近づけない。つまり、qが弱いと、どうしても学習は失敗する。

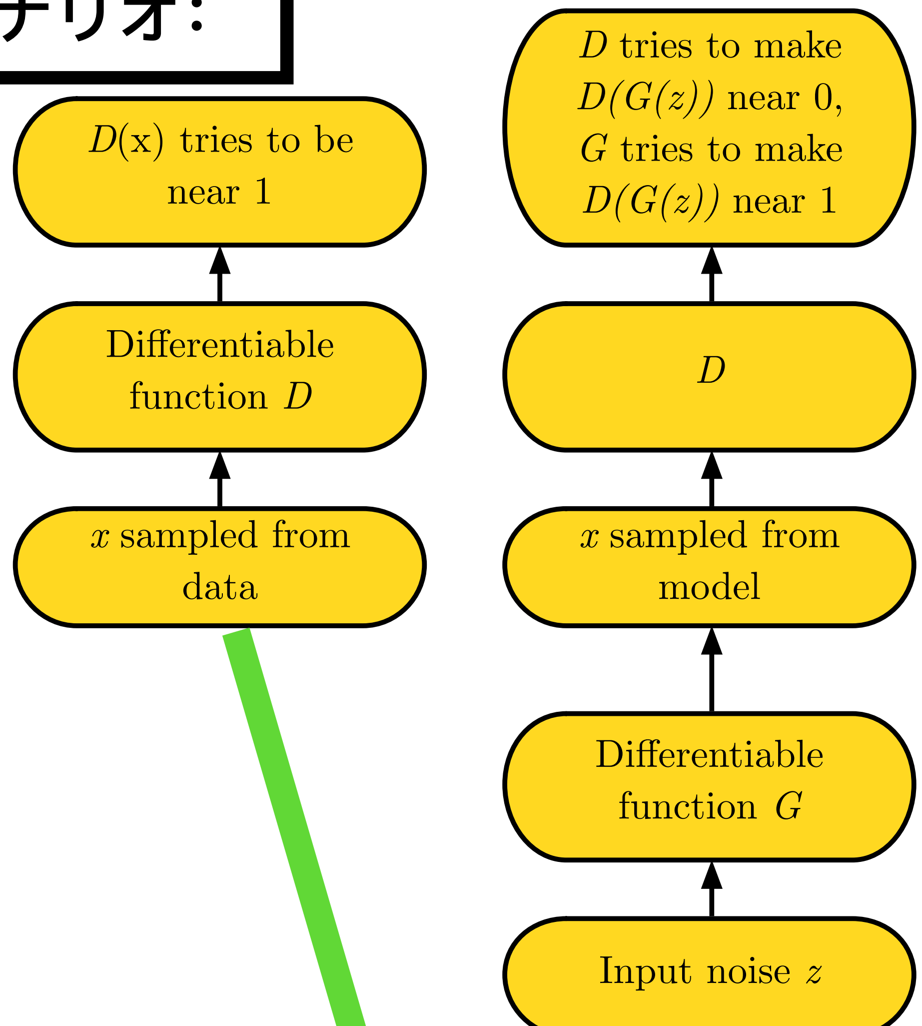
GANの設計の主なモチベーションは、近似推定もいらない、partition function gradientの近似（無向グラフモデル、Deep Boltzmann Machinesなどの学習）もいらない学習プロセスです。

Generative Adversarial Networks①Discriminator(弁別器)とGeneratorのゲーム

２シナリオ：

Discriminatorをトレーニングする時：データからのサンプル（本物）に対して、Discriminatorはできるだけ本物だと強く（確率1に近い）断定したい。





①Discriminatorをトレーニングする時：Generatorから生成したサンプルに対して、Discriminatorはできるだけ偽物だと強く（確率0に近い）断定したい。

②Generatorをトレーニングする時、パラメータが固定のDiscriminatorに対して、Generatorはできるだけ、Discriminatorが本物だと強く（確率1に近い）断定するサンプル（偽物）を生成したい。

Discriminatorのコスト関数は同じ：（GANのGeneratorのコスト関数は何バージョンがある）

$$J^{(D)}(\theta^{(D)},\theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)} logD(x) - \frac{1}{2}\mathbb{E}_{z\sim p_z(z)} log(1-D(G(z)))$$

Generative Adversarial Networks②Generatorのコスト関数

①Minimax（ゼロサムゲームの考え方、Discriminatorの得はまさにGeneratorの損、Generatorの得はまさにDiscriminatorの損）

$$J^{(G)} = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

$$V(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

収束状態

$$\theta^{(G)*} = \underset{\theta^{(G)}}{argmin}\, \underset{\theta^{(D)}}{max}\, V(\theta^{(D)}, \theta^{(G)})$$

この収束点は、Vのlocal minimaではなく、鞍点です。この鞍点、Discriminatorのparameterに対しては、local maximaです。Generatorのparameterに対しては、local minimaです。

課題：収束できないことがある

例えば：

$$v(a, b) = ab$$

player Aはaをコントロールして、最大化後abのコストを得た。player Bはbをコントロールして、最小化後-abのコストを得た。このまま繰り返して収束できない、終わらない。つまり鞍点にいけない。

②Heuristic（ヒューリスティックス）, non-saturating game（最優パフォーマンス）：

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z logD(G(z))$$

Discriminatorが間違える確率を最大化する

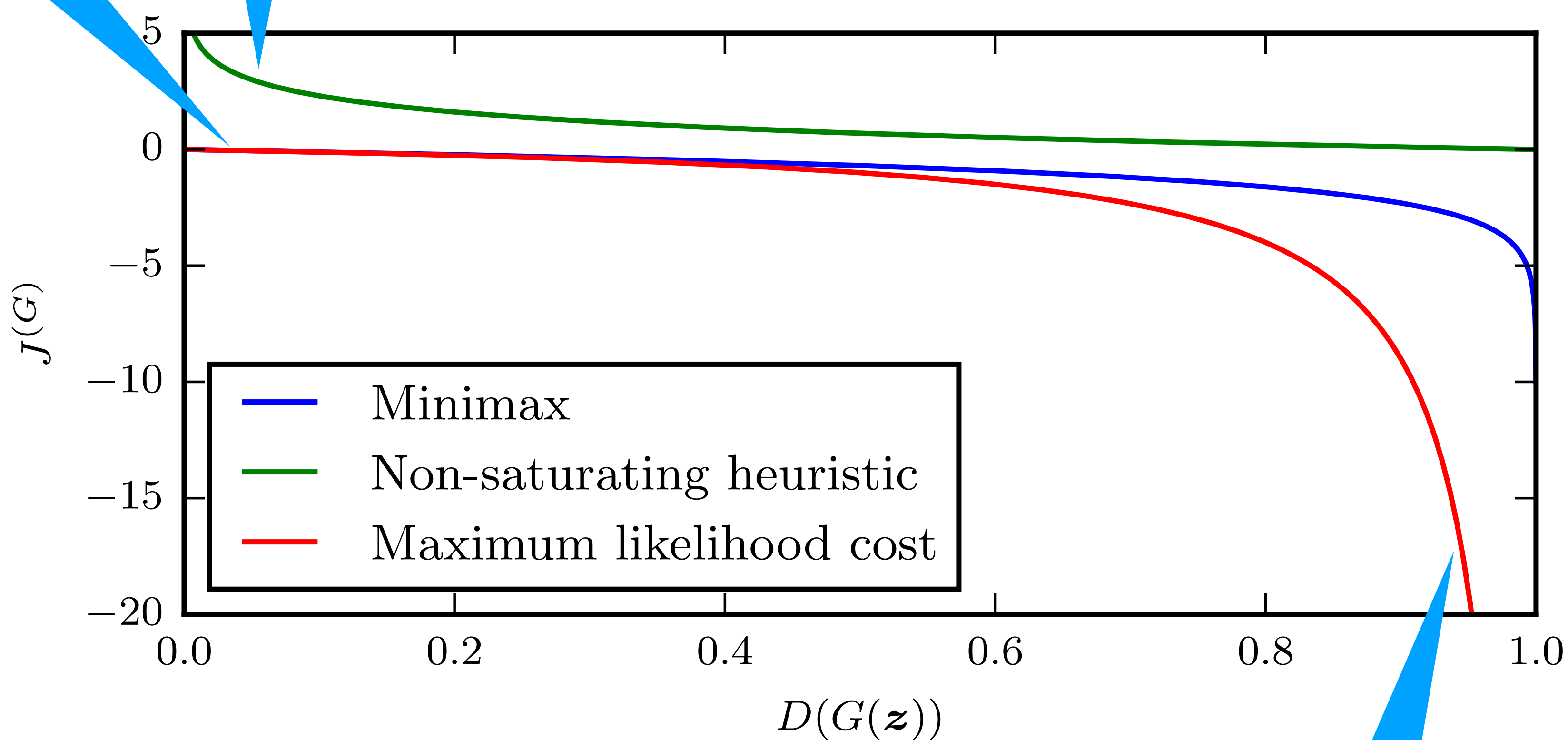
③Maximum likelihood game：KL Divergenceからの変換は略（GAN Tutorial[2016, Goodfellow]の8.3 Maximum likelihood in the GAN frameworkを参考してください）

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z exp(\sigma^{-1}(D(G(z))))$$

Generative Adversarial Networks③Generatorのコスト関数の比較

MinimaxとMaximum likelihood costは、Generatorがまだ弱い時gradientほぼゼロなので、gradient descentでは学習しにくい。

Non-saturating heuristicのみ、DiscriminatorがGeneratorから生成されたサンプルを強く否定する時も大きいgradientを持っている。つまり最初の学習はしやい。



Non-saturating heuristicとMaximum likelihood costのJ(G)は直接にデータを参照しない。データからの情報はDiscriminatorが学習したパラメータの中にある。つまりoverfittingはないはず。

また、Maximum likelihood costは、sample varianceが大きい。D(G(z))が1に近い時のgradientは結構急なので、学習は安定（収束）できない。variance reductionが必要。

（前スライドと共にGAN Tutorial[2016, Goodfellow]の3.2 Cost functionsを参考してください）

DCGANはGANのトレーニングの不安定性を解決するため提出された。

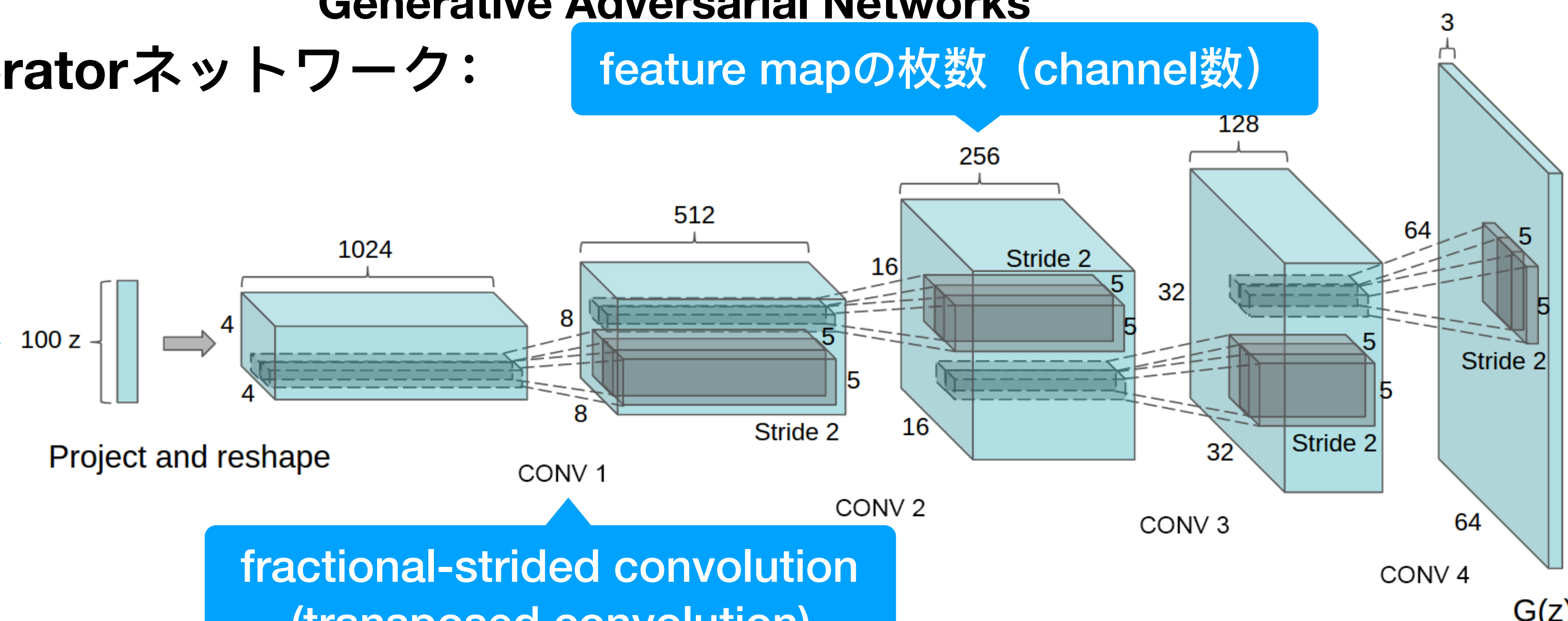
Generative Adversarial Networks④主なGANモデルのベース：DCGAN (deep convolutional GAN)

参考：[2016, Radford et al.] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

DCGANのGeneratorネットワーク：

feature mapの枚数（channel数）

VAEのdecoderも同じような実現



fractional-strided convolution (transposed convolution)

考え方：CNNのような構造のGANを作る。改善のキーポイントは主に以下の３点：

①all convolutional netを使う。poolingをなくして、strided convolutionを使う。パラメータ化のdownsampling（Discriminator）及びupsampling（Generator）は学習可能。

②全部の層にBatch Normalizationを適用する。（generatorのアウトプット層やdiscriminatorのインプット層以外Directly applying batchnorm to all layers resulted in sample oscillation and model instability.） This proved critical to get deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs.

③Adam (2014, Kingma et al.) optimizerを使う。（4. Details of Adversarial Training）その他、全結合層を無くす、GeneratorにReLU（output層のみtanh）を使う、DiscriminatorにLeakyReLUを使うこと。

Generative Adversarial Networks⑤DCGAN の実装プロセス

直感的にGANを理解するために、DCGANの実装プロセスを紹介する。コードレベルの実装は次回。参考：Deep Learning with Python, 8.5 Introduction to generative adversarial networks, p310

簡単に言うと、epoch毎に、D(x)とD(G(z))の2回トレーニングを実施

Discriminatorの学習。xはGeneratorから生成したサンプルとデータからのサンプルを混ぜたもの。

Generatorの学習。Dは固定です。Dが間違えるようにGeneratorのパラメータを調整する。D(G(z))はGANという。

①潜在スペースからランダムpoints（ランダムノイズ）を取る。
②Generatorでランダムノイズからイメージを生成する。
③生成した画像を、データセットの画像と混ぜる。
④混ぜた画像（Generatorが生成した画像のターゲットは“偽物(fake)”、データセットの画像のターゲットは“本物(real)”）にDiscriminatorをトレーニングする。
⑤潜在スペースから新しいランダムpointsを取る。
⑥ランダムpointsにDiscriminator(Generator(インプット))をトレーニングする。ターゲットは“本物(real)”です。トレーニング中Discriminator(インプット)のパラメータは固定(frozen)。

参考：18.6 Noise-Contrastive Estimation (Chapter 18 Confronting the Partition Function)

NCEのidea：良い生成モデルは、データをノイズと区別できるべき。

GANのidea：良い生成モデルは、分類器がデータと区別できないサンプルを生成できるべき

①Partition Functionも予測する： $logp_{model}(x) = log\tilde{p}_{model}(x;\theta) + c$

Partition Function（分配関数, Z(θ)）はp(x)を正規化する（分布になるように）ための関数：

$$p(x;\theta) = \frac{1}{Z(\theta)}\tilde{p}(x;\theta) \qquad Z(\theta) = \int \tilde{p}(x)dx; \qquad Z(\theta) = \sum_x \tilde{p}(x);$$

cは$-logZ(\theta)$の予測値

②ノイズモデルを導入して、教師あり学習に変換する：

$$p_{joint}(y=1) = \frac{1}{2}$$

$$p_{joint}(x|y=1) = p_{model}(x)$$

$$p_{joint}(x|y=0) = p_{noise}(x)$$

ノイズモデルを導入して、
supervised学習になれる

$$\theta, c = \underset{\theta,c}{argmax}\, \mathbb{E}_{x,y\sim p_{train}} logp_{joint}(y|x)$$

$$p_{joint}(y=1|x) = \frac{p_{model}(x)}{p_{model}(x)+p_{noise}(x)} = \frac{1}{1+\frac{p_{noise}(x)}{p_{model}(x)}} = \frac{1}{1+exp(log\frac{p_{noise}(x)}{p_{model}(x)})}$$

$$= \sigma(-log\frac{p_{noise}(x)}{p_{model}(x)}) = \sigma(logp_{model}(x) - logp_{noise}(x))$$

学習中もしノイズモデルが進化しなければ、生成モデルも強くなれない。もし学習中常に現状のモデル分布を次のステップのノイズ分布として使えば、生成モデルはどんどん強くなれそう（self-contrastive estimation）。この考え方はGANと同じ。The key limitation of NCE is that its “discriminator” is defined by the ratio of the probability densities of the noise distribution and the model distribution, and thus require the ability to evaluate and back propagate through both densities. [2014, Goodfellow et al.] Generative Adversarial Nets

補足：①VAEとGANの違い②まだ読んでいない資料

①GANはxに対して微分を計算する必要があるので、離散サンプル（顕在変数）を扱えない。VAEはzに対して微分を計算する必要があるので、離散潜在変数を扱えない。参考：[2014, Goodfellow et al.] Generative Adversarial Netsの2 Related Work。arXiv:1406.2661v1にはない。GANs require differentiation through the visible units, and thus cannot model discrete data, while VAEs require differentiation through the hidden units, and thus cannot have discrete latent variables.

$$GAN : Discriminator(Generator(z))$$

GANの学習、backpropagationできるために、Generator(z)が離散になっちゃダメ

VAEの学習、backpropagationできるために、Encoder(x)が離散になっちゃダメ

$$VAE : Decoder(Encoder(x))$$

②

[2017, Kingma et al.] An Introduction to Variational Autoencoders (https://arxiv.org/abs/1906.02691)

また、沢山の最新GAN論文。