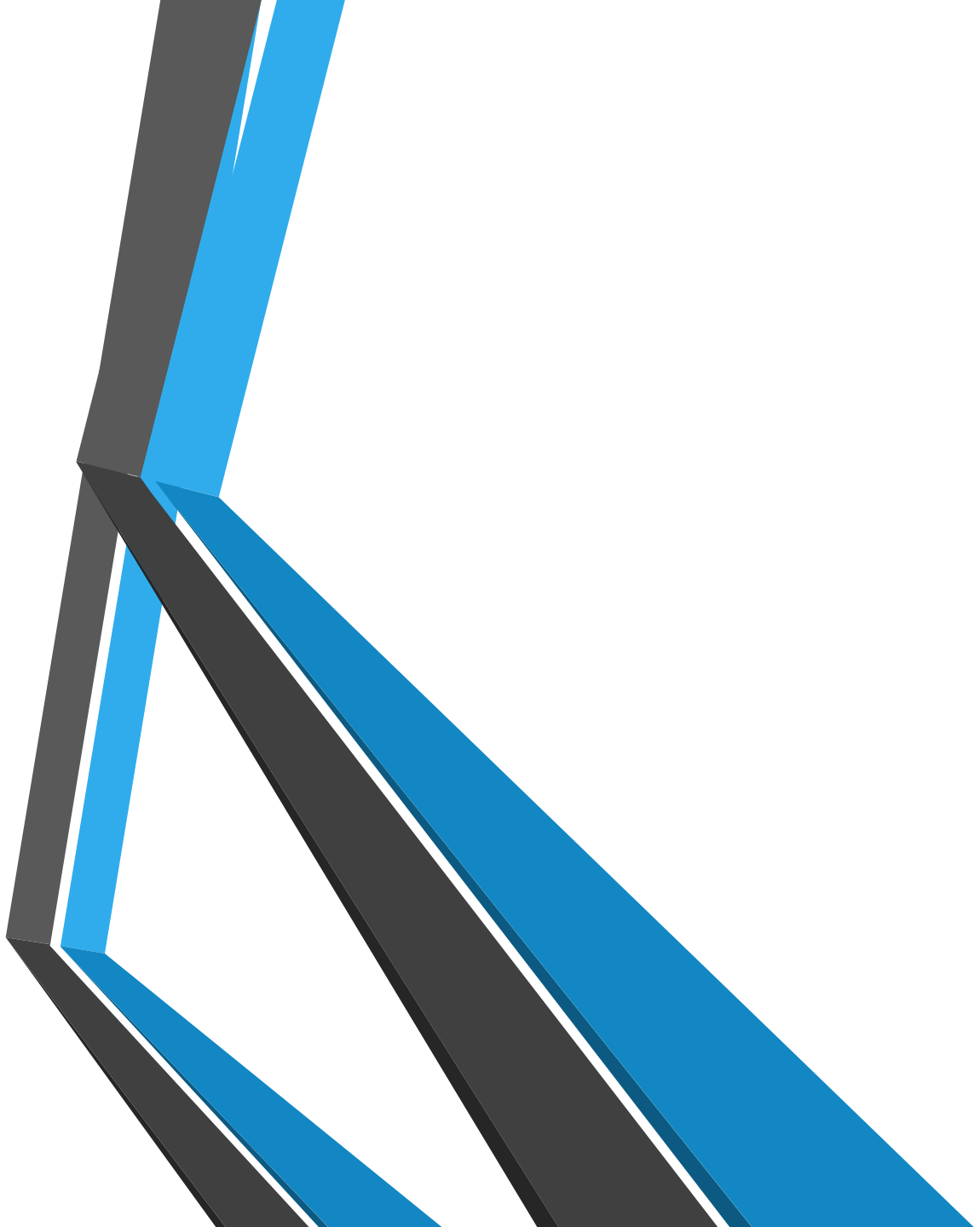


Autoencoder

Prof. Clodoaldo A M Lima

Sumário

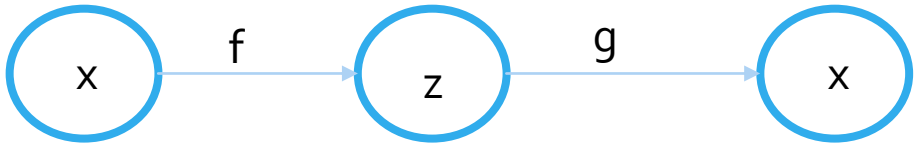
- Autoencoder
  - Introdução
  - Estrutura Geral
  - Definição
  - Taxonomia
- Redes Generativa Adversárias
  - Definição
  - Treinamento
  - Exemplos

Introdução

- Enquanto muitos algoritmos de aprendizado de máquina são capazes de tratar com entradas brutas (sem pré-processamento), é também verdade que, para a maioria, seu desempenho é degradado quando o numero de variáveis cresce.
  - Este é devido principalmente ao problema conhecido como maldição da dimensionalidade.
- Autoencoder foram propostos nos anos 80
- A arquitetura mais comum em aprendizado profundo é encoder-decoder (AE - autoencoder)
- AE foram introduzidos como uma forma de conduzir o pré-treinamento em Redes Neurais Artificiais
- Nem todos os modelos de AE são necessariamente Redes Neurais com múltiplas camadas escondidas
- Recentemente, conexões teóricas entre autoencoders e modelos de variáveis latentes levaram os autoencoder à vanguarda da modelagem generativa

Estrutura Geral

- A estrutura básica de um AE, é mostrado na figura abaixo, inclui uma entrada x, a qual é mapeada para uma codificação z via um encoder, representado por uma função *f*
- Esta codificação é mapeada de volta para a reconstrução r por meio de um decoder, representado por função *g*

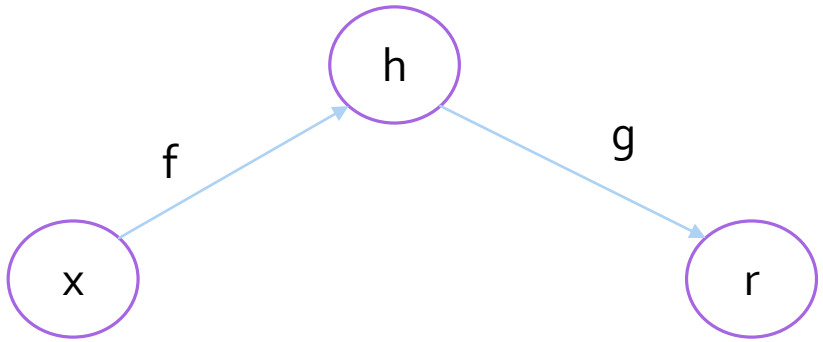


Definição

- AE são modelos com estrutura simétrica, onde a camada do meio representa a codificação dos dados de entrada.
- AEs são treinados para reconstruir suas entradas na camada de saída, sujeito a alguma restrição, que preveni a copia dados ao longo da rede.
- Embora o termo autoencoder é o mais popular atualmente, eles são conhecidos como redes neurais auto-associativas.

Definição

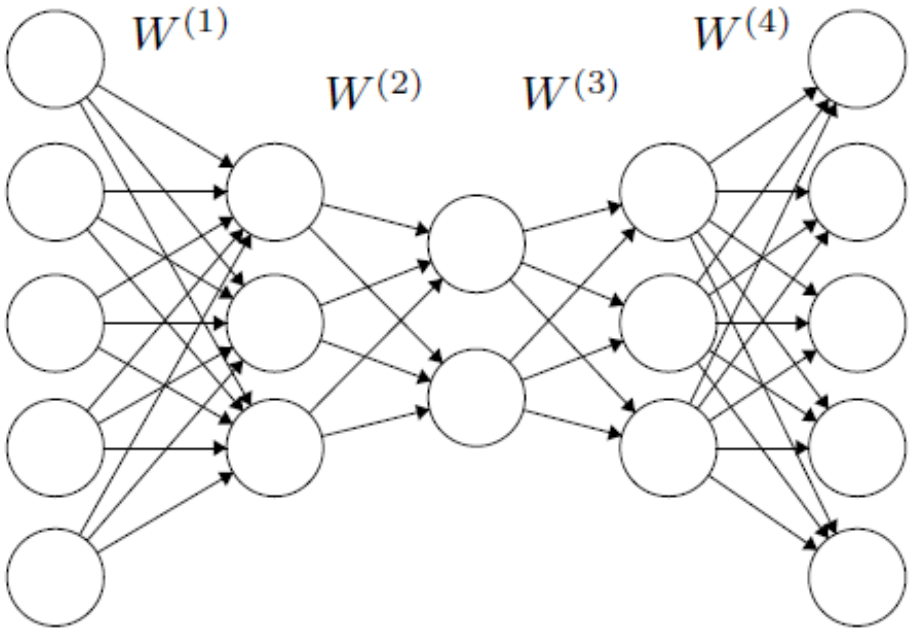
- Treinado por gradiente descendente com função perda de reconstrução: mede a diferença entre entrada e saída, por exemplo, MSE:
  - Codificador: h=f(x)
  - Decodificador: r = g(h)



Representação vs Simplicidade

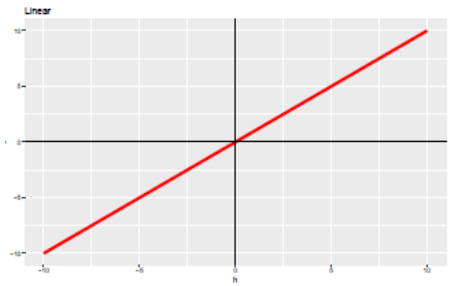
- O codificador pode aprender a função identidade precisamente: $g\big(f(x)\big) = x$
- Necessidade de restringir a complexidade:
  - por restrição na arquitetura
  - por inserindo penalidade na representação interna
- Dessa forma, o codificador precisa aprender a estrutura do conjunto de treinamento.
- Qualidade de reconstrução vs. simplicidade de representação.
  - extremos: identidade vs. saída constante.

Rede Neural para AE

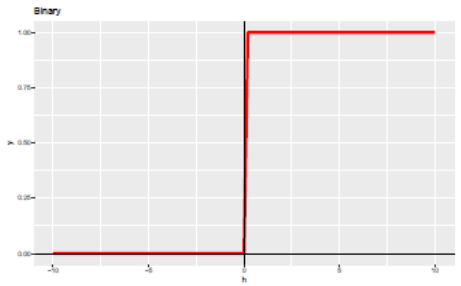


- Um arquitetura possível para um autoencoder com duas camadas de variáveis de codificação
- W denota matriz de pesos

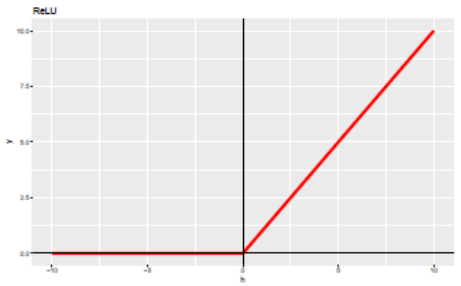
Funções de Ativações



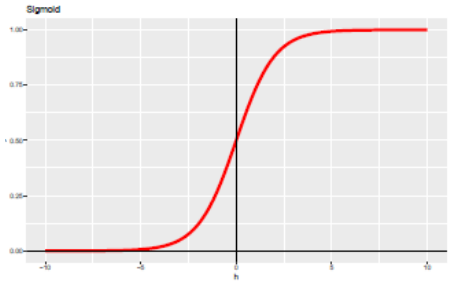
(a) Linear



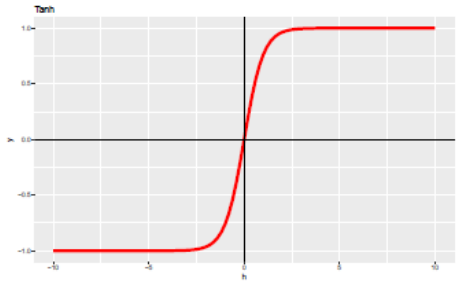
(b) Binary/Boolean



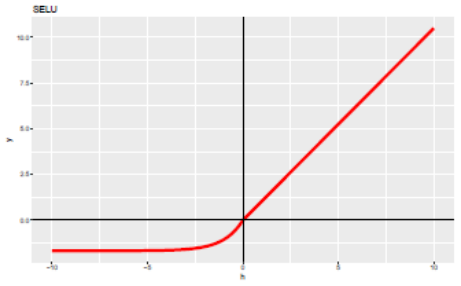
(c) ReLU



(d) Logistic



(e) Tanh



(f) SELU

$$ReLU(x) = \begin{cases} 0, se\ x \leq 0 \\ x, se\ x > 0 \end{cases}$$

$$SELU(x) = \lambda \begin{cases} \alpha e^x - \alpha, se\ x \leq 0 \\ x \qquad\qquad , se\ x > 0 \end{cases}$$

$$Logistic(x) = \frac{1}{1 + e^x}$$

Aplicações de autoencoder

- Aplicações
  - redução de dimensionalidade
    - visualização
    - extração de característica
    - aumento da precisão de predição
    - aumento da velocidade de predição
    - Diminuição dos requisitos de memória
  - pré-treinamento não supervisionado

Estrutura dos AEs

- Os EAs podem ser agrupados de acordo com:
  - estrutura,
  - o algoritmo de aprendizado,
  - a função de perda,
  - função de ativação
  - área em que são aplicados
- Dependendo da dimensionalidade da camada de codificação
  - Undercomplete
  - Overcomplete

Estrutura dos AEs

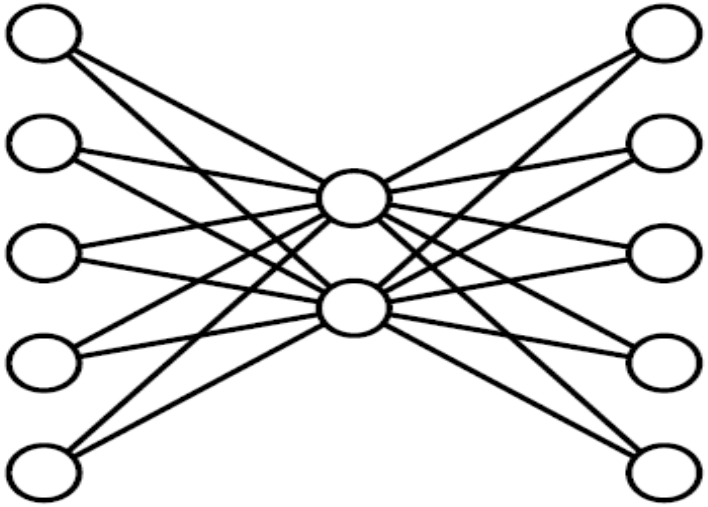
- ***Undercomplete*** - camada de codificação tem uma dimensionalidade menor que a entrada. O número menor de unidades impõe uma restrição; portanto, durante o treinamento, o AE é forçado a aprender uma representação mais compacta. Isso é obtido, fundindo as características originais de acordo com os pesos atribuídos através do processo de aprendizagem

- ***Overcomplete*** - camada de codificação tem as mesmas ou mais unidades que a entrada, neste caso, o AE poderia simplesmente aprender a função de identidade, copiando a entrada na saída. Para evitar esse comportamento, normalmente outras restrições são aplicadas, que serão discutido mais a frente.

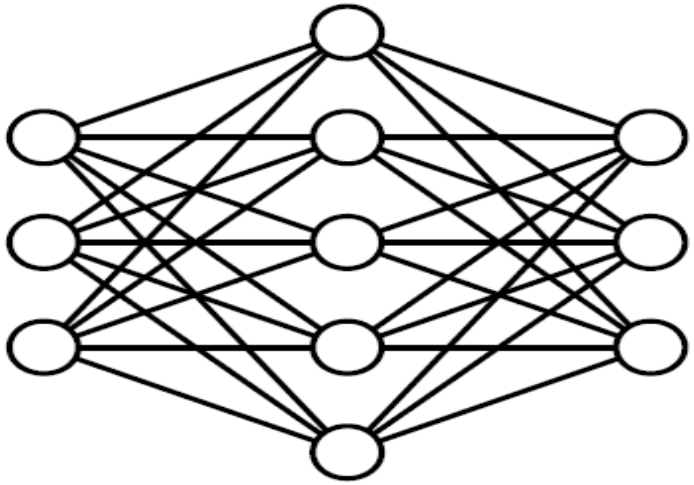
Estrutura dos AEs

- De acordo com o número de camadas
  - Shallow (Raso), quando compreende apenas três camadas (entrada, codificação e saída). É o modelo mais simples de AE, já que existe apenas uma camada oculta (a codificação).
  - Profundo, quando possui mais de uma camada oculta. este tipo de AE pode ser treinado camada por camada, conforme várias redes rasas de AEs são empilhadas ou como uma RNA profunda

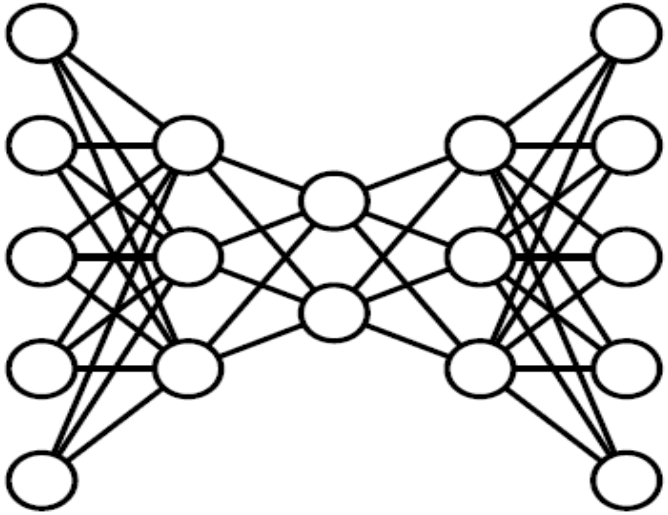
- Undecomplete

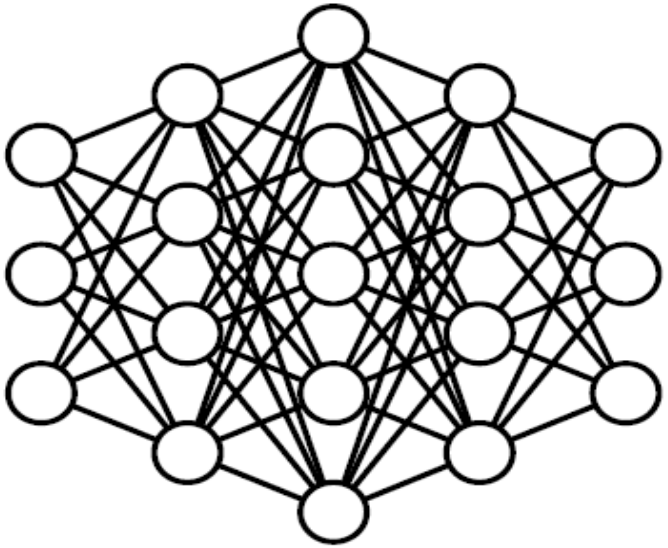


- Overcomplete

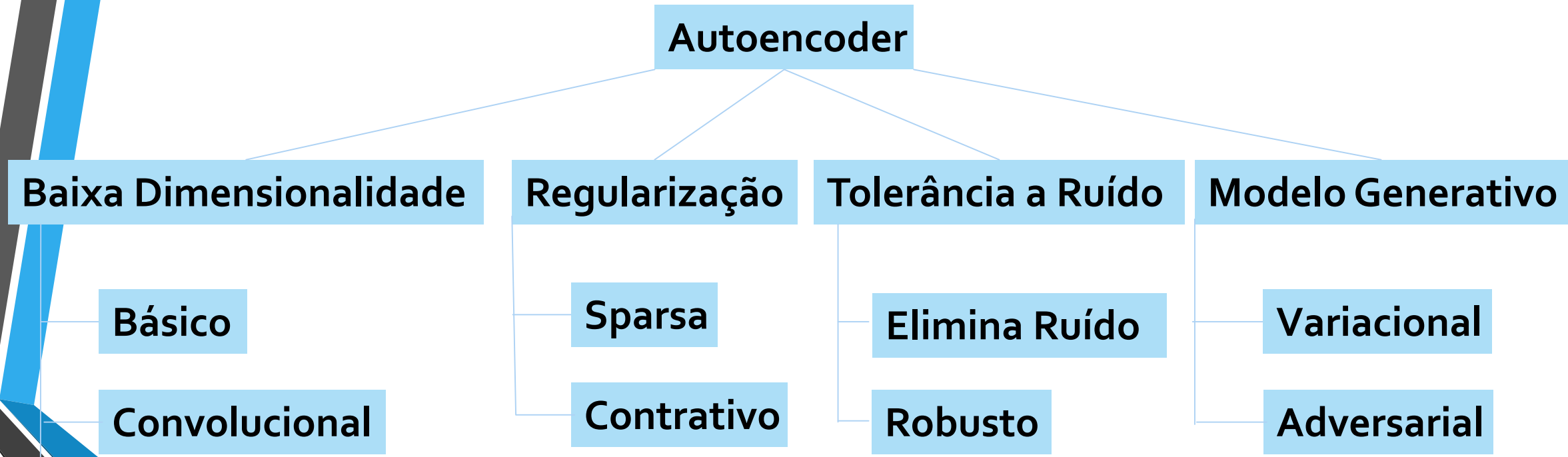


- Shallow
- Deep





Taxanomia dos AEs



Taxanomia de AEs

- Baixa dimensionalidade.
  - Dados de alta dimensão podem ser um problema ao utilizar a maioria dos classificadores e especialmente redes tradicionais, pois não realizam nenhum tipo de aprendizado de característica de alto nível e são forçados a otimizar uma quantidade enorme de parâmetros. Esta tarefa pode ser facilitada apenas diminuindo a dimensionalidade dos dados, e esse é o objetivo do EA básico. A diminuição da dimensionalidade de tipos de dados específicos, como imagens ou sequências, podem ser tratados por EA específicos
- Regularização.
  - Usa uma função de perda que incentiva o modelo a ter outras propriedades além da capacidade de copiar sua entrada para sua saída. Essas outras propriedades incluem esparsidade da representação, pequena derivada da representação e robustez ao ruído ou à falta de entradas. Um autoencoder regularizado pode ser não-linear e overcomplete, mas ainda pode aprender algo útil sobre a distribuição de dados, mesmo se a capacidade do modelo for grande o suficiente para aprender uma função de identidade trivial

Taxanomia de AEs

- Tolerância a ruído.
  - Além de diferentes propriedades, é desejável para o espaço codificado ser robusto a dados ruidosos.
- Modelo Generativo.
  - A transformação do espaço de característica original no espaço de codificação pode não ser o principal objetivo de um AE. Ocasionalmente, será útil mapear novas amostras no espaço codificado para espaço de característica original. Nesse caso, é necessário um modelo generativo.

AE Básico

- O objetivo principal da maioria dos AEs é executar um processo de fusão de características onde as novas características apresentam algumas propriedades, como menor dimensionalidade, maior esparsidade ou desejável propriedades analíticas.
- O modelo resultante é capaz de mapear novas instâncias no espaço de característica latente.
- Todos AEs compartilham uma origem comum, que pode ser chamada de AE básico

Estrutura

- É uma Rede Neural Artificial *feedforward*, onde as camadas possuem quantidade de unidades simétrica. As camadas não precisam ser simétricas no sentido de funções de ativação ou matrizes de peso.
  - $y = f(x) = \sigma_1\left(w^{(1)}x + b^{(1)}\right)$,
  - r= $g(x) = \sigma_2\left(w^{(2)}y + b^{(2)}\right)$
- EAs profundos são a extensão natural dessa definição para um número maior de camadas.
- Vamos chamar a composição de funções no codificador f, e a composição de funções no decodificador g

Função Objetiva

- Os EAs geralmente baseiam sua função objetiva em uma função de perda por instância

$$\Im(W; b; S) = \sum_{x \in S} \mathcal{L}\big(x, (g \circ f)(x)\big)$$

- onde f e g são funções de codificação e decodificação determinado pelos pesos W e bisa b, assumindo funções de ativação fixas, e S é um conjunto de amostras.
- Por exemplo, uma função de perda típica é a média quadrática erro

$$\mathcal{L}_{MSE}(u, v) = \|u - v\|_2^2$$

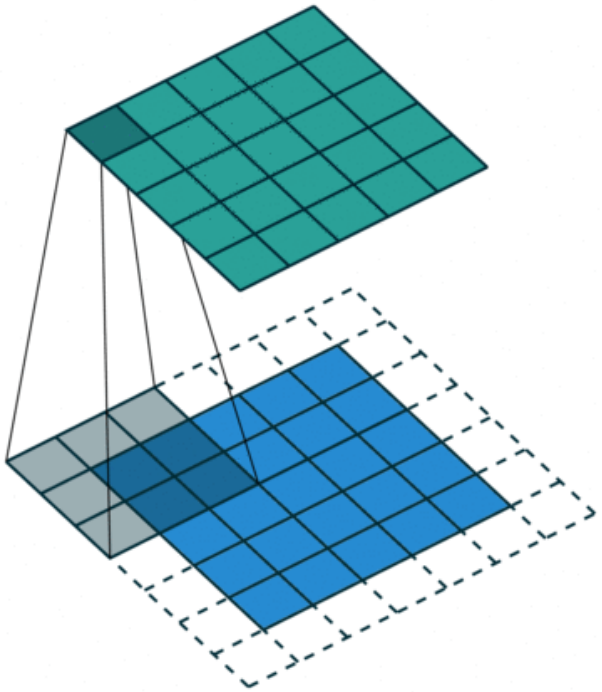
Treinamento

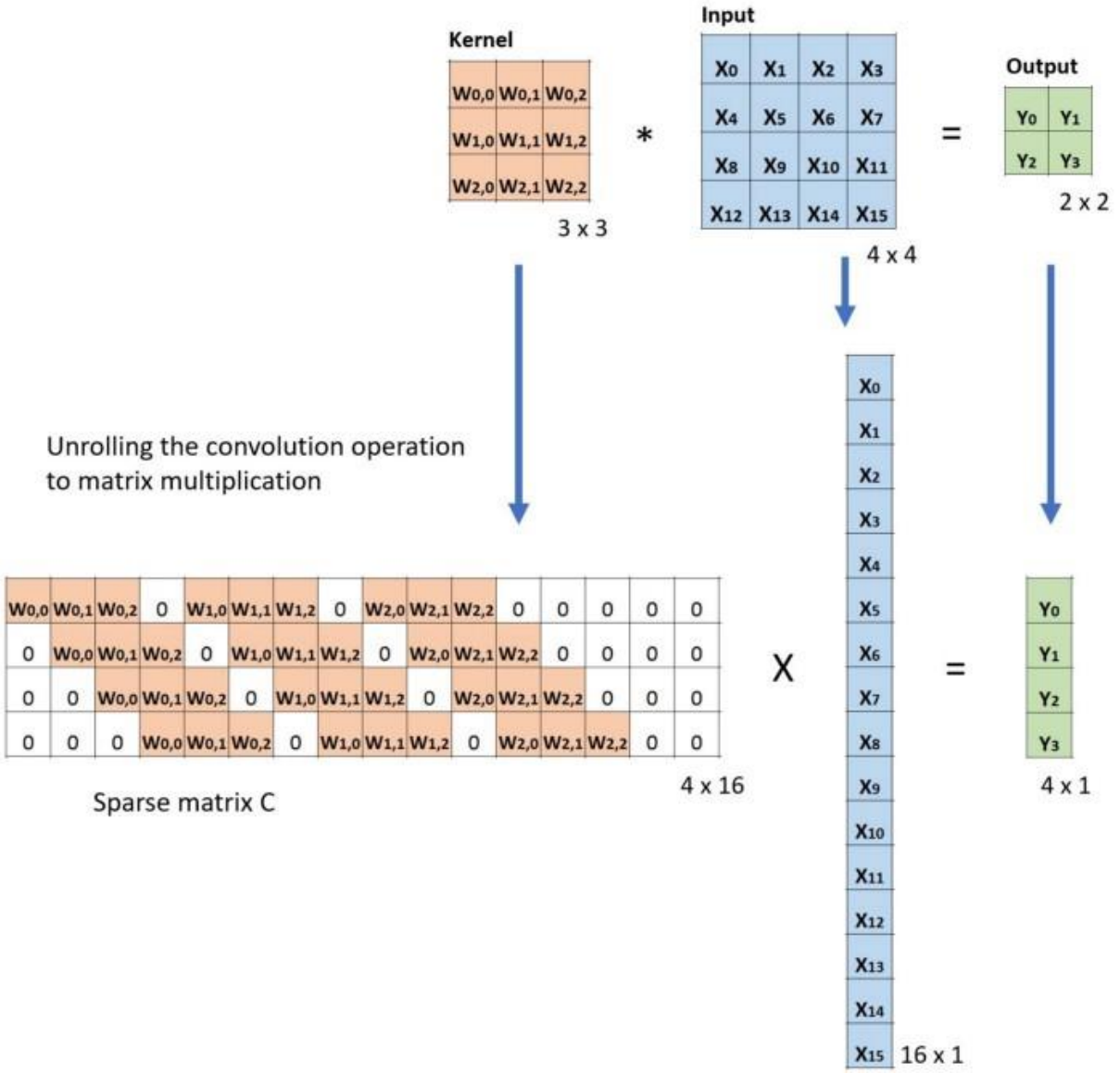
- Algoritmos para otimizar os pesos e bias em AEs são gradiente descendente estocástico e suas variações.
- AEs, como muito outros algoritmos de aprendizado de máquina, são susceptíveis a sobre ajustes dos dados de treinamento
  - Para evitar sobre ajuste um termo de regularização pode ser adicionado para a função objetiva que causa um decaimento nos pesos

$$\Im(W; b; S) = \sum_{x \in S} \mathcal{L}\big(x, (g \circ f)(x)\big) + \lambda \sum_{i} w_i^2$$

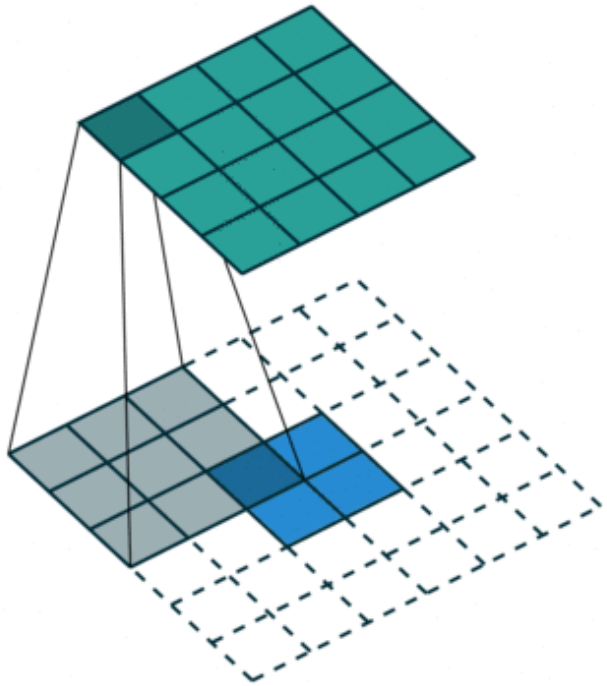
  - Outras restrições e regularizações podem ser aplicadas.

Convolução




Kernel
W0,0 W0,1 W0,2
W1,0 W1,1 W1,2
W2,0 W2,1 W2,2
3 x 3
*
Input
X0 X1 X2 X3
X4 X5 X6 X7
X8 X9 X10 X11
X12 X13 X14 X15
4 x 4
=
Output
Y0 Y1
Y2 Y3
2 x 2
Unrolling the convolution operation
to matrix multiplication
W0,0 W0,1 W0,2 0 W1,0 W1,1 W1,2 0 W2,0 W2,1 W2,2 0 0 0 0 0
0 W0,0 W0,1 W0,2 0 W1,0 W1,1 W1,2 0 W2,0 W2,1 W2,2 0 0 0 0
0 0 W0,0 W0,1 W0,2 0 W1,0 W1,1 W1,2 0 W2,0 W2,1 W2,2 0 0 0
0 0 0 W0,0 W0,1 W0,2 0 W1,0 W1,1 W1,2 0 W2,0 W2,1 W2,2 0 0
4 x 16
Sparse matrix C
X
X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
16 x 1
=
Y0 Y1 Y2 Y3
4 x 1

Convolução Transposta



$W_{0,0}$	0	0	0
$W_{0,1}$	$W_{0,0}$	0	0
$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	0
0	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$
$W_{1,0}$	0	$W_{0,2}$	$W_{0,1}$
$W_{1,1}$	$W_{1,0}$	0	$W_{0,2}$
$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	0
0	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$
$W_{2,0}$	0	$W_{1,2}$	$W_{1,1}$
$W_{2,1}$	$W_{2,0}$	0	$W_{1,2}$
$W_{2,2}$	$W_{2,1}$	$W_{2,0}$	0
0	$W_{2,2}$	$W_{2,1}$	$W_{2,0}$
0	0	$W_{2,2}$	$W_{2,1}$
0	0	0	$W_{2,2}$
0	0	0	0
0	0	0	0

16 x 4

Sparse matrix C^T

X

Y_0
Y_1
Y_2
Y_3

4 x 1

=

X_0
X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_8
X_9
X_{10}
X_{11}
X_{12}
X_{13}
X_{14}
X_{15}

16 x 1

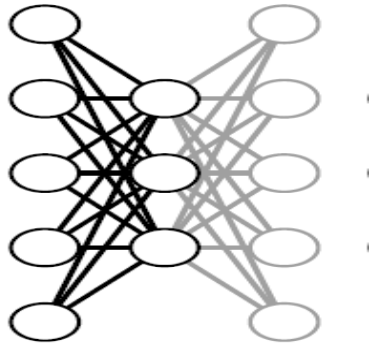
→

X_0	X_1	X_2	X_3
X_4	X_5	X_6	X_7
X_8	X_9	X_{10}	X_{11}
X_{12}	X_{13}	X_{14}	X_{15}

4 x 4

Empilhamento (Stacking)

- Quando os AEs são profundos, o sucesso do processo de treinamento depende fortemente de uma boa inicialização dos pesos, uma vez que pode variar de dezenas, a centenas de milhares.
- Essa inicialização de peso pode ser realizada empilhando sucessivamente AEs rasos, ou seja, treinando AE camada por camada de maneira gulosa.
- O processo de treinamento começa treinando apenas a primeira camada oculta como a codificação de um AE raso, como mostra a rede ao lado.



Empilhamento (Stacking)

- Após esta etapa, a segundo camada oculta é treinada, usando a primeira camada oculta como entrada.
- As entradas são calculadas através de um passagem para frente das entradas originais pela primeira camada, com os pesos determinados na etapa anterior.
- Cada camada sucessiva até a codificação é treinada da mesma maneira
- Após esse treinamento em camadas, os pesos iniciais para todas as camadas anteriores e incluindo a camada de codificação terá sido calculado.
- As camadas restante são adicionadas simetricamente com matrizes de peso resultantes da transposição de cada camada correspondente
- Finalmente, pode ser realizada uma fase de ajuste, otimizando os pesos retropropagando os gradientes através do estrutura completa com dados de treinamento.

AEs Regularizados

- AEs *undercomplete*, com dimensão de código menor que a dimensão de entrada, podem aprender as características mais importantes da distribuição de dados. Esses AEs falham em aprender algo útil se o codificador e o decodificador tiverem muita capacidade.
- Codificações produzidas pelos AEs básicos geralmente não apresentam propriedades especiais.
- Quando algumas propriedades desejáveis são requeridas paras as novas características, algumas regularizações pode ser obtidas adicionando uma penalização para a função objetiva:

$$\Im(W; b; S) = \sum_{x \in S} \mathcal{L}\big(x, (g \circ f)(x)\big) + \lambda\, \Omega(W; b; S)$$

AE Esparsos

- A esparsidade em uma representação significa que a maioria dos valores para um amostra é zero ou quase zero.
- Representações esparsas assemelham-se ao comportamento de células simples no córtex visual primário dos mamíferos, que se acredita ter evoluído para descobrir estratégias de codificação eficientes.
- Isso motiva o uso de transformações de amostras em códigos esparsos em aprendizado de máquina.
- Codificação esparsa também podem ser overcomplete. Este não era necessariamente o caso dos AEs básicos, onde um código overcomplete seria copiado das entradas para saídas

AEs Esparsos

- Quando a esparsidade é desejada na codificação gerada por AE, as ativações da camada de codificação precisam ter valores em média baixo, o que significa unidades na camada oculta geralmente não disparam (valor menor que 1).
- A função de ativação usada naqueles unidades determinaram este valor baixo: no caso de sigmoide e ativações ReLU, os valores baixos estarão próximos de 0; este valor será -1 no caso de tanh e -$\lambda\alpha$ no caso de um SELU
- Uma forma comum de introduzir a esparsidade em um AE é adicionar uma penalidade à função de perda.
- Para comparar o valor desejado das ativações para uma determinada unidade, estas podem ser modeladas como uma variável aleatória de Bernoulli, assumindo que uma unidade pode dispara ou não.

$$\hat{\rho}_i = \frac{1}{|S|}\sum_{x \in S} f_i(x)$$

AE Esparsos

- Para uma entrada específica x, seja o valor médio de ativação de uma unidade na camada oculta, onde $f = (f_1, f_2, \cdots f_c)$, c é o número de unidades na codificação.
- $\widehat{\rho_i}$ será a média da Distribuição de Bernoulli associado.

- Seja ρ ativação média desejada.

- O divergente Kullback-Leibler entre a variável aleatória da unidade *i* e aquela que corresponde à ativação da unidade desejada medi como diferente são ambas as distribuições.

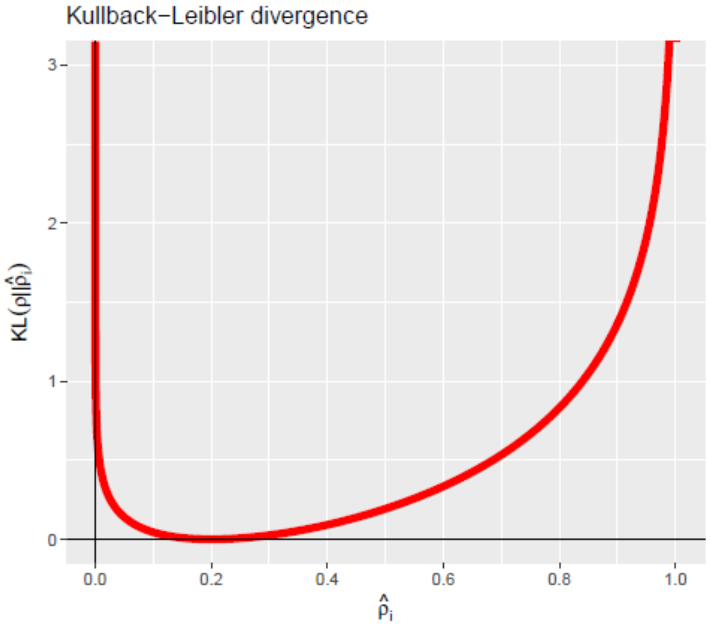
$$\mathrm{KL}(\rho||\hat{\rho}_i) = \rho log\frac{\rho}{\hat{\rho}_i} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_i}$$

AE Esparsos

$$\Im(W;b;S) = \sum_{x \in S} \mathcal{L}\big(x, (g \circ f)(x)\big) + \lambda\, \Omega(W;b;S)$$

- O termo de penalização para função objetiva

$$\Omega(W;b;S) = \sum_{i=1}^{c} KL(\rho || \hat{\rho}_i)$$



ρ=0.2

AE contrativo

- Alta sensibilidade a perturbações nas amostras de entrada pode levar um AE a gerar codificações muito diferentes.
- Isto é geralmente inconveniente, que é a motivação por trás do EA contrativo.
- Alcança invariância local para mudanças em várias direções em torno das amostras de treinamento e é capaz de descobrir mais facilmente estruturas de dimensões inferiores nos dados.
- A sensibilidade para pequenas alterações na entrada pode ser medida como a norma Frobenius $\|.\|_F$ da matriz jacobiana do codificador Jf

AE contrativo

$$\|J_f(x)\|_F^2 = \sum_{j=1}^{d} \sum_{i=1}^{c} \left(\frac{\partial f_i}{\partial x_j}(x) \right)^2$$

- Quanto maior esse valor, mais instáveis são as codificações para perturbações nas entradas.
- Uma regularização é construída, inserindo essa medida na função objetiva do EA contrativo:

$$\Omega_{\mathrm{CAE}}(W, b; S) = \sum_{x \in S} \|J_f(x)\|_F^2$$

AE contrativo

- Um caso particular dessa contração induzida é o uso do decaimento de peso l_2 com um codificador linear: nessa situação, a única maneira de produzir uma contração é manter pesos pequenos.

- No caso não linear, no entanto, a contração pode ser encorajado empurrando unidades ocultas para as regiões de saturação da função de ativação.

Tolerância a Ruído

- Um AE básico pode aprender um espaço de característica latente a partir de conjunto de amostras, mas não garante estabilidade na presença de instâncias ruidosas, nem é capaz de remover ruídos ao reconstruir novas amostras.

- Duas variantes que abordam esse problema são discutidas:
  - EA elimina ruído (denoising)
  - EAs robustos.

AE Elimina Ruído

- Um AE denoising ou DAE aprende a gerar características robustas a partir de entradas, reconstruindo partes parcialmente destruídas das amostras.
- A estrutura e os parâmetros de um DEA são idênticos aos de um EA básico. A diferença reside em um corromper as entradas aplicadas durante a fase de treinamento.
- A técnica de corromper consiste em escolher aleatoriamente uma quantidade de características para cada amostra de treinamento e torná-la igual a o.
- As reconstruções dos EA são, no entanto, comparadas às entradas originais não corrompidas. O EA será treinado para descobrir os valores ausentes.

AE Elimina Ruído

- Um DEA não precisa de mais restrições ou regularizações para aprender uma codificação significativa a partir dos dados, o que significa que podem ser overcomplete, se desejado.
- Quando possui mais de uma camada oculta, pode ser treinada em camadas. Para isso, entradas não corrompidas são computadas como saídas das camadas anteriores, essas são corrompidas e fornecido à rede.
- Note que depois que DAE é treinado, é usado para calcular o nível mais alto das representações sem corromper os dados de entrada.
- Outras formas possíveis:
  - Utilizar um ruído gaussiano aditivo $\hat{x} \sim \mathcal{N}(x, \sigma^2 I)$
  - Utilizar ruído sal e pimenta (*salt-and-pepper*), que define uma fração dos elementos da entrada ao mínimo ou máximo

AE Robusto

- Treinar um AE para recuperar dados corrompidos não é a única maneira de induzir tolerância ao ruído no modelo.
- Uma alternativa é modificar a função de perda usada para minimizar o erro de reconstrução, a fim de diminuir a sensibilidade a diferentes tipos de ruído
- Robust Stacked AEs aplica essa ideia e consegue ser menos afetado pelo ruído não gaussiano do que os EAs padrão.

Robust stacked AEs

- Usam uma função de perda diferente baseada na correntropia, uma medida de similaridade localizada, definida por:

$$\mathcal{L}_{\mathrm{MCC}}(u, v) = -\sum_{k=1}^{d} \mathcal{K}_\sigma(u_k - v_k),$$

$$\text{where } \mathcal{K}_\sigma(\alpha) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\alpha^2}{2\sigma^2}\right),$$

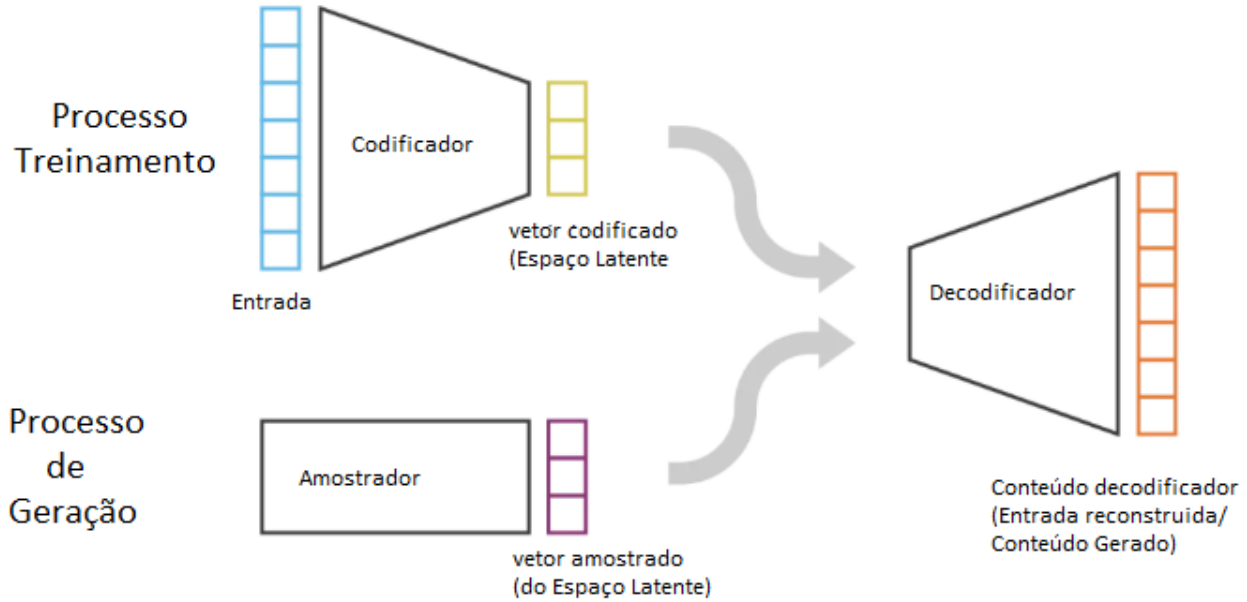
- σ é um parâmetro para o kernel K
- A correntropia mede especificamente a densidade de probabilidade que dois eventos são iguais. Essa métrica é menos afetada por *outliers* do que o MSE. Os EA robustos tentam maximizar essa medida (equivalentemente, minimizar correntropia negativa), que se traduz em maior resiliência ao ruído não gaussiano.

Modelos Generativos

- Uma vez que o Autoencoder foi treinado, temos um codificador e um decodificador, mas ainda não há uma foram real de produzir nova saída (conteúdo).

- Se o espaço latente for regular o suficiente, poderíamos pegar um ponto aleatoriamente nesse espaço latente e decodificá-lo para obter um novo conteúdo. O decodificador agiria mais ou menos como o gerador de uma Rede Adversária Generativa

Modelos Generativos

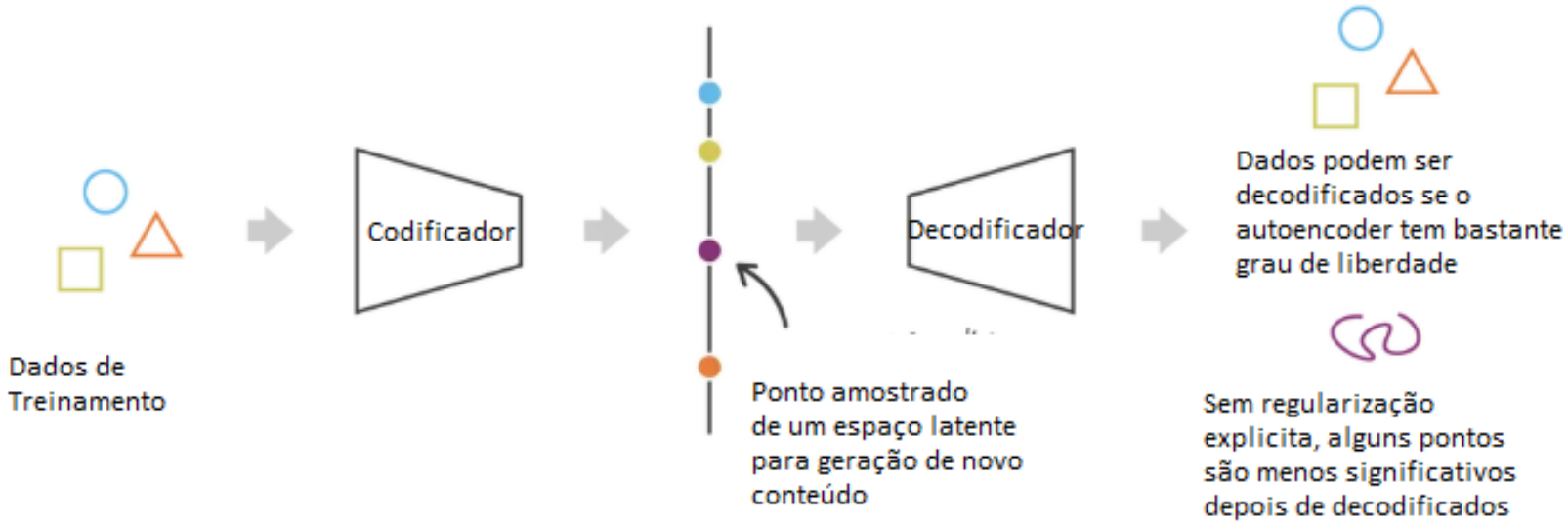


Modelos Generativos

- A regularidade do espaço latente para Autoencoders é um ponto difícil, pois depende da distribuição dos dados no espaço inicial, da dimensão do espaço latente e da arquitetura do codificador.

- Portanto, é bastante difícil (se não impossível) garantir, a priori, que o codificador organize o espaço latente de maneira inteligente, compatível com o processo generativo que acabamos de descrever.

Modelos Generativos



Modelos generativos

- AEs geralmente podem reconstruir dados codificados, mas não são necessariamente capazes de gerar resultados significativos a partir das codificações
- AEs fornecem mecanismos diferentes para reduzir a dimensionalidade de um conjunto de variáveis, logo é possível produzir um modelo generativo a partir dos dados de treinamento
  - Impor regularidade no espaço latente → regularização
- Modelos Generativo aprendem uma distribuição para podem coletar novas amostras, diferentes das observadas.
- AEs variacionais e contraditórios aprendem um modelo dos dados a partir dos quais novas instâncias podem ser geradas.

AE Variacional

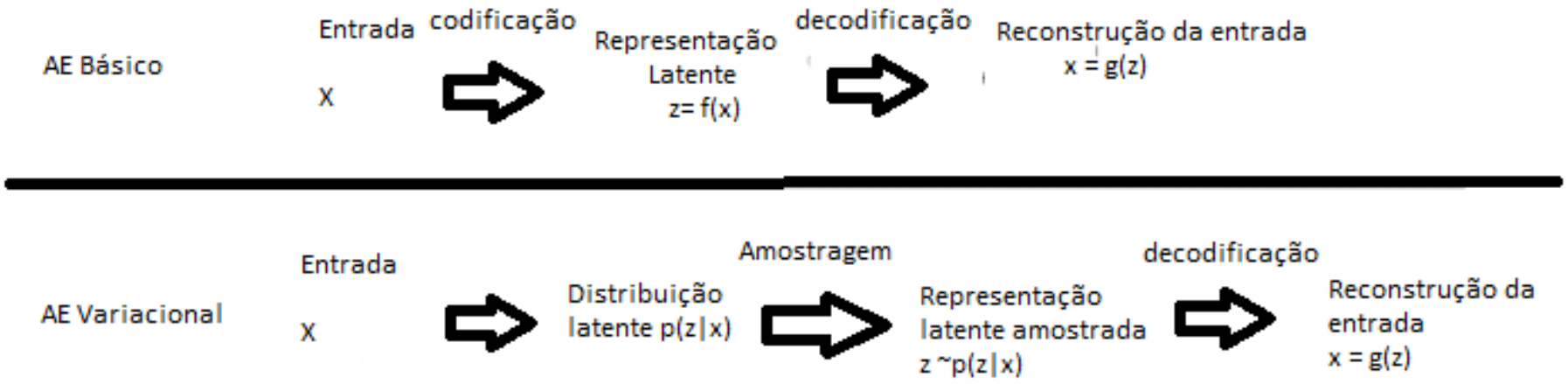
- Uma solução possível para obter essa regularidade é introduzir regularização explícita durante o processo de treinamento.

- Um autoencoder variacional pode ser definido como um autoencoder cujo treinamento é regularizado para evitar sobreajuste e garantir que o espaço latente tenha boas propriedades que possibilitem processos generativos.

AE Variacional

- Um autoencoder variacional é uma arquitetura composta por um codificador e um decodificador, treinada para minimizar o erro de reconstrução entre os dados decodificados e os dados iniciais.
- No entanto, para introduzir alguma regularização do espaço latente, procedemos a uma ligeira modificação do processo de codificação / decodificação: em vez de codificar uma entrada como um único ponto, a codificamos como uma distribuição no espaço latente.
- O modelo é treinado da seguinte maneira:
  - Primeiro, a entrada é codificada como distribuição no espaço latente.
  - Segundo, um ponto do espaço latente é amostrado a partir dessa distribuição.
  - Terceiro, o ponto amostrado é decodificado e o erro de reconstrução pode ser calculado.
  - Finalmente, o erro de reconstrução é retropropagado pela rede.

AE Variacional

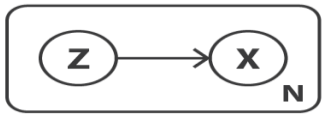


AE Variacional

- Na prática, as distribuições codificadas são escolhidas serem normais, de modo que o codificador possa ser treinado para retornar a média e a matriz de covariância que descrevem essas gaussianos.
- A razão pela qual uma entrada é codificada como uma distribuição com alguma variância em vez de um único ponto é que torna possível expressar muito naturalmente a regularização do espaço latente: as distribuições retornadas pelo codificador são impostas estarem próximas a uma distribuição normal padrão.

Formulação

- Os Autoencoders Variacional são auto-codificadores que codificam entradas como distribuições em vez de pontos e cujo espaço latente "organização" é regularizado restringindo as distribuições retornadas pelo codificador para estarem próximas de um gaussiano padrão.
- Denotamos por x, a variável que representa os dados e assumimos que x é gerado a partir de uma variável latente z.
- Para cada amostra de entrada, o processo generativo é realizado em duas etapas
  - primeiro, uma representação latente z é amostrada da distribuição a priori $p(z)$
  - segundo, os dados x são amostrados da distribuição de probabilidade condicional $p(x|z)$



Formulação

- O "decodificador probabilístico" é definido por $p(x|z)$, que descreve a distribuição da variável decodificada dada a codificada , enquanto o "codificador probabilístico" é definido por $p(z|x)$, que descreve a distribuição da variável codificada, dada a decodificada .

- De acordo com o teorema de Bayes
  - $p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du}$

Como obter $p(z|x)$?

Formulação

- Assume-se que representações codificadas z no espaço latente seguem uma distribuição a priori $p(z)$
- Dada uma função f pertencente a uma família de funções denotadas por F, temos que
  - $p(z) \equiv \mathcal{N}(0, I)$
  - $p(x|z) \equiv \mathcal{N}(f(x), cI) \qquad f \in F \qquad c > 0$
- Logo que p (z) e p (x | z) são ambos distribuição gaussiana.
  - Portanto, se $E(x|z) = f(z) = z$, isso implicaria que $p(z|x)$também deveria seguir uma distribuição gaussiana e, em teoria, poderíamos tentar expressar a média e a matriz de covariância de $p(z|x)$ com relação às médias e matrizes de covariância de $p(z)$ e $p(x|z)$
  - na prática, essa condição não é atendida e precisamos usar uma técnica de aproximação como inferência variacional que torne a abordagem bastante geral e mais robusta a algumas mudanças na hipótese do modelo

Inferência Variacional

- Em estatística, a inferência variacional é uma técnica para aproximar distribuições complexas.
- A ideia é definir uma família de distribuição parametrizada (por exemplo, a família de Gaussianas, cujos parâmetros são a média e a covariância) e procurar a melhor aproximação de nossa distribuição alvo entre essa família.
- O melhor elemento da família é aquele que minimiza uma determinada medição de erro de aproximação (na maioria das vezes a divergência de Kullback-Leibler entre aproximação e alvo) e é encontrada por descida gradiente sobre os parâmetros que descrevem a família

Inferência Variacional

- Aproximado $p(z|x)$ por uma distribuição gaussiana $q_x(z)$ cuja média e covariância são definidas por duas funções, g e h, do parâmetro x.
- Essas duas funções devem pertencer, respectivamente, às famílias de funções G e H que serão especificadas mais tarde, mas que devem ser parametrizadas. Assim, podemos denotar
  - $q_x(z)=\mathcal{N}(g(x),h(x)) \qquad g \in G \qquad h \in H$
- Precisamos agora encontrar a melhor aproximação entre essa família otimizando as funções g e h (de fato, seus parâmetros) para minimizar a divergente de Kullback-Leibler entre a aproximação e a função desejada $p(z|x)$

Inferência Variacional

- Em outras palavras, estamos procurando a função ideal g * e h * de modo que

$$
\begin{aligned}
(g^*, h^*) &= \underset{(g,h)\in G\times H}{\arg\min}\ KL(q_x(z), p(z|x)) \\
&= \underset{(g,h)\in G\times H}{\arg\min}\left(\mathbb{E}_{z\sim q_x}(\log q_x(z)) - \mathbb{E}_{z\sim q_x}\left(\log \frac{p(x|z)p(z)}{p(x)}\right)\right) \\
&= \underset{(g,h)\in G\times H}{\arg\min}\ (\mathbb{E}_{z\sim q_x}(\log q_x(z)) - \mathbb{E}_{z\sim q_x}(\log p(z)) - \mathbb{E}_{z\sim q_x}(\log p(x|z)) + \mathbb{E}_{z\sim q_x}(\log p(x))) \\
&= \underset{(g,h)\in G\times H}{\arg\max}\ (\mathbb{E}_{z\sim q_x}(\log p(x|z)) - KL(q_x(z), p(z))) \\
&= \underset{(g,h)\in G\times H}{\arg\max}\left(\mathbb{E}_{z\sim q_x}\left(-\frac{||x - f(z)||^2}{2c}\right) - KL(q_x(z), p(z))\right)
\end{aligned}
$$

Inferência Variacional

- f também é desconhecido, logo

$$(f^*, g^*, h^*) = \underset{(f,g,h)\in F\times G\times H}{\arg\max} \left(\mathbb{E}_{z\sim q_x} \left(-\frac{||x - f(z)||^2}{2c} \right) - KL(q_x(z), p(z)) \right)$$

- Podemos identificar nessa função objetivo os elementos introduzidos na descrição intuitiva de autoencoder variacional: o erro de reconstrução entre x e $f(z)$ e o termo de regularização dado pela divergência KL entre $q_x(z)$ e $p(z)$ (que é um gaussiano padrão).
- Também podemos notar a constante c que regula o equilíbrio entre os dois termos anteriores. Quanto maior for c, mais assumimos uma alta variação em torno de $f(z)$ para o decodificador probabilístico em nosso modelo e, portanto, favorecemos o termo de regularização sobre o termo de reconstrução (e o oposto acontece se c for baixo).

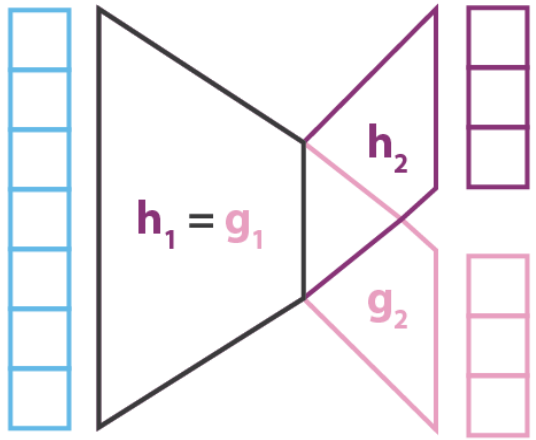
Contexto de Redes Neurais

- Na prática, g e h não são definidos por duas redes completamente independentes, mas compartilham uma parte de sua arquitetura e seus pesos, de modo que temos

$$g(x) = g_2(g_1(x)) \qquad h(x) = h_2(h_1(x)) \qquad g_1(x) = h_1(x)$$

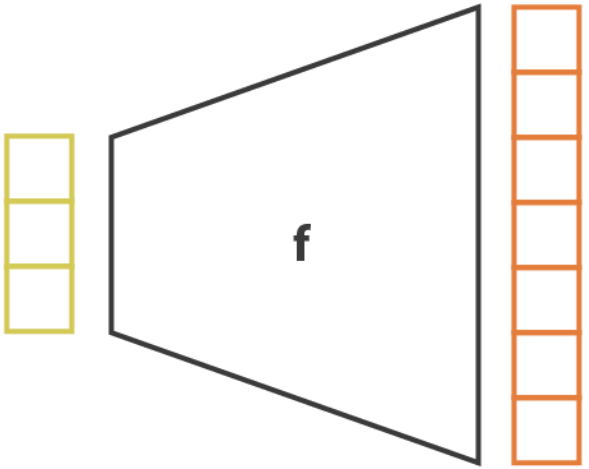
- Como h (x) define a matriz de covariância de q_x (z), deve ser uma matriz quadrada.
- No entanto, para simplificar o cálculo e reduzir o número de parâmetros, assumimos que p(z|x), q_x (z), são uma distribuição gaussiana multidimensional com matriz de covariância diagonal (assunção de independência de variáveis) .
- Com essa suposição, h (x) é simplesmente o vetor dos elementos diagonais da matriz de covariância e, então, tem o mesmo tamanho de g (x). Desta forma reduzimos a família de distribuições que consideramos para inferência variacional e, portanto, a aproximação de p(z|x) obtida pode ser menos precisa.

Contexto de Redes Neurais



x

$\mu_x = \mathrm{g}(\mathrm{x})$
$\sigma_x = h(x)$
$\zeta \sim \mathcal{N}(0, I)$



z

$\hat{x}$

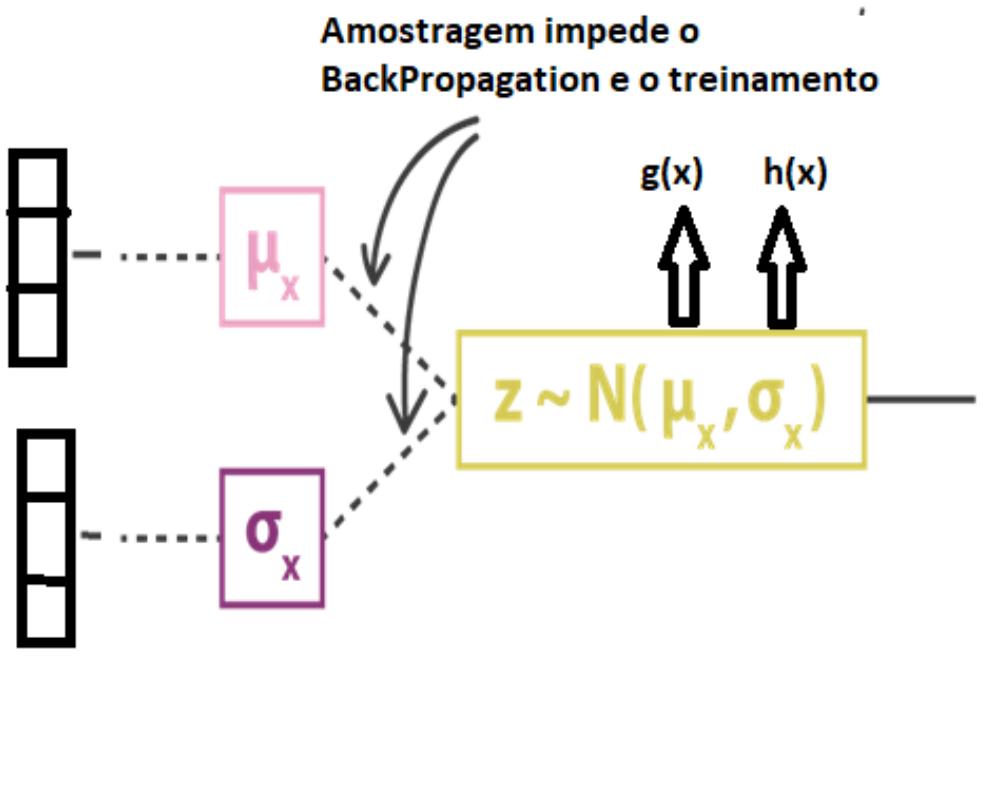
Contexto de Redes Neurais

- **Truque de reparametrização** , é usado para tornar possível a aplicação do gradiente descendente, apesar da amostragem aleatória que ocorre na metade da arquitetura, e consiste em usar o fato de que se z é uma variável aleatória com uma distribuição gaussiana com média g (x) e matriz de covariância $H(x) = h(x)h(x)^T$ então z pode ser expresso como:

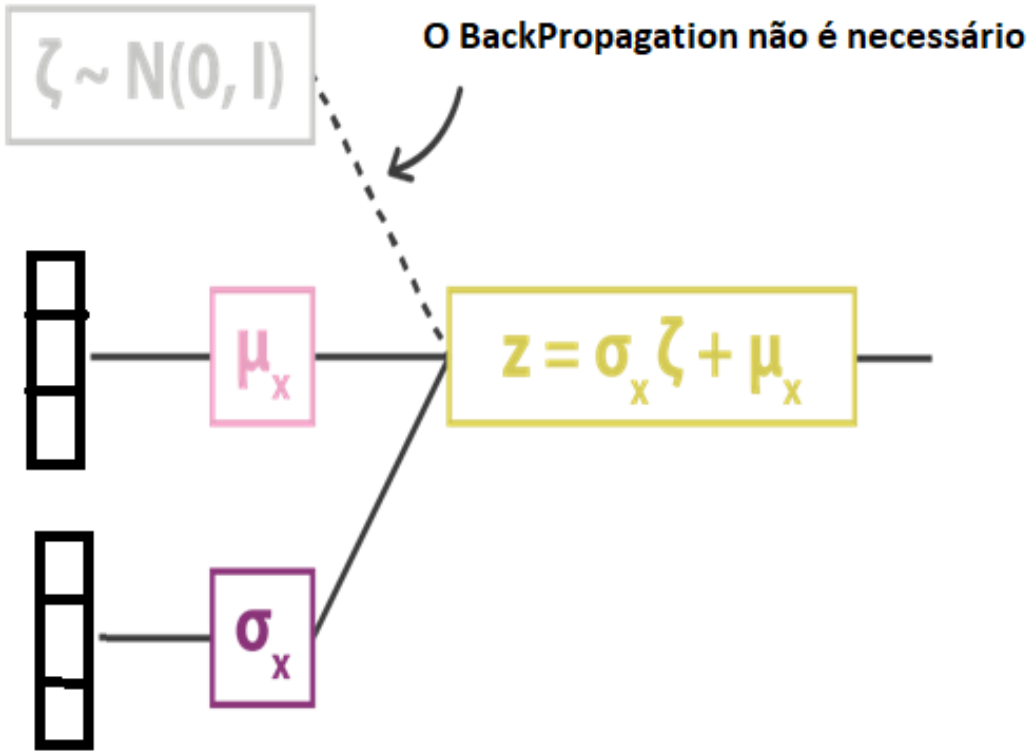
$$z = h(x)\zeta + g(x) \qquad \zeta \sim \mathcal{N}(0, I)$$

Nenhum problema para o BackPropagation

BackPropagation não é possível devido a amostragem

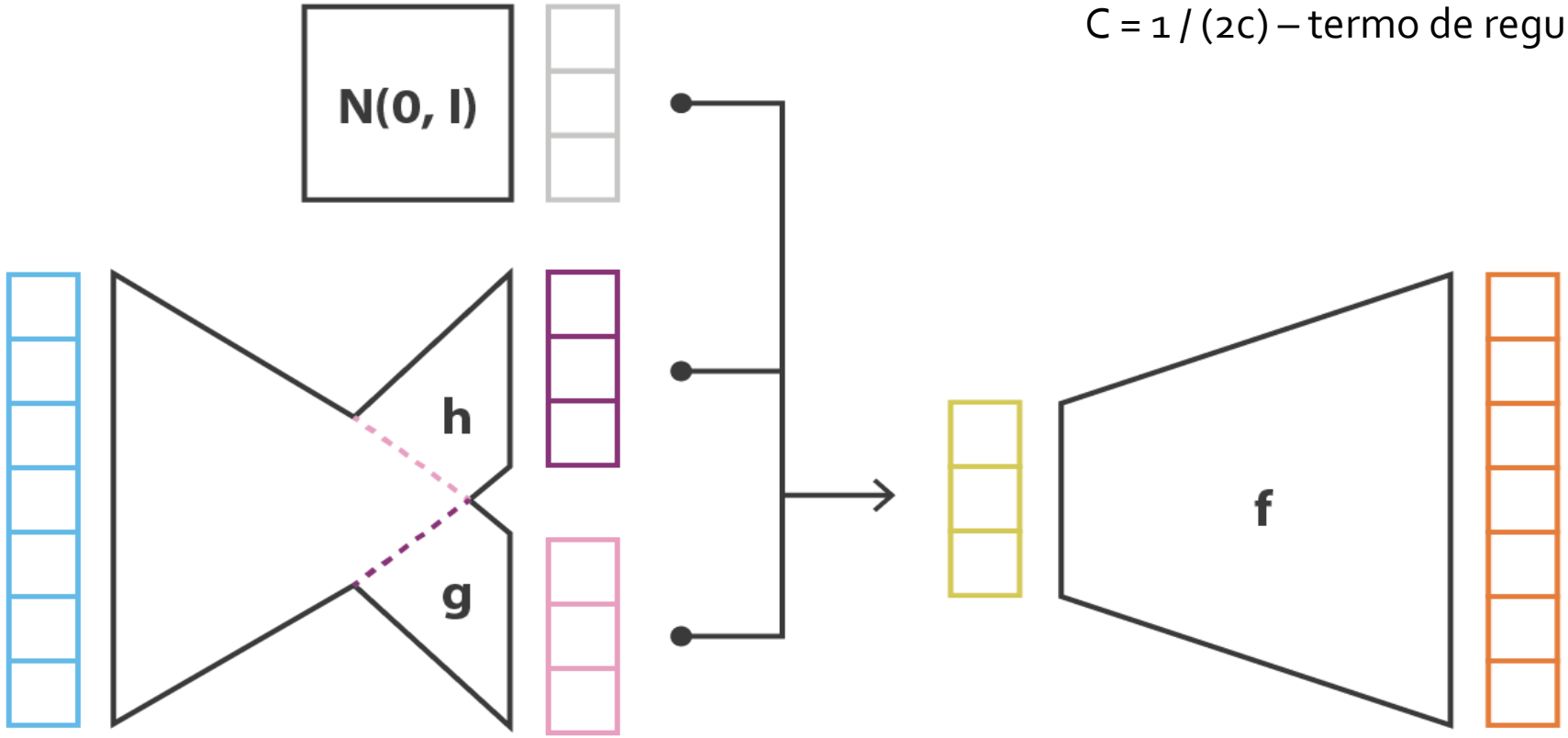


Amostragem sem o truque de reparametrização



Amostragem com o truque de reparametrização

C = 1 / (2c) – termo de regularização



x

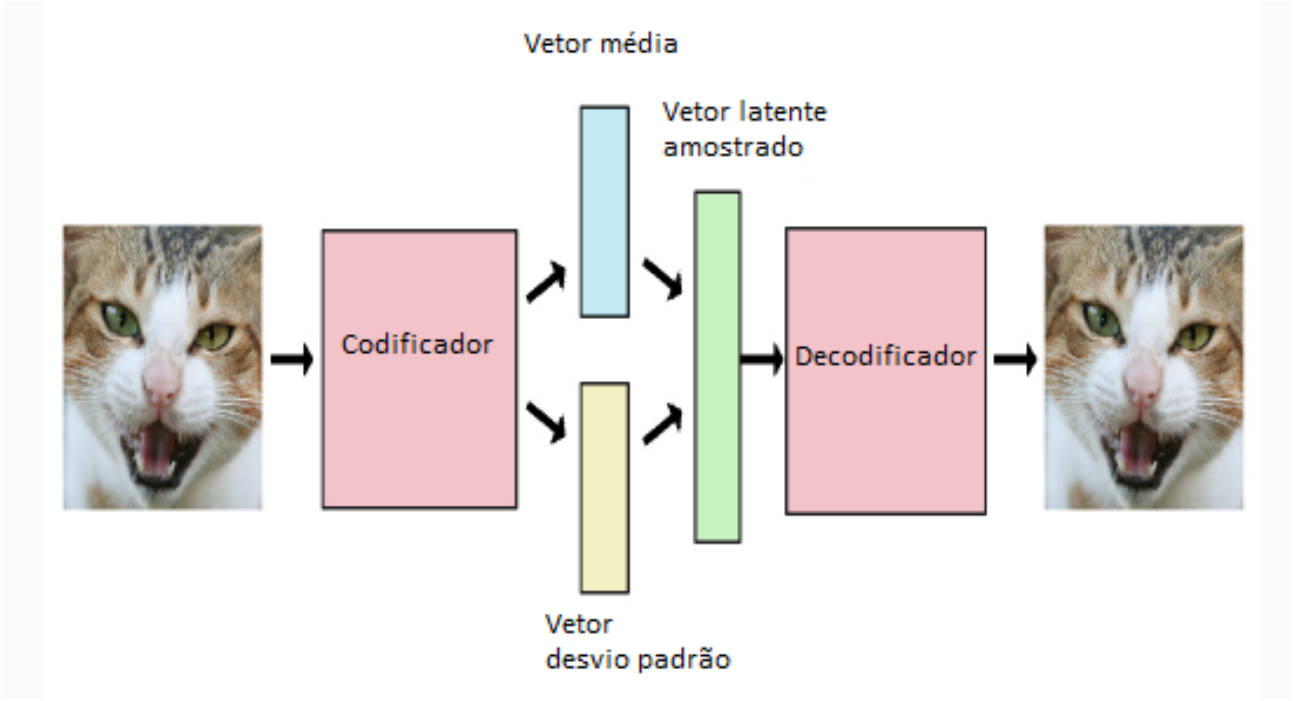
$\mu_x = \mathrm{g(x)}$
$\sigma_x = h(x)$
$\zeta \sim \mathcal{N}(0, I)$

$z = \zeta\sigma_x + \mu_x$

$\hat{x} = f(x)$

$$fob = C\|x - \hat{x}\|^2 + \mathrm{KL}[\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0, I)] = C\|x - \hat{x}\|^2 + \mathrm{KL}[\mathcal{N}\big(g(x), f(x)\big), \mathcal{N}(0, I)]$$

AE Variacional - Exemplo



AE Adversários

- Traz o conceito de Redes Generativas Adversárias ao campo dos EAs.
- Este modela a codificação impondo uma distribuição apriori e, em seguida, treina um EA padrão e, concomitantemente, uma rede tenta distinguir a codificação a partir de amostras da apriori imposta.
- Desde que o gerador (o codificador) é treinado para enganar o discriminador também, as codificações tendem a seguir a distribuição imposta.
- Portanto, os EA adversários também são capazes de gerar novas amostras significativas.

Rede Generativa Adversária

Geração de Variáveis Aleatórias

- Um computador é capaz, usando um gerador de números pseudo-aleatórios, de gerar uma sequência de números que segue aproximadamente uma distribuição aleatória uniforme entre 0 e 1
- O caso uniforme é muito simples, no qual variáveis aleatórias mais complexas podem ser construídas
- Alguns processos de geração de variáveis aleatórias
  - amostragem por rejeição
  - método de transformação inversa

Amostragem por rejeição

- Expressa a variável aleatória como resultado de um processo que consiste em amostrar não a distribuição complexa, mas uma distribuição simples bem conhecida e para aceitar ou rejeitar o valor amostrado, dependendo de alguma condição.
- Repetindo esse processo até que o valor amostrado seja aceito, podemos mostrar que, com a condição correta de aceitação, o valor que será efetivamente amostrado seguirá a distribuição correta.

Transformação inversa

- Método de transformação inversa é uma maneira de gerar uma variável aleatória que segue uma determinada distribuição, fazendo uma variável aleatória uniforme passar por uma "função de transformação".
- Essa noção de "método de transformação inversa" pode, de fato, ser estendida à noção de "método de transformação" que consiste, de maneira mais geral, na geração de variáveis aleatórias em função de algumas variáveis aleatórias mais simples (não necessariamente uniformes, afunção de transformação é inverso CDF).
  - Função de distribuição cumulativa (CDF)
- Conceitualmente, o objetivo da "função de transformação" é deformar / remodelar a distribuição de probabilidade inicial

Modelos Generativos

- Suponha que estamos interessados em gerar imagens quadradas em preto e branco de cães com um tamanho de *n* por *n* pixels
- Podemos remodelar cada dado como um vetor $N = n \times n$ dimensional (empilhando colunas umas sobre as outras), de modo que uma imagem de cachorro possa ser representada por um vetor.
- No entanto, isso não significa que todos os vetores representem um cão que foi moldado de volta a um quadrado
- Portanto, podemos dizer que os vetores N dimensionais que efetivamente dão algo que se parece com um cachorro são distribuídos de acordo com uma distribuição de probabilidade muito específica em todo o espaço vetorial N dimensional
- Existe, nesse espaço vetorial N dimensional, distribuições de probabilidade para imagens de gatos, pássaros e assim por diante.

Modelos Generativos

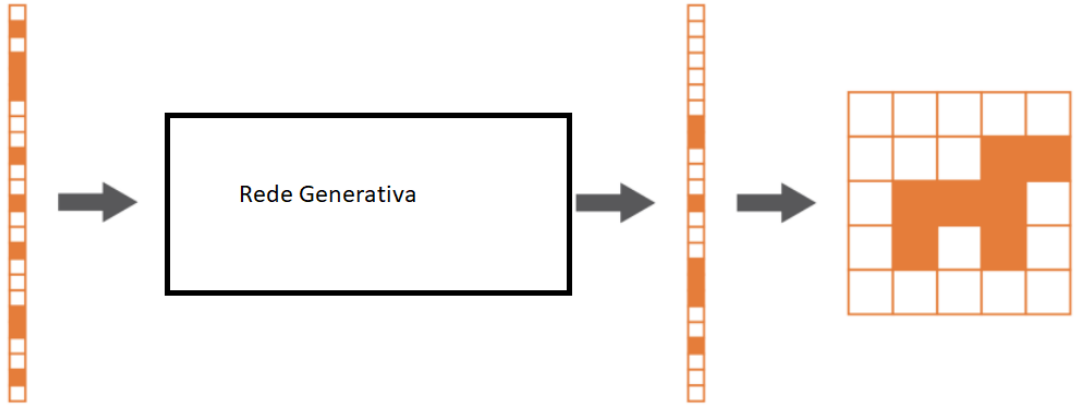
- Então, o problema de gerar uma nova imagem do cão é equivalente ao problema de gerar um novo vetor seguindo a "distribuição de probabilidade do cão" no espaço vetorial N dimensional.
- Então, de fato, estamos enfrentando um problema de gerar uma variável aleatória com relação a uma distribuição de probabilidade específica.
- Dois pontos importantes.
  - A “distribuição de probabilidade canina” que mencionamos é uma distribuição muito complexa em um espaço muito grande.
  - Mesmo se pudermos assumir a existência de tal distribuição subjacente (na verdade existem imagens que se parecem com cachorro e outras que não), obviamente não sabemos como expressar explicitamente essa distribuição.
- Os dois pontos anteriores dificultam bastante o processo de geração de variáveis aleatórias a partir dessa distribuição.

Modelos Generativos

- Nosso primeiro problema ao tentar gerar nossa nova imagem de cachorro é que a "distribuição de probabilidade de cachorro" no espaço vetorial N dimensional é muito complexa e não sabemos como gerar diretamente variáveis aleatórias complexas.
- No entanto, podemos gerar *N* variáveis aleatórias uniformes não correlacionadas, e utilizar o método de transformação.
- Para fazer isso, precisamos expressar nossa variável aleatória *N* dimensional como resultado de uma função muito complexa aplicada a uma variável aleatória *N* dimensional simples!
- A função de transformação não é tão simples, quanto tomar a inversa de forma fechada da Função de Distribuição Cumulativa
- A função de transformação não pode ser expressa explicitamente e, então, precisamos aprender com os dados.

Modelos Generativos

- A ideia é modelar a função de transformação por uma rede neural que recebe como entrada uma variável aleatória uniforme simples *N* dimensional e retorna como saída outra variável aleatória *N* dimensional que deve seguir, após o treinamento, a "distribuição de probabilidade do cão" correta .
- Depois que a arquitetura da rede for projetada, recisamos treiná-la.



Modelos Generativos

- Agora, ainda precisamos treinar (otimizar) a rede para expressar a função de transformação correta.
- Para isso, podemos sugerir dois métodos diferentes de treinamento:
  - Método direto - consiste em comparar as distribuições de probabilidade verdadeira e gerada e retropropagar a diferença (o erro) através da rede. Essa é a ideia que governa as Redes de Correspondência Generativas (GMNs).
  - Método indireto. - treinamos a rede generativa, fazendo com que essas duas distribuições passem por uma tarefa escolhida de forma que o processo de otimização da rede generativa em relação à tarefa imponha que a distribuição gerada esteja próxima da verdadeira distribuição. Essa última ideia é a que está por trás das Redes Adversárias Generativas (GANs)

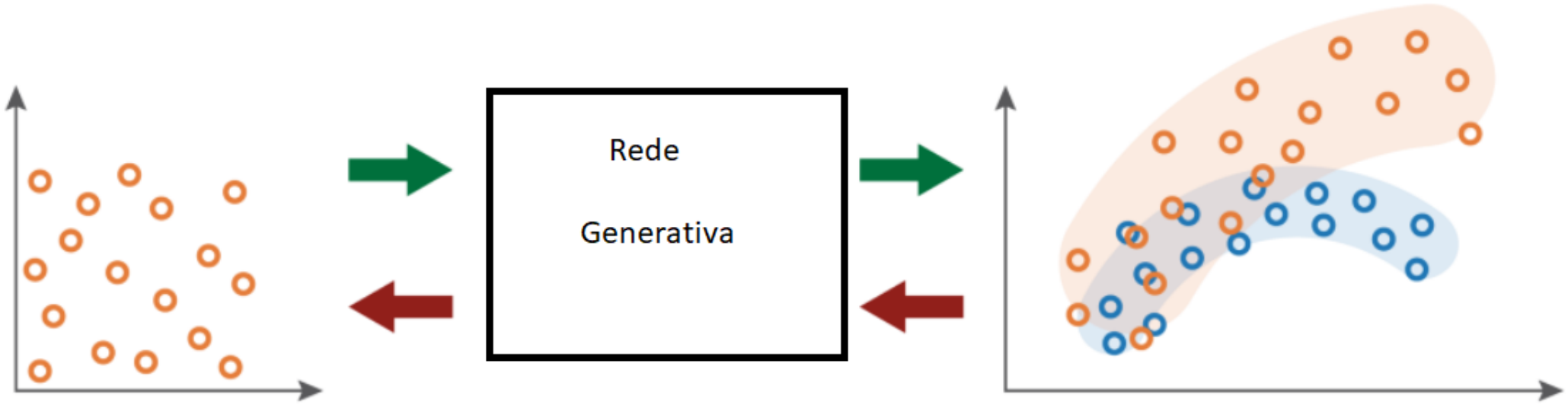
Comparando duas distribuições de probabilidade

- Não sabemos como expressar explicitamente a verdadeira "distribuição de probabilidade de cães" e também podemos dizer que a distribuição gerada é complexa demais para ser expressa explicitamente
- Se tivermos uma maneira de comparar distribuições de probabilidade com base em amostras, podemos usá-la para treinar a rede.
- Dada uma amostra de dados verdadeiros, podemos, a cada iteração do processo de treinamento, produzir uma amostra de dados gerados.
- Como comparar efetivamente duas distribuições com base em sua amostras?

Rede de Correspondência Generativas (Generative Matching Networks)

- Abordagem da Discrepância Média Máxima (MMD)
- A ideia das GMNs é otimizar a rede repetindo os seguintes etapas:
  - 1) Gerar algumas entradas uniformes
  - 2) Passar essas entradas pela rede e coletar as saídas geradas
  - 3) Comparar a verdadeira "distribuição de probabilidade de cães" e a gerada com base nas amostras disponíveis (por exemplo, calcule a distância MMD entre a amostra de imagens reais de cães e a amostra de imagens geradas)
  - 4) Usar retropropagação para fazer uma etapa de descida de gradiente para diminuir a distância (por exemplo, MMD) entre distribuições verdadeiras e geradas

Rede de Correspondência Generativas



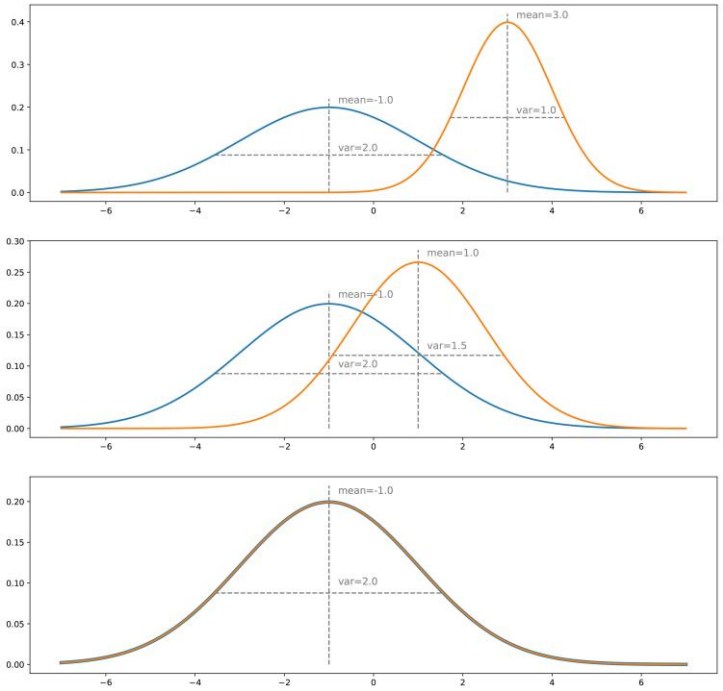
Redes Generativas Adversárias

- A ideia que governa as GANs consiste em substituir essa comparação direta por uma indireta que assume a forma de uma tarefa sobre essas duas distribuições.
- O treinamento da rede generativa é então realizado com relação a essa tarefa, de modo que força a distribuição gerada a se aproximar cada vez mais da distribuição verdadeira.
- A tarefa utilizada nas GANs é uma tarefa de discriminação entre amostras verdadeiras e geradas. Ou podemos dizer uma tarefa de "não discriminação", pois queremos que a discriminação falhe o máximo possível.
- Portanto, em uma arquitetura GAN, temos um discriminador, que coleta amostras de dados verdadeiros e gerados e tenta classificá-los da melhor maneira possível, e um gerador treinado para enganar o discriminador o máximo possível.

Caso ideal: gerador e discriminador perfeitos

- Suponha que tenhamos uma distribuição verdadeira, por exemplo, uma gaussiana unidimensional e que desejamos um gerador que faça amostras dessa distribuição de probabilidade.
- O que chamamos de método de treinamento "direto" consiste em ajustar iterativamente o gerador (iterações de descida de gradiente) para corrigir a diferença / erro medido entre distribuições verdadeiras e geradas.
- Por fim, supondo que o processo de otimização seja perfeito, devemos terminar com a distribuição gerada que corresponda exatamente à verdadeira distribuição.

Caso ideal: gerador e discriminador perfeitos



- Se as duas distribuições estiverem distantes, o discriminador será capaz de classificar facilmente e com um alto nível de confiança a maioria dos pontos que apresentamos. Se queremos enganar o discriminador, precisamos aproximar a distribuição gerada da verdadeira.
- O discriminador terá mais dificuldade em prever a classe quando as duas distribuições forem iguais em todos os pontos.

Complexidade do Método Indireto

- Parece correto se perguntar se esse método indireto é realmente uma boa ideia.
- De fato, parece ser mais complicado (precisamos otimizar o gerador com base em uma tarefa, em vez de diretamente nas distribuições) e requer um discriminador que consideramos aqui como um determinado oráculo, mas que, na realidade, não é conhecido. nem perfeito.
- Para o primeiro ponto, a dificuldade de comparar diretamente duas distribuições de probabilidade com base em amostras contrabalança a aparente maior complexidade do método indireto. Para o segundo ponto, é óbvio que o discriminador não é conhecido. No entanto, pode ser aprendido

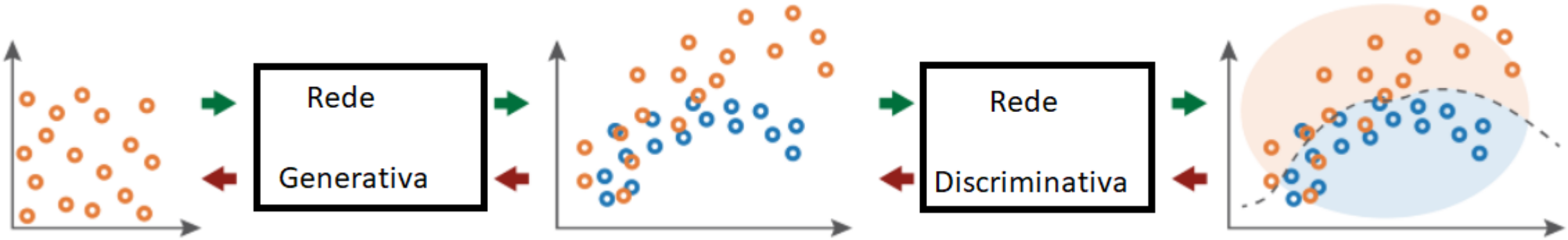
A aproximação: redes neurais adversárias

- O gerador é uma rede neural que modela uma função de transformação. Ele assume como entrada uma variável aleatória simples e deve retornar, uma vez treinada, uma variável aleatória que segue a distribuição alvo.
- Como é muito complicado e desconhecido, decidimos modelar o discriminador com outra rede neural. Essa rede neural modela uma função discriminativa.
- Ele assume como entrada um ponto (no nosso exemplo de cachorro, um vetor N dimensional) e retorna como saída a probabilidade desse ponto ser "verdadeiro".

A aproximação: redes neurais adversárias

- Uma vez definidas, as duas redes podem ser treinadas em conjunto (ao mesmo tempo) com objetivos opostos:
  - o objetivo do gerador é enganar o discriminador; portanto, a rede neural generativa é treinada para maximizar o erro de classificação final (entre dados verdadeiros e gerados)
  - o objetivo do discriminador é detectar dados falsos gerados, a rede neural discriminativa será treinada para minimizar o erro de classificação final

A aproximação: redes neurais adversárias

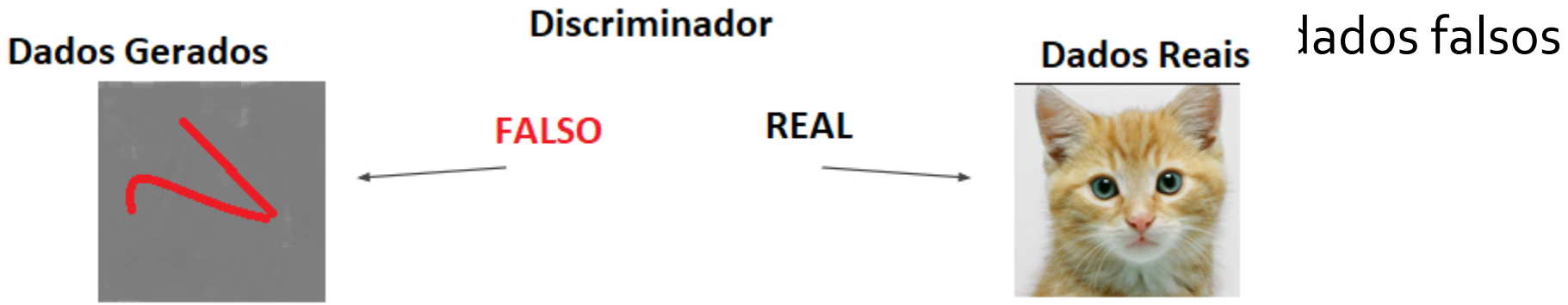


Rede Adversária Generativa (GAN)

- È composto por duas partes
  - O gerador aprende a gerar dados plausíveis. As instâncias geradas se tornam exemplos negativos de treinamento para o discriminador.
  - O discriminador aprende a distinguir os dados falsos do gerador dos dados reais. O discriminador penaliza o gerador por produzir resultados não plausíveis.

Rede Adversária Generativa (GAN)

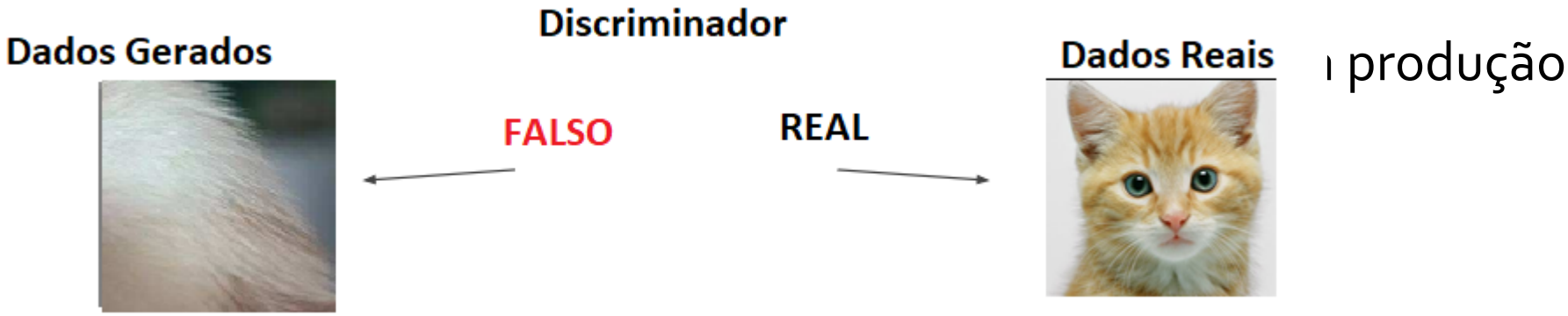
- Qua
  e o c



dados falsos

Rede Adversária Generativa (GAN)

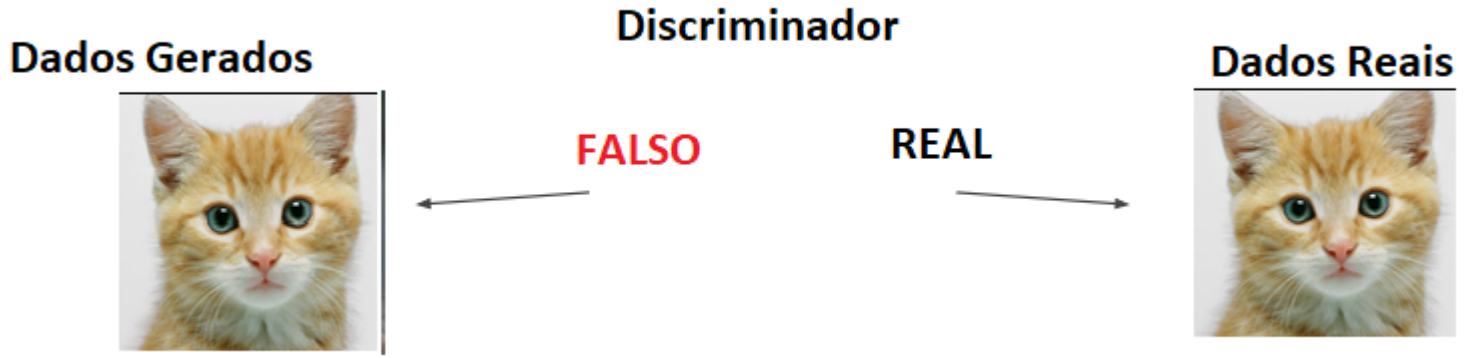
- À m
  que



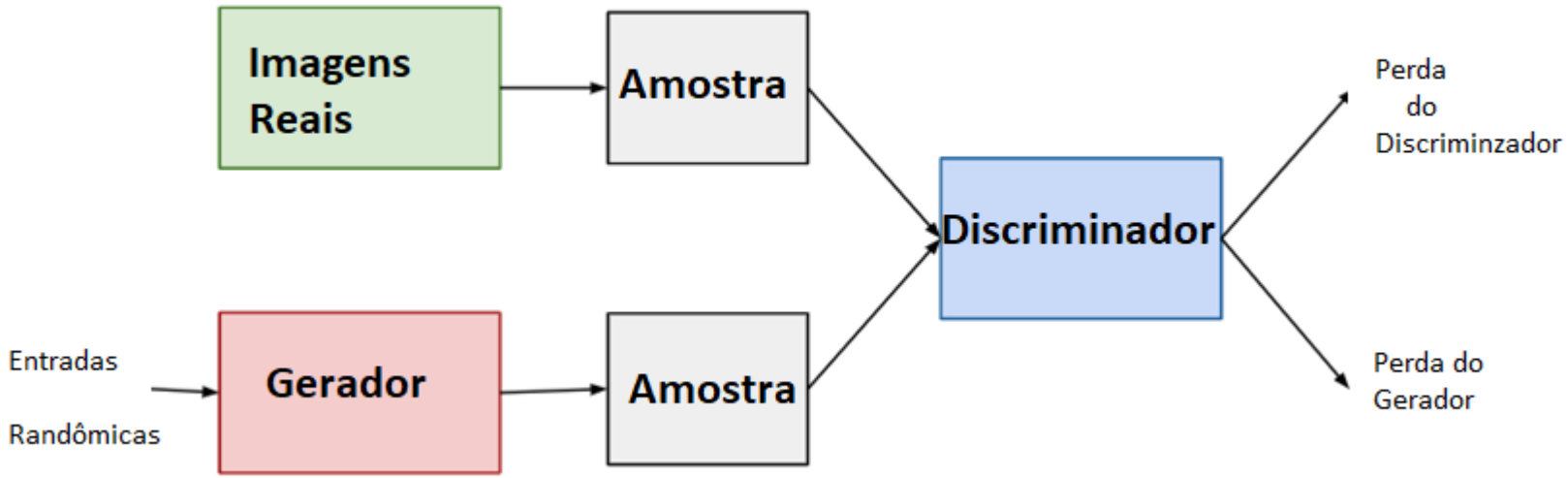
produção

Rede Adversária Generativa (GAN)

- Finalmente, se o treinamento do gerador for bem sucedido, discriminador fica p[illegible] sificar dado



Rede Adversária Generativa (GAN)



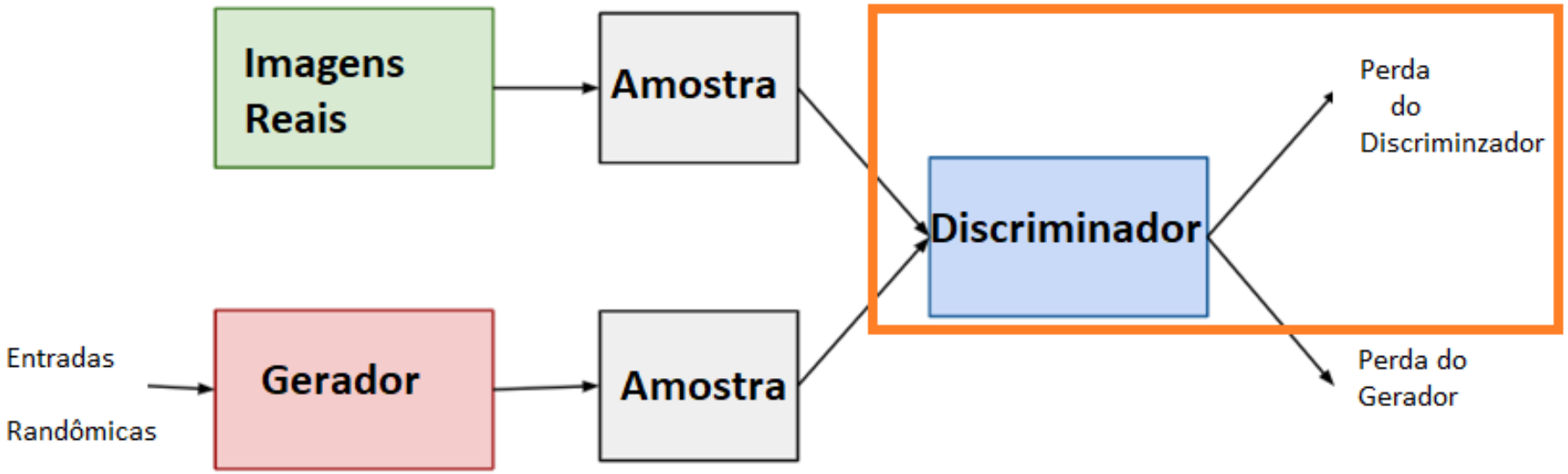
- Tanto o gerador quanto o discriminador são redes neurais. A saída do gerador é conectada diretamente à entrada do discriminador.

  Por meio da retropropagação, a classificação do discriminador fornece um sinal que o gerador usa para atualizar seus pesos.

O Discriminador

- O discriminador é um classificador.
- Ele tenta distinguir dados reais dos dados criados pelo gerador.
- Ele pode usar qualquer arquitetura de rede apropriada ao tipo de dados que está sendo classificando.

BackPropagation no treinamento Discriminador



Dados de treinamento do Discriminador

- Os dados de treinamento do discriminador vêm de duas fontes:
  - Instâncias de dados reais, como imagens reais de pessoas. O discriminador usa essas instâncias como exemplos positivos durante o treinamento.
  - Instâncias de dados falsas criadas pelo gerador. O discriminador usa essas instâncias como exemplos negativos durante o treinamento.
- Na figura anterior, as duas caixas "Amostra" representam essas duas fontes de dados que alimentam o discriminador.
- Durante o treinamento do discriminador, o gerador não é treinado. Seus pesos permanecem constantes enquanto produz exemplos para que discriminador seja treinado.

Treinamento do Discriminador

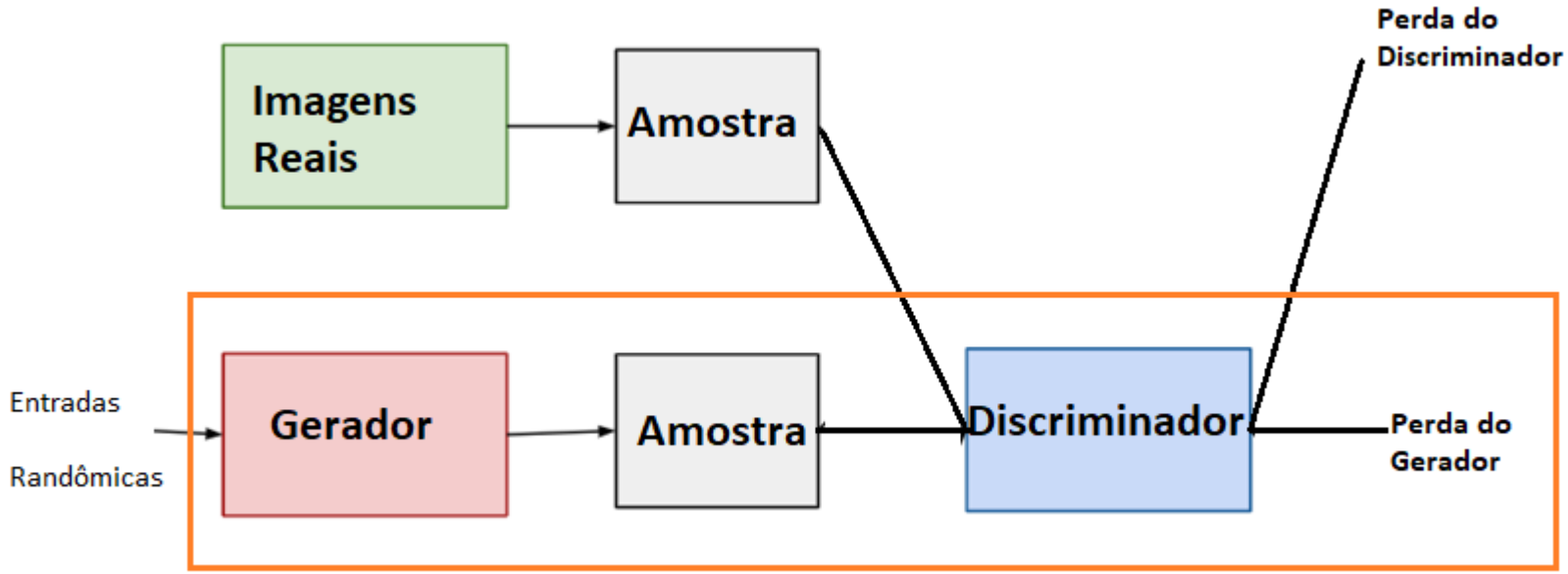
- O discriminador se conecta a duas funções de perda.
- Durante o treinamento do discriminador, o discriminador ignora a perda do gerador e apenas usa a perda do discriminador. Usamos a perda do gerador durante o treinamento do gerador, conforme descrito a frente.

- Durante o treinamento do discriminador:
  - O discriminador classifica dados reais e dados falsos do gerador.
  - A perda do discriminador penaliza o discriminador por classificar incorretamente uma instância real como falsa ou uma instância falsa como real.
  - O discriminador atualiza seus pesos por retropropagação da perda de discriminador através da rede de discriminadores.

O Gerador

- A parte geradora de uma GAN aprende a criar dados falsos incorporando feedback do discriminador. Aprende a fazer o discriminador classificar sua saída como real.
- O treinamento do gerador requer uma integração mais estreita entre o gerador e o discriminador. A parte da GAN que treina o gerador inclui:
  - entrada aleatória
  - rede geradora, que transforma a entrada aleatória em uma instância de dados
  - rede discriminadora, que classifica os dados gerados
  - saída discriminadora
  - perda de gerador, que penaliza o gerador por não enganar o discriminador

Backpropagation para Gerador



Entrada aleatória

- Como redes neurais precisam de alguma forma de entrada. Normalmente, inserimos dados com o que queremos fazer algo, como uma instância sobre o que queremos classificar ou fazer uma previsão.
- Como usar a entrada para uma rede que gera instâncias de dados totalmente novas?
- Na sua forma mais básica, um GAN recebe um ruído aleatório como entrada. O gerador então transforma esse ruído em uma saída significativa. Ao exibir o ruído, podemos obter a GAN para produzir uma ampla variedade de dados, amostrando diferentes locais na distribuição alvo.
- Resultados experimentais sugeri que a distribuição do ruído não importa muito; portanto, podemos escolher algo fácil de amostrar como uma distribuição uniforme. Por conveniência, o espaço do qual o ruído é amostrado é menor que a dimensão do espaço de saída.

Usando o discriminador para treinar o gerador

- Para treinar uma rede neural, alteramos os pesos da rede para reduzir o erro ou a perda de sua saída. No entanto, o gerador não está diretamente conectado à perda que estamos tentando afetar. O gerador alimenta a rede discriminadora e produz a saída que estamos tentando afetar. A perda do gerador penaliza o gerador por produzir uma amostra que a rede discriminadora classifica como falsa.

- Esse pedaço extra de rede deve ser incluído na retropropagação. A retropropagação ajusta cada peso na direção correta, calculando o impacto do peso na saída - como a saída mudaria se você alterasse o peso. Mas o impacto do peso de um gerador depende do impacto dos pesos do discriminador. Assim, a retropropagação começa na saída e volta através do discriminador para o gerador.

Usando o discriminador para treinar o gerador

- Ao mesmo tempo, não queremos que o discriminador mude durante o treinamento do gerador. Tentar atingir um alvo em movimento tornaria um problema ainda mais difícil para o gerador.
- Então, treinamos o gerador com o seguinte procedimento:
  - Gerar uma amostra de ruído aleatório.
  - Produzir a saída do gerador a partir de ruído aleatório amostrado.
  - Obter a classificação discriminadora "Real" ou "Fake" para a saída do gerador.
  - Calcular a perda da classificação discriminadora.
  - Retropropagar através do discriminador e do gerador para obter gradientes.
  - Usar o gradiente para alterar apenas os pesos do gerador.
- Esta é uma iteração do treinamento do gerador.

Treinamento GAN

- Como uma GAN contém duas redes treinadas separadamente, seu algoritmo de treinamento deve abordar duas complicações:
  - As GANs devem conciliar dois tipos diferentes de treinamento
    - (gerador e discriminador).
  - A convergência do GAN é difícil de identificar.

Treinamento Alternado

- O gerador e o discriminador têm diferentes processos de treinamento. Então, como treinamos o GAN como um todo?

- O treinamento da GAN prossegue em períodos alternados:
  - 1) O discriminador é treinado por uma ou mais épocas.
  - 2) O gerador é treinado por uma ou mais épocas.
  - 3) Repita as etapas 1 e 2 para continuar treinando as redes de gerador e discriminador.

Treinamento Alternado

- Mantemos o gerador constante durante a fase de treinamento do discriminador. Como o treinamento de discriminadores tenta descobrir como distinguir dados reais de falsos, ele precisa aprender a reconhecer as falhas do gerador. Esse é um problema diferente para um gerador completamente treinado do que para um gerador não treinado que produz saída aleatória.

- Da mesma forma, mantemos o discriminador constante durante a fase de treinamento do gerador. Caso contrário, o gerador tentaria atingir um alvo em movimento e talvez nunca convergisse.

- É isso que permite que as GANs resolvam problemas de geradores intratáveis. Nós nos deparamos com o problema generativo difícil, começando com um problema de classificação muito mais simples. Por outro lado, se você não pode treinar um classificador para diferenciar dados reais e gerados, mesmo para a saída inicial do gerador aleatório, não poderá iniciar o treinamento da GAN.

Algumas variações de GAN

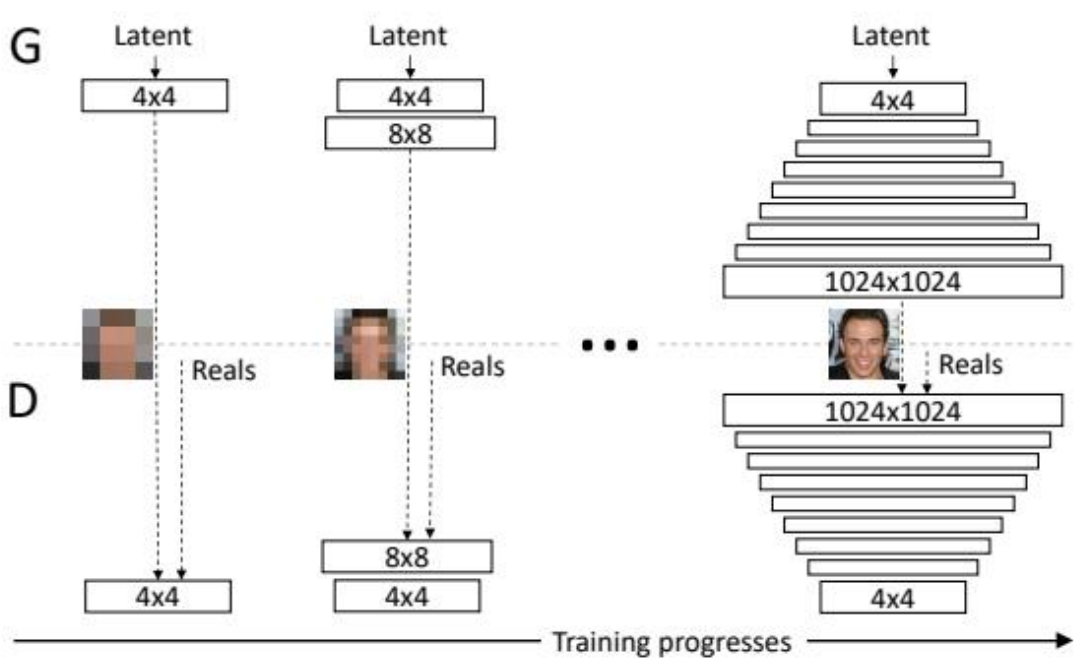
- Há algumas variações de GAN e aplicações.
- A seguir apresentamos uma amostra das variações do GAN para dar uma ideia das possibilidades.

GANs progressivos

- Em um GAN progressiva, as primeiras camadas do gerador produzem imagens de resolução muito baixa e as camadas subsequentes adicionam detalhes.

- Essa técnica permite que a GAN treine mais rapidamente que GANs não progressivos e produz imagens de alta resolução.

GANs Progressivas



GANs Progressivas

- O treinamento começa com o gerador (G) e o discriminador (D) com uma resolução espacial baixa de 4 × 4 pixels. À medida que o treinamento avança, adicionamos camadas progressivamente a G e D, aumentando assim a resolução espacial das imagens geradas. Todas as camadas existentes permanecem treináveis durante todo o processo.

- Aqui N × N refere-se a camadas convolucionais que operam em N × N espacial resolução. Isso permite uma síntese estável em altas resoluções e também acelera o treinamento consideravelmente.

GANs condicionais

- As GANs condicionais são treinados em um conjunto de dados rotulados e permitem especificar o rótulo para cada instância gerada.
- Por exemplo, um MNIST GAN incondicional produziria dígitos aleatórios, enquanto um MNIST GAN condicional permitiria especificar qual dígito a GAN deve gerar.
- Em vez de modelar a probabilidade conjunta P (X, Y), as GANs condicionais modelam a probabilidade condicional P (X | Y).

Tradução Imagem para Imagem

- As GANs pegam uma imagem como entrada e mapeiam-na para uma imagem de saída gerada com propriedades diferentes.

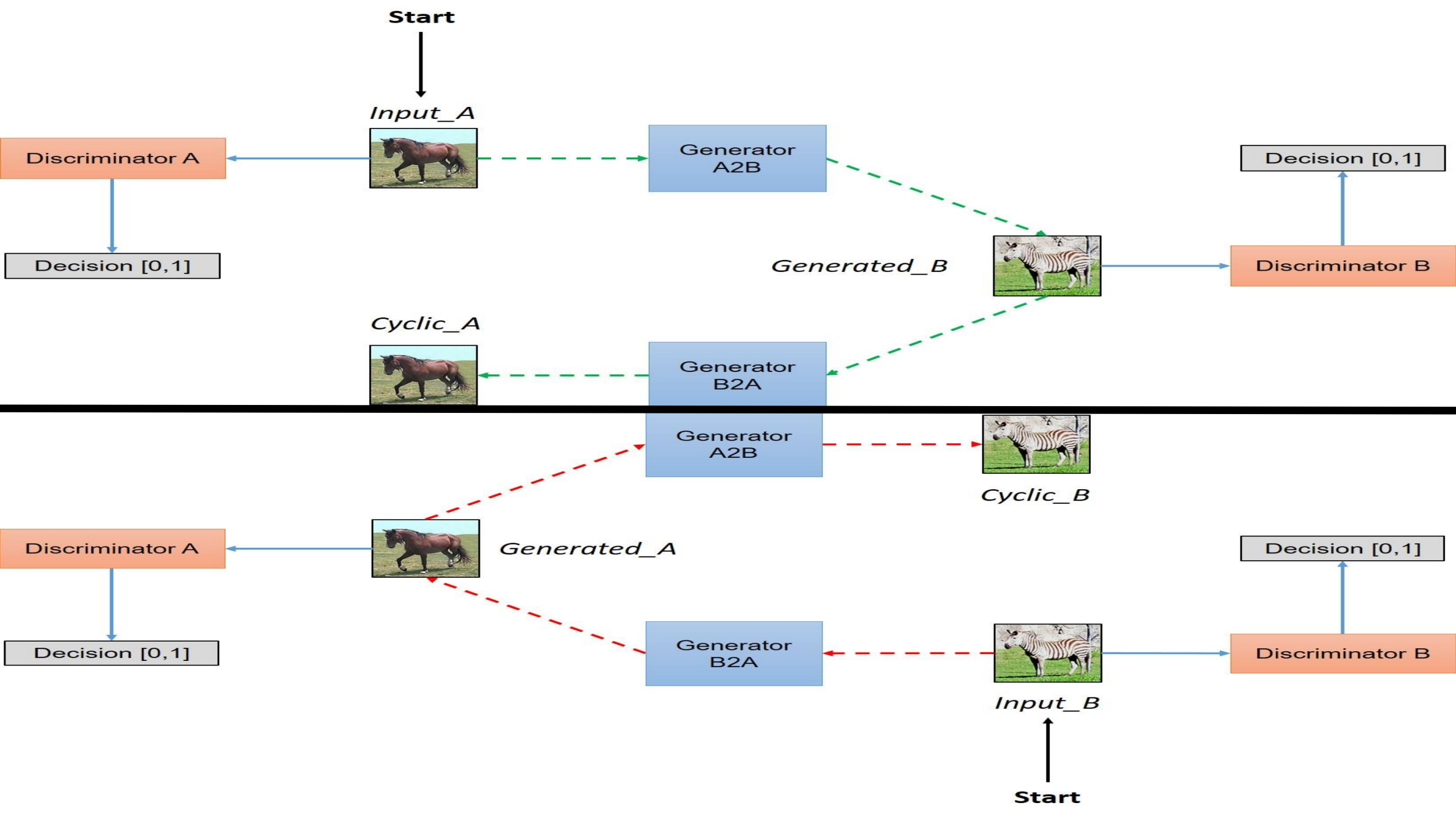
- Por exemplo, pode treinar um GAN para tirar esboços de bolsas e transformá-los em imagens foto realistas de bolsas.

Tradução Imagem para Imagem

- Nesses casos, a perda é uma combinação ponderada da perda baseada na discriminador usual e uma perda em pixel que penaliza o gerador por se afastar da imagem de origem.

CycleGAN

- Os CycleGANs aprendem a transformar imagens de um conjunto em imagens que poderiam pertencer a outro conjunto.
- Por exemplo, um CycleGAN produziu a imagem abaixo à direita quando recebeu a imagem à esquerda como entrada. Ele pegou a imagem de um cavalo e transformou-a na imagem de uma zebra.
- Os dados de treinamento do CycleGAN são simplesmente dois conjuntos de imagens
  - um conjunto de imagens de cavalos e um conjunto de imagens de zebra.


Start
Input_A
Discriminator A
Decision [0,1]
Generator A2B
Generated_B
Decision [0,1]
Discriminator B
Cyclic_A
Generator B2A
Generator A2B
Cyclic_B
Discriminator A
Generated_A
Decision [0,1]
Generator B2A
Input_B
Discriminator B
Start