Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Autoencoders e GANs

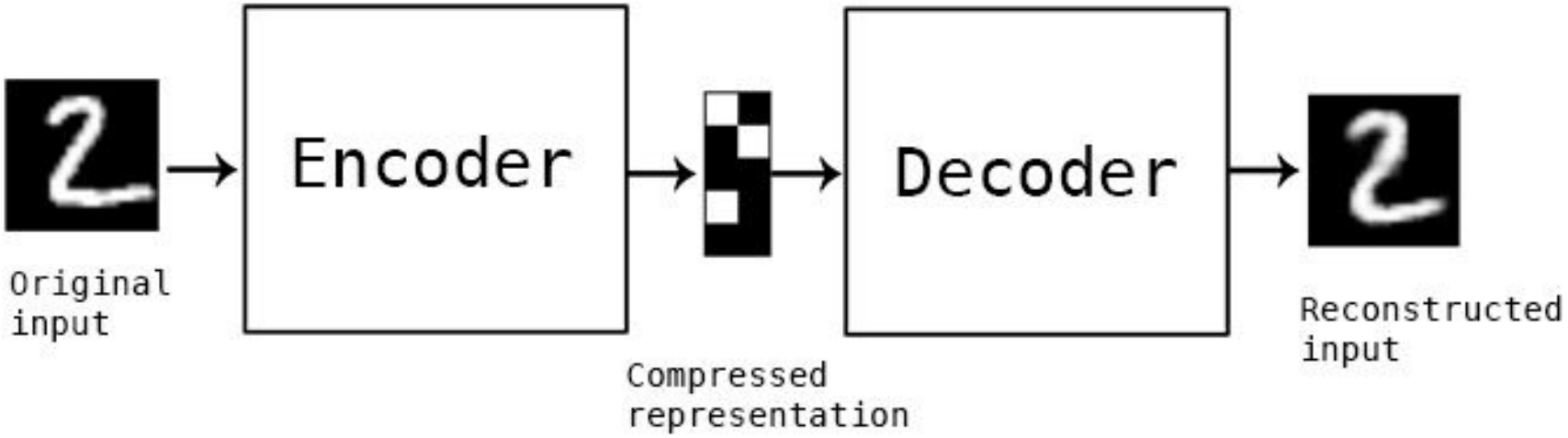
Tiago Maritan

(tiago@ci.ufpb.br)

Autoencoders

Autoencoders

- RNAs capazes de aprender representações densas dos dados, chamadas de **representações latentes** ou **codificações**
  - De forma **não supervisionada**



Autoencoders

- Basicamente aprendem a copiar as suas entradas para as saídas

- A princípio parece fácil, mas a ideia é aplicar restrições para evitar cópia das entradas de forma trivial.
  - Limitar o tamanho das **representações latentes** (dimensionalidade menor que os dados de entrada)
  - Adicionar ruído as entradas e treinar a rede para recuperar as entradas originais

- **Representações latentes** são derivadas do autoencoder que aprendem a função de identidade com algumas restrições

Autoencoders - Aplicações

- Autoencoders podem ser aplicados para:
  - **Redução de dimensionalidade**
  - **Extração de características**
  - **Pré-treinamento não supervisionado**
  - **Modelos generativos**
    - Gerar novos dados que se parecem com os dados de treinamento

Representação de Dados Eficiente

- Qual sequência é mais fácil de memorizar?
  - 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
  - 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

Representação de Dados Eficiente

- Qual sequência é mais fácil de memorizar?
  - 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
  - 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

Sequência é menor

Representação de Dados Eficiente

- Qual sequência é mais fácil de memorizar?
  - 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
  - 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

Sequência menor

Sequência maior, mas com um padrão!

Mais fácil de memorizar!!!

É uma lista de números pares de 50 a 14

Basta lembrar o padrão (números pares decrescentes) e os números inicial e final (50 e 14)

Representação de Dados Eficiente

- Se fossemos bons em memorizar sequências longas de forma fácil e rápida, não nos importaríamos com a existência de um padrão na 2a sequência

- O fato de ser difícil reconhecer sequências longas é o que o torna útil para o reconhecimento de padrões

Representação de Dados Eficiente

- William Chase e Herbert Simon estudaram a **relação entre memória, percepção e correspondência de padrões** em 1973[1]

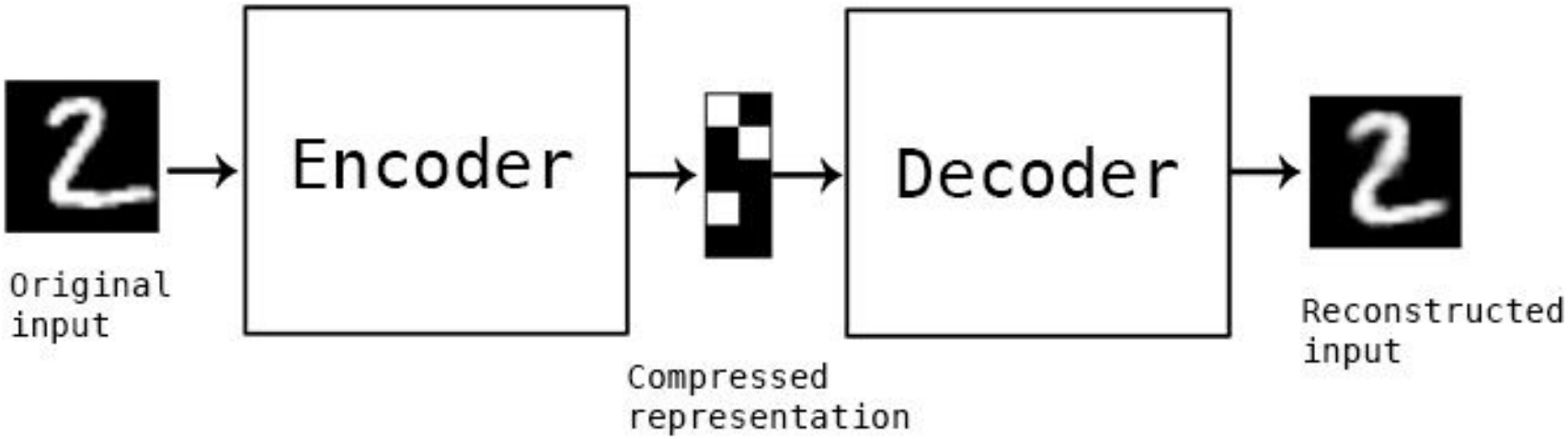
- Observaram que:
  - Jogadores de xadrez experientes conseguiam memorizar a posição de todas as peças olhando para o tabuleiro por 5 segundos;
    - Impossível para a maioria das pessoas;
  - Contudo, isso só acontecia quando as peças eram colocadas em posições realistas (posição de jogo), não quando as peças eram colocadas aleatoriamente
  - Eles não tem memória tão melhor. Apenas enxergam os padrões do xadrez com mais facilidade, graças à experiência com o jogo

[1] **William Chase e Herbert A. Simon, "Perception in Chess", Cognitive Psychology, 4, 55-81, 1973**

Representação de Dados Eficiente

- Portanto, restringir um autoencoder é o que o leva a descobrir e explorar padrões nos dados

- Ele observa as entradas e as converte numa representação latente eficiente

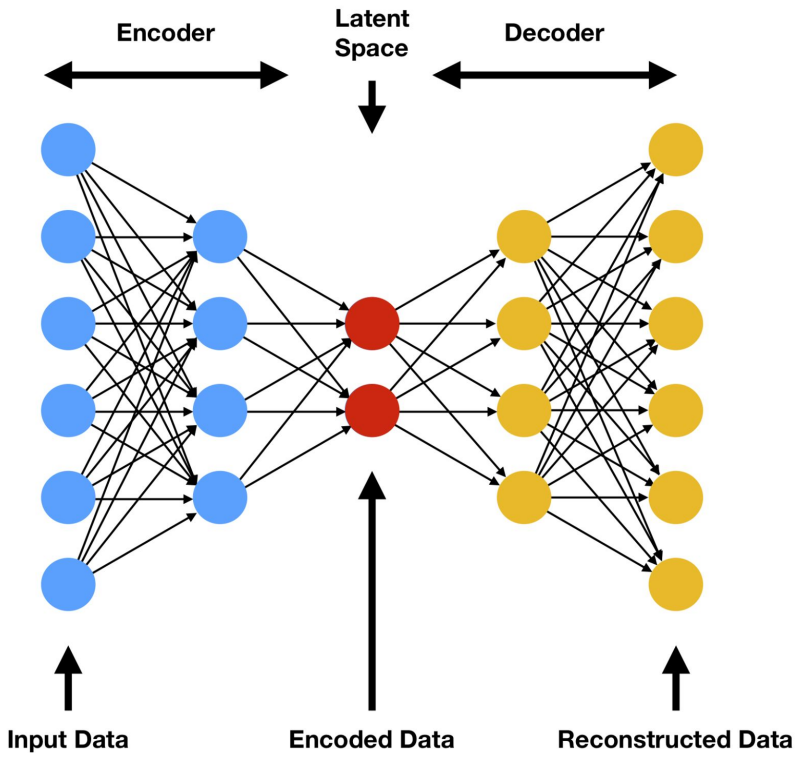


Autoencoders como um MLP

- Um autoencoder pode ser implementado como um MLP, em que:

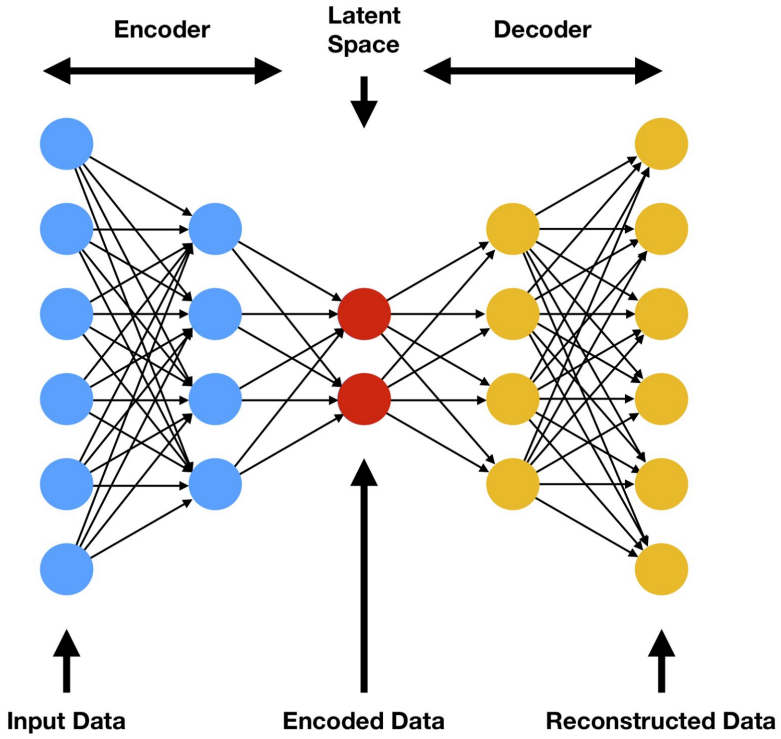
  <u>nº neurônios de saída é igual ao ao nº de entradas</u>

- Quando a representação interna é menor que os dados de entrada, elé considerado um **<u>autoencoder subcompleto</u>**



PCA como um Autoencoder Linear Subcompleto

- Um autoencoder que usa funções de ativação lineares e a função de custo é o erro médio quadrático, funciona como um PCA (Análise de Componentes Principais).



PCA como um Autoencoder Linear Subcompleto

- Implementação: (função de ativação = linear, Função custo = MSE)

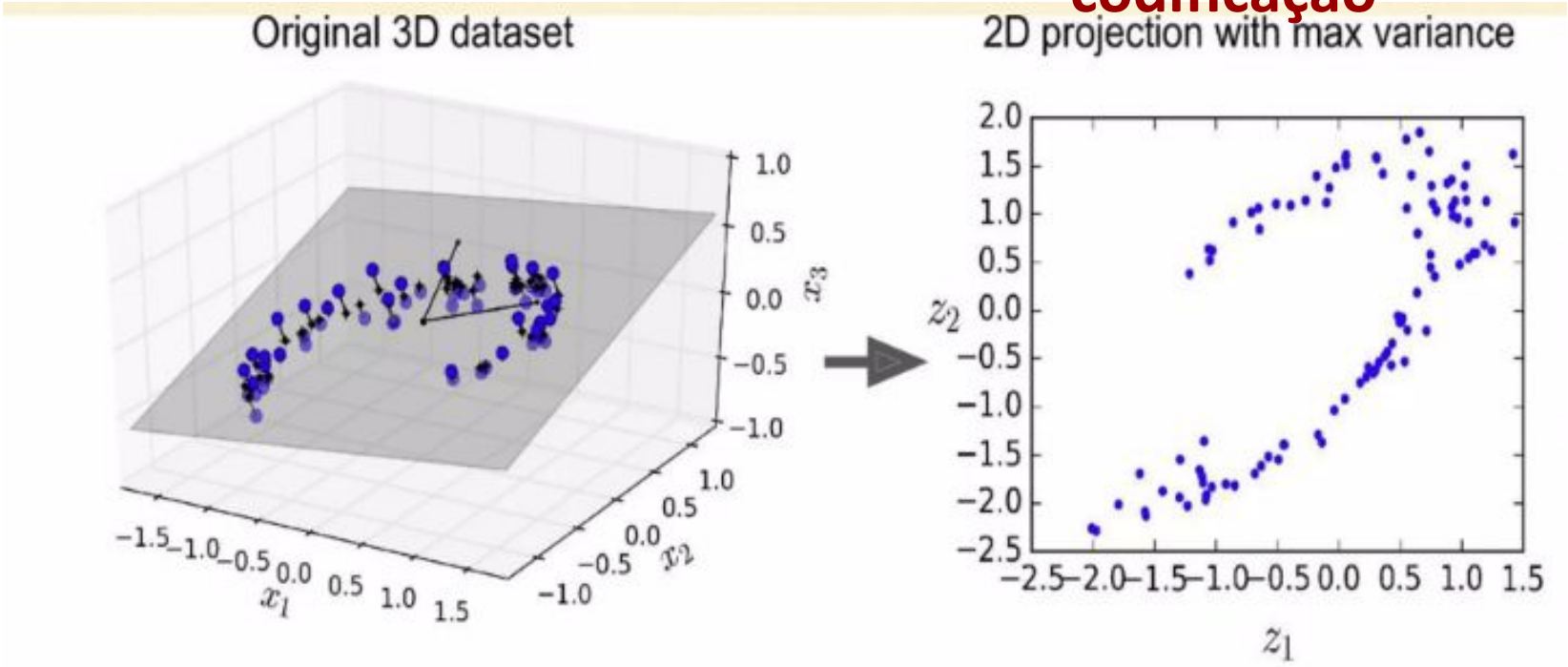
```
encoder = keras.models.Sequential(
            [keras.layers.Dense(2, input_shape=[3])])

decoder = keras.models.Sequential(
            [keras.layers.Dense(3, input_shape=[2])])

autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=
                keras.optimizers.SGD(learning_rate=1.5))
```

PCA como um Autoencoder Linear Subcompleto

- Exemplo: Autoencoder linear simples realizando o PCA em um conjunto de dados 3D, projetando-o em 2D

Dados de entrada

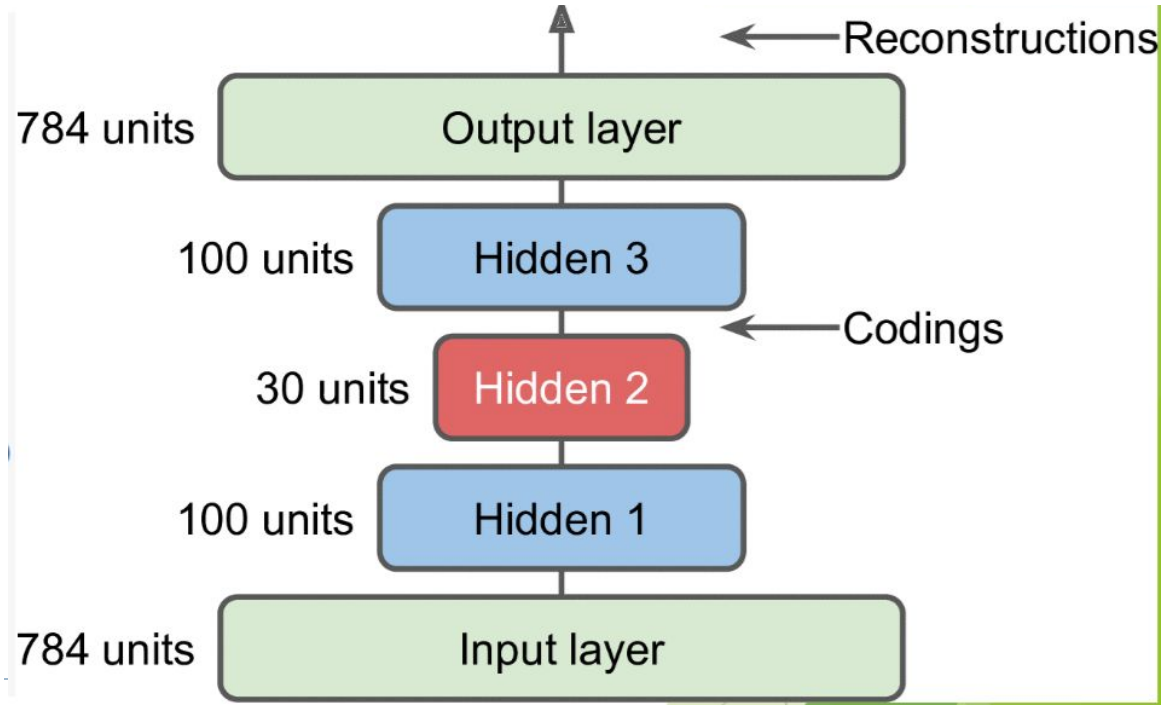
Saída da camada de codificação



Deep Autoencoders (Autoencoders Empilhados)

- Adicionar mais camadas a um autoencoder o ajuda a compreender codificações (representações latentes) mais complexas.

- A arquitetura de um Deep Autoencoder é geralmente simétrica com relação a camada central oculta (ou a camada de codificação)



Deep Autoencoders - Implementação

```
stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])

stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu",
                       input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])

stacked_ae = keras.models.Sequential([stacked_encoder,
                                      stacked_decoder])
stacked_ae.compile(loss = "binary_crossentropy",
           optimizer=keras.optimizers.SGD(learning_rate=1.5)
history = stacked_ae.fit(X_train, X_train, epochs=20,
                         validation_data=(X_valid, X_valid))
```

Deep Autoencoders (Autoencoders Empilhados)

- Uma forma de verificar se um autoencoder está devidamente treinamento é comparar entradas e as saídas

- Ex: Reconstruções com o Fashion MNIST para um autoencoder MLP com uma arquitetura 784 (entrada): 100: 30: 100: 784 (saída)
  - Reconstruções visíveis mas com perdas

Entradas

Saídas

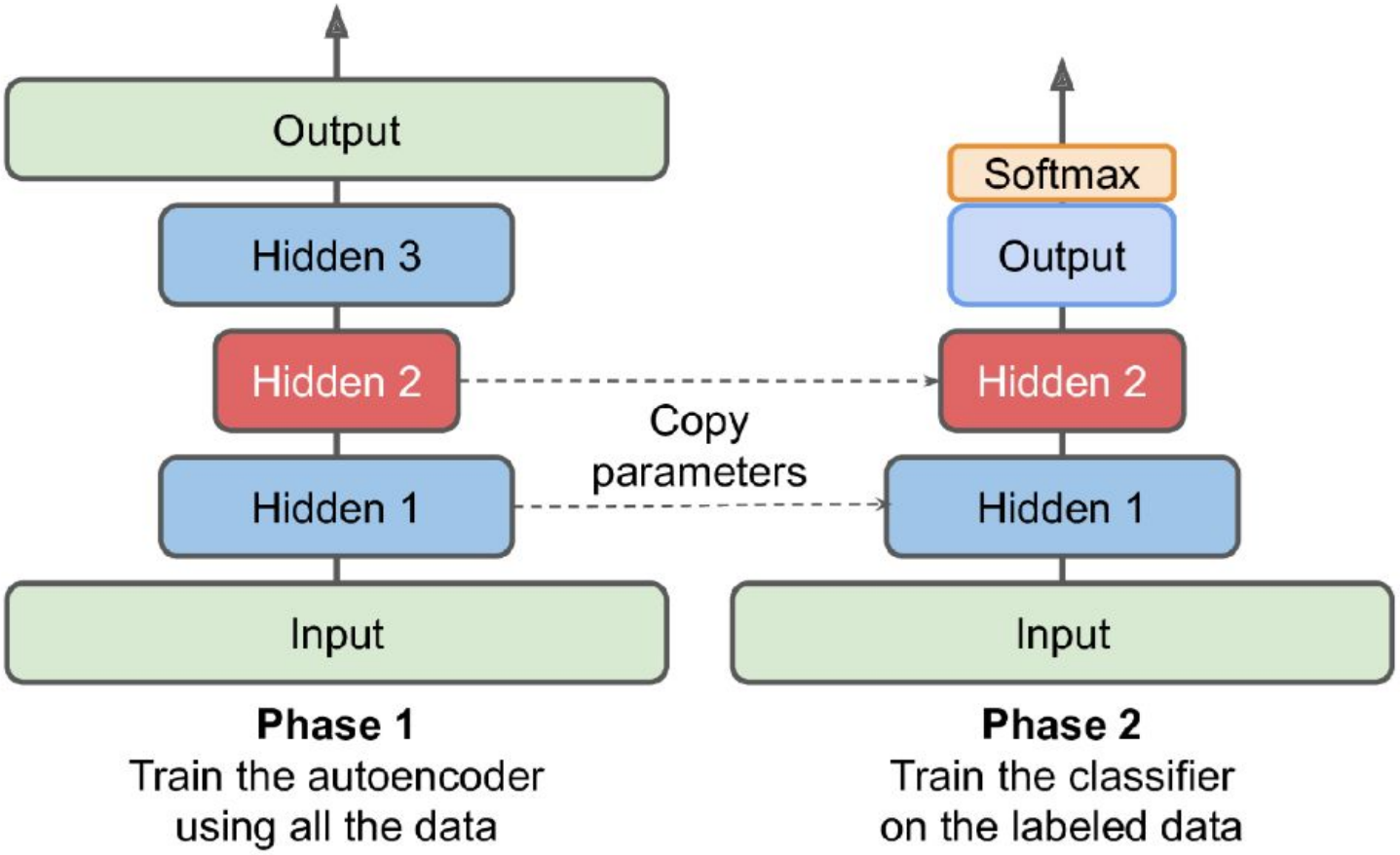
Amarrar os pesos

- Quando o autoencoder é ordenadamente simétrico, podemos **"amarrar" os pesos das camadas do decodificador nos pesos das camadas do codificador**

$$\mathbf{W}_{N-L+1} = \mathbf{W}_{L}^{T} \text{ (com } L = 1,2\ldots, N/2)$$

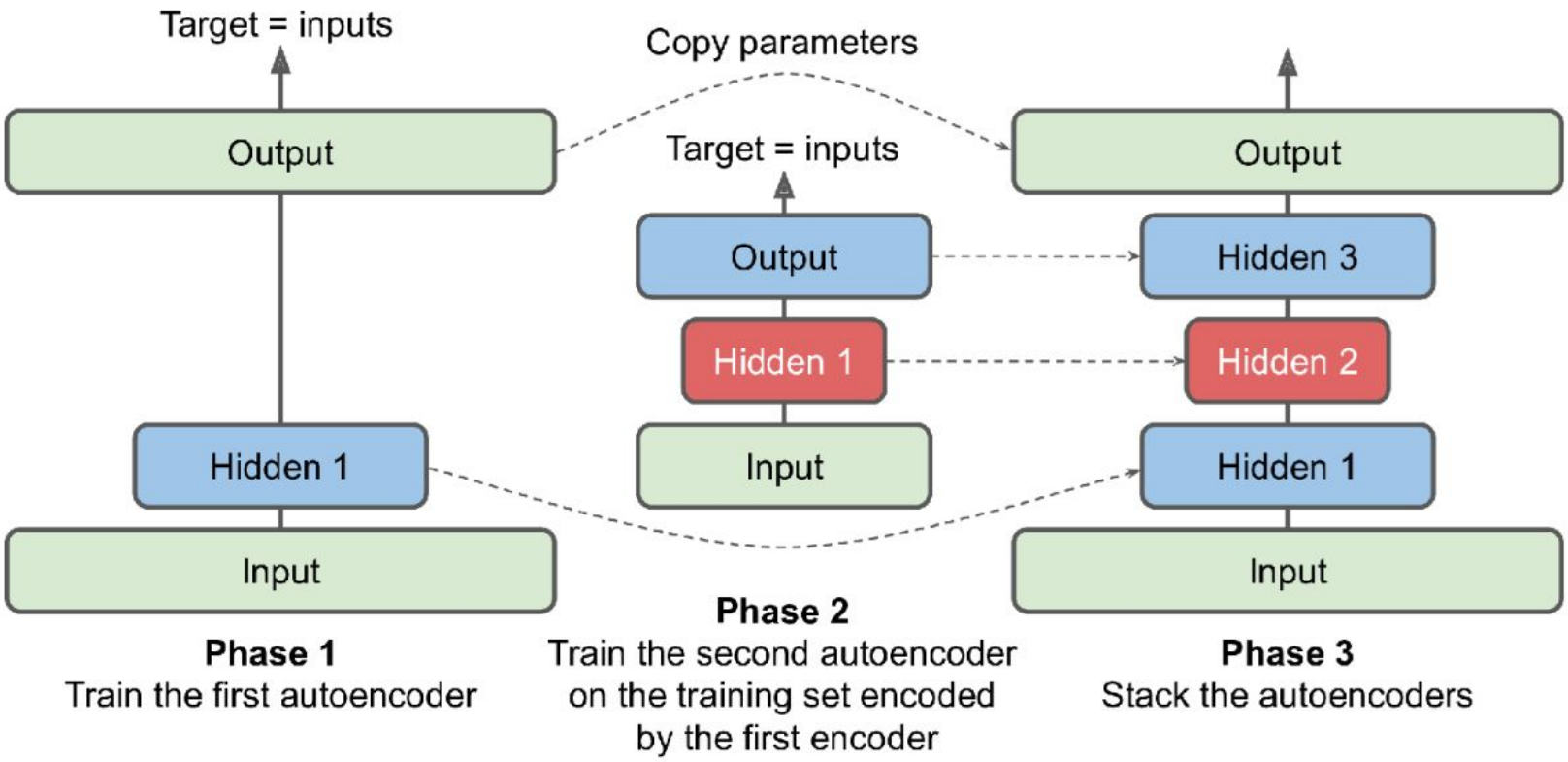
- Reduz pela metade o número de pesos do modelo, acelera o treinamento e reduz o risco de overfitting

Pré-treinamento não supervisionado com Autoencoders Empilhados



Treinando um Autoencoder de cada vez

- **Treinamento Guloso em Camadas**
  - Em vez de treinar o autoencoder todo de uma vez, podemos treinar um autoencoder pouco profundo de cada vez, e em seguida empilhá-los em um único autoencoder empilhado



Treinando um Autoencoder de cada vez

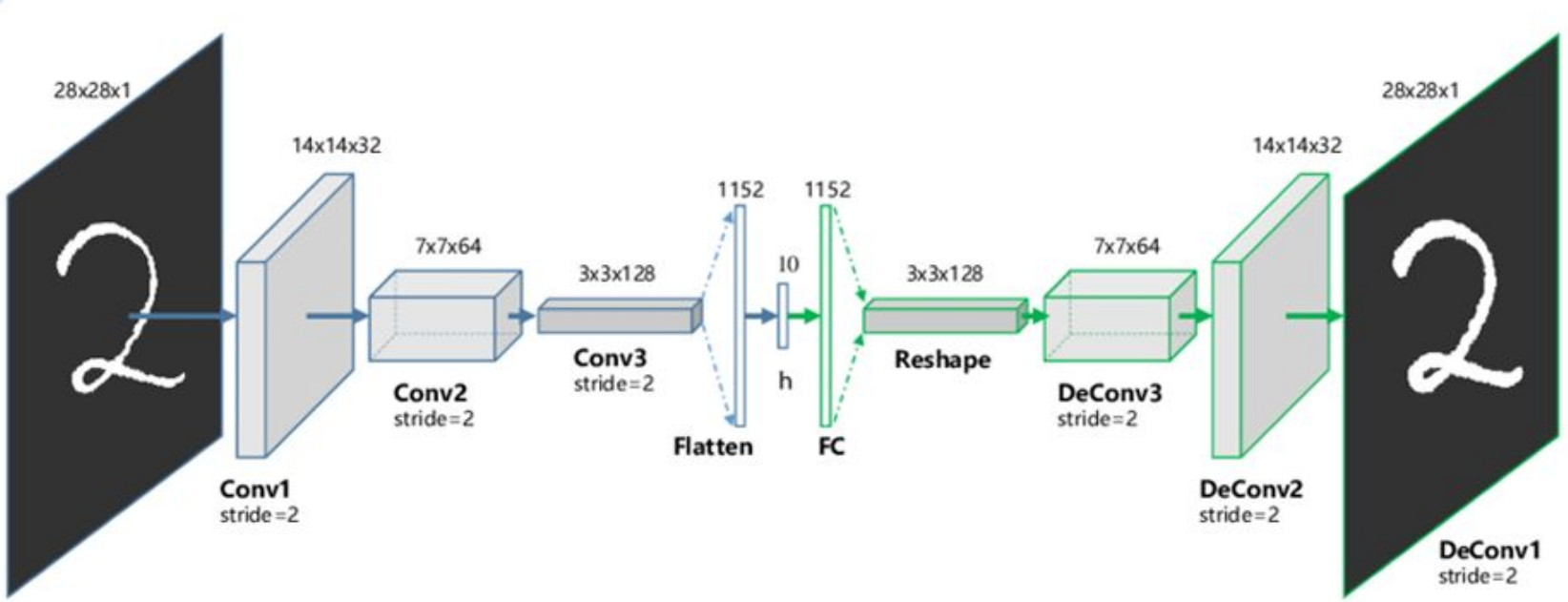
- **Treinamento Guloso em Camadas**
  - Em 2006, Geofrey Hinton[1] descobriu que redes profundas podem ser treinadas de uma forma não supervisionada com o treinamento guloso (ganancioso) em camadas;
    - Fez isso com RMBs (Máquinas Restritas de Boltzmann)
  - Um dos desencadeadores do tsunami do Aprendizado Profundo
  - Em 2007, Yoshua Bengio[2] aplicaram isso com autoencoders
  - Durante muitos anos foi a única maneira eficiente de treinar redes profundas

[1] G. Hinton et al. (2006), A Fast Learning Algorithm for Deep Belief Nets, Neural Computation 18, 1527–1554.

[2] Y. Bengio et al. (2007), Greedy Layer-Wise Training of Deep Networks, Anais da Conferência Internacional sobre Sistemas de Processamento de Informação Neural, 153-160

Autoencoders Convolucionais

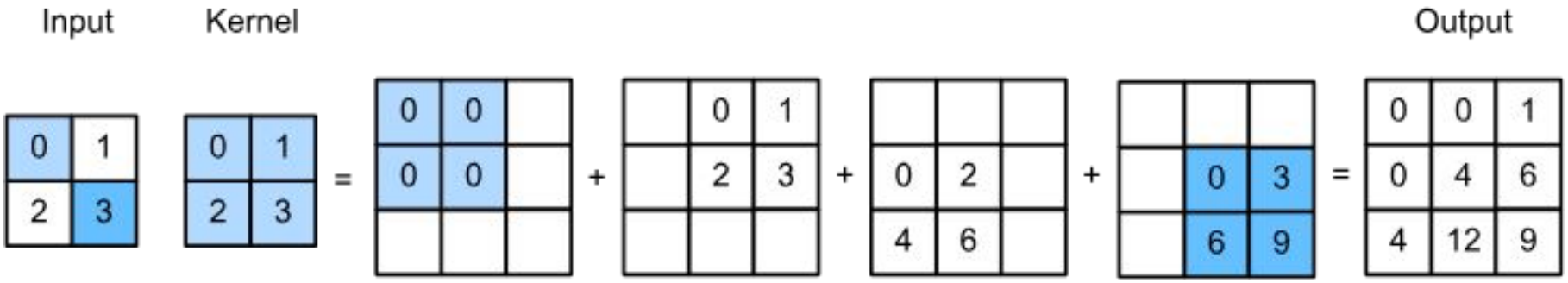
- Codificador é uma CNN normal: camadas conv + pooling
  - Reduz a dimensionalidade espacial das entradas e aumenta a profundidade (número de mapas de características)
- Decodificador: camadas de conv transposta (ou “deconv”)
  - Aumenta a imagem e reduz a profundidade



Convolução Transposta (ou Deconvolução)

- A Convolução Transposta (ou Deconvolução) aumenta a largura e altura de entrada, tentando prever valores para cada pixel
  - Deconvolução não é um termo muito adequado, convolução transposta representa melhor a operação implementada

- Exemplo:

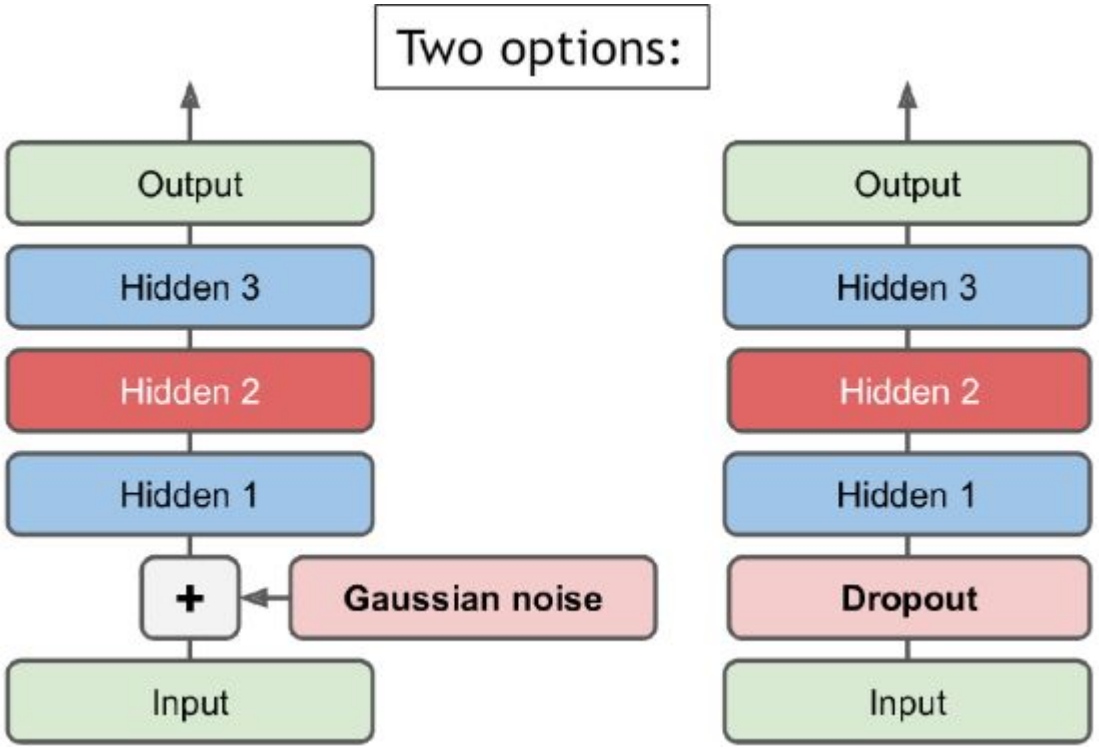


Autoencoders Recorrentes

- Usado para sequências temporais ou textos
- Codificador é uma RNN de sequência para vetor
  - Compacta a sequência em um único vetor
- Decodificador é uma RNN de vetor para sequência

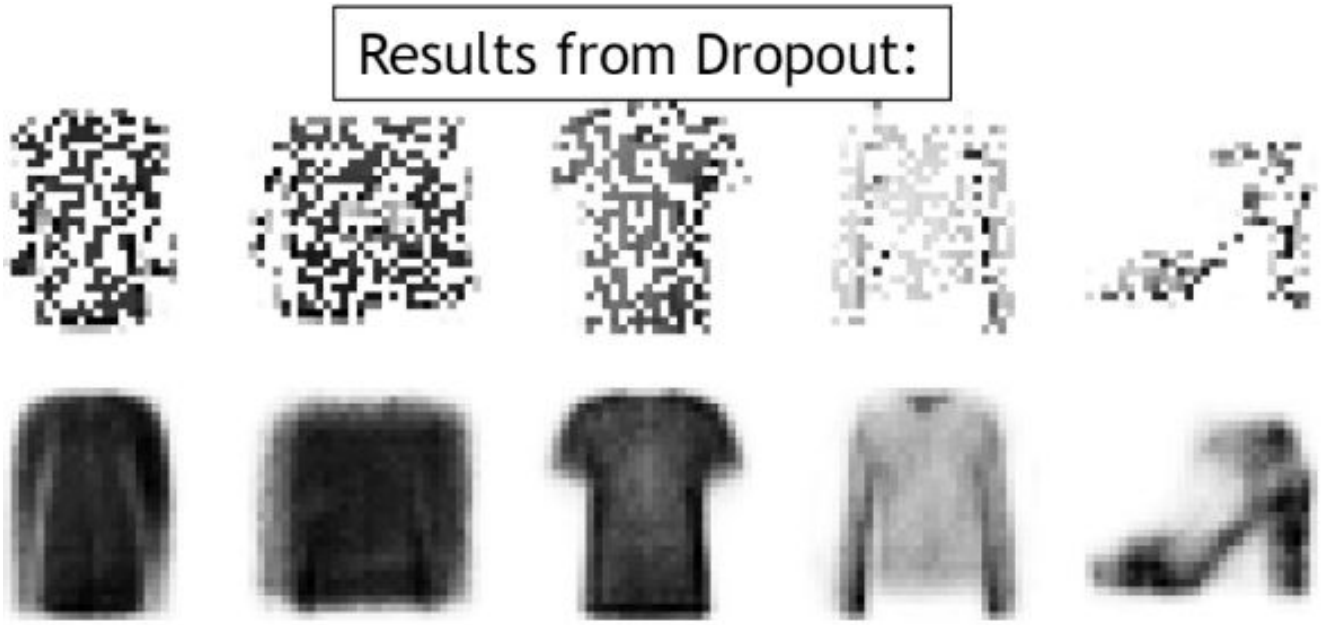
Denoising Autoencoders (Autoencoders que eliminam ruído)

- Um autoencoder pode aprender **features úteis** adicionando ruído a entrada e treinando-o para recuperar as entradas sem ruído.



Denoising Autoencoders (Autoencoders que eliminam ruído)

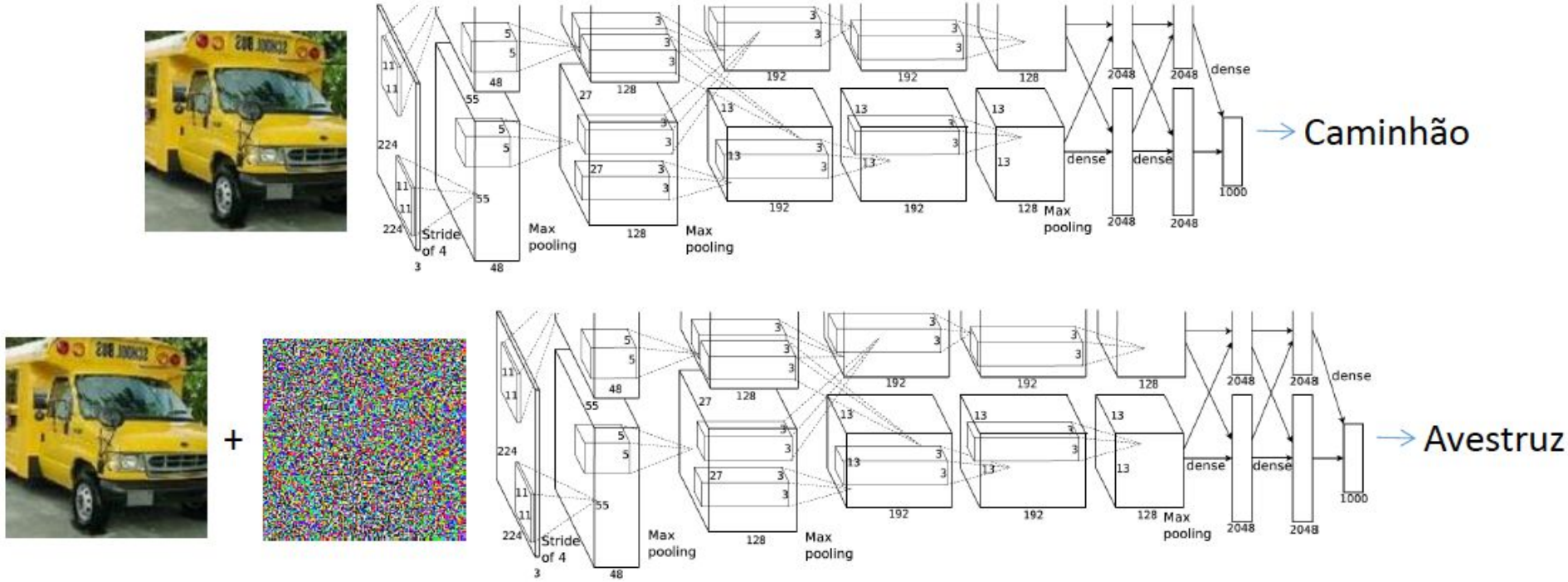
- Exemplo: Resultado com aplicação de Dropout
  - Imagens com ruído com metade dos pixels desativados vs imagens reconstruídas



Redes Generativas Adversárias (GANS)

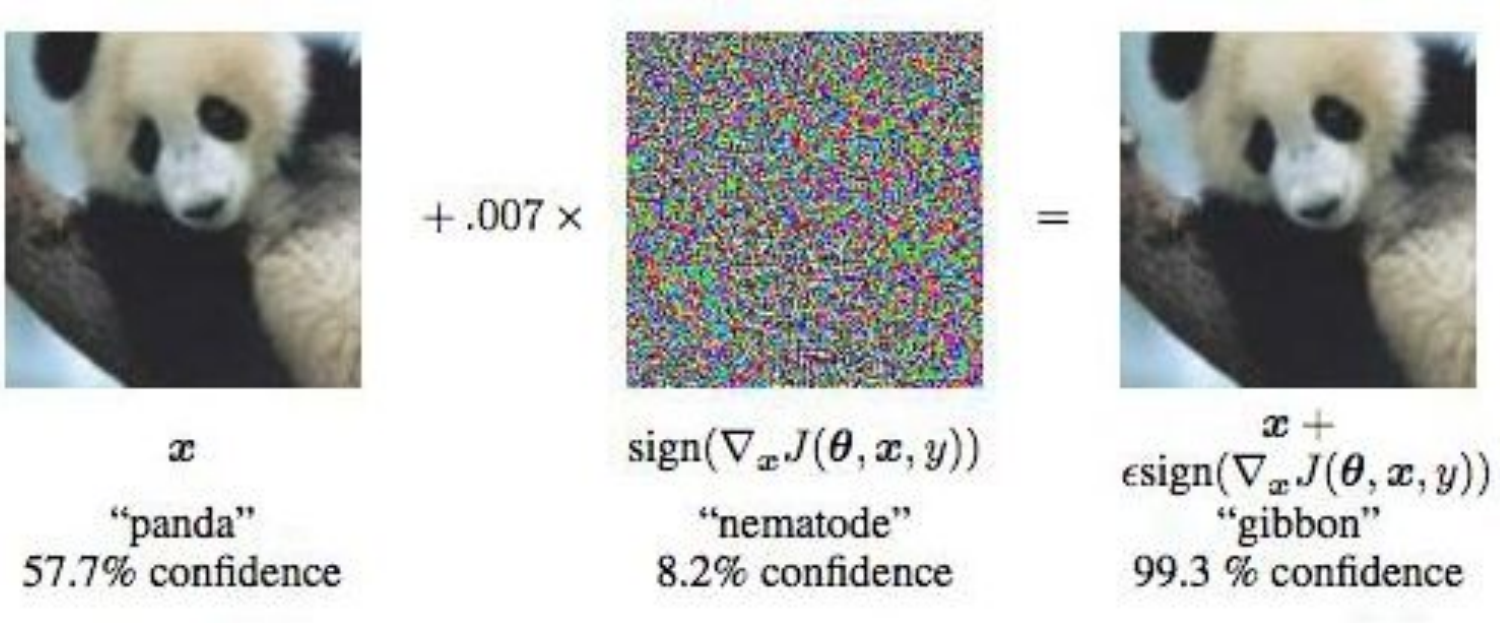
Redes Adversárias (GANs)

- Ajusta a imagem de entrada para **enganar a rede**.

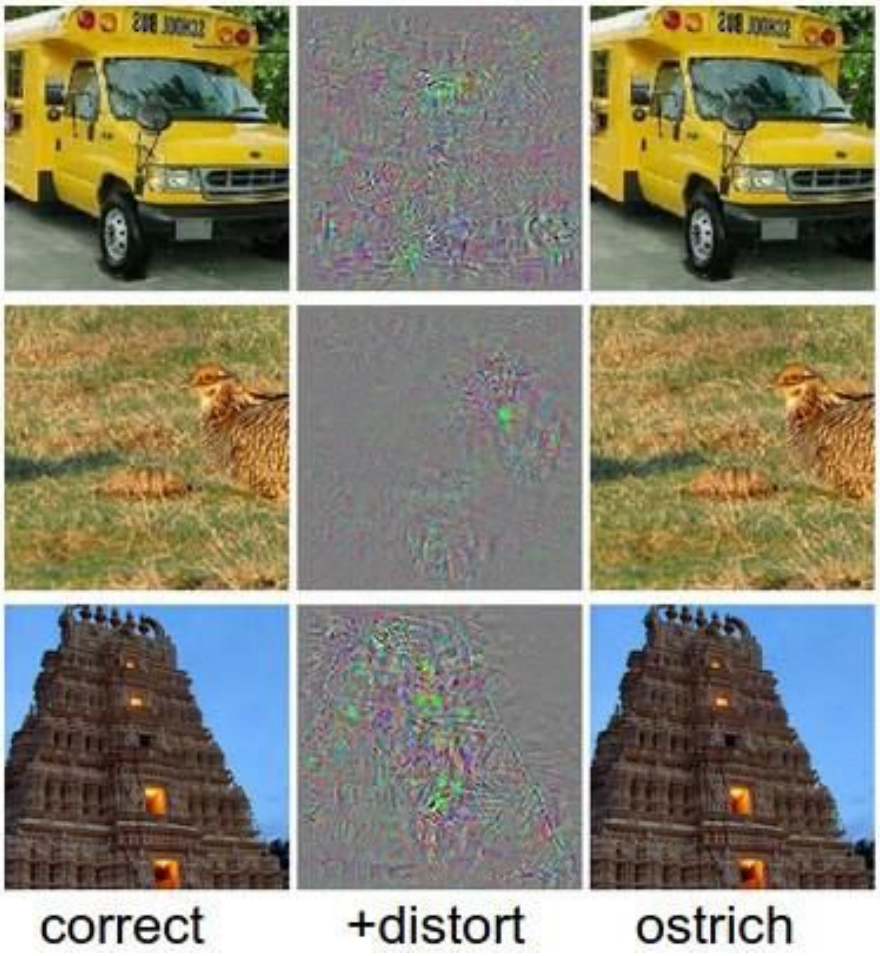


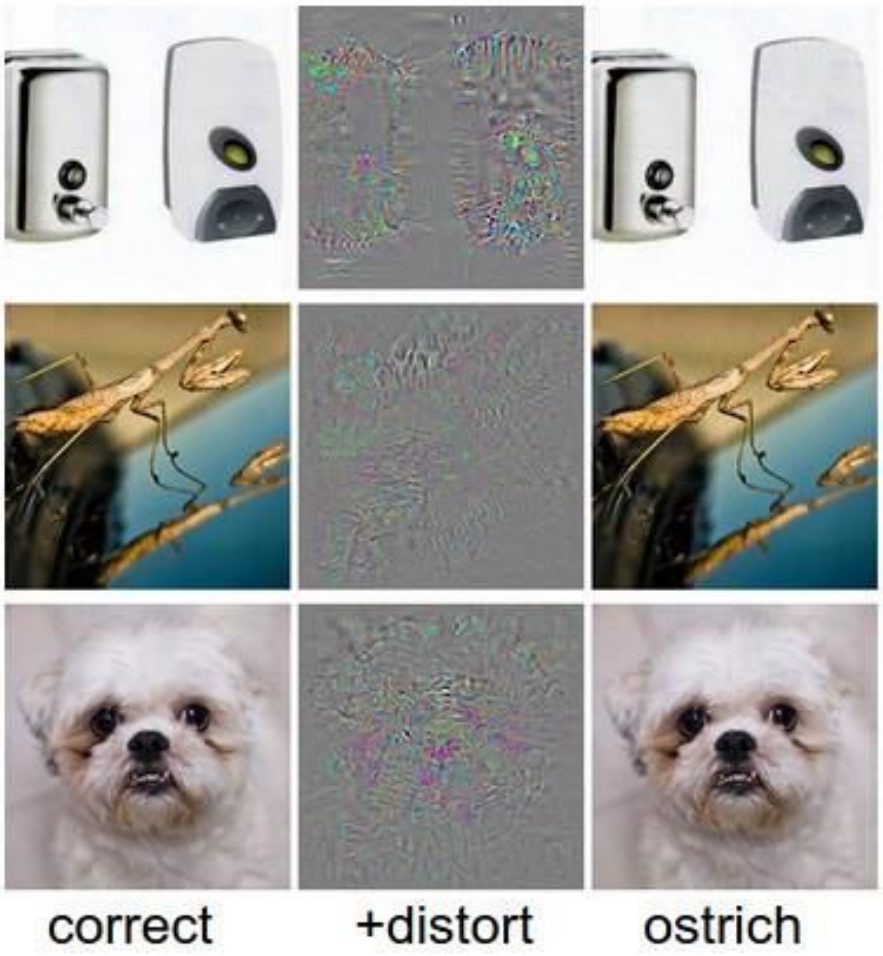
Redes Adversárias (GANs)

- Ajustar (por backpropagation) os pixels da imagem de entrada, até obter a saída desejada, com a confiança que se deseja

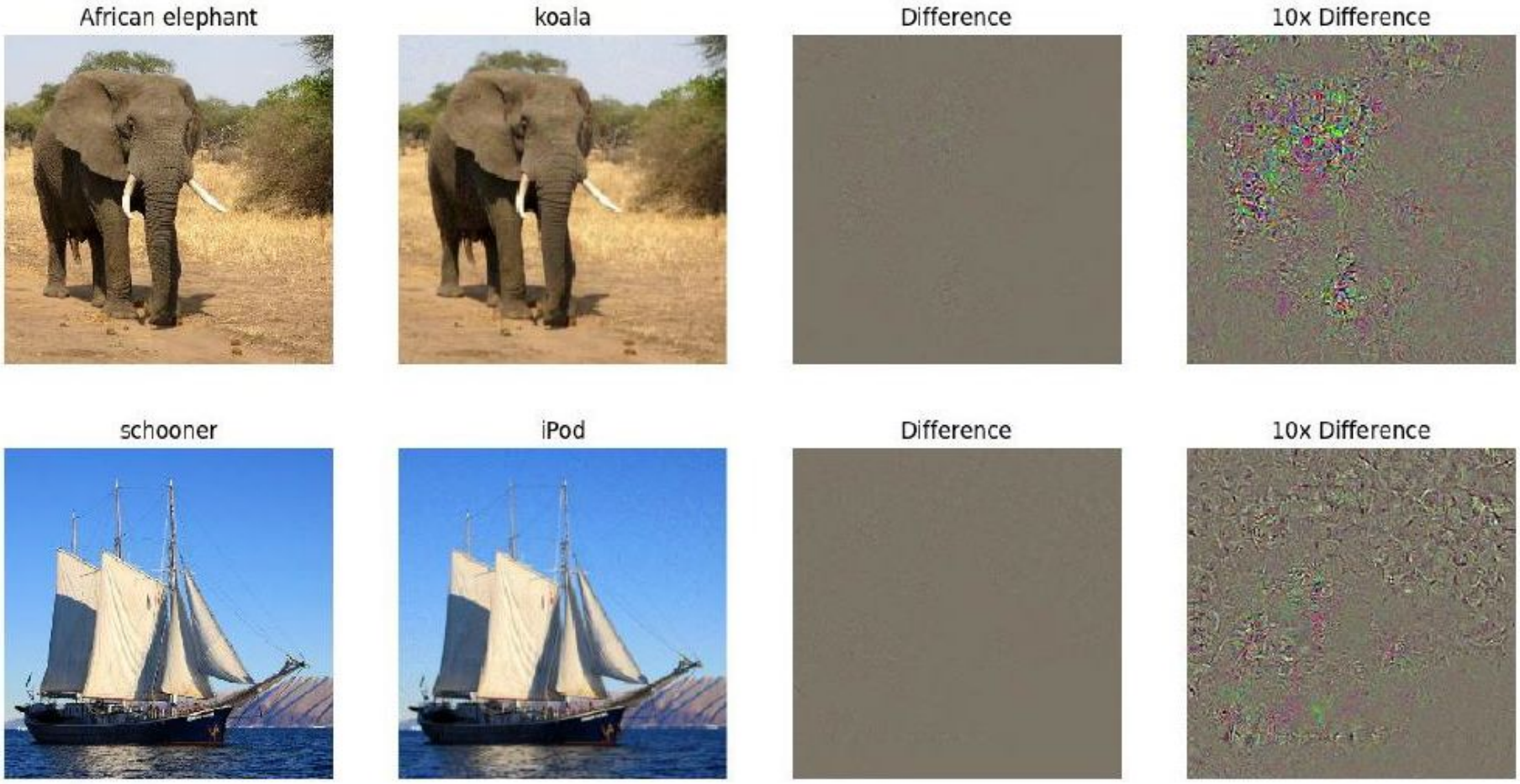


Redes Adversárias (GANs)



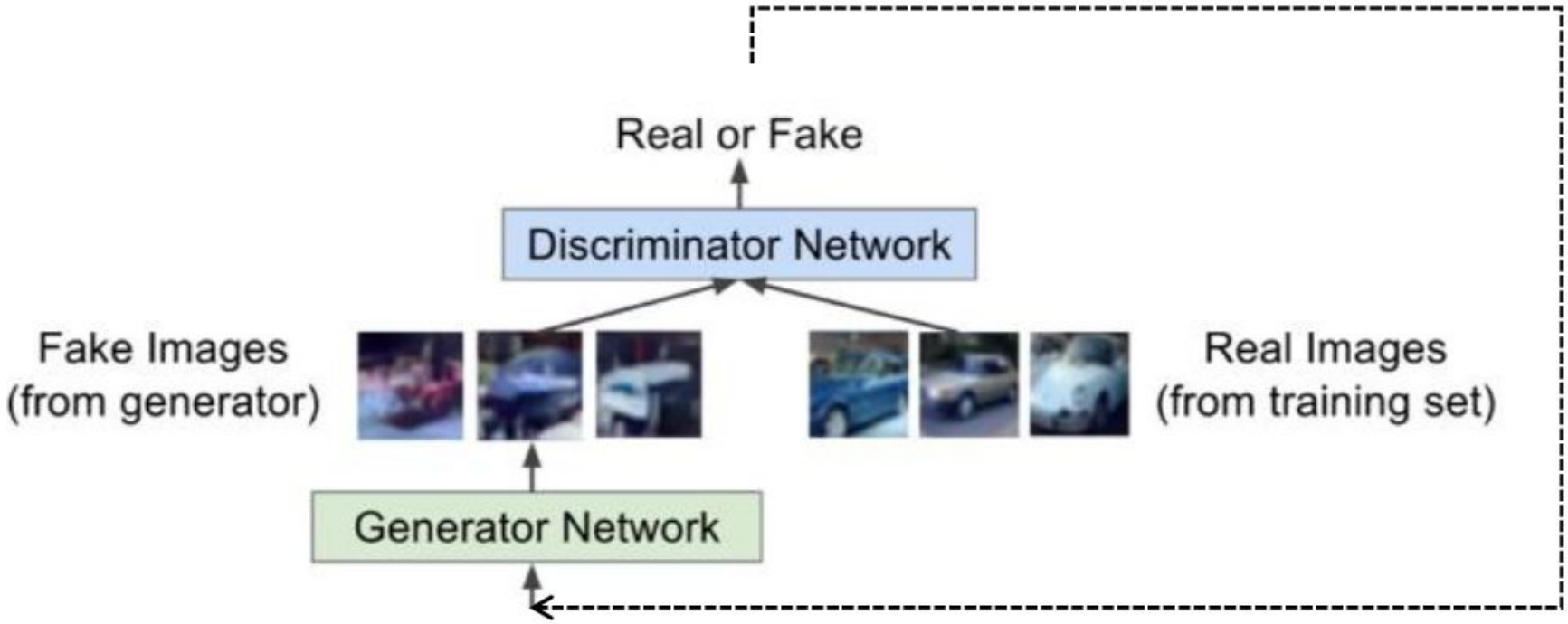


Redes Adversárias (GANs)



Redes Adversárias Generativas (GANs)

- Solução: Treinar rede com exemplos adversários
- Jogos de dois jogadores



Redes Adversárias Generativas (GANs)

- Treinamento adversário é considerado uma das ideias mais importantes dos últimos anos

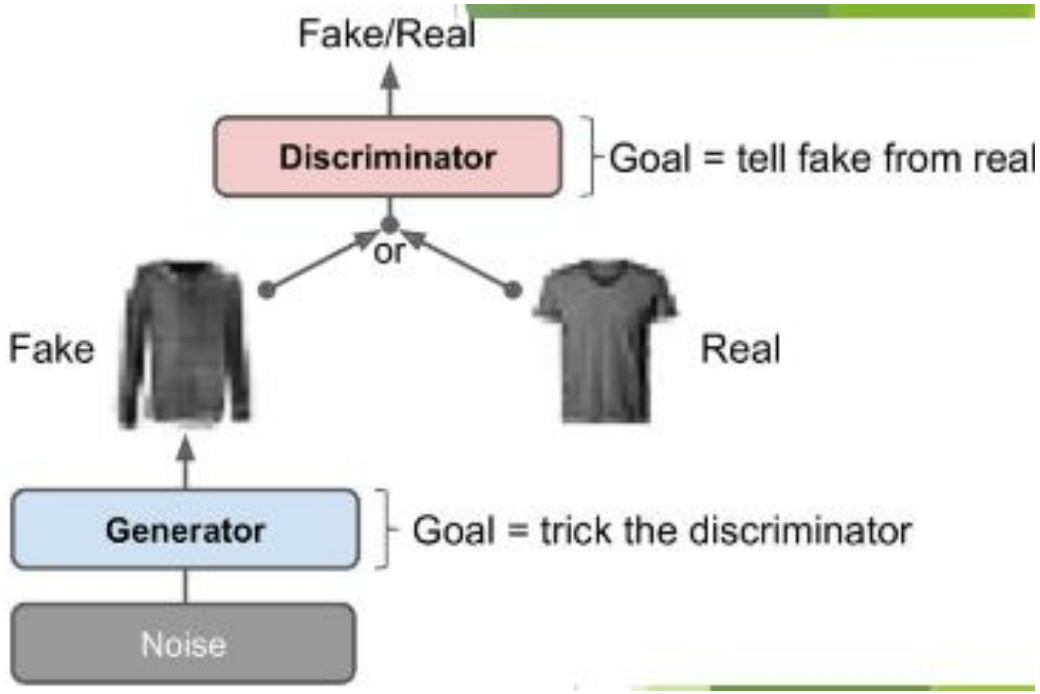
- Yann Lecun chegou a dizer que era **a ideia mais interessante dos últimos 10 anos em aprendizagem de máquina**

- Ex: Geração artificial de imagens (rostos, quartos)
  StyleGAN: https://thispersondoesnotexist.com
  StyleGAN: https://thisrentaldoesnotexist.com

GANs - Aplicações

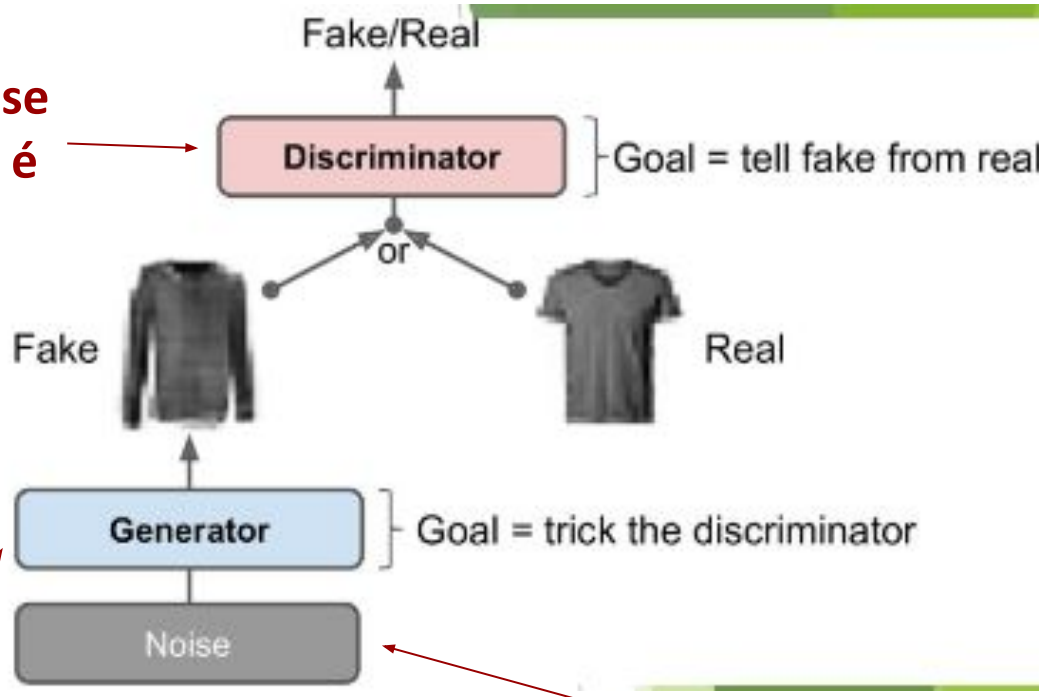
- Geração de imagens artificiais
- Super-resolução (aumento de resolução de imagens)
  - https://github.com/AnjanaGJoseph/Super-Resolution-GAN
- Colorização
  - https://github.com/jantic/DeOldify
- Edição de imagens expressiva
  - Ex: Remoção de photobombing (intrometido)
- Transformação de esboços em imagens realísticas
- Predição de próximos quadros de um vídeo
- Data augmentation
- Identificação de pontos em modelos

GANs - Como funcionam



GANs - Como funcionam

Tenta discriminar se a imagem gerada é real ou falsa



Obtém uma distribuição aleatória como entrada (ex: gaussiana), e gera alguns dados (ex: imagens)

Entradas aleatórias funcionam como representação latente da imagem a ser gerada

GANs - Treinamento

- Cada iteração de treinamento é dividida em 2 fases:
  1. <u>Treinamento da rede discriminativa:</u>
     - Amostra-se um batch contendo: imagens do conjunto de treinamento + imagens falsas geradas pela rede generativa.
     - Rótulos:
       - 0 - imagens falsas;
       - 1 - imagens reais
     - Treina-se apenas a rede discriminativa com esse batch
     - *Retropropagação otimiza somente pesos da rede discriminativa*

GANs - Treinamento

- Cada iteração de treinamento é dividida em 2 fases:
  2. Treinamento da rede generativa:
     - Gera-se um outro batch de imagens falsas e usa a rede discriminativa para dizer se as imagens são reais ou falsas
     - Não são adicionais imagens reais no batch, e todos os rótulos são definidos como 1 (real)
       - O alvo é enganar a rede discriminativa
       - Queremos que a rede gerativa crie imagens que a rede discriminativa acreditem ser reais
     - Retropropagação otimiza somente os pesos da rede generativa
     - Pesos da rede discriminativa ficam congelados nesta etapa;

GANs - Treinamento

- Rede Generativa nunca vê imagens reais, mas aos poucos aprende a produzir imagens falsas convincentes!

- Como isso acontece?
  - Os gradientes voltam para a rede discriminativa!
  - Quanto melhor fica a rede discriminativa, mais informações sobre as imagens reais ficam nesses gradientes já usados
  - Permite-se o progresso da rede generativa

GANs - Implementação

- Rede Generativa é semelhante ao Decoder de um Autoencoder. Ex:

```
np.random.seed(42)
tf.random.set_seed(42)
codings_size = 30

generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu",
                       input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

GANs - Implementação

- Rede Discriminativa é um classificador binário regular. Ex:

```
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])

gan = keras.models.Sequential([generator,
                               discriminator])
```

GANs - Implementação

- Compilando os modelos
  - Rede generativa por meio do modelo `gan`, então não precisamos compilá-lo

```
discriminator.compile(loss="binary_crossentropy",
                      optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy",
            optimizer="rmsprop")
```

GANs - Implementação

- Compilando os modelos
  - Rede generativa por meio do modelo `gan`, então não precisamos compilá-lo

```
discriminator.compile(loss="binary_crossentropy",
                      optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy",
            optimizer="rmsprop")
```

Não deve ser treinada na 2a fase do treinamento

São classificadores binários, então podemos usar a perda de entropia cruzada binária

GANs - Implementação

- Treinamento
  - Como o loop de treinamento é incomum, não podemos usar o método `fit()`.

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size,
                                            codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch],
                                        axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            ...
```

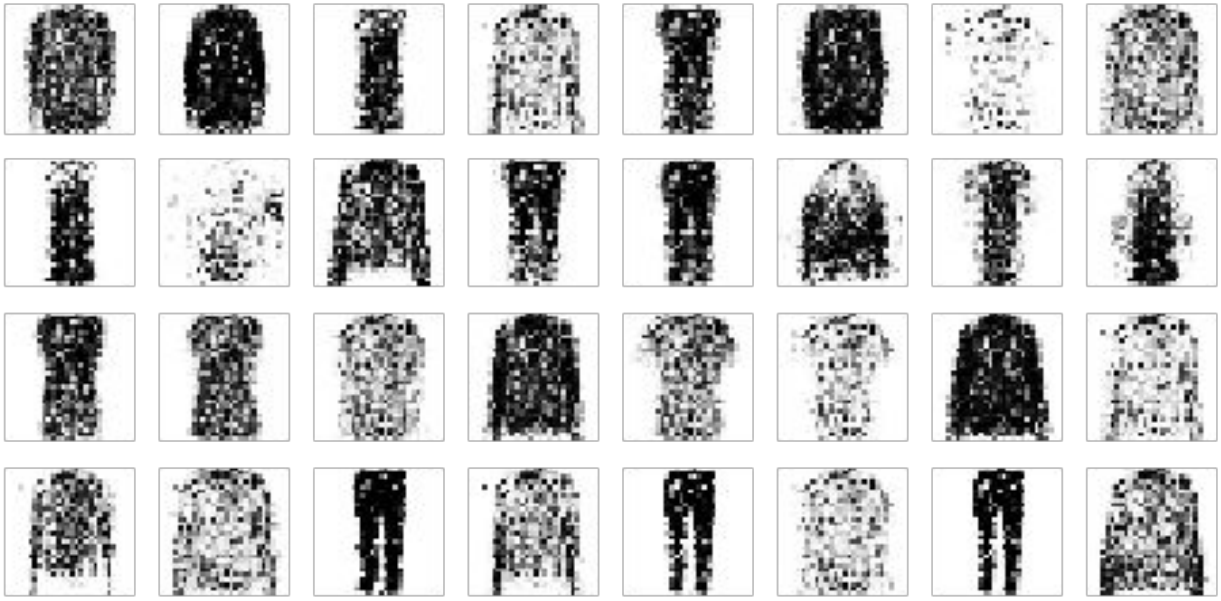
GANs - Implementação

- Treinamento

```
            ...
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size,
                                            codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size, n_epochs=1)
```

GANs - Implementação

- Ex: Imagens geradas pela GAN após uma época de treinamento



Desafios de Treinar as GANs

- **Modo Colapso**
  - **Maior dificuldade!!!**
  - Quando as saídas da rede generativa se tornam gradualmente menos diversificadas!

  - Ex: Rede generativa ficou melhor na produção convincente de sapatos do que outras classes
    - Engana a rede discriminativa
    - Incentiva a rede generativa a produzir ainda mais imagens de sapatos
    - Rede discriminativa só verá imagens falsas de sapatos e esquecerá como discriminar imagens falsas de outras classes

Desafios de Treinar as GANs

- **Oscilação e Parâmetros instáveis**
  - Competição entre as redes pode fazer com que parâmetros oscilem e tornem-se instáveis
  - Treinamento pode começar de forma adequada e divergir sem motivo aparente!
  - GANs são, portanto, muito sensíveis aos hiperparâmetros!!!
  - Necessário gastar um bom tempo ajustando-os!

Desafios de Treinar as GANs

- Esses problemas têm desafiado os pesquisadores desde 2014

- Muitos artigos publicados sobre o tema, envolvendo:
  - Propostas de novas funções de custo
  - Técnicas para estabilizar o treinamento
  - Técnicas para evitar o modo colapso

- Campo de pesquisa muito vivo!!!
  - Dinâmica das GANs ainda não é totalmente compreendida

GANs Convolucionais Profundas - DCGANs

- Inicialmente complicado ter GANs convolucionais profundas pois treinamento era muito instável!

- Em 2016, Alec. Radford et al.[1] testou muitas arquiteturas e hiperparâmetros diferentes e conseguiu sucesso na tarefa!
  - Arquitetura denominada de DCGANs

- Propôs um conjunto de diretrizes para construção de GANs convolucionais estáveis

[1] Alec Radford et al. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", arXiv:1511.06434, 2016

GANs Convolucionais Profundas - DCGANs

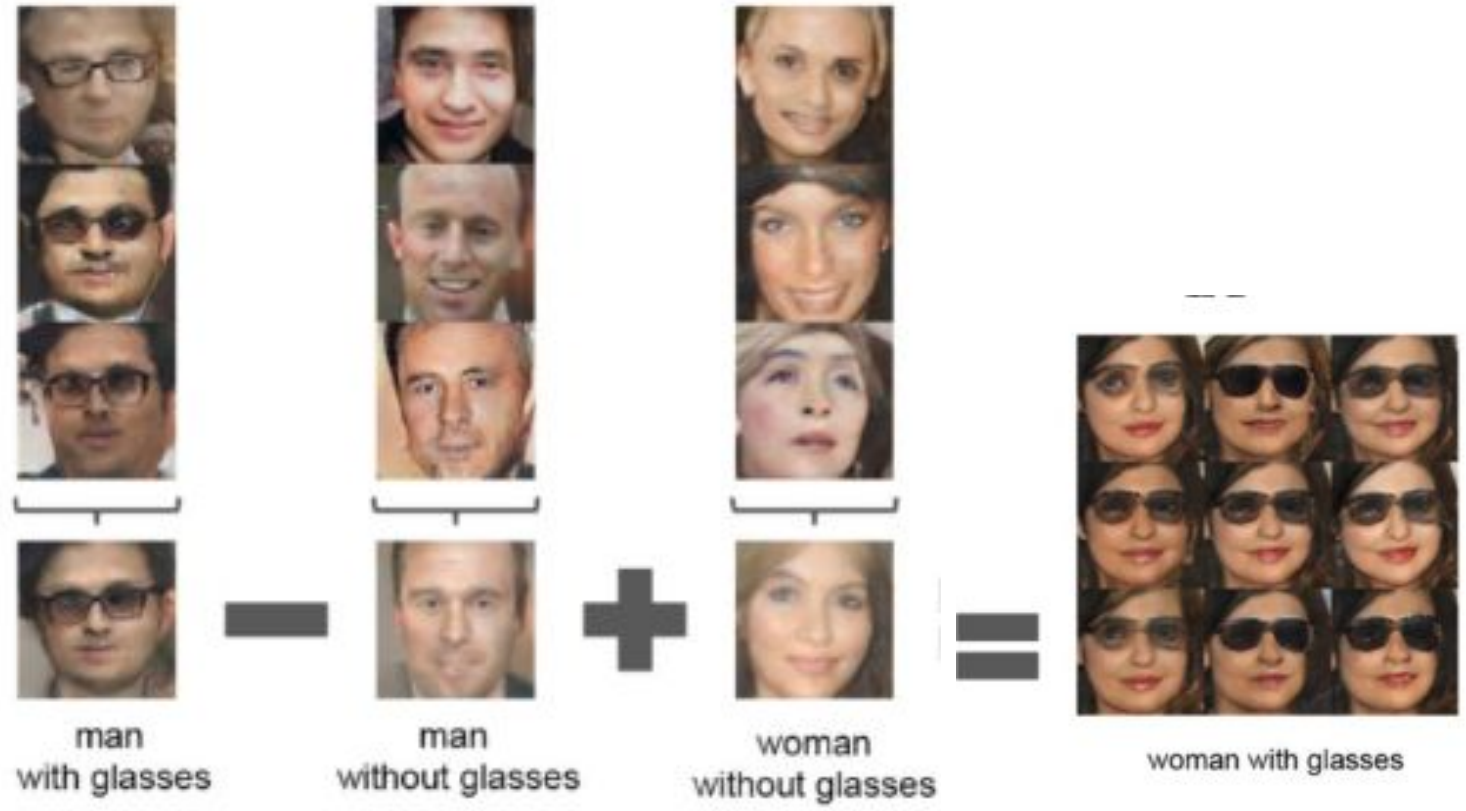
- Principais diretrizes:
  - Usar Convoluções com Stride e Convoluções Transpostas, e em vez de Pooling
  - Usar normalização em batch nas 2 redes
  - Remover Camadas Densas
  - Função de ativação:
    - Generativa: ReLU em todas as camadas, e tanh na saída
    - Discriminativa: Leaky ReLU

GANs Convolucionais Profundas - DCGANs

- Ex 2: Imagens geradas pela DCGAN após 50 épocas para o Fashion MNIST

GANs Convolucionais Profundas - DCGANs

- Ex 1: Imagens geradas pela DCGAN para rostos



GANs Convolucionais Profundas - DCGANs

- DCGANs possuem dificuldades com imagens grandes!

- Gera imagens com características convincentes localmente, mas que apresentam inconsistências gerais!
  - Ex: camisa com uma manga mais longa que a outra

GAN Condicional - CGAN[1]

- Permite controlar a classe de cada imagem produzida

- Como?
  - Adicionando a classe de cada imagem como uma entrada extra nas redes generativa e discriminativa
  - Com isso, redes aprendem a aparência de cada classe!

[1] Mehdi Mirza e Simon Osindero, "Conditional Generative Adversarial Nets", arXiv:1411.1784, 2014

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Autoencoders e GANs

Tiago Maritan

(tiago@ci.ufpb.br)