Lista Final — MNIST

Relatório

Grupo EuQuipe: Vinícius Claudino Ferraz. 2019435823

(1)

Iniciei com os dados originais, em $n = 784$ colunas. Utilizei tanto a linguagem R quanto o Python.

Tentei rodar classificador Bayes.R com a classe 1, todo o resto 2. Utilizei o k-means com $k = 20$.

Resultado: impossível utilizar o pdfnvar$(\cdots)$. A matriz inteira era quase inteiramente igual a zero.

(2)

Rodei ELM.R.

Rodei MLP.R.

Tentei combinar ambos acima. parte2.R.

Gerei uma submissão com o Python: parte3.py.

Comparei os resultados. parte4.R.

Pouca diferença. Os 2 métodos dão praticamente a mesma submissão.

(3)

Fui procurar extração de características. O PCA.

Deu erro de coluna identicamente zero. Reduzi para $n = 74$ colunas.

O PCA acusou que com apenas 20 colunas e autovetores, eu já teria acima de 90% das características. Mas eu descartei isso. Afinal, devia ser uma competição de acurácias.

Rodei os itens 1 e 2 novamente.

(4)

Tentei minha própria ideia. Segundo grau: elipsoides, paraboloides, hiperboloides.

Exemplo: XOR. Acrescente a coluna xy, porque x^2 e y^2 são os mesmos.

```
x y xy (x + y - 2xy)
0 0 0        0
0 1 0        1
1 0 0        1
1 1 1        0
```

Logo, os pesos são $w = (1, 1, -2)$.

Entretanto, meu (x,y) são 74 colunas, mais 74 da representação de cada coluna ao quadrado, mais $\binom{74}{2} = \frac{74 \cdot 73}{2} = 2701$ de $x_i \cdot x_j$.

Total de $n = 2849$ colunas.

Rodei o item 2 novamente.

(5)

Tornei a ler o enunciado do trabalho. Estado da arte.

https://en.wikipedia.org/wiki/MNIST_database

O melhor da Wikipedia possui os seguintes dados:

Tipo: Convolutional neural network.

Classificador: Committee of 20 CNNS with Squeeze-and-Excitation Networks.

Preprocessamento: Data augmentation.

Erro: 0,17% https://github.com/Matuzas77/MNIST-0.17.git

Está em Python. Demorei muito a concluir que 4 épocas rodavam em 2 dias (48 horas de processamento); desisti porque 20 épocas rodariam em 10 dias. Não tenho no-break.

(6)

O segundo melhor da Wikipedia possui os seguintes dados:
Tipo: Random Multimodel Deep Learning (RMDL).
Classificador: 10 DNN — 10 RNN — 10 CNN.
Preprocessamento: Nenhum.
Erro: 0,18% https://github.com/kk7nc/RMDL/blob/master/README.rst
Manipulei muito para rodar em 13 horas e alguns minutos.
Comparei com ELM e MLP, a submissão dos 3 é praticamente a mesma.
$245/4000 = 6{,}125\%$ de diferença com ELM/MLP.
Vamos agora juntar as peças.

Descrição do problema e do banco de dados

Seja o conjunto de cores $C = \{0,1,2,\cdots,255\}$. Seja o conjunto de classes $Y = \{1,5,6,7\}$.

Treinamento:

x_t são 13017 vetores de C^{784}. Cada vetor é um bitmap 28×28. A impressão desses bitmaps é um algarismo de Y escrito à mão.

y_t são 13017 escalares de Y.

O nosso objetivo é encontrar uma função f que aproxima $y_t = f(x_t)$, daí fazer a previsão de:

x_v são 4000 vetores de C.

y_v são 4000 escalares de Y a submeter. Tecnicamente, dada qualquer escrita à mão, queremos reconhecer se ela mais se parece com 1, 5, 6 ou 7.

Revisão bibliográfica sobre o assunto

— O que é ELM?

— Segundo BRAGA, “As ELMs (Extreme Learning Machines) [HZS04], ou Máquinas de Aprendizado Extremo, se baseiam nos mesmos princípios dos modelos descritos nos parágrafos anteriores", (trata-se de redes neurais com camada escondida; $H = \Phi_h(xZ)$; $\Phi_o(HW) = Y$ é a saída; x é a entrada), “porém, com uma interpretação particular do Teorema de Cover [Cov65]. Como no Teorema não há nenhuma restrição sobre a forma do mapeamento $\phi_h(x_i, Z)$, o princípio das ELMs é que a matriz Z seja selecionada de forma aleatória e que o número de funções $h_i(x, z_i)$ seja suficientemente grande para garantir a separabilidade no espaço da camada intermediária. Uma vez garantida uma projeção linearmente separável, o problema volta a ser o de encontrar o separador linear, o que pode ser feito de maneira direta através da pseudo-inversa".

— O que é MLP?

— Ainda segundo BRAGA, As redes MLP (Perceptron de Múltiplas Camadas) “são caracterizadas por uma estrutura de neurônios com duas ou mais camadas alimentadas para frente (feed-forward). Os neurônios das camadas intermediárias possuem tipicamente funções de ativação sigmoidais enquanto que os neurônios da camada de saída podem ser implementados também com funções de ativação lineares, dependendo do problema a ser resolvido. Neste caso, quando as funções de saída são lineares, as redes MLP e as ELMs são idênticas, diferenciando-se somente pela forma como a matriz Z é obtida. Usualmente as saídas sigmoidais são utilizadas em problemas de classificação, enquanto que as saídas lineares são utilizadas em problemas de aproximação de funções ou de previsão. Tipicamente, ambos os mapeamentos, $\Phi_h(X, Z)$ e $\Phi_o(H, W)$ são obtidos de forma iterativa durante o treinamento."

— O que é backpropagation?

— Uma otimização no ajuste dos pesos Z e W em uma rede MLP.

— O que é CNN?

— Segundo a Wikipedia, “No contexto de inteligência artificial e aprendizagem de máquina, uma rede neural convolucional (CNN do inglês Convolutional Neural network ou ConvNet) é uma classe de rede neural artificial do tipo feed-forward, que vem sendo aplicada com sucesso no processamento e análise de imagens digitais.

Uma CNN usa uma variação de perceptrons multicamada desenvolvidos de modo a demandar o mínimo pré-processamento possível. Essas redes também são conhecidas como redes neurais artificiais invariantes a deslocamento (shift invariant) ou invariantes a espaço (space invariant), em ambos os casos representadas pela sigla em inglês SIANN.

As redes convolucionais são inspiradas nos processos biológicos. Nelas o padrão de conectividade entre os neurônios é inspirado na organização do córtex visual dos animais. Neurônios corticais individuais respondem a estímulos apenas em regiões restritas do campo de visão conhecidas como campos receptivos. Os campos receptivos de diferentes neurônios se sobrepõem parcialmente de forma a cobrir todo o campo de visão.

Uma CNN tende a demandar um nivel minimo de pre-processamento quando comparada a outros algoritmos de classificação de imagens. Isso significa que a rede “aprende" os filtros que em um algoritmo tradicional precisariam ser implementados manualmente. Essa independencia de um conhecimento a priori e do esforço humano no desenvolvimento de suas funcionalidades basicas pode ser considerada a maior vantagem de sua aplicação.

Esse tipo de rede é usada principalmente em reconhecimento de imagens e processamento de vídeo, embora já tenha sido aplicada com sucesso em experimentos envolvendo processamento de voz e linguagem natural.

Na saúde usa-se esta metodologia com algoritmos específicos, recorrendo a um grande número de fotografias clínicas, para o diagnóstico da retinopatia diabética e do cancro da pele, com resultados muito precisos e comparáveis aos clínicos especializados."

— Por que o CNN demorou tanto?

— Estou vendo no código chamadas de model.fit com tolerâncias baixíssimas de 10^{-3}, 10^{-4}, 10^{-5}, totalizando $20 \cdot (13+3+3) = 380$ execuções.

— O que é RMDL?

— Segundo os autores que o idealizaram, “This approach, called Random Multimodel Deep Learning (RMDL), uses three different deep learning architectures: Deep Neural Networks (DNN), Convolutional Neural Netwroks (CNN), and Recurrent Neural Networks (RNN). Test results with a variety of data types demonstrate that this new approach is highly accurate, robust and efficient.

RDML searches across randomly generated hyperparameters for the number of hidden layers and nodes (density) in each hidden layer in the DNN. CNN has been well designed for image classification. RMDL finds choices for hyperparameters in CNN using random feature maps and random numbers of hidden layers. CNN can be used for more than image data. The structures for CNN used by RMDL are 1D convolutional layer for text, 2D for images and 3D for video processings. RNN architectures are used primarily for text classification. RMDL uses two specific RNN structures: Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM). The number of GRU or LSTM units and hidden layers used by the RDML are also the results of search over randomly generated hyperparameters."

Metodologia

Como disse no início, fui rodando do mais simples para o mais complexo e comparando. Submeti RMDL, o qual nas comparações deixava todos os outros para trás.

Método do ELM, com $n = 2849$ colunas:

Em YELM.R, não dividi Yhat pelo valor absoluto. Utilizei os parâmetros abaixo.

```
p <- 2849
par <- 1
res <- treinaELM(xin,yin,p,par)
```

Gerei arquivos valid_ELM_1.csv, valid_ELM_5.csv, valid_ELM_6.csv e valid_ELM_7.csv. Eram sempre duas classes; por exemplo, se 5, então classe 1; se diferente de 5, então $y \leftarrow -1$.

Método do MLP:

Não dividi y_v pelo valor absoluto. Utilizei os parâmetros abaixo.

```
p <- 2850
dimx <- 2850
dimy <- 1
Z <- matrix(runif(dimxp - dimx) - 0.5, nrow = dimx, ncol = p - 1)
W <- matrix(runif(p) - 0.5, nrow = p, ncol = dimy)
tol <- 0.01
eta <- 0.01
maxepocas <- 4
[...]
ei2 <- 0
erros <- 0
xseq <- sample(N)
for (i in 1:N) {
  irand <- xseq[i]
  xatual[1:(dimx-1),1] <- xtrain[irand,]
  xatual[dimx,1] <- 1
  yatual <- ytrain[irand,]
```

```
  U <- t(xatual) %*% Z # xatual eh dimx x 1 e Z eh dimx x (p-1)
  H <- tanh(U)
  Haug <- cbind(H, 1) # Haug eh 1xp
  O <- t(Haug %*% W) # 1xp x p x dimy
  yhat <- tanh(O)    # dimy x 1
  yhat <- yhat/abs(yhat)
  e <- yatual - yhat
  if (yatual != yhat)
    erros <- erros + 1
  flinhaO <- sech2(O)
  dO <- t(e * flinhaO)         # .*
  Wminus <- W[-p,]             # retirar polarizacao
  ehidden <- dO %*% t(Wminus) # dO eh dimy x 1, W eh p x dimy
  flinhaU <- sech2(U)
  dU <- ehidden * flinhaU      # .* ; ehidden eh 1x(p - 1)
  W <- W + eta * (t(Haug) %*% dO)
  Z <- Z + eta * (xatual %*% dU)
  ei2 <- ei2 + (e %*% t(e))
}
```

Gerei arquivos valid_MLP_1.csv, valid_MLP_5.csv, valid_MLP_6.csv e valid_MLP_7.csv.

Combinação:

Eu tinha vários números próximos de -1 e de 1. Escolhi a classe pela distância até o 1. Exemplo: se a maior dentre $(1{,}5{,}6{,}7)$ fosse a da última coluna, então a classe a submeter era a 7.

Primeiro gerei saídas separadas para ELM e MLP. Se o máximo fosse negativo, poderia ser um “sem-classe", ou seja, classe zero. Daí contei os certos e os errados.

Na hora de combinar, escolhi o máximo dentre as colunas $ELM_{(1,5,6,7)}$, $MLP_{(1,5,6,7)}$, de forma que nenhum y retornasse classe zero.

ELM retornou 299 zeros. MLP retornou 104 zeros, mais 237 divergências em relação ao ELM.

Dos 299, 195 foram bem cobertos. Portanto, as incertezas são:

$299/4000 = 7{,}475\%$ no ELM.

$341/4000 = 8{,}525\%$ no MLP/ELM.

Criei um projeto Python só para tirar um número sequencial à esquerda que o R gerava. A saída do Python bem que poderia ser uma submissão.

Método do RMDL, com $n = 784$ colunas. Os parâmetros utilizados foram os seguintes:

```
n_epochs = [10, 5, 4] ## 100 DNN-RNN-CNN
Random_Deep = [3, 3, 3] ## 5 min + 1 hour + 12 hours. DNN-RNN-CNN
Z = RMDL.Image_Classification(x_train, y_train, x_test, y_test,
                              shape=(28, 28, 1),
                              batch_size=128,
                              sparse_categorical=True,
                              random_deep=Random_Deep,
                              epochs=n_epochs)
```

Reduzi para $(10+5+4)\cdot 3 = 57$ épocas.

A função Image_Classification foi alterada para retornar final_y, o qual seria gravado em csv.

Havia um erro de vez em quando de números negativos na indexação, conforme trecho abaixo.

```
def Build_Model_CNN_Image(shape, nclasses, sparse_categorical,
                          min_hidden_layer_cnn, max_hidden_layer_cnn,
                          min_nodes_cnn,
                          max_nodes_cnn, random_optimizor, dropout):
   [...]
   try:
      model.add(MaxPooling2D(pool_size=(2, 2)))
   except ValueError:
      print("*** without max pool")
```

Depois que encapsulei em try, rodou até o fim, sem parar quando alguma célula de shape[i] era igual a 1.

Resultados — Acurácia

ELM — Dividi ao meio. Metade treinamento, metade teste. Calculei a acurácia 5 vezes e encontrei:

```
[1,] 97,80014
[2,] 97,84204
[3,] 97,87721
[4,] 97,86190
[5,] 97,74787
```

Cuja média é 97,82583 e cujo desvio padrão é 0,05230234.

MLP — Também dividi ao meio. Metade treinamento, metade teste. Calculei o MSE = Mean Square Error 5 vezes e encontrei:

```
[1,] 0,09465787 = 154,008 * 4/6508 => 2,366447% = 100 - 97,63355325%
[2,] 1,12153437 =                                  71,96164075%
[3,] 0,10480727 =                                  97,37981825%
[4,] 0,10391901 =                                  97,40202475%
[5,] 0,14617593 =                                  96,34560175%
```

Cuja média é 92,14452775 e cujo desvio padrão é 11,29354786.

RMDL — A cada iteração, o programa imprimia acurácia de treinamento de 99,6%.

Conclusão

Fizemos o que foi possível.

Os fontes e os dados intermediários são importantes.

Backup no meu Google Drive.

Reitero que submeti o segundo melhor por não ter um hardware bom o bastante.

Vinicius Claudino Ferraz, 29/agosto/2021.