Variational Autoencoders

Prof. Jefersson A. dos Santos

jefersson@dcc.ufmg.br

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

Roteiro

Aulas anteriores

- Otimização
- Redes Neurais
- Redes Convolucionais
- Aplicações de CNNs
- Autoencoders

Aula de Hoje

- Variational Autoencoders (VAE)
  - Definição e Motivação
  - Arquitetura de um VAE
  - Modelagem probabilística
  - Função de Perda
  - Backpropagation
    - Reparametrization Trick
  - Aplicações

Variational Autoencoders

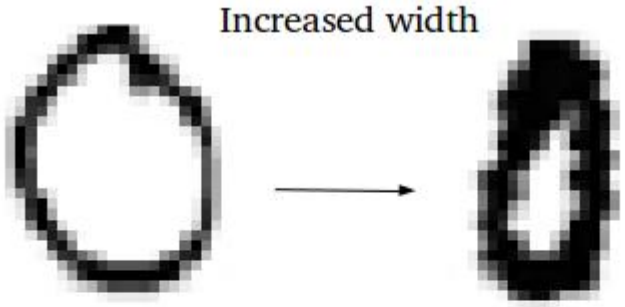
Introdução

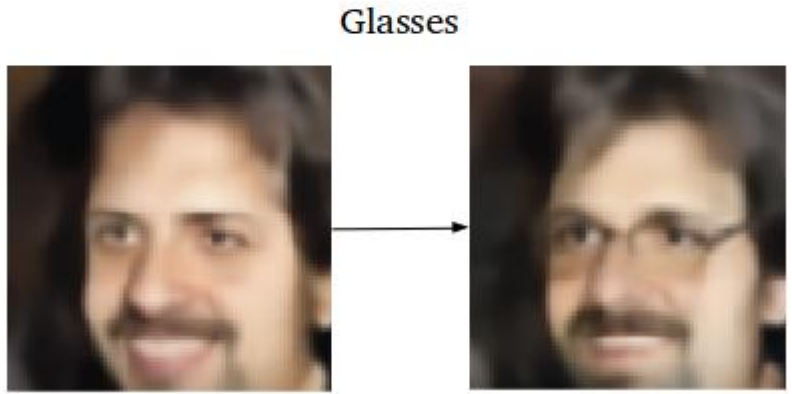
Variational autoencoder

- VAE = variational autoencoder
- As VAEs são um exemplo de modelos chamados modelos generativos.
- Esses modelos podem ser usados para gerar exemplos de dados de entrada.
- Eles aprendem os padrões estatísticos dos dados de entrada.
- Depois de treinar um VAE para aprender features de alguns dados x em R^n
    - (por exemplo, x poderia ser imagens de rostos)
- O VAE seria capaz de gerar novas amostras de imagens de rostos similares aos da amostra de treinamento.

Variational autoencoders

- Ao usar o VAE, você pode gerar uma saída nova e aleatória, semelhante aos dados de treinamento.
- Você pode também alterar ou explorar variações nos dados que já possui.
- Pode fazer isto não apenas de forma aleatória, mas em uma direção específica desejada.
- É aqui que os VAEs funcionam bem





Variational Autoencoders

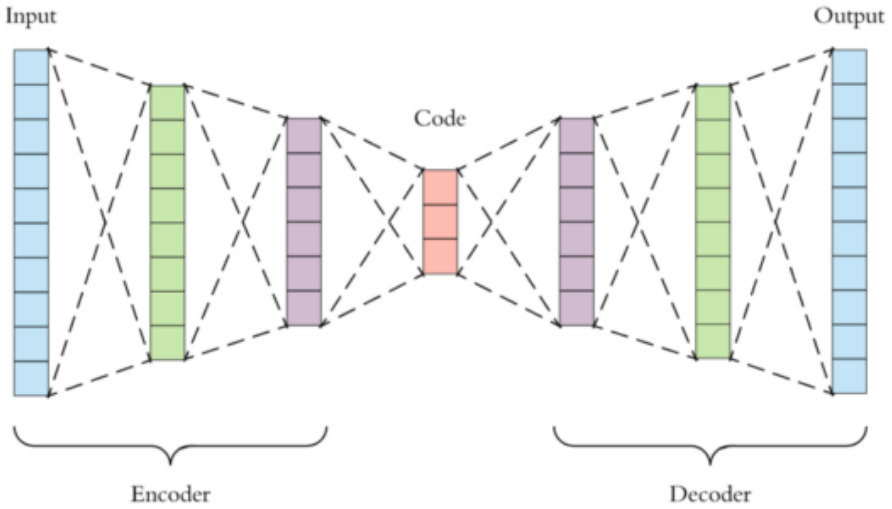
Diferenças para os Autoencoders

AE versus VAE

Autoencoder:
- determinístico
- codifica diretamente a entrada x

Por exemplo, camada CODE pode conter um vetor *encoding* de dimensão 30

Gera dados similares à entrada
Não existe variabilidade
Não existe flexibilidade

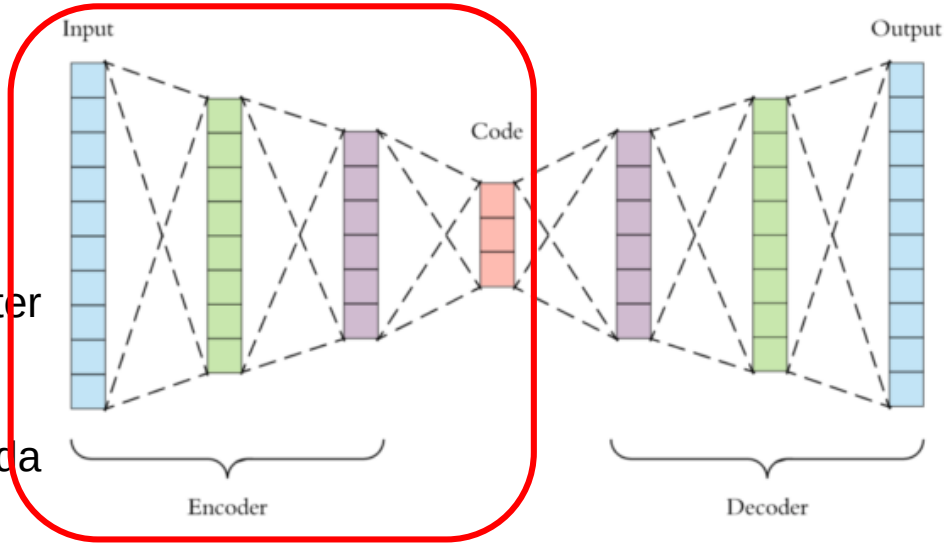


AE versus VAE

Autoencoder:
- determinístico
- codifica diretamente a entrada x

Por exemplo, camada CODE pode conter um vetor *encoding* de dimensão 30

Gera dados similares à entrada
Não existe variabilidade
Não existe flexibilidade



interesse

AE versus VAE

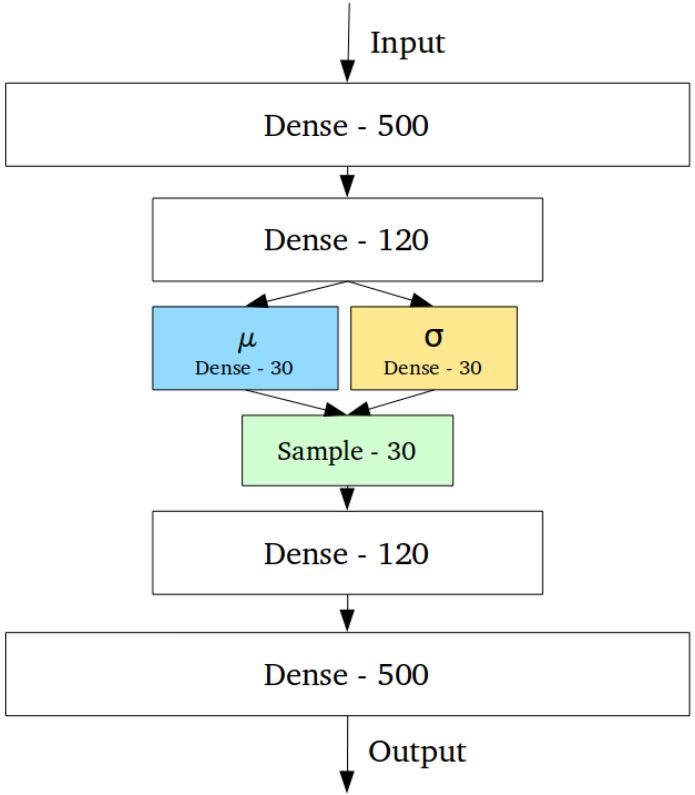
Ao invés de codificar **um** vetor de dim 30 encoding a entrada…

VAE codifica DOIS vetores de dim 30:

- um vetor de médias (dim 30)
- um vetor de DPs (dim 30)

A seguir, gera um vetor aleatório de dimensão 30 (usualmente gaussiano)

- usa o vetor de médias
- usa o vetor de DPs



AE versus VAE

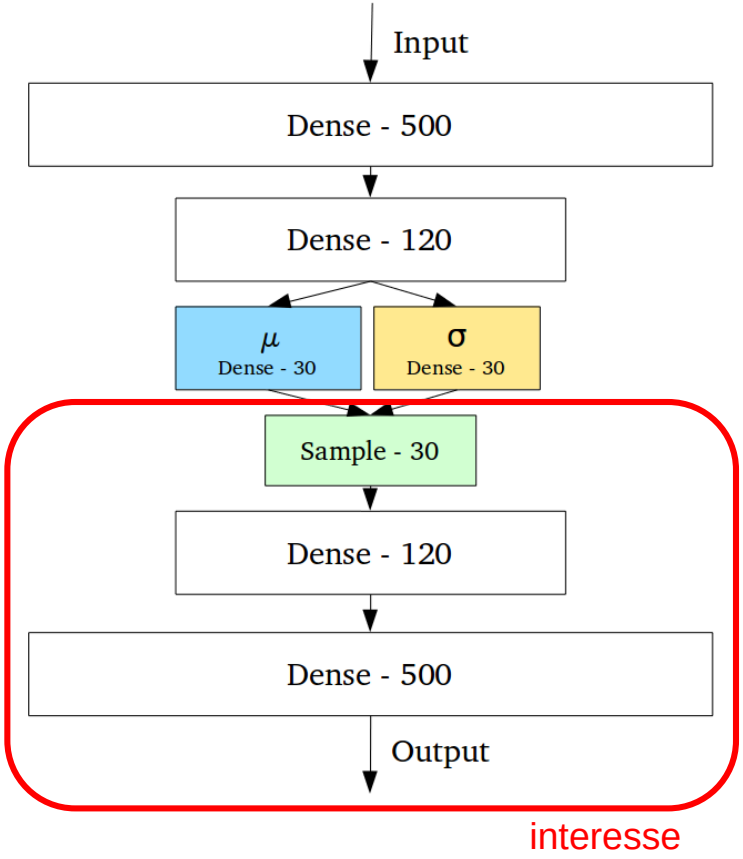
Ao invés de codificar **um** vetor de dim 30 encoding a entrada…

VAE codifica DOIS vetores de dim 30:

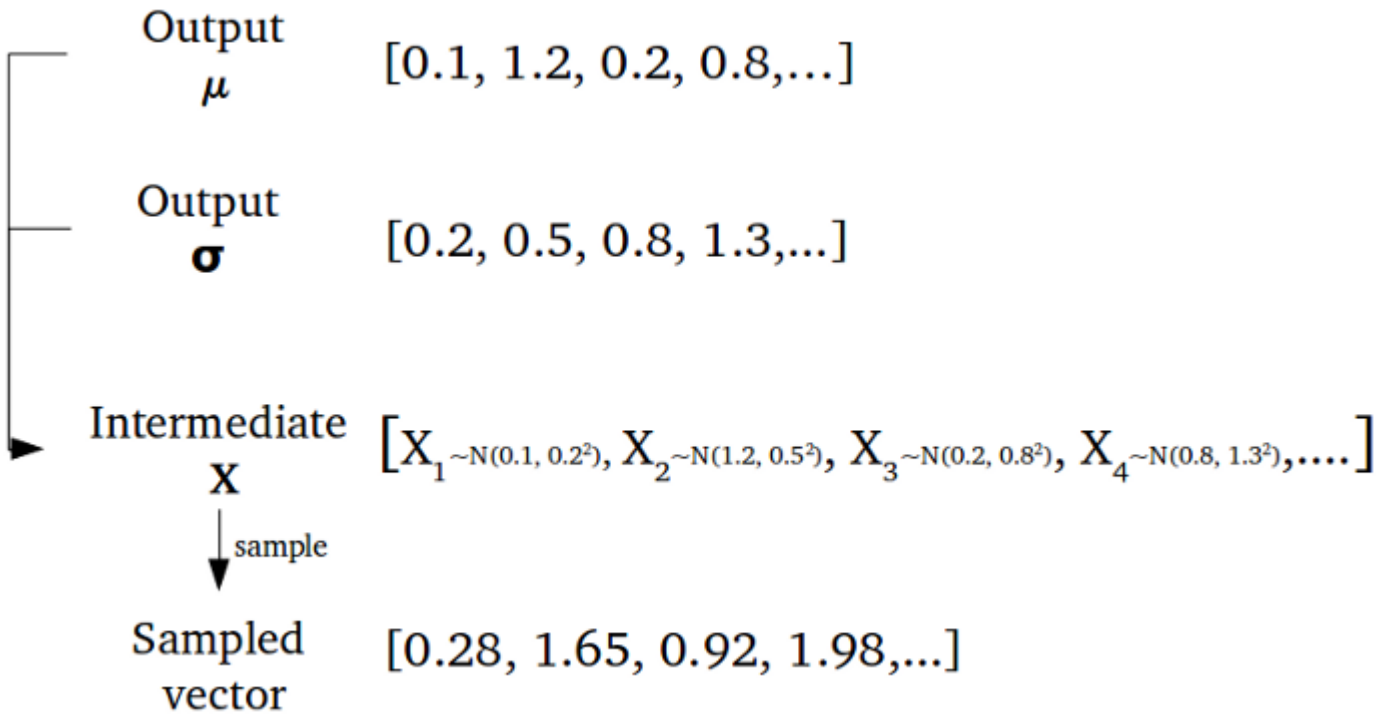
- um vetor de médias (dim 30)
- um vetor de DPs (dim 30)

A seguir, gera um vetor aleatório de dimensão 30 (usualmente gaussiano)

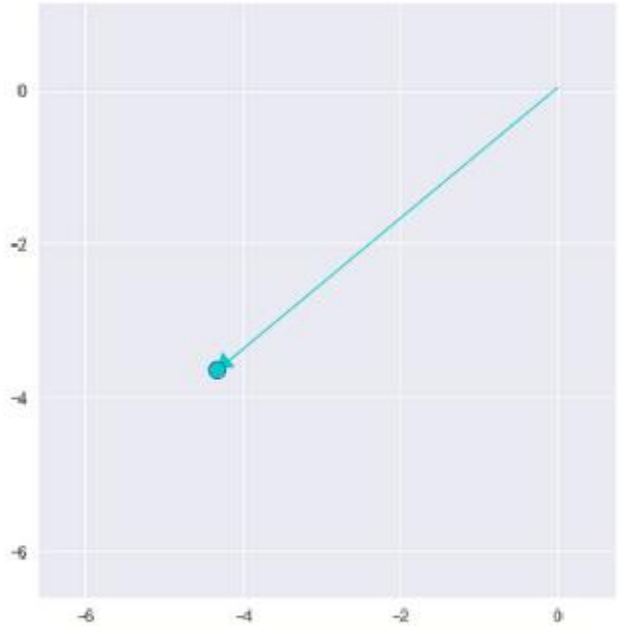
- usa o vetor de médias
- usa o vetor de DPs



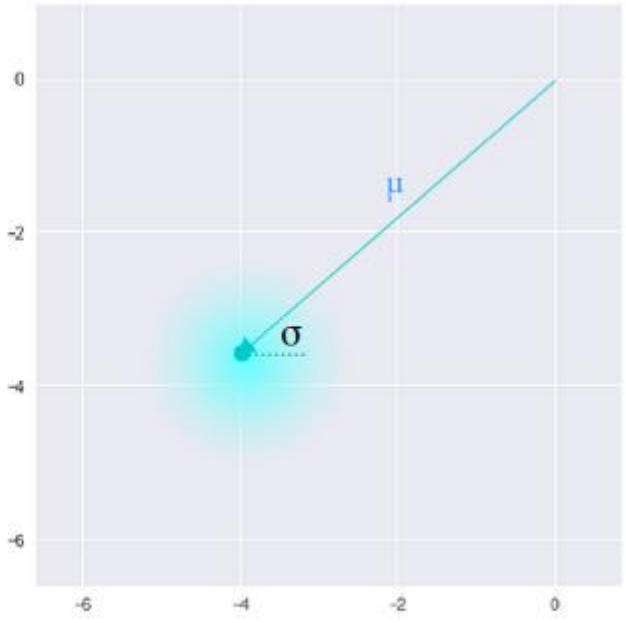
Camada interna do VAE



Autoencoders versus variational autoencoder



Standard Autoencoder
(direct encoding coordinates)



Variational Autoencoder
(μ and σ initialize a probability distribution)

Um ponto versus uma distribuição

- O vetor de médias controla onde a codificação de uma entrada deve ser centralizada
- O vetor de DPs controla a “área”, quanto a codificação pode variar o redor da média.
- Novas codificações são geradas **aleatoriamente** de qualquer lugar dentro do "círculo" (a distribuição)
- O decodificador aprende não apenas um único ponto no espaço latente referente a uma amostra dessa classe
- Ele aprende outros pontos próximos que se referem ao mesmo exemplo.

Variational Autoencoders

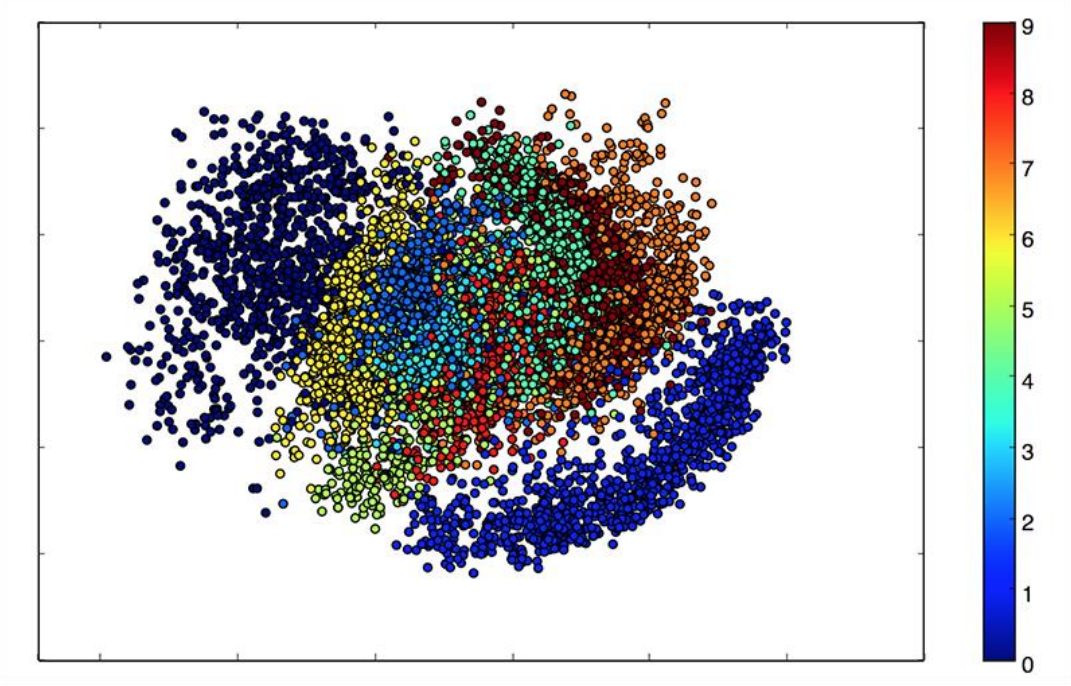
Funcionamento

Como funcionam?

- Uma rede encoder transforma as amostras de entrada x em dois parâmetros em um espaço latente:
  - z_{mean} e $z_{log\ sigma}$.
  - A seguir, geramos aleatoriamente pontos z de distribuição normal:
    - $z = z_{mean} + \exp(z_{log\ sigma}) * epsilon$,
    - onde epsilon é uma v.a. normal N(0,1).
- Uma rede decoder mapeia esses pontos z do espaço latente de volta para o espaço dos dados de entrada originais.
- Os parâmetros das duas redes são treinados através de uma função de perda que soma dois termos:
  - uma perda de reconstrução que força as amostras decodificadas a parecer com as entradas iniciais (como nos autoencoders)
  - e a divergência de Kullback-Leibler entre a distribuição latente aprendida e a distribuição anterior, atuando como um termo de regularização.

Adaptado das aulas do Prof. Renato Assunção

Exemplo com MNIST com z bivariado

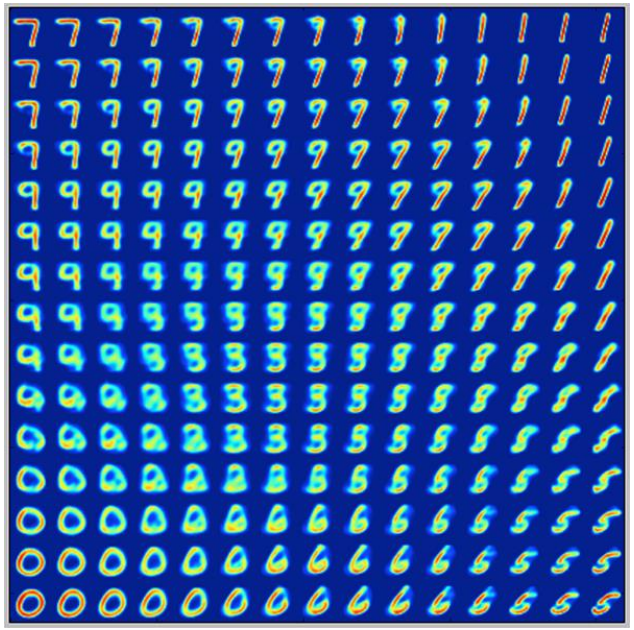


Cada um desses clusters coloridos é um tipo de dígito. Clusters próximos são dígitos que são estruturalmente semelhantes (isto é, dígitos que compartilham informações no espaço latente).

De https://blog.keras.io/building-autoencoders-in-keras.html

Gerando novos dígitos

Como o VAE é um modelo generativo, também podemos usá-lo para gerar novos dígitos.
Vamos varrer o plano latente (z_1, z_2), amostrando pontos latentes em intervalos regulares e gerando o dígito correspondente para cada um desses pontos.
Isso nos dá uma visualização da variedade latente que "gera" os dígitos MNIST.



De https://blog.keras.io/building-autoencoders-in-keras.html

Autoencoder

- Vimos autoencoders:
  - Encoder: redução de dimensionalidade nos dados, $\mathbf{x} \in \mathbb{R}^n$ obtendo features .$\mathbf{z} \in \mathbb{R}^d$
  - Decodificador: Mapeie os recursos $\mathbf{z} \in \mathbb{R}^d$ para reproduzir a entrada, $\hat{\mathbf{x}} \in \mathbb{R}^n$
- Assim, o autoencoder implementa o seguinte problema:
  - Seja $\mathbf{x} \in \mathbb{R}^n$
    $f(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^d$
  - e $g(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^n$
  - Defina $\hat{\mathbf{x}} = g(f(\mathbf{x}))$
  - Defina uma função de perda, $\mathcal{L}(\hat{\mathbf{x}}, \mathbf{x})$
  - minimizar L em relação aos parâmetros de f () e g ().
- Existem diferentes funções de perda; uma comum é a perda quadrada: $\mathcal{L}(\hat{\mathbf{x}}, \mathbf{x}) = ||\hat{\mathbf{x}} - \mathbf{x}||^2$

Adaptado das aulas do Prof. Renato Assunção

Variational Autoencoder

- VAEs foram foram introduzidos por Kingma e Welling(2014).
- Um bom tutorial é Doersch (2016) e as aulas de J.C.Kao na UCLA.
- Em vez de aprender f(.) e g(.), os VAEs aprendem:
  - a distribuição de probabilidade dos encodings dada a entrada
  - e a distribuição da entrada dadas as ativações dos encodings
- Isto é, VAEs fornecem versões probabilísticas de f(.) e g (.).
- O VAE aprenderá:
  - $p(\mathbf{z}|\mathbf{x})$: a distribuição dos encodings z dada a entrada x.
  - $p(\mathbf{x}|\mathbf{z})$: a distribuição da entrada dadas as características z.
- Essas distribuições são parametrizadas por redes neurais
  - elas podem expressar transformações não-lineares
  - são treinados via stochastic gradient descent

Adaptado das aulas do Prof. Renato Assunção

Diagrama conceitual

- Onde vamos chegar →
- Este diagrama conceitual do VAE é semelhante a um
- autoencoder, com uma grande mudança:
  - em vez de inferir z diretamente de x
  - $\hat{\mathbf{x}}$ diretamente de z
  - nós inferimos distribuições de probabilidade
- Em vez de aprender z = f (x), aprendemos q(z|x)
- Em vez de $\hat{\mathbf{x}}$ = g(z), aprendemos p(x|z).
- Para obter valores para z e $\hat{\mathbf{x}}$ nós simulamos dessas distribuições aprendidas.

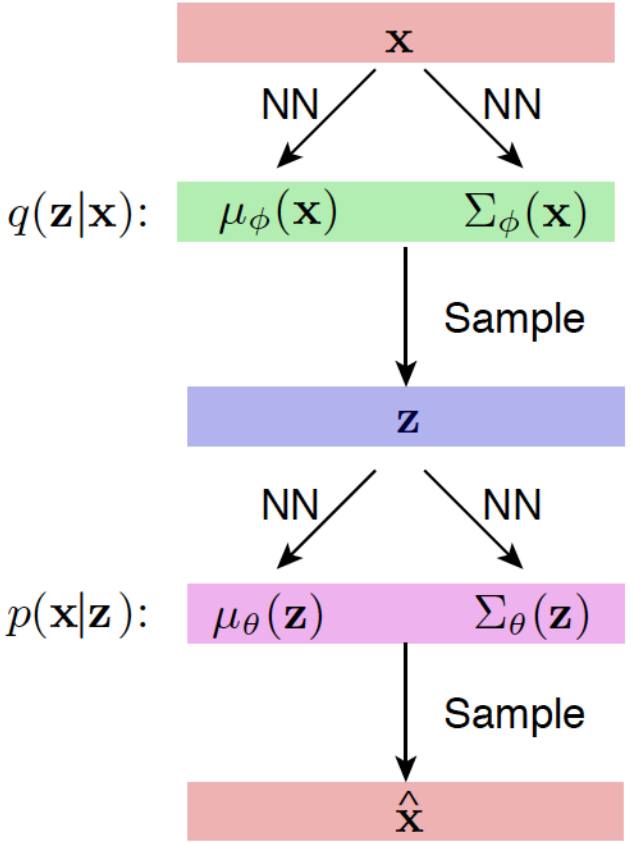
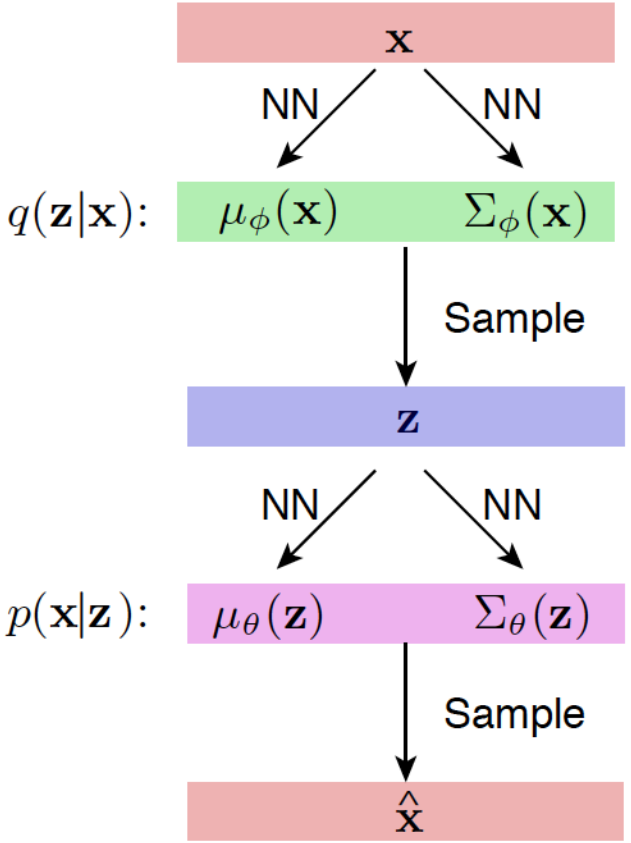


Diagrama conceitual

- Onde vamos chegar →
- Este diagrama conceitual do VAE é semelhante a um
- autoencoder, com uma grande mudança:
  - em vez de inferir z diretamente de x
  - $\hat{\mathbf{x}}$ diretamente de z
  - nós inferimos distribuições de probabilidade
- Em vez de aprender z = f (x), aprendemos q(z|x)
- Em vez de $\hat{\mathbf{x}}$ = g(z), aprendemos p(x|z).
- Para obter valores para z e $\hat{\mathbf{x}}$ nós simulamos dessas distribuições aprendidas.

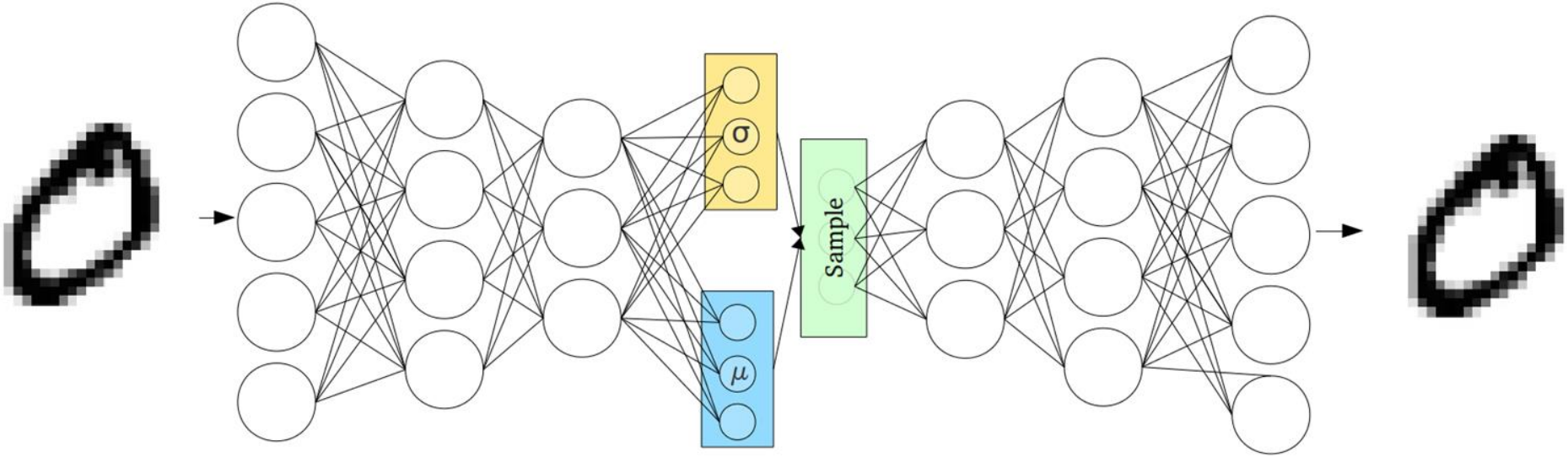
VAE é uma versão probabilística do autoencoder



Formulando o VAE

- Abordaremos o VAE a partir do contexto de um modelo generativo.
- Queremos gerar amostras da distribuição de dados, p(x).
- Em vez de inferir p(x) diretamente, podemos usar o que chamamos de modelos de variáveis latentes.
- Modelos de variáveis latentes modelam os dados, x, como decorrentes de uma variável não observada (e, portanto, latente), z.
- Pode não ser fácil modelar p(x) diretamente, mas pode ser mais fácil escolher alguma distribuição p(z) e, em vez disso, modelar p (x | z).
- Intuitivamente, isso significa que nós modelamos como x surge a partir de algumas variáveis latentes, z, que possuem sua própria variabilidade.

VAE



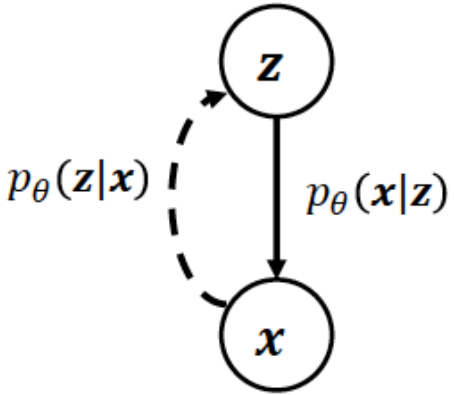
Mistura elementos de redes neurais + modelos gráficos probabilísticos!

Modelos Gráficos Probabilísticos

(Probabilistic Graphical Models, PGM, em inglês)

- Sub-área de pesquisa dentro de Aprendizagem de Máquina
- Objetivo:
  - criar uma estrutura matemática unindo grafos e probabilidade que sirva para modelar situações complexas que envolvam aleatoriedade ou incerteza.

Nós: variáveis aleatórias
Arestas: uma representação da estrutura de dependência condicional entre as variáveis.



Modelos Gráficos Probabilísticos

(Probabilistic Graphical Models, PGM, em inglês)

- Principais tipos:
  - redes bayesianas (grafos direcionados)
  - campos aleatórios de Markov (grafos não-direcionados).
- Passado:
  - embora fossem comuns, eram sempre limitados a situações simples que permitiam um tratamento matemático analítico.
- Presente:
  - algoritmos eficientes permitem a utilização de modelos probabilísticos para tomada de decisões em situações complexa → centenas ou milhares de variáveis ou condicionantes.
- Aplicações incluem:
  - processamento de imagens e sinais, visão computacional, robótica, sistemas expecialistas, redes sociais, redes regulatórias de proteínas, recuperação de informação, entre outras.

Modelos Gráficos Probabilísticos

(Probabilistic Graphical Models, PGM, em inglês)

- Principais tipos:
  - **redes bayesianas (grafos direcionados)**
  - campos aleatórios de Markov (grafos não-direcionados).
- Passado:
  - embora fossem comuns, eram sempre limitados a situações simples que permitiam um tratamento matemático analítico.
- Presente:
  - algoritmos eficientes permitem a utilização de modelos probabilísticos para tomada de decisões em situações complexa → centenas ou milhares de variáveis ou condicionantes.
- Aplicações incluem:
  - processamento de imagens e sinais, visão computacional, robótica, sistemas expecialistas, redes sociais, redes regulatórias de proteínas, recuperação de informação, entre outras.

Modelos Gráficos Probabilísticos

(Probabilistic Graphical Models, PGM, em inglês)

Inferência Variacional

Abordagem para se estimar distribuições a posteriori muito difíceis ou complexas

- Principais tipos:
  - **redes bayesianas (grafos direcionados)**
  - campos aleatórios de Markov (grafos não-direcionados).
- Passado:
  - embora fossem comuns, eram sempre limitados a situações simples que permitiam um tratamento matemático analítico.
- Presente:
  - algoritmos eficientes permitem a utilização de modelos probabilísticos para tomada de decisões em situações complexa → centenas ou milhares de variáveis ou condicionantes.
- Aplicações incluem:
  - processamento de imagens e sinais, visão computacional, robótica, sistemas expecialistas, redes sociais, redes regulatórias de proteínas, recuperação de informação, entre outras.

Variational Autoencoders

Gerando Amostras – *Forward Step*

Recordar fatos de probabilidade

- Suponha que $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$ sejam v.a.s i.i.d com densidade $p_Z(\mathbf{z})$
- Temos

$$\mathbb{E}(\mathbf{Z}) = \int \mathbf{z} p_Z(\mathbf{z}) d\mathbf{z} \approx \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i$$

- Seja g(Z) qualquer função matemática de Z.
- Por exemplo, $g(Z) = Z^2$ ou $g(Z) = \exp(-Z)$
- g(Z) é uma nova v.a.
- Sua esperança é

$$\mathbb{E}(g(\mathbf{Z})) = \int g(\mathbf{z})\, p_Z(\mathbf{z}) d\mathbf{z} \approx \frac{1}{n} \sum_{i=1}^{n} g(\mathbf{z}_i)$$

Recordar fatos de probabilidade

- Suponha que $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$ sejam v.a.s i.i.d com densidade $p_Z(\mathbf{z})$
- Temos

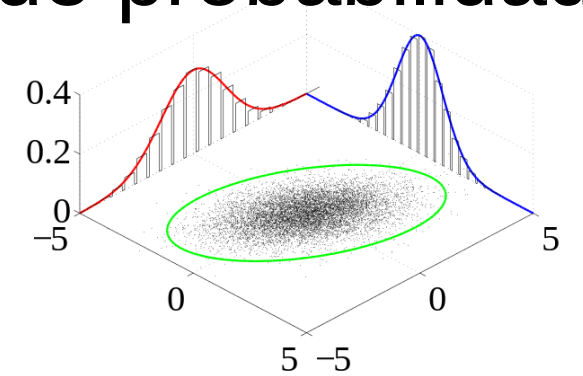
$$\mathbb{E}(\mathbf{Z}) = \int \mathbf{z} p_Z(\mathbf{z}) d\mathbf{z} \approx \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i$$

- Seja g(Z) qualquer função matemática de Z.
- Por exemplo, $g(Z) = Z^2$ ou $g(Z) = \exp(-Z)$
- g(Z) é uma nova v.a.
- Sua esperança é

$$\mathbb{E}(g(\mathbf{Z})) = \int g(\mathbf{z})\, p_Z(\mathbf{z}) d\mathbf{z} \approx \frac{1}{n} \sum_{i=1}^{n} g(\mathbf{z}_i)$$

Eventos com mesma probabilidade:
Média aritmética!

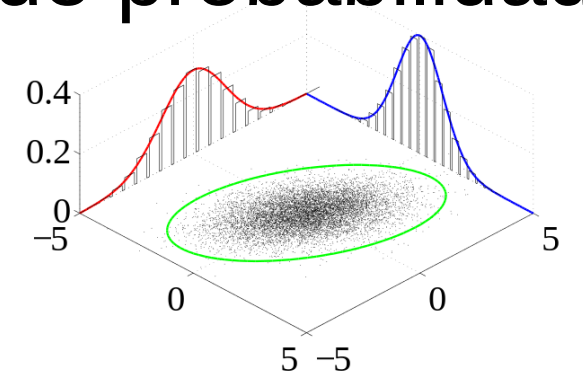
Recordar fatos de probabilidade



- Seja $(\mathbf{x}, \mathbf{z})$ um vetor com os dois grupos de variáveis, x e z.
- Densidade conjunta: $p_{XZ}(\mathbf{x}, \mathbf{z})$
- Densidade condicional: $p_{X|Z}(\mathbf{x}|\mathbf{z})$
- Densidades marginais: $p_X(\mathbf{x})$ e $p_Z(\mathbf{z})$
- Podemos escrever:

$$p_X(\mathbf{x}) = \int p_{XZ}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

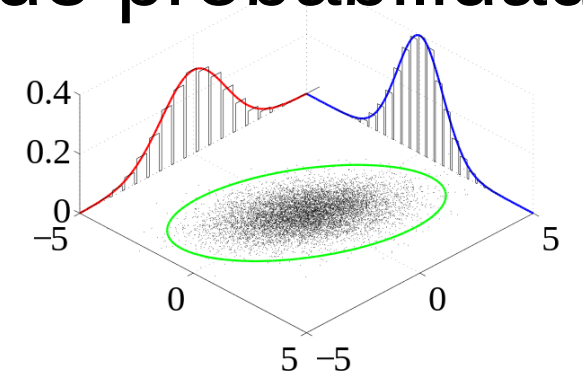
Recordar fatos de probabilidade



- Seja $(\mathbf{x}, \mathbf{z})$ um vetor com os dois grupos de variáveis, x e z.
- Densidade conjunta: $p_{XZ}(\mathbf{x}, \mathbf{z})$
- Densidade condicional: $p_{X|Z}(\mathbf{x}|\mathbf{z})$
- Densidades marginais: $p_X(\mathbf{x})$ e $p_Z(\mathbf{z})$
- Podemos escrever:

$$p_X(\mathbf{x}) = \int p_{XZ}(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_{X|Z}(\mathbf{x}|\mathbf{z}) p_Z(\mathbf{z}) d\mathbf{z}$$

Recordar fatos de probabilidade



- Seja $(\mathbf{x}, \mathbf{z})$ um vetor com os dois grupos de variáveis, x e z.
- Densidade conjunta: $p_{XZ}(\mathbf{x}, \mathbf{z})$
- Densidade condicional: $p_{X|Z}(\mathbf{x}|\mathbf{z})$
- Densidades marginais: $p_X(\mathbf{x})$ e $p_Z(\mathbf{z})$
- Podemos escrever:

$$p_X(\mathbf{x}) = \int p_{XZ}(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_{X|Z}(\mathbf{x}|\mathbf{z}) p_Z(\mathbf{z}) d\mathbf{z} = \mathbb{E}_Z \left[p_{X|Z}(\mathbf{x}|\mathbf{Z}) \right]$$

Gerando amostras de p(x)

- Se soubermos p(z) e p(x|z), é fácil gerar amostras de p(x):
  - 1) Gere uma amostra aleatória de z a partir de p(z). Chame este exemplo de $\mathbf{z}_s$
  - 2) Em seguida, gere uma amostra de $\mathbf{x}_s$ a partir de p (x | z = z_s).
  - A amostra final de x tem a densidade desejada densidade p(x)
- A amostra da etapa (2) tem probabilidade: $p(\mathbf{x}|\mathbf{z}_s)$
- Repita isto várias vezes.
- Geramos $\mathbf{z}_s(1), \mathbf{z}_s(2), \ldots, \mathbf{z}_s(n)$
- Temos então a aproximação

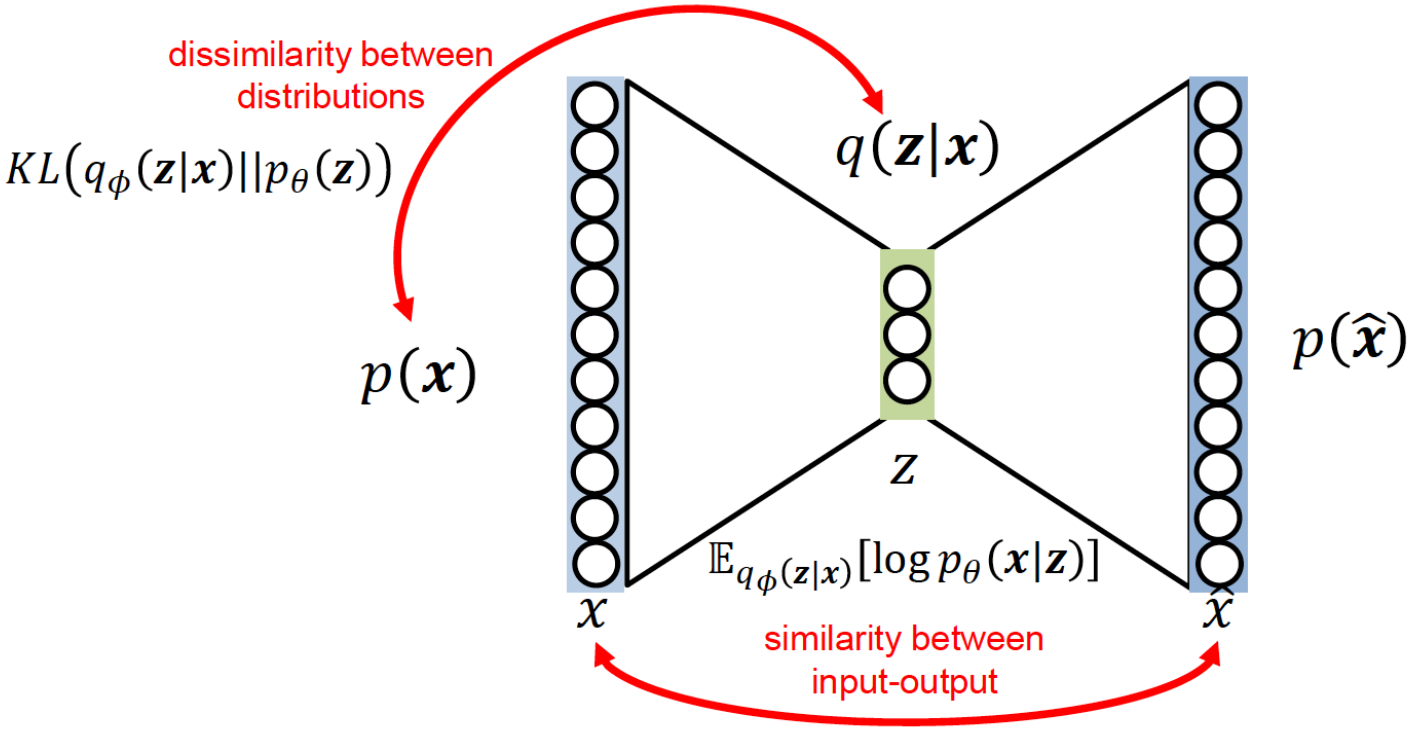
$$p_{X|Z}(\mathbf{x}) = \mathbb{E}_Z\left[p_{X|Z}(\mathbf{x}|\mathbf{Z})\right] \approx \frac{1}{n}\sum_{i=1}^{n} p(\mathbf{x}, \mathbf{z}_s(i))$$

Variational Autoencoders

Treinamento – *Loss Function*

Função de Perda

Perspectiva de Redes Neurais



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

Adaptado das aulas do Prof. Raul Feitosa (PUC-Rio)

The Kullback Leibler divergence

Mede diferença entre duas distribuições de probabilidade

Algumas definições:

- **Informação:**

$$I = -\log p(\boldsymbol{x}) ,$$ onde *x* é um evento

Observe que, quanto mais baixa a probabilidade, maior é a informação.

Ex.:

- “amanhã irá chover ou não” → probabilidade é 1, sem informação.
- “amanhã a temperatura alcançará 42 graus” → probabilidade é pequena, muita informação.

The Kullback Leibler divergence

Mede diferença entre duas distribuições de probabilidade

Algumas definições:

- **Informação esperada ou <u>Entropia</u>:**

$$H = \mathbb{E}[-\log p(\boldsymbol{x})] = -\sum_{\boldsymbol{x}} p(\boldsymbol{x}) \log p(\boldsymbol{x})$$

Observe que, quanto mais baixa a probabilidade, maior é a informação.

Ex.:

- “amanhã irá chover ou não” → probabilidade é 1, sem informação.
- “amanhã a temperatura alcançará 42 graus” → probabilidade é pequena, muita informação.

The Kullback Leibler divergence

$$KL\big(p(\boldsymbol{x})||q(\boldsymbol{x})\big) = -\sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log q(\boldsymbol{x}) + \sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log p(\boldsymbol{x})$$

$$= \sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} = \mathbb{E}_{p(\boldsymbol{x})}\left[log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$$

Some properties

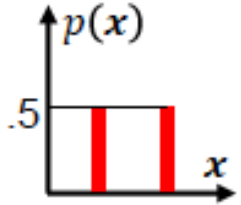
- $KL\big(p(\boldsymbol{x})||q(\boldsymbol{x})\big) \geq 0$
- $KL\big(p(\boldsymbol{x})||q(\boldsymbol{x})\big) \neq KL\big(q(\boldsymbol{x})||p(\boldsymbol{x})\big)$ not a distance!

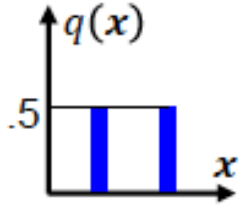
Adaptado das aulas do Prof. Raul Feitosa (PUC-Rio)

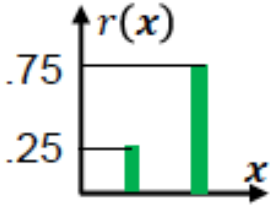
The Kullback Leibler divergence

$$KL\big(p(\boldsymbol{x})||q(\boldsymbol{x})\big) = -\sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log q(\boldsymbol{x}) + \sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log p(\boldsymbol{x}) = \sum_{\boldsymbol{x}} p(\boldsymbol{x})\ \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}$$

Examples:







$$KL\big(p(\boldsymbol{x})||q(\boldsymbol{x})\big) = 0$$

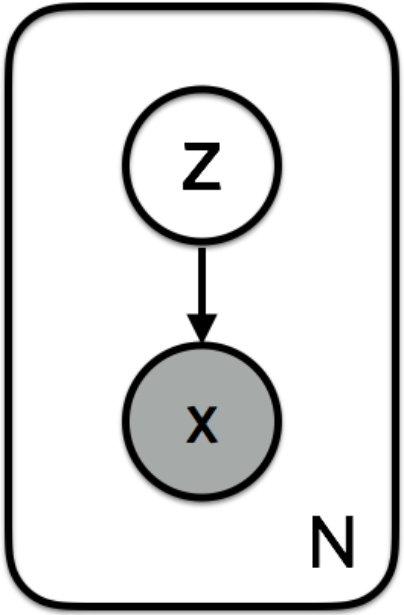
$$KL\big(p(\boldsymbol{x})||r(\boldsymbol{x})\big) = .5 \log \frac{.5}{.25} + .5 \log \frac{.5}{.75} = 0.0625$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

VAE contém um modelo de probabilidade específico dos dados x e variáveis latentes z. Podemos escrever a probabilidade conjunta do modelo como:

$$p\,(x, z) = p(x|z)\,p(z)$$



O processo generativo pode ser escrito da seguinte maneira:

- Variáveis latentes: $z_i \sim p(z)$
- Amostras $x_i \sim p(x|z)$

Função de Loss: ELBO

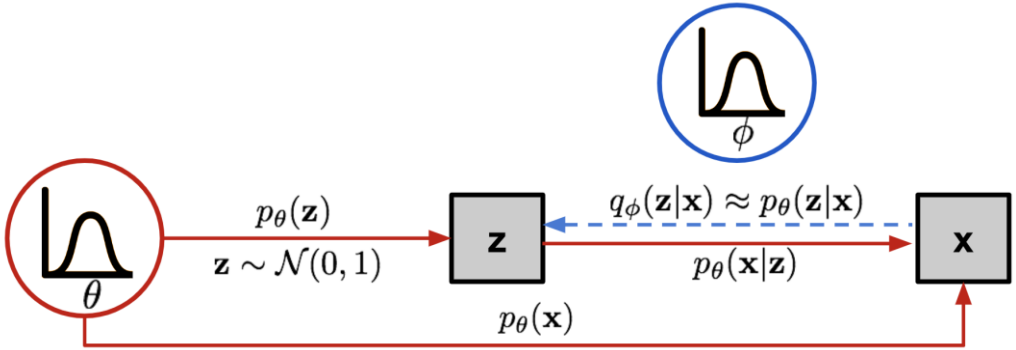
Perspectiva de Modelos Probabilísticos

- As variáveis latentes são extraídas de um *(z)* a priori.
- Os dados x têm uma probabilidade *p (x|z)* condicionada às variáveis latentes z.
- O modelo define uma distribuição de probabilidade conjunta sobre dados e variáveis latentes: *p (x, z).*
- Podemos decompor isso na probabilidade:

$$p(x, z) = p(x|z)\, p(z).$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

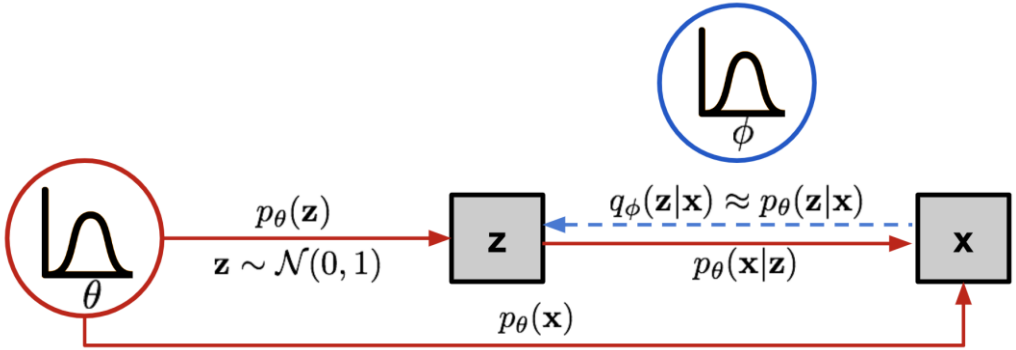


- Agora a estrutura se parece mais com um autoencoder:
  - A probabilidade condicional $p_\theta(x \mid z)$ define um modelo generativo – $p_\theta(x \mid z)$ também é conhecido como decodificador probabilístico.
  - A função de aproximação $q_\phi(z \mid x)$ é o codificador probabilístico

A probabilidade estimada $q_\phi(z \mid x)$ deve estar muito próximo do real $p_\theta(z \mid x)$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos



- Agora a estrutura se parece mais com um autoencoder:
  - A probabilidade condicional $p_\theta(x \mid z)$ define um modelo generativo – $p_\theta(x \mid z)$ também é conhecido como decodificador probabilístico.
  - A função de aproximação $q_\phi(z \mid x)$ é o codificador probabilístico

A probabilidade estimada $q_\phi(z \mid x)$ deve estar muito próximo do real $p_\theta(z \mid x)$

Como calcular?

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- **Inferência Variacional**: o objetivo é inferir bons valores das variáveis latentes, dados os dados observados, ou calcular *p(z|x)* a posteriori. De acordo com Bayes temos:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}.$$

- Observe o denominador p(x). Isso é chamado de evidência, e podemos calculá-lo marginalizando as variáveis latentes:

$$p(x) = \int p(x \mid z)p(z)dz$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- **Inferência Variacional**: o objetivo é inferir bons valores das variáveis latentes, dados os dados observados, ou calcular *p(z|x)* a posteriori. De acordo com Bayes temos:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}.$$

- Observe o denominador p(x). Isso é chamado de evidência, e podemos calculá-lo marginalizando as variáveis latentes:

$$p(x) = \int p(x \mid z)p(z)dz$$

Essa integral requer tempo exponencial para calcular, pois precisa ser avaliada em todas as configurações de variáveis latentes

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Inferência variacional aproxima do posterior com uma família de distribuições $q_\lambda(z \mid x)$
- O parâmetro variacional *λ* indexa a família de distribuições.
  - Por exemplo, se *q* for gaussiano, seria a média e a variância das variáveis latentes para cada ponto de dados:

$$\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Inferência variacional aproxima do posterior com uma família de distribuições $q_\lambda(z \mid x)$
- O parâmetro variacional *λ* indexa a família de distribuições.
  - Por exemplo, se *q* for gaussiano, seria a média e a variância das variáveis latentes para cada ponto de dados:

$$\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$$

Como saber quão bem a nossa probabilidade a posteriori *q (z|x)* se aproxima da distribuição verdadeira *p(z|x)*?

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Inferência variacional aproxima do posterior com uma família de distribuições $q_\lambda(z \mid x)$
- O parâmetro variacional *λ* indexa a família de distribuições.
  - Por exemplo, se *q* for gaussiano, seria a média e a variância das variáveis latentes para cada ponto de dados:

$$\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$$

Como saber quão bem a nossa probabilidade a posteriori *q (z|x)* se aproxima da distribuição verdadeira *p(z|x)*?

R: Podemos usar a divergência de *Kullback-Leibler*:

$$\mathbb{KL}(q_\lambda(z \mid x) \,||\, p(z \mid x)) =$$

$$\mathbf{E}_q[\log q_\lambda(z \mid x)] - \mathbf{E}_q[\log p(x, z)] + \log p(x)$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- O objetivo é encontrar os parâmetros variacionais *λ* que minimizem a divergência KL . A posterior aproximada ideal fica assim:

$$q_\lambda^*(z \mid x) = \arg\min_\lambda \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

- Mas, conforme já discutido p(x) aparece na divergência e, como já discutido, é intratável.
- Por isso, a área de inferência variacional utiliza um elemento a mais:

$$ELBO(\lambda) = \mathbf{E}_q[\log p(x, z)] - \mathbf{E}_q[\log q_\lambda(z \mid x)]$$

Pela **desigualdade de Jensen***, a divergência de Kullback-Leibler é sempre maior ou igual a zero. Isso significa que minimizar a divergência Kullback-Leibler é equivalente a maximizar o ELBO (Evidence Lower BOund).

*http://users.umiacs.umd.edu/~xyang35/files/understanding-variational-lower.pdf

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Observe que podemos combinar isso com a divergência Kullback-Leibler e reescrever as evidências como:

$$\log p(x) = ELBO(\lambda) + \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

- No modelo de autoencoder variacional, existem apenas variáveis latentes locais.
  - Nenhum ponto de dados compartilha seu z latente com a variável latente de outro ponto de dados
- Assim, podemos decompor o ELBO em uma soma em que cada termo depende de um único ponto de dados. Isso nos permite usar a descida do gradiente estocástico em relação aos parâmetros *λ :*

$$ELBO_i(\lambda) = \mathbb{E}q_\lambda(z \mid x_i)[\log p(x_i \mid z)] - \mathbb{KL}(q_\lambda(z \mid x_i) \mid\mid p(z))$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Vamos fazer a conexão com a rede neural. O passo final é parametrizar a probabilidade aproximada $q_\theta(z|x, \lambda)$ com uma rede de inferência (ou *encoder*) que recebe como dados de entrada x e gera os parâmetros λ. Parametrizamos a probabilidade p(x|z) com a rede generativa (ou *decoder*) que leva variáveis latentes e produz parâmetros para a distribuição de dados $p_\phi(x|z)$.
- Podemos escrever o ELBO e incluir os parâmetros de inferência e rede generativa como:

$$ELBO_i(\theta, \phi) = \mathbb{E}q_\theta(z \mid x_i)[\log p_\phi(x_i \mid z)] - \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

Função de Loss: ELBO

Perspectiva de Modelos Probabilísticos

- Vamos fazer a conexão com a rede neural. O passo final é parametrizar a probabilidade aproximada $q_\theta(z|x, \lambda)$ com uma rede de inferência (ou *encoder*) que recebe como dados de entrada x e gera os parâmetros λ. Parametrizamos a probabilidade p(x|z) com a rede generativa (ou *decoder*) que leva variáveis latentes e produz parâmetros para a distribuição de dados p_ϕ (x|z).
- Podemos escrever o ELBO e incluir os parâmetros de inferência e rede generativa como:

$$ELBO_i(\theta, \phi) = \mathbb{E}q_\theta(z \mid x_i)[\log p_\phi(x_i \mid z)] - \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z)).$$

- Comparando com a loss apresentada, inicialmente temos que $ELBO_i(\theta, \phi) = -l_i(\theta, \phi)$:

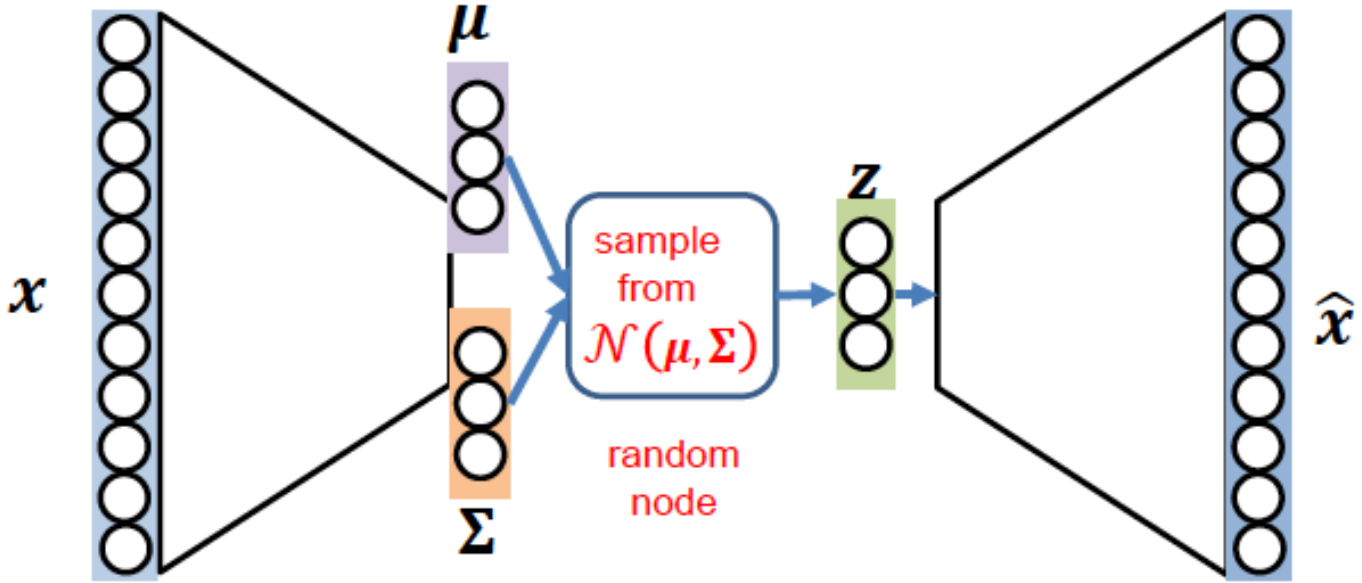
$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

Variational Autoencoders

Treinamento – *Backpropagation*

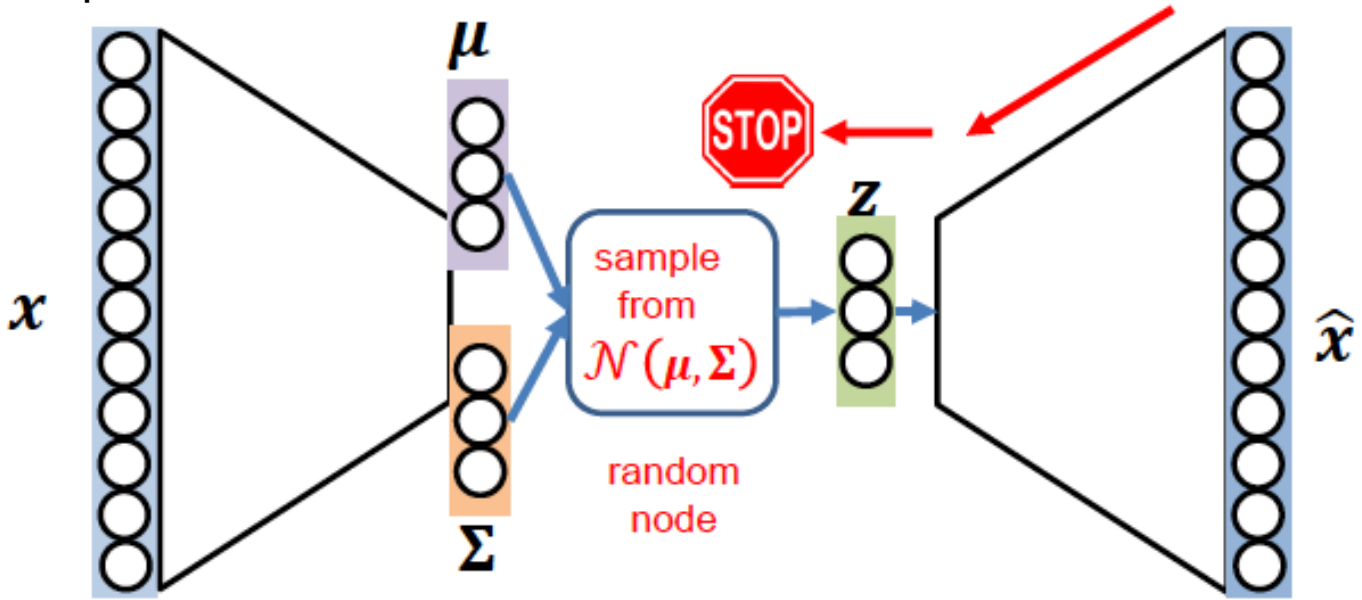
Backpropagation

Ao invés de produzir $\boldsymbol{z}$, o encoder produz $\boldsymbol{\mu}$ e $\boldsymbol{\Sigma}$ da distribuição gaussiana e gera amostras a partir disso.



Backpropagation

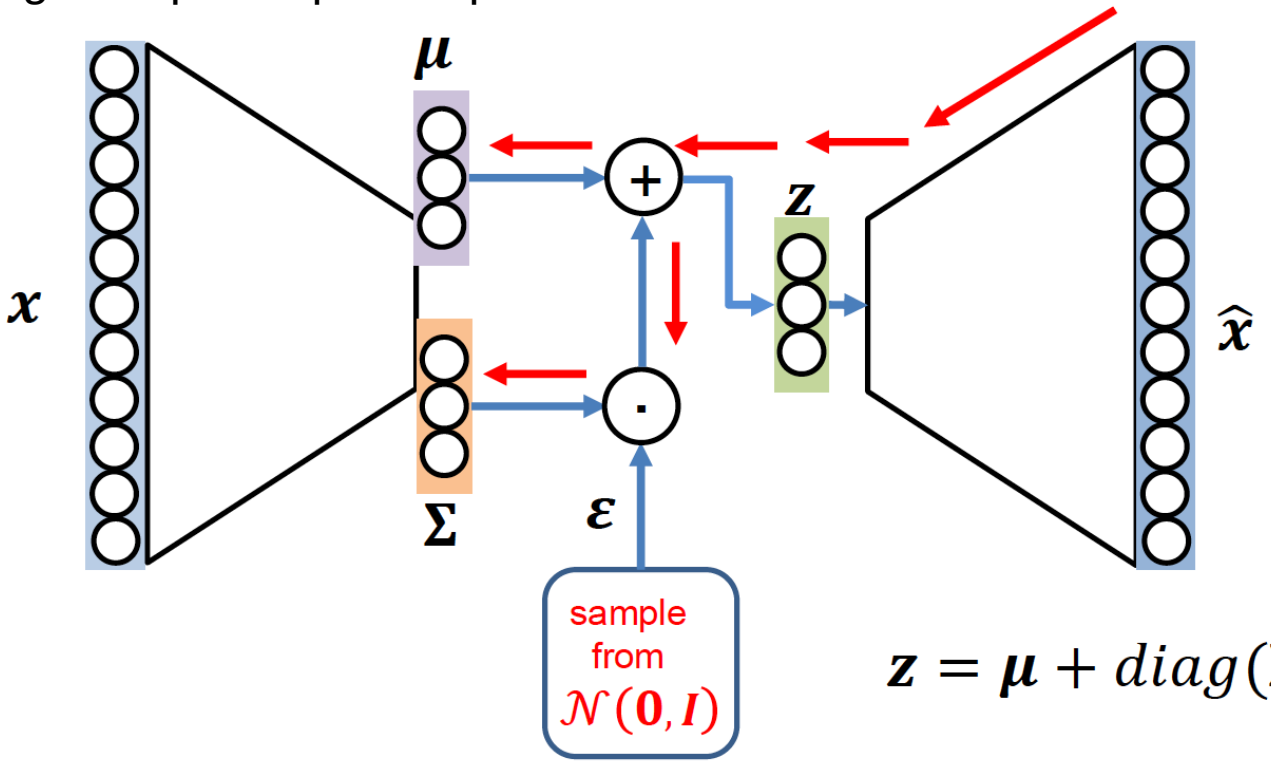
Ao invés de produzir $\boldsymbol{z}$, o encoder produz $\boldsymbol{\mu}$ e $\boldsymbol{\Sigma}$ da distribuição gaussiana e gera amostras a partir disso.



Problema: Backpropagation não pode ser computado em um nó aleatório!

Solução: Truque da Reparametrização

Introduz uma variável $\boldsymbol{\varepsilon}$ que permite reparametrizar $\boldsymbol{z}$ de forma que o *backpropagation* possa passar por nós determinísticos



$$\boldsymbol{z} = \boldsymbol{\mu} + diag(\boldsymbol{\Sigma}) \odot \boldsymbol{\varepsilon}$$

Solução: Truque da Reparametrização

Para algumas distribuições, é possível reparametrizar as amostras de maneira inteligente, de modo que a estocástica seja independente dos parâmetros.

Uma variável distribuída normalmente com média μ e σ, podemos fazer uma amostra dela assim:

$$z = \mu + \sigma \odot \epsilon,$$

where $\epsilon \sim \text{Normal}(0,1)$. Going from $\sim$ denoting a draw from the distribution to the equals sign = is the crucial step.

No autoencoder variacional, a média e a variância são produzidas por uma rede de inferência com parâmetros θ que otimizamos. O truque de reparametrização nos permite retropropagar (pegar derivadas usando a regra da cadeia) em relação a θ através da função de custo (ou ELBO), que é uma função das amostras das variáveis latentes z.

Variational Autoencoders

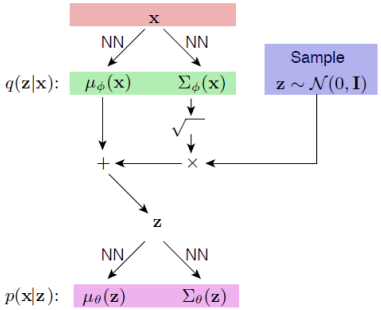
Aplicações

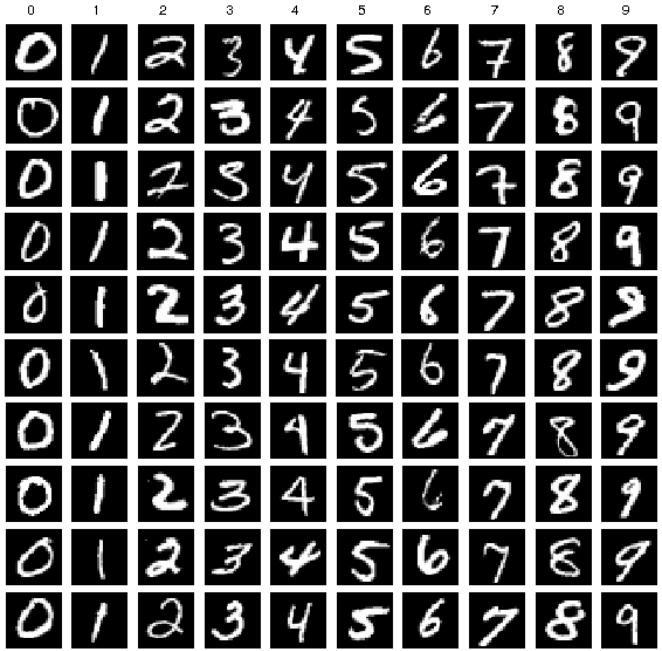
Exemplo com MNIST

De https://blog.keras.io/building-autoencoders-in-keras.htm

Uma camada para o encoder, densa (fully connected), vetor z com dimensão DOIS e com ativação ReLU para os parametros theta

```
x = Input(batch_shape=(batch_size, original_dim))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_sigma = Dense(latent_dim)(h)
```



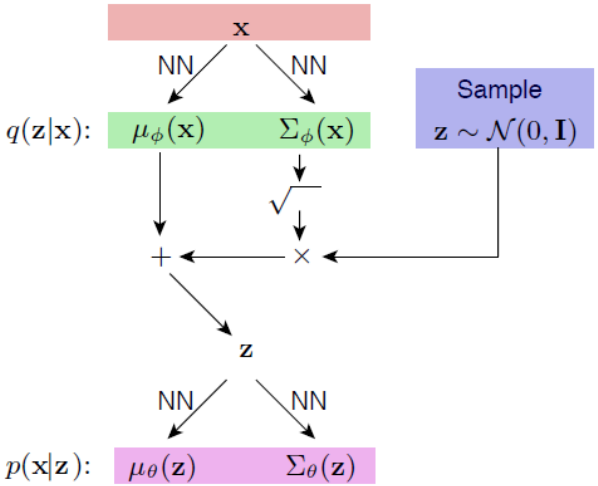


Exemplo com MNIST: https://blog.keras.io/building-autoencoders-in-keras.htm

Podemos usar esses parâmetros theta (média e DP) para amostrar novos pontos semelhantes do espaço latente (não são novas imagens ainda)

```
def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim),
                              mean=0., std=epsilon_std)
    return z_mean + K.exp(z_log_sigma) * epsilon

# note that "output_shape" isn't necessary with the TensorFlow backend
# so you could write `Lambda(sampling)([z_mean, z_log_sigma])`
z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_sigma])
```

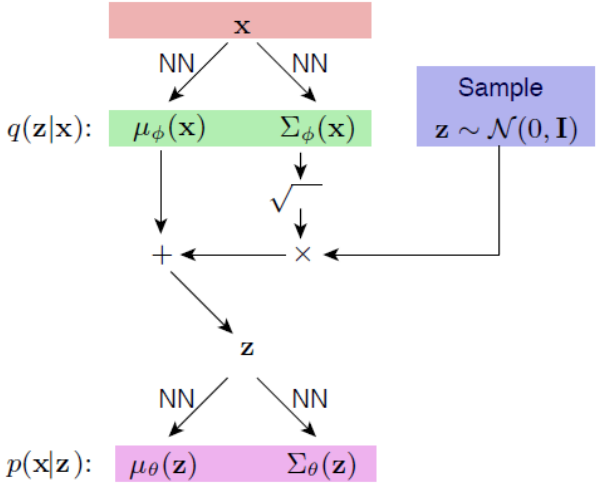


Exemplo com MNIST: https://blog.keras.io/building-autoencoders-in-keras.htm

Finalmente, podemos mapear esses pontos latentes amostrados de volta para entradas reconstruídas:

```
decoder_h = Dense(intermediate_dim, activation='relu')
decoder_mean = Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)
```

Ver código completo em
https://blog.keras.io/building-autoencoders-in-keras.html



Exemplo com MNIST: https://blog.keras.io/building-autoencoders-in-keras.htm

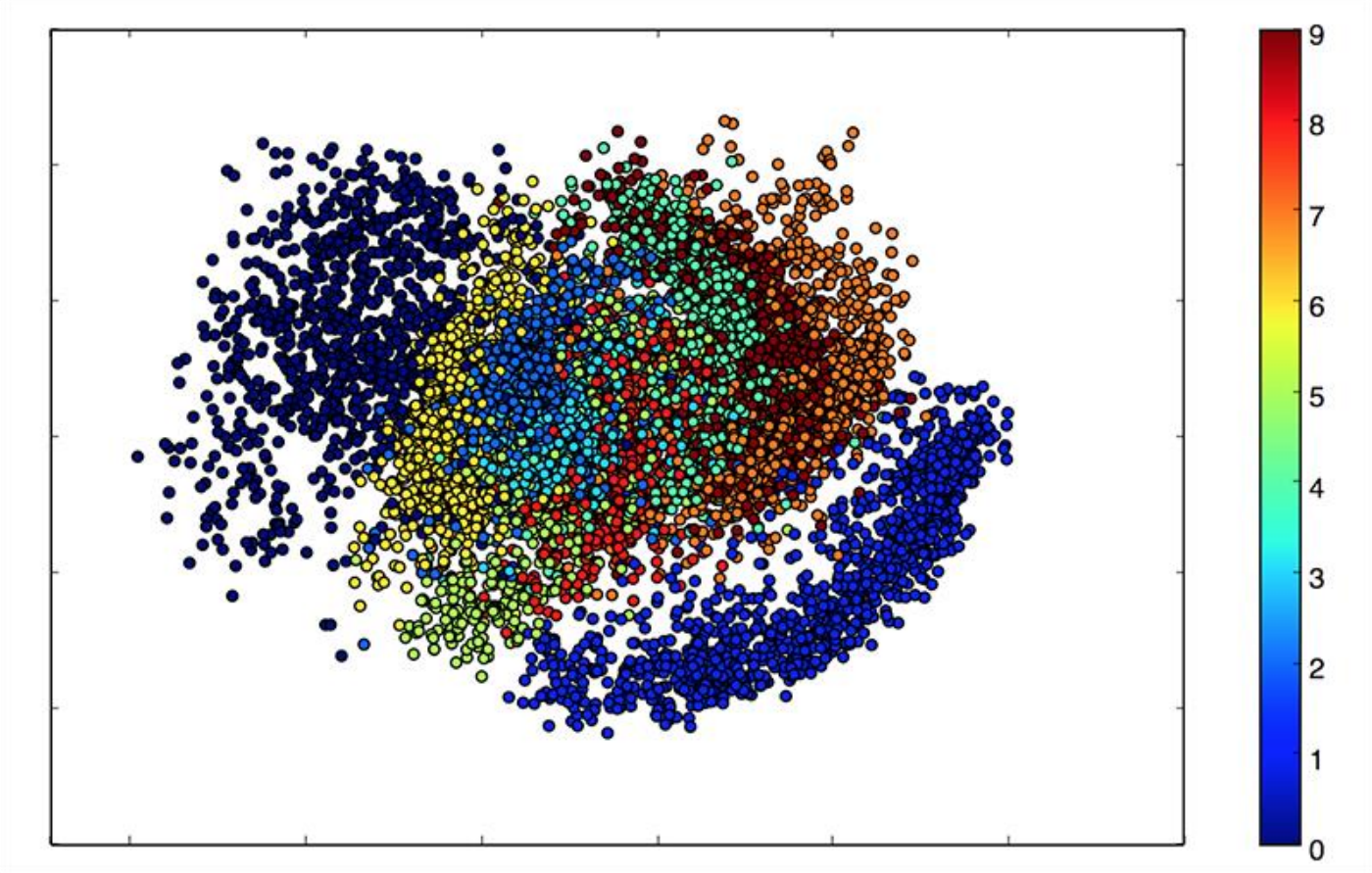
Nós treinamos o modelo usando uma função de perda personalizada: a soma de um termo de reconstrução e o termo de regularização de divergência KL

```
def vae_loss(x, x_decoded_mean):
    xent_loss = objectives.binary_crossentropy(x, x_decoded_mean)
    kl_loss = - 0.5 * K.mean(1 + z_log_sigma - K.square(z_mean) - K.exp(z_log_sigma), axis=-1)
    return xent_loss + kl_loss

vae.compile(optimizer='rmsprop', loss=vae_loss)
```

Ver código completo em
https://blog.keras.io/building-autoencoders-in-keras.html

Espaço latente z (2-dim) e os dígitos



Varrendo o espaço latente com uma grade regular

Como o VAE é um modelo generativo, também podemos usá-lo para gerar novos dígitos.

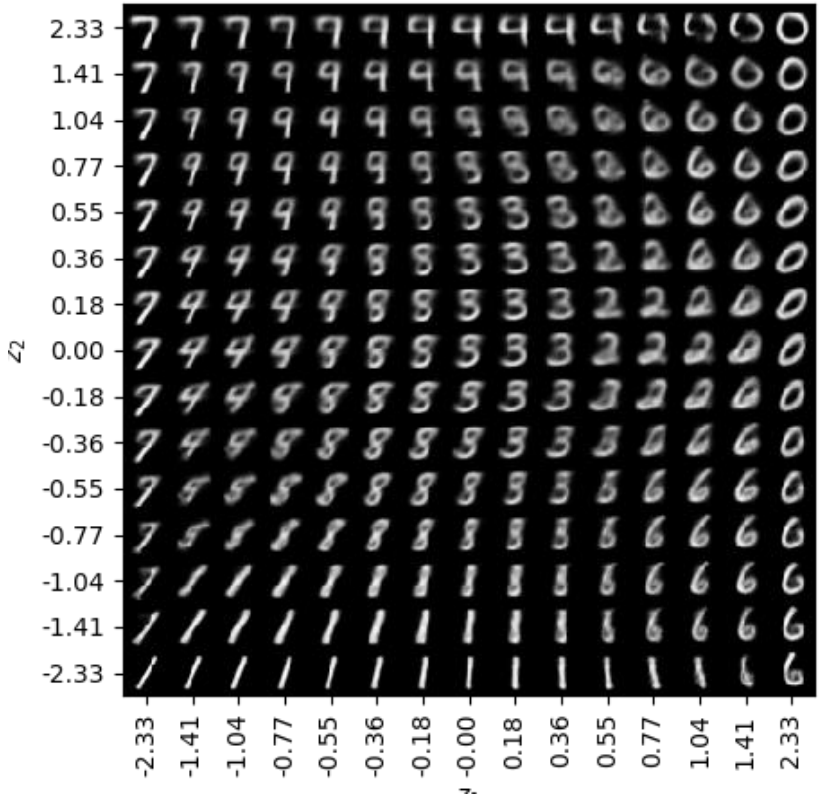
Vamos varrer o plano latente z (2-dim).

Amostrando pontos latentes z em intervalos regulares.

Geramos o dígito correspondente para cada um desses pontos.

Isso nos dá uma visualização da variedade latente que gera (ou codifica) os dígitos MNIST.

Parece que o 4 não é bem representado.... ??

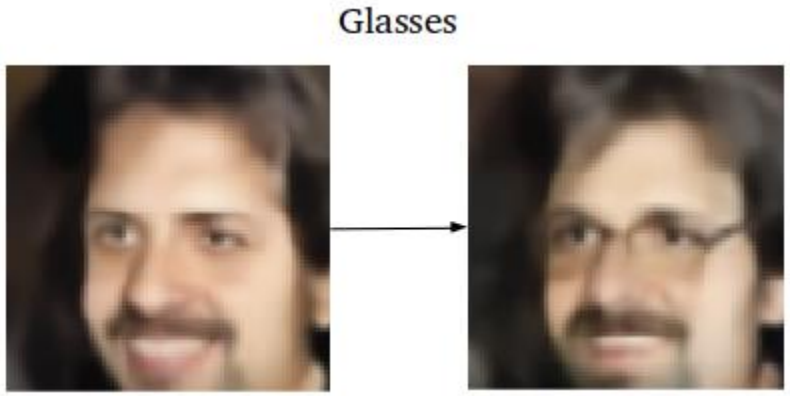


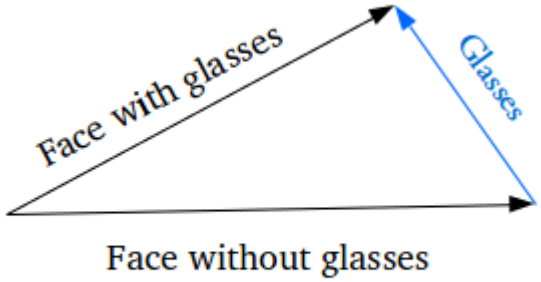
Exemplo: VAE para gerar faces fake de celebridades

- https://github.com/yzwxx/vae-celebA
- Conv nets → Input, Reconstruction, Generated Sample

Pondo óculos numa face

- Que tal gerar features específicas?
- Por exemplo, como gerar óculos em um rosto?
- Encontre duas amostras, uma com óculos, e uma sem.
- Obtenha seus vetores codificados z do encoder.
- Obtenha a diferença vetorial GLASSES entre eles.
- Adicione este novo vetor de GLASSES a qualquer outra imagem de rosto e decodifique-a.





Adding new features to samples

Resumo - VAEs

- A geração de imagens com aprendizado profundo é feita pelo aprendizado de espaços latentes que capturam informações estatísticas sobre um conjunto de dados de imagens.
- Por amostragem e decodificação de pontos do espaço latente, você pode gerar imagens nunca antes vistas.
- Existem duas ferramentas principais para fazer isso: VAEs e GANs (último módulo).
- As VAEs resultam em representações latentes contínuas e altamente estruturadas.
- Por esse motivo, eles funcionam bem para fazer edição de imagem no espaço latente: trocar de rosto, transformar um rosto carrancudo em um rosto sorridente e assim por diante.