

MAE 5905: Introdução à Ciência de Dados

Pedro A. Morettin

Instituto de Matemática e Estatística
Universidade de São Paulo
pam@ime.usp.br
<http://www.ime.usp.br/~pam>

Aula 16 - Parte 3

20 de junho de 2023

Sumário

1 Redes neurais convolucionais - CNN

2 Redes generativas adversárias - GAN

CNN

- As redes neurais convolucionais (*convolutional neural networks*, CNN) foram introduzidas por LeCun et al. (1998) e são muito eficientes para resolver problemas de classificação de imagens.
- De modo geral, as CNN são desenhadas para processar dados que surgem na forma de matrizes (*arrays*), como sequências (séries temporais e linguagem) unidimensionais, sinais de audio bidimensionais e audio e imagens tridimensionais. Como exemplo, podemos ter uma imagem colorida composta de três *arrays* bidimensionais, contendo intensidades de pixel nos três canais de cores (RGB) (LeCun et al., 2015).
- Nas CNN, para extrair padrões (*features*) dos dados, há quatro ideias básicas:
 - uso de muitas camadas;
 - camadas de convolução (*convolution layers*, CL);
 - camadas de agrupamento (*pooling layers*, PL);
 - camadas totalmente conectadas.

CNN

- A CL é responsável por extrair os padrões locais (*feature maps*) da camada precedente. Esta é feita por meio de pesos (*filter banks*) de tamanhos reduzidos. Diferentes bancos de filtros são usados para os diversos padrões da imagem, como arestas, arranjos de arestas, partes de objetos familiares, cores etc. O resultado dessa soma ponderada local é passada por meio de uma não linearidade (ReLU).
- A PL, usada após uma camada convolucional, destina-se a reduzir a dimensão dos dados de entrada e juntar características locais numa só. Pode-se usar médias ou escolher o maior valor encontrado em subregiões. Este segundo procedimento é o mais utilizado e chamado *maxpooling*. Na Figura 1 temos um exemplo com uma imagem 4×4 e um maxpooling com filtro 2×2 . Essa técnica reduz a quantidade de dados para a camada seguinte, reduzindo também o custo de processamento e memória.

CNN

2	10	8	5
5	7	2	15
5	1	6	8
4	3	5	1

 \Rightarrow

10	15
5	8

Figura 1: Exemplo de *maxpooling* no caso de uma imagem 4×4 .

CNN

- As camadas totalmente conectadas situam-se no final da rede e os padrões extraídos nas camadas de convolução anteriores são utilizadas para a classificação final da rede neural.
- As razões para essa arquitetura são (LeCun et al, 2015):
 - (i) em *arrays*, como imagens, grupos locais de valores são correlacionados, formando padrões (*motifs*) facilmente detectados;
 - (ii) as estatísticas locais dessas *arrays* são invariantes em relação à localização, donde a ideia de que os mesmos pesos são compartilhados por unidades em diferentes localizações.
- Os cálculos com CNN envolvem contrações em escalas múltiplas, linearização de simetrias hierárquicas e separação esparsa. Em muitas aplicações o número de amostras cresce linearmente com o número de dimensões.
- Como todo algoritmo de aprendizagem, uma CNN é baseada em alguma suposição de suavidade (regularidade) do classificador, digamos, $f(\mathbf{x})$, sendo \mathbf{x} o vetor de dados, e a natureza dessa regularidade é o problema matemático mais importante.
- A ideia é reduzir a dimensão de \mathbf{x} e isso pode ser feito definindo-se uma nova variável $\phi(\mathbf{x})$, em que ϕ é um operador contração, que reduz a variabilidade de \mathbf{x} , aliada à separação de valores distintos de $f(\mathbf{x})$. Os aspectos matemáticos de uma CNN estão descritas em Mallat (2026) e Kohler et al. (2022).

CNN - Exemplo 1

- Vejamos um exemplo de convolução com uma série temporal fictícia com $n = 10$ observações como entrada:

1,2	0,9	-0,8	0,7	1,5	-1,3	-1,0	0,7	1,3	1,4
-----	-----	------	-----	-----	------	------	-----	-----	-----

Consideremos um filtro com coeficientes:

1	2	1
---	---	---

- A convolução dos três primeiros valores da série com os pesos do filtro resulta $(1, 2) \times 1 + (0, 9) \times 2 + (-0, 8) \times 1 = 2, 2$. Deslocando-se uma unidade de tempo e efetuando o produto dos valores seguintes pelos coeficientes do filtro obtemos o valor 0. Continuando, obtemos a série de saída

2,2	0	2,1	2,4	-2, 1	-2,6	1,7	4,7
-----	---	-----	-----	-------	------	-----	-----

CNN - Exemplo 1

- Para que tenhamos convolução e *maxpooling*, temos que adicionar dois zeros no começo e final da série (*padding*):

0	0	1,2	0,9	-0,8	0,7	1,5	-1,3	-1,0	0,7	1,3	1,4	0	0
---	---	-----	-----	------	-----	-----	------	------	-----	-----	-----	---	---

- A série convolvida e a saída após tomar o máximo de cada três observações estão mostradas a seguir:

1,2	3,3	2,2	0	2,1	2,4	-2,1	-2,6	1,7	4,7	4,1	1,4
-----	-----	-----	---	-----	-----	------	------	-----	-----	-----	-----

3,3	2,4	1,7	4,7
-----	-----	-----	-----

- A Figura 2 ilustra uma CNN com série temporal como entrada. Se a entrada for outra *array*, como uma imagem, o esquema é o mesmo, obtendo-se não mais sequências unidimensionais, mas matrizes, como na Figura 1.

CNN - Exemplo 1

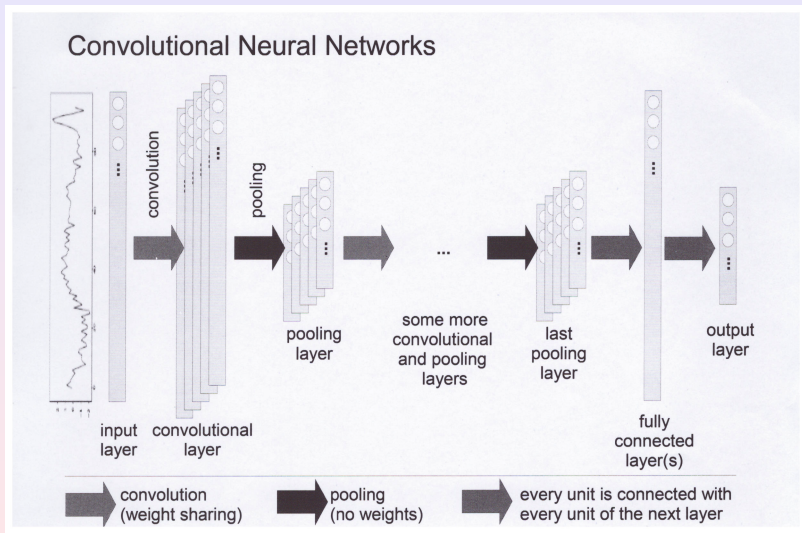


Figure 2: Rede neural convolucional.

CNN - Exemplo 2

- Vamos usar os dados de fotos de dígitos escritos a mão, **Mnist** (Modified NIST(National Institute of Standards and Technology)).
- Esses dados contém 60.000 imagens de treinamento e 10.000 imagens de teste, dos quais a metade de cada conjunto foi retirada conjunto de treinamento do NIST, as outras duas metades foram retiradas do conjunto teste do NIST.
- Esses dados foram usados com diversos tipos de classificadores, dentre os quais classificadores lineares, KNN, SVM, florestas aleatórias e diversos tipos de redes neurais, incluindo as CNN. Usaremos um código constante do site

https://rpubs.com/juanhklopper/example_of_a_CNN

CNN - Exemplo 2

Comentaremos alguns comandos:

- A primeira imagem é um 5:

```
> y_train[1,] # a primeira imagem \ 'e um 5
```

```
[1] 0 0 0 0 0 1 0 0 0 0
```

- Criando o modelo:

```
model <- keras_model_sequential() %>%  
  layer_conv_2d(filters = 16,  
                kernel_size = c(3,3),  
                activation = 'relu',  
                input_shape = input_shape) %>%  
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%  
  layer_dropout(rate = 0.25) %>%  
  layer_flatten() %>%  
  layer_dense(units = 10,  
              activation = 'relu') %>%  
  layer_dropout(rate = 0.5) %>%  
  layer_dense(units = num_classes,  
              activation = 'softmax')
```

CNN - Exemplo 2

- O comando `summary()` mostra que foram aprendidos 27.320 parâmetros.
`> model %>% summary()`,

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 16)	160
max_pooling2d (MaxPooling2D)	(None, 13, 13, 16)	0
dropout_1 (Dropout)	(None, 13, 13, 16)	0
flatten (Flatten)	(None, 2704)	0
dense_1 (Dense)	(None, 10)	27050
dropout (Dropout)	(None, 10)	0
dense (Dense)	(None, 10)	110
Total params: 27,320		
Trainable params: 27,320		
Non-trainable params: 0		

CNN - Exemplo 2

- O modelo é compilado com a função perda entropia cruzada e métrica acurácia:

```
model %>% compile(  
  loss = loss_categorical_crossentropy,  
  optimizer = optimizer_adadelta(),  
  metrics = c('accuracy')  
)
```

- Número de mini-batches e número de épocas:

```
batch_size <- 128  
epochs <- 50
```

- Treinando o modelo com 50 iterações (épocas):

```
model %>% fit(  
  x_train, y_train,  
  batch_size = batch_size,  
  epochs = epochs,  
  validation_split = 0.2  
)
```

CNN - Exemplo 2

- Avaliando a perda e acurácia no conjunto teste:

```
> score <- model \%>\% evaluate(x_test,  
+                               y_test)  
313/313 - 1s 2ms/step - loss: 0.1569 - accuracy: 0.9723
```

- Ou seja, a perda (dada pela entropia cruzada) no conjunto teste foi 0,1569 e a acurácia no conjunto teste 0,9723.
- O gráfico da Figura 3 mostra a evolução das perdas (medidas pela entropia cruzada) e da acurácia do procedimento.
- Um código em Python para o mesmo conjunto de dados pode ser encontrado em

https://keras.io/examples/vision/mnist_convnet/

CNN - Exemplo 2

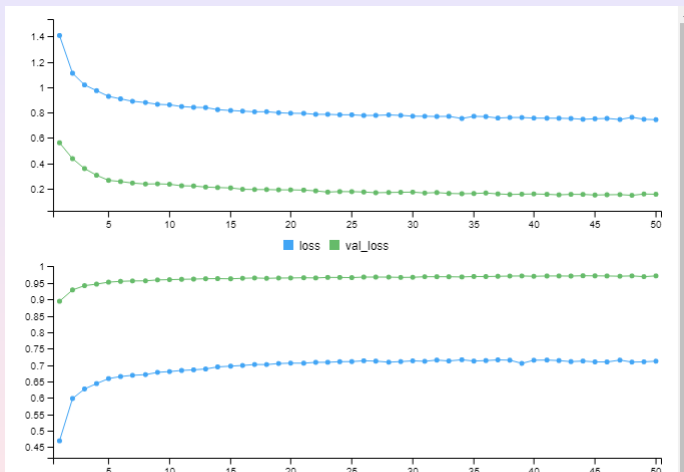


Figura 3: Perdas (entropia, painel superior) e acurácia(painel inferior) para o Exemplo 2.

Redes generativas adversárias

- As redes generativas adversárias (*generative adversarial networks*, GAN) foram introduzidas por Goodfellow et al. (2014). Elas consistem de duas redes neurais treinadas em oposição uma à outra: o **gerador** (generator) G e o **discriminador** (discriminator) D .
- Considere uma matriz de dados \mathbf{X} , e suponha que temos exemplos, considerados amostras i.i.d. de \mathbf{X} , com densidade de probabilidade $p(\mathbf{x})$, desconhecida.
- O gerador G tem como entrada um vetor de ruídos aleatórios \mathbf{z} e saída uma imagem $X_{\text{falso}} = G(\mathbf{z})$, com a mesma distribuição $p(\mathbf{x})$ dos dados reais.
- Por sua vez, o discriminador D depara-se com um exemplo real, $\mathbf{X} \sim p(\mathbf{x})$ ou um exemplo falso, X_{falso} gerado por G .
- Quem é apresentado é decidido jogando-se uma moeda honesta (a probabilidade de ser falso é $1/2$). A moeda é lançada independentemente em cada rodada do jogo.
- O discriminador usa uma função $D(\mathbf{x}) = P(\mathbf{x} \text{ real})$ (o classificador), tal que se $D(\mathbf{x}) > 0,5$ então \mathbf{x} é real; se $D(\mathbf{x}) < 0,5$, então \mathbf{x} é falso.

Redes generativas adversárias

- Essa é a **regra de classificação**. D gostaria que $D(\mathbf{x}) \approx 1$ quando \mathbf{x} for real e $D(\mathbf{x}) \approx 0$ quando falso.
- Na Figura 4 temos o esquema de uma rede neural GAN.

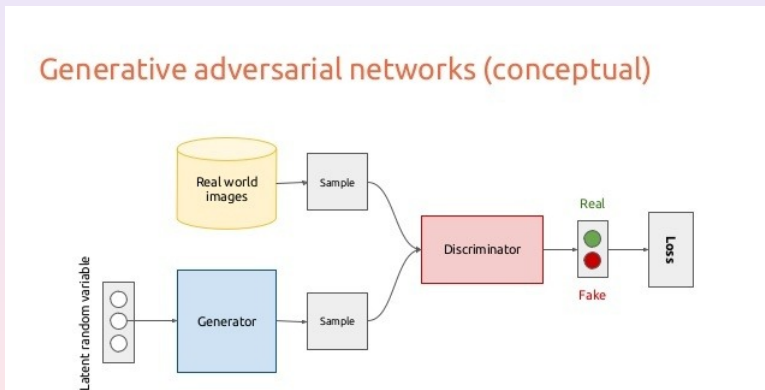


Figura 4: Rede neural geradora adversária.

Redes generativas adversárias

- Para gerar \mathbf{X} a partir de \mathbf{Z} usamos o Método Monte Carlo. De modo geral, a função de distribuição acumulada (f.d.a.) de um vetor $\mathbf{X} = (X_1, \dots, X_d)$ pode ser escrita como

$$F_{X_1, \dots, X_d}(x_1, \dots, x_d) = F_{X_1}(x_1)F_{X_2|X_1}(x_2|x_1) \cdots F_{X_d|X_1, \dots, X_{d-1}}(x_d|x_1, \dots, x_{d-1}).$$

- A seguir, geramos d variáveis aleatórias i.i.d. $\mathcal{U}(0, 1)$, digamos U_1, \dots, U_d e geramos o vetor \mathbf{X} por meio de:

$$\begin{aligned}x_1 &= F_{X_1}^{-1}(u_1), \\x_2 &= F_{X_2|X_1}^{-1}(u_2) \\&\vdots \\x_d &= F_{X_d|X_1, \dots, X_{d-1}}^{-1}(u_d).\end{aligned}$$

- Em nosso caso, sabemos que existe uma função G que transforma ruídos \mathbf{Z} num vetor \mathbf{X} com a densidade $f(\mathbf{x})$. Mas nesse caso, não conhecemos a distribuição de \mathbf{X} e será difícil obter a f.d.a.

Redes generativas adversárias

- Como mencionamos acima, a solução é imaginar que exista essa G na forma de uma rede neural e o gerador $G(\mathbf{z})$ terá parâmetros $\theta^{(G)}$ (os pesos da rede neural). Teremos que aprender os parâmetros dessa rede de forma aproximada.
- A função perda do discriminador é dada por

$$J^{(D)}(\theta) = -\frac{1}{2}E_{p_{\text{real}}}[\log(D(\mathbf{X}))] - \frac{1}{2}E_{\mathbf{z}}[\log((1 - D(G(\mathbf{z})))]. \quad (1)$$

- O tipo de jogo a usar é o de soma zero: a perda de um jogador é o ganho do outro e a soma das perdas dos dois jogadores é zero. Neste caso, a função perda do gerador é dada por

$$J^{(G)}(\theta^{(G)}, \theta^{(D)}) = -J^{(D)}(\theta^{(G)}, \theta^{(D)}). \quad (2)$$

- A solução de um jogo de soma zero é chamada **solução minimax**. O gerador G quer determinar seus pesos $\theta^{(G)}$ que minimizem sua perda, dada por

$$J^{(G)}(\theta^{(G)}, \theta^{(D)}) = \frac{1}{2}E_{p_{\text{dados}}}[\log(D(\mathbf{X}))] + \frac{1}{2}E_{\mathbf{z}}[\log((1 - D(G(\mathbf{z})))]. \quad (3)$$

Redes generativas adversárias

- Note que $\theta^{(G)}$ somente aparece na segunda parcela de (3). Portanto, podemos ignorar a primeira parcela ao otimizar essa expressão e usando apenas os dados \mathbf{Z} . A solução que minimiza as duas perdas individualmente é a **solução minimax**; para o jogador G será aquela que minimiza a perda $J^{(G)}$ sobre os parâmetros $\theta^{(G)}$ enquanto maximiza esta perda sobre $\theta^{(D)}$ e procedimento similar para $J^{(D)}$.
- Um método iterativo usa otimização via gradiente descendente estocástico e este pode ou não convergir. Estudos mostram que GANs podem produzir bons resultados em dados com baixas variabilidade e resolução.
- Várias variantes do GAN surgiram para melhorá-lo:
 - DCGAN: usa redes convolucionais profundas;
 - cGAN, ACGAN: conditional GAN
 - WGAN, WGAN-GP: diferentes funções perda.
- Para detalhes veja Assunção (2022).

Redes generativas adversárias - Exemplo

- Neste exemplo, voltamos a usar os dados Mnist e uma versão do GAN, a ACGAN. Um código em R pode ser encontrado em

```
https://github.com/rstudio/keras/blob/main/vignettes/  
examples/mnist\_acgan.R
```

- Vemos, abaixo, a última iteração do algoritmo, indicando as perdas do discriminador, 1,3604, e a do gerador, 0,9898.

Epoch 20/20

235/235 [====] - 203s 863ms/step - d_loss: 1.3604 - g_loss: 0.9898

- Na Figura 5 notamos que, enquanto a função perda do discriminador diminui ao longo das épocas, o mesmo não acontece com a perda do gerador. Na Figura 6 temos o dígito 6 previsto corretamente pelo modelo.

Redes generativas adversárias - Exemplo

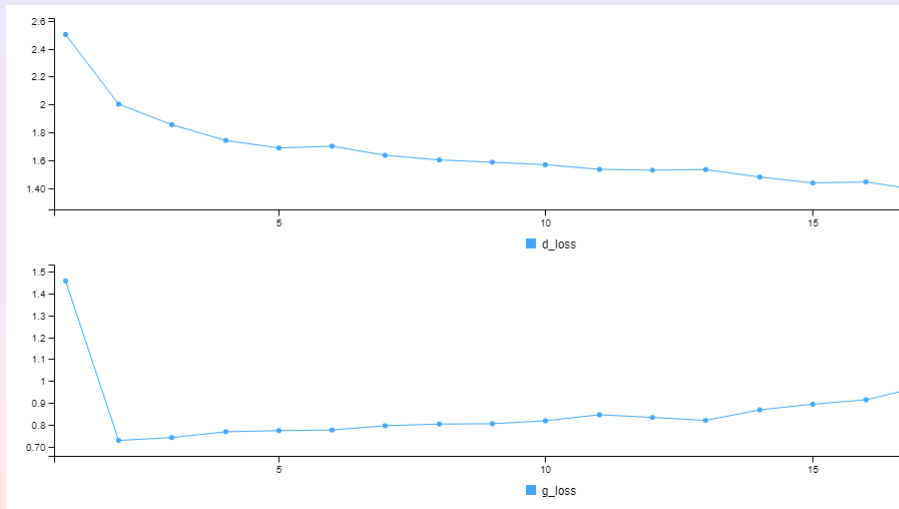


Figura 5: Comportamento das perdas do gerador e discriminador para o Exemplo.

Redes generativas adversárias - Exemplo

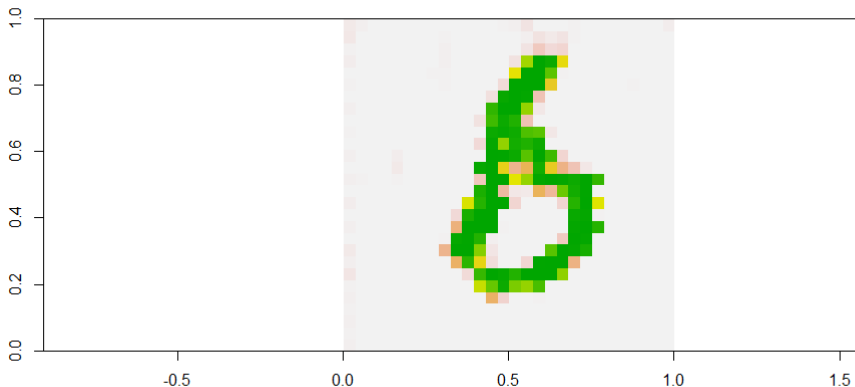


Figura 6: Dígito 6 previsto pelo modelo.

Referências

Assunção, R. (2022). Deep Learning. Short Course, SINAPE 2022, Gramado, RS.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative adversarial networks, arXiv:1406.2661.

Morettin, P. A. and Singer, J. M. (2023). *Estatística e Ciência de Dados*. 2a. edição. LTC.