COPPE/UFRJ (21/OUT/2019)
Redes Neurais sem Peso

Trabalho I – Analise da Base NMINST com WISARD

Nome: Luiz Marcio Faria de Aquino Viana
CPF: 024.723.347-10
RG: 08855128-8 IFP-RJ

1) INTRODUCAO

O objetivo deste trabalho e realizar uma analise de desempenho do modelo de rede neural sem peso, WISARD, no reconhecimento de números manuscritos. Para este estudo utilizamos como referencia a base NMINST e efetuamos uma analise comparativa entre o modelo de rede neural sem peso WISARD e o modelo de rede neural convolucional profunda com o objetivo de avaliar o tempo de treinamento, o tempo de classificação, e o desmpenho final de cada um dos modelos de aprendizado de maquina avaliados.

Neste experimento, verificamos uma das características da rede neural WISARD que e a rapidez obtida no treinamento. Esta característica permite que padrões de imagens sejam aprendidos muito rapidamente.

No experimento a rede neural WISARD efetuou o aprendizado da base NMINST em apenas 2,1732 segundos em media, enquanto a rede neural convolucional profunda analisada efetuou o mesmo treinamento em 135,5393 segundos. Isto e, o tempo gasto pela rede neural WISARD no treinamento dos dados foi de 1,6 % do tempo gasto pela rede neural convolucional profunda avaliada.

Porem, verificamos que também existem desvantagens no uso da rede neural WISARD em relação a rede neurais convolucional profunda. No experimento a rede neural WISARD obteve um resultado pior na classificação dos dados, onde o tempo médio de classificação foi de 97.5255 segundos, enquanto o algoritmo utilizado pela rede neural convolucional profunda obteve um tempo médio de 4.3903 segundos. Isto e, o tempo gasto pela rede neural convolucional profunda na classificação dos dados foi de 4,5 % do tempo gasto pela rede neural WISARD.

Além disso, as redes neurais WISARD apresentaram um resultado pior quando comparamos a precisão na classificação dos dados. No experimento, a rede neural convolucional profunda analisada obteve um resultado de 98,3% de acerto, enquanto a rede neural WISARD obteve um resultado de 70,7%.

2) CONCEITOS DE REDES NEURAIS CONVOLUCIONAIS PROFUNDAS

Redes neurais profundas, são modelos de redes neurais fortemente conectadas que apresentam uma camada de entrada, duas ou mais camadas ocultas e uma camada de saída.

As redes neurais convolucionais, são modelos de redes neurais profundas que são largamente utilizadas na classificação de imagens. Este modelo de rede neural procura separar em cada camada as características que definem as imagens.

Quando estudamos imagens com maior profundidade de cores, como imagens RGB, podemos separar os elementos que formam a imagem em três vetores, correspondendo as intensidades de vermelho, verde e azul presentes na imagem. Desta forma, as características das imagens são melhor preservadas e o resultado obtido no treinamento das redes neurais organizadas desta forma, redes neurais convolucionais, são muito melhores do que os resultados obtidos pelas redes neurais tradicionais.

No experimento realizado utilizamos uma rede convolucional profunda com uma camada de entrada, duas camadas convolucionais ocultas, com 64 e 32 features, e uma camada de saída softmax com 10 elementos, representando cada um dos algarismos aprendidos. Esta rede neural convolucional profunda foi treinada através de 1 época de treinamento com uma base de 60.000 imagens de caracteres manuscritos.

Apos o treinamento o modelo de rede neural convolucional profunda foi aplicado na classificação de uma base com 10.000 imagens de caracteres manuscritos.

3) CONCEITOS DE REDES NEURAIS SEM PESO

As redes neurais sem peso procuram mapear os bits da imagem em um vetor de endereçamento de memoria que armazena o aprendizado obtido durante a fase de treinamento da rede neural.

No experimento utilizamos uma rede neural WISARD com 3 bits de endereçamento. Desta forma, três pixels aleatórios da imagem são mapeados em três bits de endereçamento. Permitindo que ate $2^3 = 8$ posições de memoria sejam mapeadas por cada elemento.

Para permitir este mapeamento, a imagem foi preprocessada, e a profundidade de cores reduzida para uma imagem em preto e branco, *Figura 1*. Em seguida, a rede neural WISARD foi treinada com uma base de treinamento com 60.000 imagens de caracteres manuscritos.

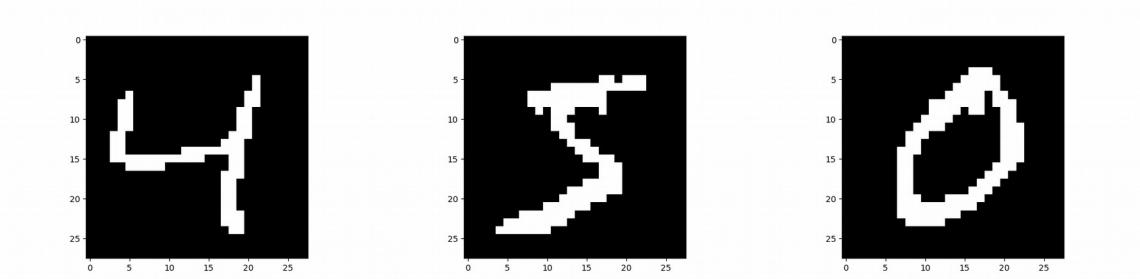

Figura 1: *Exemplo de imagem em preto e branco obtida apos o preprocessamento dos dados*

Apos o treinamento o modelo de rede neural WISARD foi aplicado na classificação de uma base com 10.000 imagens de caracteres manuscritos. A *Figura 2*, apresenta dois exemplos de imagens mentais geradas durante o aprendizado da rede neural WISARD.

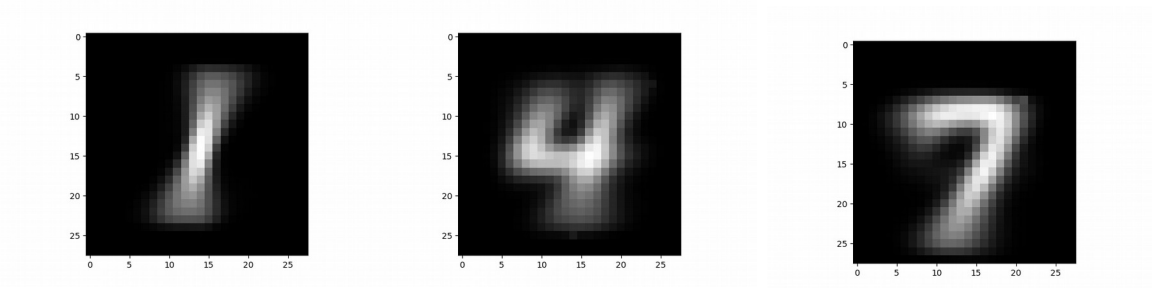

Figura 2: *Exemplo de imagem mental obtida apos o treinamento da rede neural WISARD*

A mesma base de treinamento e teste foram aplicadas aos modelos de rede neural WISARD e de rede neural convolucional profunda analisados.

4) RESULTADOS OBTIDOS COM REDES NEURAIS CONVOLUCIONAIS PROFUNDAS

Para realização do experimento, o modelo de rede neural convolucional profunda foi executado 5 vezes para uma base de treinamento com 60.000 imagens de caracteres manuscritos e uma base de teste com 10.000 imagens de caracteres manuscritos.

Em seguida, o tempo de treinamento, o tempo de classificação e o resultado foram registrados na *Tabela 1*.

	Rede Neural Convolucional Profunda		
	Treinamento	**Classificacao**	**Resultado**
Teste 1	136.2262	4.3991	0.9857
Teste 2	135.2238	4.3792	0.9826
Teste 3	134.5444	4.3672	0.9813
Teste 4	135.2767	4.4176	0.9827
Teste 5	136.4254	4.3885	0.9841
Media =	135.5393	4.3903	0.9833

Tabela 1: *Resultados obtidos no treinamento de uma rede neural convolucional profunda L=2 e M=64*

Podemos observar na *Tabela 1*, que a rede neural convolucional profunda obtém excelente resultado na classificação dos dados, com uma taxa de acerto de 98,33%. Entretanto, o tempo de aprendizado deste modelo de rede neural e bastante lento, sendo que para 1 época de treinamento, foram necessários 136,4254 segundos de processamento.

5) RESULTADOS OBTIDOS COM REDES NEURAIS SEM PESO - WISARD

Para realização do experimento, o modelo de rede neural WISARD também foi executado 5 vezes para a mesma base de treinamento com 60.000 imagens de caracteres manuscritos e a mesma base de teste com 10.000 imagens de caracteres manuscritos.

Em seguida, o tempo de treinamento, o tempo de classificação e o resultado foram registrados na *Tabela 2*.

	WISARD		
	Treinamento	**Classificacao**	**Resultado**
Teste 1	2.1707	97.4596	70.8900
Teste 2	2.1686	102.8278	69.9500
Teste 3	2.1699	97.1135	70.8500
Teste 4	2.1772	98.4594	69.5900
Teste 5	2.1794	91.7673	72.1200
Media =	2.1732	97.5255	70.6800

Tabela 2: *Resultados obtidos no treinamento de uma rede neural WISARD com 3 bits de mapeamento*

Podemos observar na *Tabela 2*, que a rede neural WISARD obtém excelente resultado no tempo de treinamento dos dados. Obtendo um resultado médio de 2.1732 segundos para o treinamento da base. Este desempenho representa apenas 1,6% do tempo de aprendizado de 1 época da rede neural convolucional profunda analisada.

Entretanto, a taxa de acerto da rede neural WISARD foi de apenas 70,68%. Este valor e muito menor do que o registrado pela rede neural convolucional profunda analisada. Porem, este valor pode não ser apropriado para todas as aplicações de aprendizado de maquina, mas devido a rapidez obtida no aprendizado de padrões de imagens, a rede neural WISARD possui inúmeras aplicações praticas.

Verificamos também neste experimento, que o tempo de classificação da base de teste com 10.000 imagens manuscritas usando a rede neural WISARD foi muito pior que o tempo registrado pela rede neural convolucional profunda analisada. Neste experimento, a rede neural convolucional profunda registrou um tempo que foi 4,5% do tempo registrado pela rede neural WISARD.

Acredito que esta diferença de desempenho na classificação dos dados, seja resultado de uma fraca otimização na implementação do algoritmo de classificação pela biblioteca wisardpkg para a linguagem python, que foi utilizada neste experimento.

6) CONCLUSÃO

Concluímos neste experimento que a aplicação da rede neural WISARD oferece uma grande vantagem no treinamento dos dados, com um tempo de processamento que e 1,6% do tempo registrado por uma rede convolucional profunda.

Podemos concluir também, que se efetuarmos uma otimização na implementação da biblioteca wisardpkg para a linguagem python, podemos obter também excelente desempenho na classificação dos dados, reduzindo a diferença entre o tempo médio registrado pela rede neural WISARD e a rede neural convolucional profunda utilizada na comparação.

Neste experimento, podemos concluir também que a rapidez obtida no treinamento de padrões de imagens tem um preço, que e a baixa taxa de acerto obtida, com 70,68% de acerto, contra 98,33% de acerto das redes convolucionais profundas.

ANEXO I – CÓDIGO REDE NEURAL CONVOLUCIONAL PROFUNDA (L=2 / M=64 / EPOCH=1)

```
import datetime
import csv
import numpy as np
import matplotlib.pyplot as plt
import keras
import keras.callbacks

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist
from keras.utils import to_categorical
from sklearn import datasets, svm
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from scipy.sparse import coo_matrix

class LossHistory(keras.callbacks.Callback):
	def on_train_begin(self, logs={}):
		self.losses = []
		self.acc = []
		self.list_X = []
		self.curr_X = 0
		self.list_Epoch = []
		self.list_Epoch.append(0.0)

	def on_batch_end(self, batch, logs={}):
		self.curr_X = self.curr_X + 1
		self.losses.append(logs.get('loss'))
		self.acc.append(logs.get('acc'))
		self.list_X.append(self.curr_X)

	def on_epoch_end(self, epoch, logs={}):
		self.list_Epoch.append(self.curr_X)

def calc_test(arr1, arr2):
	n = 0
	n_win = 0
	for v1 in arr1:
		v2 = arr2[n]
		n = n + 1
		if v1 == v2:
			n_win = n_win + 1
	result = (n_win / n * 100.0)
	return result

def plot_learning_curve(title, train_scores, train_sizes, train_epoch):
	plt.figure()
	plt.title(title)
	plt.xlabel("Training epoch")
	plt.ylabel("Score")
	plt.grid()
	plt.plot(train_sizes, train_scores)
	plt.legend(loc="best")
	n = 0
	for val_x in train_epoch:
		val_str = "Epoch " + str(n)
		plt.text(val_x, 0, val_str)
		n = n + 1
	return plt

startTime = datetime.datetime.now()
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train /= 255
X_test /= 255

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

n_train = len(X_train)
n_test = len(X_test)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Data Preparation - Elapsed Time")
print(elapsedTime)

model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

train_hist = LossHistory()

startTime = datetime.datetime.now()

hist = model.fit(X_train, y_train, batch_size=32, nb_epoch=1, verbose=1, callbacks=[train_hist])

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Train - Elapsed Time")
print(elapsedTime)

startTime = datetime.datetime.now()

accuracy = model.evaluate(X_test, y_test)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Classify and Test - Elapsed Time")
print(elapsedTime)

print("Result")
print(accuracy)

plot_learning_curve(
              "L_2-M_64-Epoch_10-Losses",
              train_hist.losses,
              train_hist.list_X,
              train_hist.list_Epoch)

plot_learning_curve(
              "L_2-M_64-Epoch_10-Acc",
              train_hist.acc,
              train_hist.list_X,
              train_hist.list_Epoch)

plt.show()
```

ANEXO II – CODIGO REDE NEURAL WISARD (3 BITS ENDERESSAMENTO)

```
def calc_test(arr1, arr2):
	n = 0
	n_win = 0
	for v1 in arr1:
		v2 = arr2[n]
		n = n + 1
		if v1 == v2:
			n_win = n_win + 1
	result = (n_win / n * 100.0)
	return result

def convert_to_list(arr):
	lst = []
	for arr_row in arr:
		lst_row = []
		for arr_col in arr_row:
			lst_row.append(arr_col)
		lst.append(lst_row)
	return lst

def convert_to_list2(arr):
	lst = []
	for arr_col in arr:
		lst.append(arr_col)
	return lst

def plot_learning_curve(title, train_scores, train_sizes, train_epoch):
	plt.figure()
	plt.title(title)
	plt.xlabel("Training epoch")
	plt.ylabel("Score")
	plt.grid()
	plt.plot(train_sizes, train_scores)
	plt.legend(loc="best")
	n = 0
	for val_x in train_epoch:
		val_str = "Epoch " + str(n)
		plt.text(val_x, 0, val_str)
		n = n + 1
	return plt

def show_mental_image(s):
	img = np.array(s, dtype='float')
	pixels = img.reshape((28, 28))
	plt.imshow(pixels, cmap='gray')
	plt.show()

import datetime
import wisardpkg as wp
import tensorflow as tf
import numpy as np
import math as m
import matplotlib.pyplot as plt

from array import *
from matplotlib import pyplot as plt
from synthesizer import Player, Synthesizer, Waveform

startTime = datetime.datetime.now()

(X, y), (X_t, y_t) = tf.keras.datasets.mnist.load_data()

X = X.reshape(60000,784)
X_t = X_t.reshape(10000,784)

X = X / 128
X_t = X_t / 128

X = X.astype('int')
y = y.astype('str')
```

```
X_t = X_t.astype('int')
y_t = y_t.astype('str')

X1 = []
y1 = []

X_t1 = []
y_t1 = []

X1 = convert_to_list(X)
y1 = convert_to_list2(y)

X_t1 = convert_to_list(X_t)
y_t1 = convert_to_list2(y_t)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Data Preparation - Elapsed Time")
print(elapsedTime)

show_mental_image(X[0])

wsd = wp.Wisard(3, ignoreZeros=False, verbose=False)

startTime = datetime.datetime.now()

wsd.train(X1, y1)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Train - Elapsed Time")
print(elapsedTime)

startTime = datetime.datetime.now()

out = wsd.classify(X_t1)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Classify - Elapsed Time")
print(elapsedTime)

startTime = datetime.datetime.now()

rst = calc_test(out, y_t1)

endTime = datetime.datetime.now()

elapsedTime = endTime - startTime
print("Test - Elapsed Time")
print(elapsedTime)

print("Result")
print(rst)

mentalImages = wsd.getMentalImages()
s0 = mentalImages["0"]
show_mental_image(s0)
```