# Variational AutoEncoder

K.Nitta

# 自己紹介

- 名前 : K.Nitta
- 所属 : 金沢の大学(石川の高専から編入)
- Twitter : @one_meets_seven

# もくじ

- 変分近似
- Variational AutoEncoder

# 変分近似

# ベイズの基礎

- 計算ルール
  - 和 $p(x) = \int p(x,y)dy$
  - 積 $p(x,y) = p(y|x)p(x)$
- ベイズの定理

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

  - p(y|x)をp(x|y)にひっくり返すことができる(その逆も然り)

# 機械学習らしい概念

- パラメータθ データ点の集合D

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

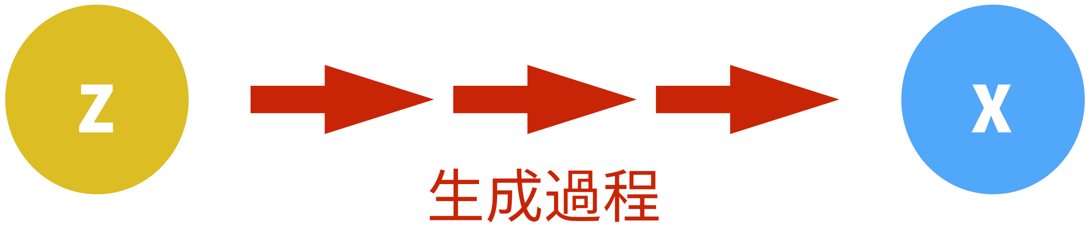
  - p(D|θ) 尤度関数
    - パラメータがある特定の値に条件付けされたとき、データDがどれだけモデルから発生しやすいか
  - p(θ) 事前分布
    - θの分布に関する事前の仮説
  - p(θ|D) 事後分布
    - 尤度関数を通して更新されたθの分布
  - p(D) エビデンス(周辺尤度)
    - 事後確率の正規化を保障 $p(x|\mathcal{D}) = \sum_{\theta} p(x|\theta, \mathcal{D})p(\theta|\mathcal{D})$

# 生成モデル

- 観測したデータxはとある因子zから生成されたと考える


  - 因子zはたとえば正規分布(ガウス分布)と考える

- 生成過程は一般的に未知




  - 教師なしデータxを活用して生成過程を逆向きに辿ってxからzを推論できる

# 変分近似(Variational Bayes)

- log p(x)を最大化したい(対数尤度)
  - データが発生する分布p(x)が大きいほど実際のデータに即している
- 真の事後分布p(z|x)は未知
  - 近似の事後分布**q(z|x)**を仮定
- 対数尤度を分解
  - q(z|x)とp(z|x)を測るKL-divergence項と下限Lに分解
    - 更に下限Lは事後分布q(z|x)と事前分布p(z)を測るKL-divergence項と事後分布q(z|x)からみた対数尤度p(x|z)のcross entropy項に分解できる
    - 下限Lを大きくすれば、KL(q||p)項が小さくなる(非負)。
    - よって下限Lを大きくすることで対数尤度を大きくできる。(ELBO)

# 対数尤度の分解

$$
\begin{aligned}
\log p(x) &= \log p(x) \int q(z|x) dz \\
&= \log \frac{p(x,z)}{p(z|x)} \int q(z|x) dz \\
&= \int q(z|x) \log \frac{q(z|x)}{p(z|x)} \frac{p(x,z)}{q(z|x)} dz \\
&= -\int q(z|x) \log \frac{p(z|x)}{q(z|x)} dz + \int q(z|x) log \frac{p(x,z)}{q(z|x)} dz \\
&= KL(q(z|x)||p(z|x)) + \mathcal{L}
\end{aligned}
$$

$$
\log p(x) = KL(q(z|x)||p(z|x)) + \mathcal{L}
$$

$$
\mathcal{L} \;=\; -KL(q(z|x)||p(z)) \;+\; \mathbb{E}_{q(z|x)}[\log p(x|z)]
$$

# 変分近似とは

- 下限L(x)を最大化すれば、尤度p(x)の最大化と近似できる。
  - このアプローチをDeep Neural Networkと組み合わせ応用したものが **Variational AutoEncoder** (Kingma (2013); Rezende et al. (2014))

# Variational AutoEncoder

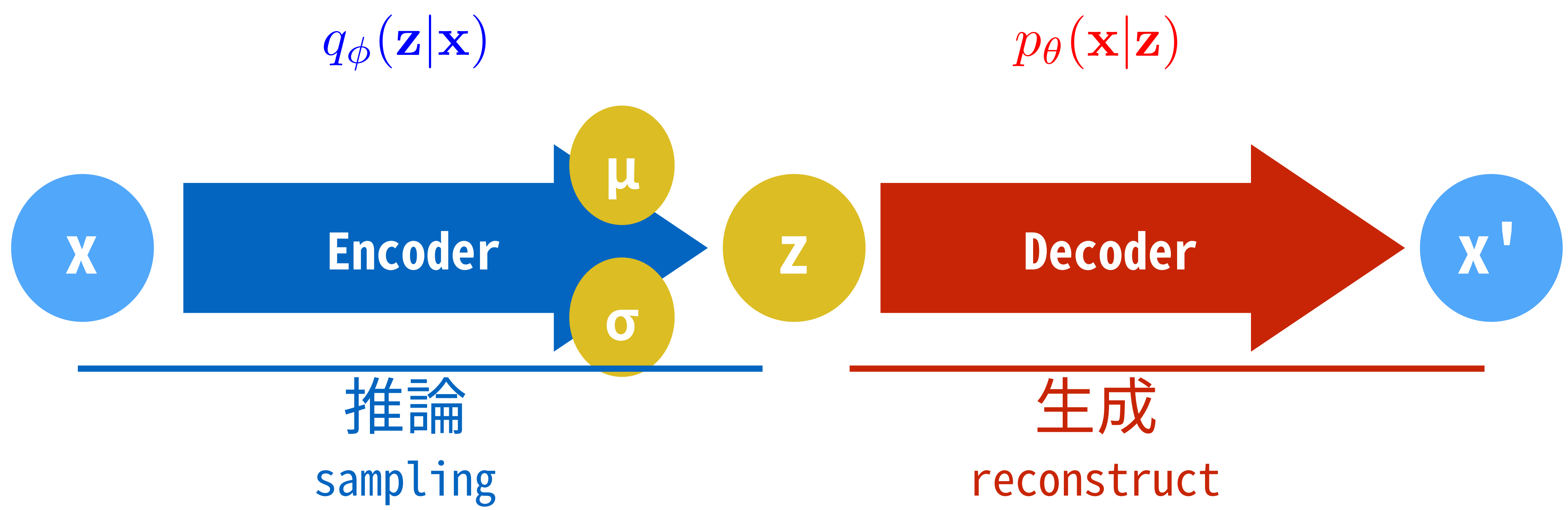
# 概要

- ニューラルネットワークでVariational Bayes(変分近似)
- 微分可能な推論(inference)+生成(generative)モデル

  $q_\phi(\mathbf{z}|\mathbf{x})$ $p_\theta(\mathbf{x}|\mathbf{z})$

- オートエンコーダのEncoderが $q_\phi(\mathbf{z}|\mathbf{x})$
  - つまりデータxから因子zを推論する(パラメータφ)
- Decoderが $p_\theta(\mathbf{x}|\mathbf{z})$
  - つまりEncoderで求めたzからもう一度データx'を生成する(パラメータθ)

# 図




生成モデルの損失関数は-logp(x) (負の対数尤度)
下限L(x)は

$$\mathcal{L}(x;\theta,\phi) = \underbrace{-KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{regularization term(正規化項)}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruct error(復元誤差)}}$$

regularization term(正規化項) reconstruct error(復元誤差)

# ~Gaussian~

$$\mathcal{L}(x;\theta,\phi) = \underline{-KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))} + \underline{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}$$

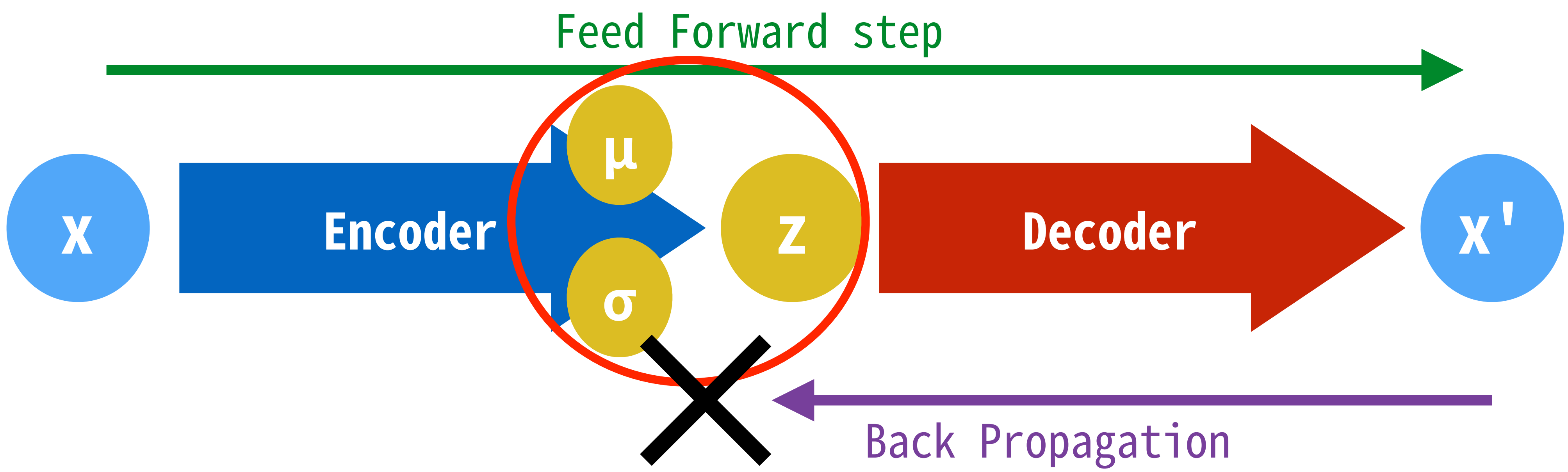
$$\mathcal{L}(x;\theta,\phi) = -\frac{1}{2}\sum^{d}(1+\log(\sigma_d^2)-\mu_d^2-\sigma_d) + \frac{1}{L}\sum^{l}\log p_\theta(\mathbf{x}|z_l)$$

zがガウス分布と仮定している場合、
KL-divergenceは解析的に求められる

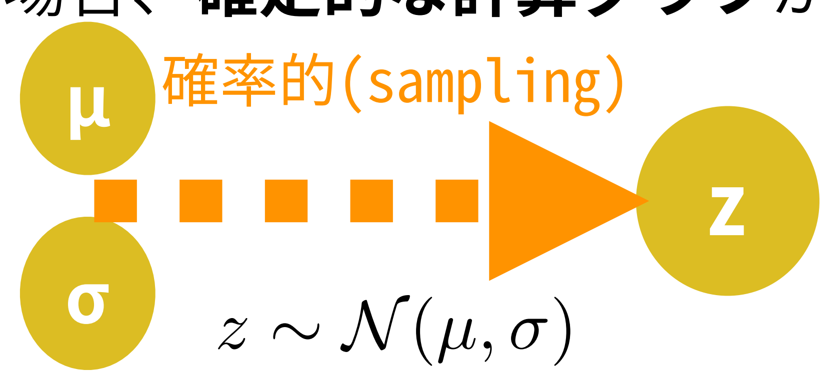
q(z|x)からサンプリングしたL個で
期待値を近似する
(論文では大きなミニバッチサイズを
計算することでL=1でも
いい結果が得られる)

# 確率的計算の誤差逆伝播

- -L(x;θ, φ)を損失関数にして勾配降下法をしていきたい
  - しかし、フィードフォワードステップに確率的計算がある

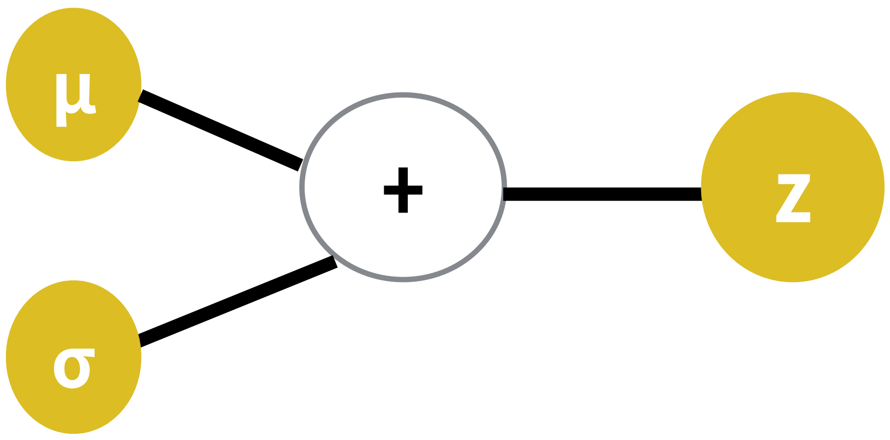



- 赤丸の箇所が $z \sim \mathcal{N}(\mu, \sigma)$ の場合、**確定的な計算グラフ**が途切れてしまう
  - よって誤差逆伝播が行えない



# Reparametrization trick(SGVB)

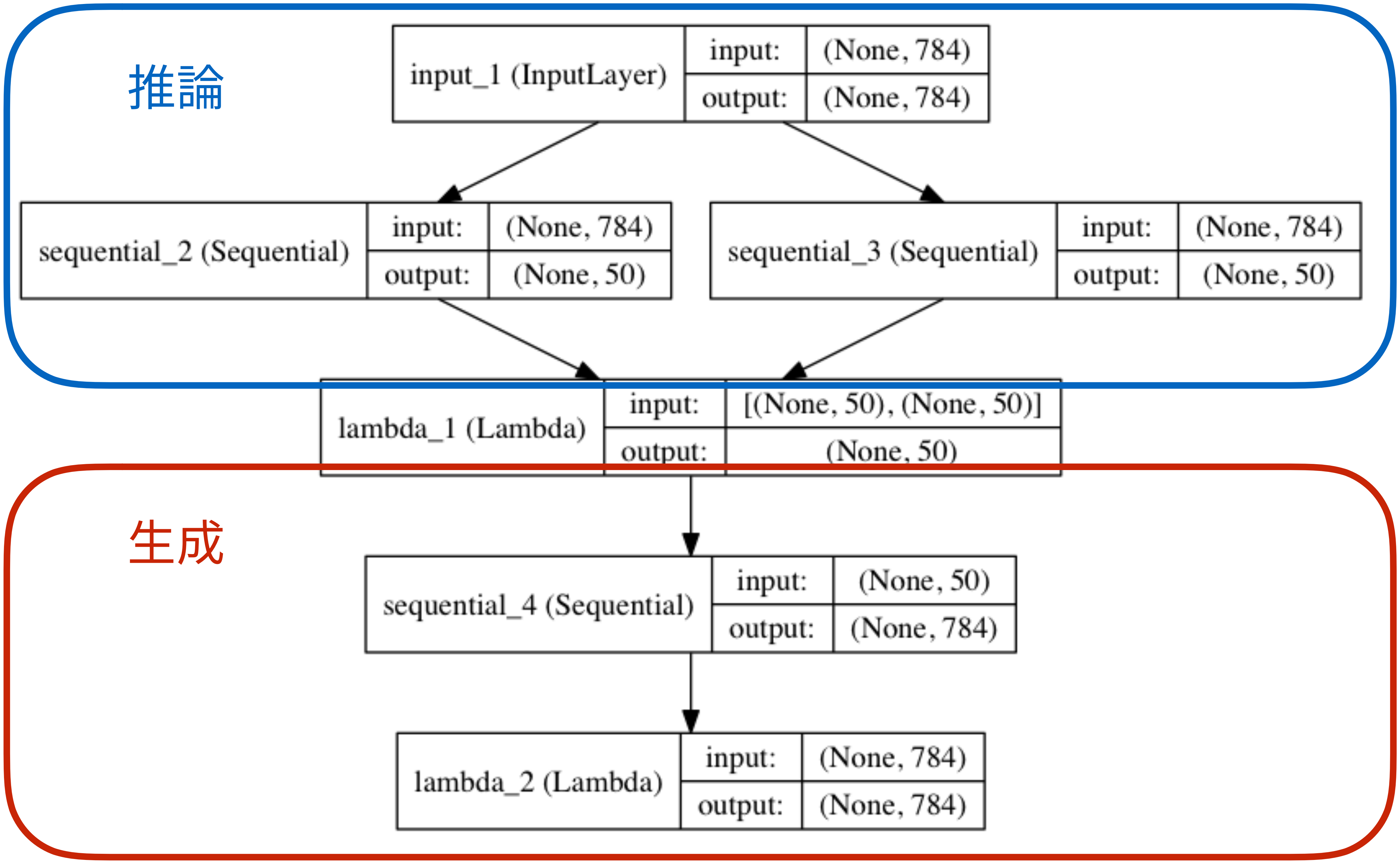
- $\epsilon \sim \mathcal{N}(0, I)$ をサンプリングして $z = \mu + \epsilon\sigma$ で近似する
  - 無事計算グラフが逆向きでも繋がる(微分可能)



# 実装(Keras)（※自分用メモ）

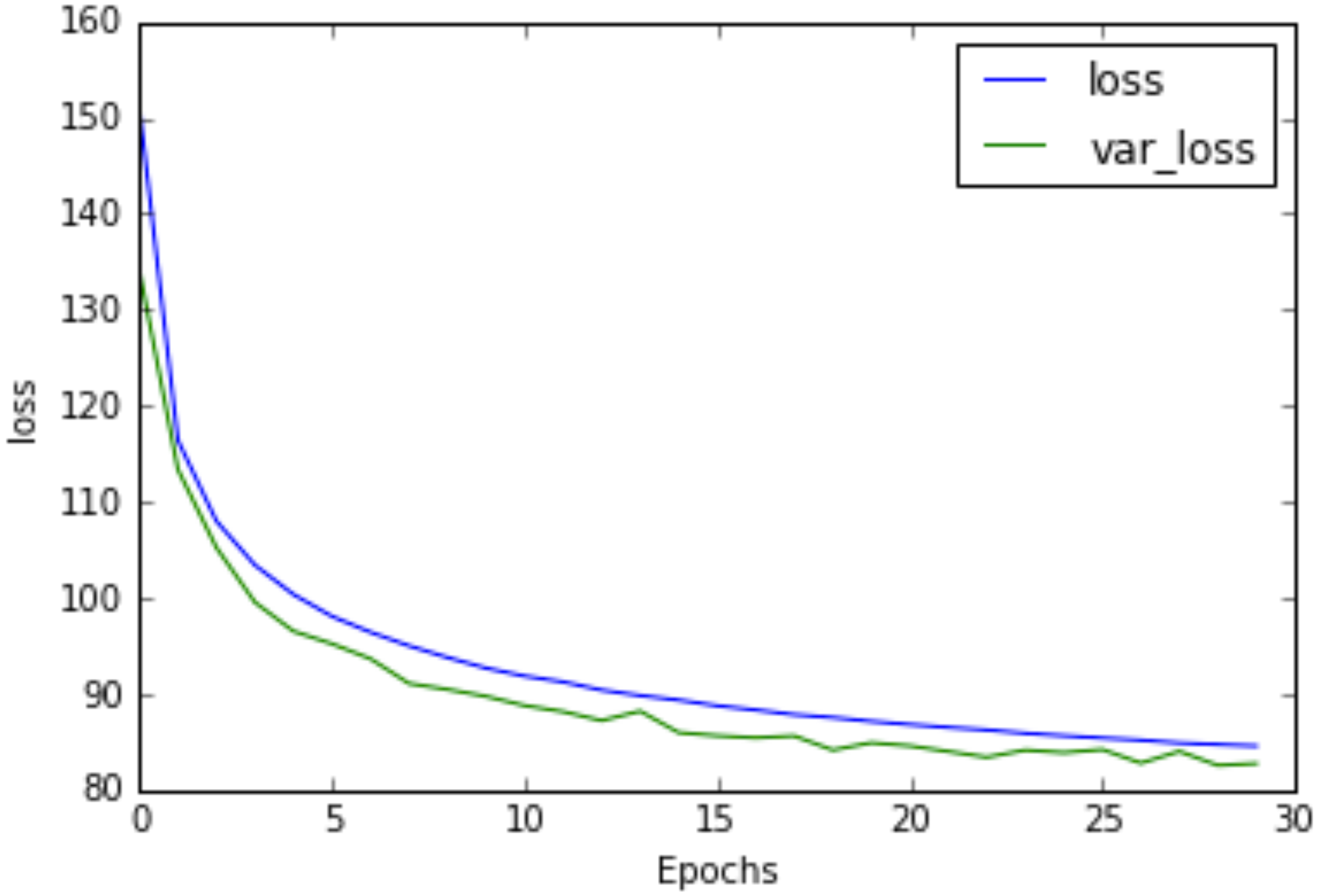
- 推論 $q_\phi(\mathbf{z}|\mathbf{x})$ と生成 $p_\theta(\mathbf{x}|\mathbf{z})$ をSequential()で分けて深層にする
  - self.q_z_xやself.p_x_zのようにインスタンス変数を保持する
- training
  - 入力はx, 出力はp(x|z)のサンプリング
  - cost関数は-L(x;ϕ,θ)
- encoder
  - 入力はx, 出力はq(z|x)のサンプリング
- decoder
  - 入力はz, 出力はp(x|z)のサンプリング

# 実装(Keras) (※自分用メモ)



# 結果

- MNISTタスク
  - 入力は784次元、zは50次元
    - Encoder部分 784-300-300-50
    - Decoder部分 50-300-300-784
  - 活性化関数
    - 中間層はsoftplus, 出力はsigmoid
  - 最適化
    - Adam
    - ミニバッチサイズ100
  - エポック数30
    - loss: 84.5658
    - val_loss: 82.7433



# 2つの潜在変数間を可視化

In [24]:

```
target1 = X_train[0:1] # 5
target2 = X_train[8:9] # 1
latent1 = encoder.predict(target1, batch_size=1)
latent2 = encoder.predict(target2, batch_size=1)

fig = plt.figure(figsize=(14, 14))
for i, d in enumerate(np.linspace(0, 1, 10)):
    latent = latent1 + d * (latent2-latent1)
    reconstruct_image = decoder.predict(latent, batch_size=1)
    ax = fig.add_subplot(1, 10, i+1, xticks=[], yticks=[])
    ax.imshow(reconstruct_image.reshape(28, 28), 'gray')
plt.show()
```

# VAEまとめ

- AutoEncoderにプラス要素
  - 中間層にノイズεが加わった
  - 損失関数に正則化項(KL-divergence)を加えた
- 生成モデル
  - データから生成過程を推論できる
  - 生成過程に沿った新しいデータを生成できる
  - 二点間のアナロジーを再生成できる
  - 対数尤度は求められない
  - KL項を解析的に導出できるように単純な正規分布(例えば)を仮定にする