Profs. Fernando J. Von Zuben & Rosana Veroneze
DCA/FEEC/Unicamp

IA353 - Exercícios de Fixação de Conceitos - EFC 3
Atividade Individual - Peso 5

Especificação: Trabalhe no ambiente Anaconda e elabore o seu relatório como um Jupyter notebook. Todas as atividades deverão, assim, compor um único relatório no Jupyter notebook, a ser encaminhado à professora Rosana Veroneze (rveroneze@gmail.com) até 23h59 do dia 27 de junho de 2019. Inicie o **assunto** da mensagem a ser enviada com o termo [**IA353**].

Questão 1: Esta questão aborda conceitos de interpretabilidade em aprendizado de máquina, evidenciando dois paradigmas de grande relevância e focando num subconjunto de técnicas, dentre várias possibilidades que vêm sendo propostas na literatura. A busca por modelos de aprendizado interpretáveis está longe de ser recente, mas até recentemente havia uma predominância de foco em acurácia e se defendia a existência de um compromisso entre acurácia e interpretabilidade, de modo que a busca por mais acurácia tenderia a gerar modelos menos interpretáveis, da mesma forma que modelos mais interpretáveis não conseguiriam atingir os mesmos níveis de acurácia que modelos "caixa-preta". As últimas conquistas voltadas para interpretabilidade em aprendizado de máquina, no entanto, demonstram que acurácia e interpretabilidade podem caminhar juntas, trazendo maior confiabilidade para aplicações de inteligência artificial.

Tomando o mesmo problema de classificação de dados da base MNIST, o objetivo desta questão é analisar a interpretabilidade uma RNA treinada. Para tanto, vamos utilizar o código a seguir para treinar uma RNA.

```
import keras

mnist = keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = keras.models.Sequential()
model.add(keras.layers.Conv2D(32, kernel_size=(3, 3),
 activation='relu',input_shape=(28, 28, 1)))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(0.25))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(10, activation='softmax'))

model.get_config()

model.compile(optimizer='adam',
```

```
 loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
evaluation = model.evaluate(x_test, y_test)

model.save('mnist_model.h5')
```

a) Utilize a biblioteca `innvestigate` [1] para analisar as predições da RNA e comparar diferentes métodos de *explicação* [4]. Uma explicação é a coleção de características do domínio interpretável, que contribuíram para que uma dada amostra produzisse uma decisão (de classificação ou regressão, por exemplo) [4]. Utilize os seguintes métodos da biblioteca `innvestigate`: `Gradient, SmoothGrad, DeepTaylor, LRPAlphaBeta, LRPEpsilon, LRPZ`. Um código que pode servir de ponto de partida é fornecido a seguir. Escolha 6 imagens aleatórias de 3 classes distintas para analisar, portanto teremos 2 imagens por classe. Escolha parâmetros adequados para os métodos `LRPAlphaBeta` e `LRPEpsilon` (use os mesmos parâmetros para as 6 imagens escolhidas). A Fig. 1 apresenta um modelo de como os resultados devem ser exibidos. Analise o resultado obtido.

```
import keras
import innvestigate
import matplotlib.pyplot as plot

mnist = keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = keras.models.load_model('mnist_model.h5')
model_wo_sm = innvestigate.utils.keras.graph.model_wo_softmax(model)

imagem = x_test[0:1]
plot.imshow(imagem.squeeze(), cmap='gray', interpolation='nearest')

analyzer = innvestigate.analyzer.LRPEpsilon(model=model_wo_sm,
 epsilon=1)
analysis = analyzer.analyze(imagem)
plot.imshow(analysis.squeeze(), cmap='seismic',
 interpolation='nearest')
```

b) Utilize a biblioteca `keras-vis` [3] para interpretar as predições da RNA para cada uma das 10 classes. Neste caso, procuramos por um padrão de entrada que produza uma resposta máxima do modelo para uma métrica de interesse [4]. Um código que pode servir de ponto de partida é fornecido a seguir. Escolha valores adequados para os parâmetros que determinam os pesos da *Total variation* e *L-p norm* (veja dicas em `https://github.com/raghakot/keras-vis/blob/master/examples/mnist/activation_maximization.ipynb`). Plote os gráficos das imagens obtidas para cada uma das 10 classes e analise o resultado obtido.

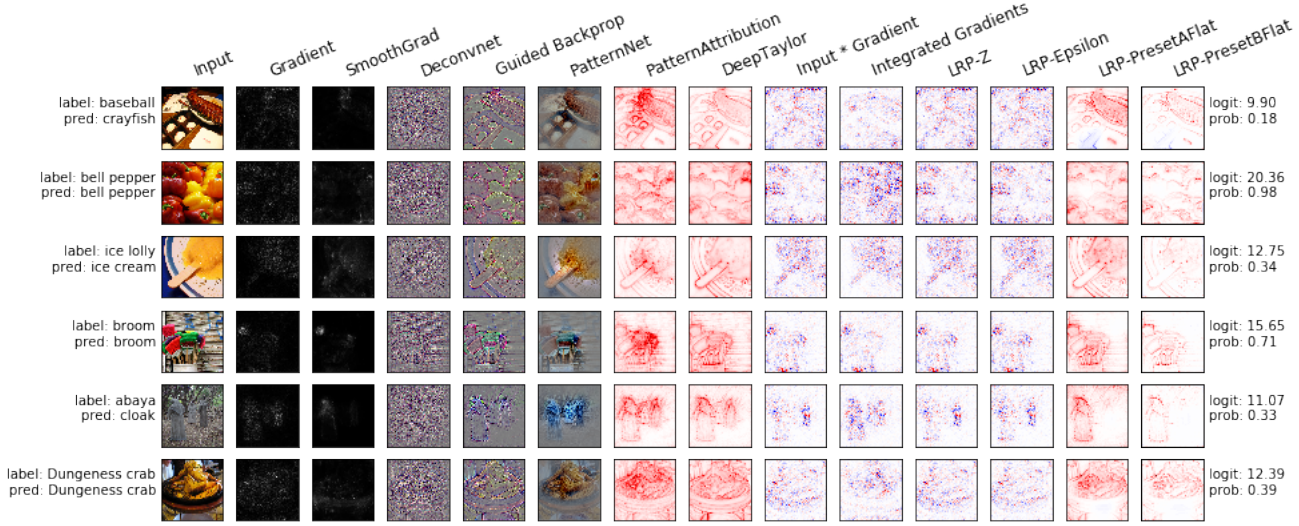

Figura 1: Exemplo de como exibir os resultados da Questão 1a. Fonte: `https://github.com/albermax/innvestigate`.

```
import keras
from vis.visualization import visualize_activation
from vis.utils import utils
import matplotlib.pyplot as plot

model = keras.models.load_model('mnist_model.h5')
layer_idx = utils.find_layer_idx(model, 'dense_2')
model.layers[layer_idx].activation = keras.activations.linear
model = utils.apply_modifications(model)

filter_idx = 9
img = visualize_activation(model, layer_idx,
 filter_indices=filter_idx, input_range=(0., 1.), verbose=True,
 max_iter=1000, tv_weight=1., lp_norm_weight=0.)
plot.imshow(img.squeeze(), cmap='seismic', interpolation='nearest')
```

Questão 2: O treinamento de redes neurais adversárias foi proposto em 2014 [2], no contexto de síntese de modelos gerativos, e foi considerada a ideia mais interessante em aprendizado de máquina da última década. As GANs conseguem conduzir o processo de treinamento a desempenhos nunca antes vislumbrados em aprendizado de máquina, permitindo afirmar que as redes neurais gerativas podem capturar "qualquer" distribuição exibida pelos dados de treinamento, habilidade esta que torna "ilimitado" o potencial de aplicação dessas máquinas de aprendizado.

Assim, esta atividade envolve a execução completa do treinamento de uma GAN. Para isso, vamos utilizar a biblioteca `Keras-GAN` disponível em `https://github.com/eriklindernoren/Keras-GAN`. O objetivo é conseguir executar o exemplo disponível em `https://github.com/eriklindernoren/Keras-GAN/blob/master/gan/gan.py`. O código original executa o treinamento por 30.000 épocas e salva exemplos de imagens a cada 200 épocas. Você pode alterar esses parâmetros conforme a sua necessidade. Reporte os valores utilizados para esses dois parâmetros e

exiba: a primeira imagem obtida, uma imagem obtida no meio do treinamento, a penúltima imagem obtida e a imagem obtida após a última época de treinamento. Para cada imagem, indique a época do treinamento em que foi obtida.

Questão 3: Para fechar o elenco de atividades envolvendo a base de dados MNIST, cabe treinar um autoencoder e constatar o sucesso da tarefa de codificação e de decodificação, indicando que duas variáveis latentes podem ser suficientes para codificar a essência de toda a variedade exibida pelos 10 dígitos manuscritos.

Esta atividade envolve a execução completa de um notebook já proposto na literatura, com pequenas modificações. Os resultados não serão idênticos, pois cada treinamento vai produzir uma solução distinta para o manifold. Realize o treinamento de um autoencoder para os dados do MNIST. O código é fornecido em `https://www.kaggle.com/apapiu/manifold-learning-and-autoencoders` e os dados devem ser baixados de `https://www.kaggle.com/c/digit-recognizer/data`.

A modificação que deve ser feita é a seguinte: na parte **Generating new digits by moving in the latent 2D - space**, o autor do código fez duas células para plotar o manifold: `#moving along the x axis` e `#moving along the y axis`. Unifique essas duas células e explore melhor o intervalo de variação dos eixos x e y, de modo que mais dígitos diferentes sejam plotados.

Questão 4: Esta atividade envolve o processamento de linguagem natural, mais especificamente a síntese de texto empregando redes neurais recorrentes. Deve-se trabalhar com redes já treinadas e também treinar a sua própria rede neural para síntese de textos em português. Evidentemente que não se busca aqui algo próximo ao estado-da-arte, mas resultados convincentes.

Para isso, vamos utilizar a biblioteca `textgenrnn` disponível em `https://github.com/minimaxir/textgenrnn`. O autor da biblioteca nos disponibiliza algumas redes já treinadas. Escolha uma delas e gere frases para diferentes valores do parâmetro `temperature`. Com base nisso, explique a funcionalidade do parâmetro `temperature`. A seguir, treine uma rede recorrente com base em algum texto em português ao nível de palavra (ver parâmetro `word_level`). Livros em português podem ser encontrados em `http://www.gutenberg.org/browse/languages/pt`. Após o treinamento, gere algumas frases de modo iterativo. Um código de exemplo é fornecido a seguir.

```
from textgenrnn import textgenrnn

file_path = "domcasmurro.txt"
textgen.reset()
textgen.train_from_file(file_path, new_model=True, num_epochs=20,
 gen_epochs=10, word_level=True)
textgen.generate(interactive=True, top_n=5)
textgen.save('domcasmurro.hdf5')
```

Questão 5:

Outra frente de grande repercussão em aprendizado de máquina é o aprendizado por reforço, que tem viabilizado o aprendizado de uma ampla gama de tarefas, que vão desde soluções estado-da-arte para jogos de tabuleiro até complexas soluções para robótica autônoma.

Assim, nesta atividade, vamos trabalhar com aprendizagem por reforço com a finalidade de criar um agente capaz de aprender através de tentativa e erro a resolver o *cartpole pro-*

blem. Para isso, vamos utilizar a biblioteca `keras-rl` [5]. O autor disponibiliza em `https://github.com/keras-rl/keras-rl/tree/master/examples` 4 maneiras diferentes de resolver esse problema (*Cross-Entropy Method*, *Deep Q Learning* (DQN), *Duel DQN*, e *State-Action-Reward-State-Action*). Esta atividade envolve a execução desses 4 códigos e a comparação dos resultados obtidos. Para que a comparação seja mais justa, procure treinar as 4 opções utilizando o mesmo tempo computacional.

Referências

[1] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, *innvestigate neural networks!*, arXiv preprint arXiv:1808.04260, (2018).

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in Advances in neural information processing systems, 2014, pp. 2672–2680.

[3] R. Kotikalapudi and contributors, *keras-vis*. `https://github.com/raghakot/keras-vis`, 2017.

[4] G. Montavon, W. Samek, and K.-R. Müller, *Methods for interpreting and understanding deep neural networks*, Digital Signal Processing, 73 (2018), pp. 1–15.

[5] M. Plappert, *keras-rl*. `https://github.com/keras-rl/keras-rl`, 2016.