Deep learning

Conceitos básicos e exemplos de aplicação

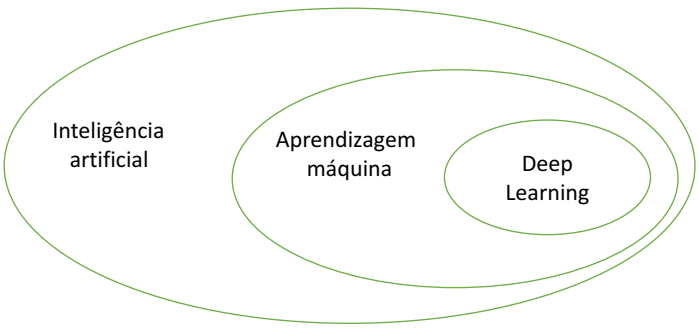
com ***keras*** e ***tensorflow***

Exemplos adaptados de F. Chollet, "Deep Learning with Python"

Deep learning: o que é ?

Área da **Aprendizagem Máquina** caracterizada pela maior complexidade dos modelos e pela capacidade de aprender representações dos dados de entrada

Modelos de deep learning consistem em camadas sucessivas de representações, sendo estas em número tipicamente elevado (deep)



Deep learning: o que é ?

Modelos usados baseiam-se em redes neuronais estruturadas em diversas camadas de processamento

Diversos tipos de neurónios e de arquiteturas usadas para diferentes tipos de problemas: redes feedforward, redes recorrentes, redes convolucionais, etc.

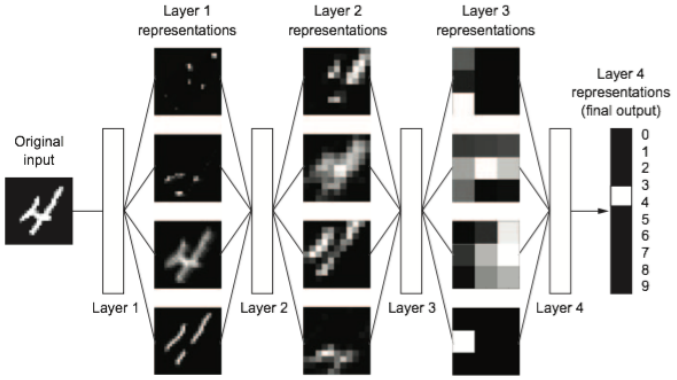
Existem modelos de deep learning para problemas de aprendizagem supervisionada e não supervisionada, bem como aprendizagem por reforço

Deep learning: o que é ?

Diversas camadas processam informação criando representações distintas e tipicamente mais abstratas dos inputs

No caso da aprendizagem supervisionada, a última camada representa o output

Aprendizagem feita por métodos de gradiente descendente



Deep learning: áreas de aplicação / resultados

Os modelos de DL têm sido aplicados em diversos campos com resultados de boa qualidade, incluindo:

- Classificação de imagens (e.g. ImageNet)
- Reconhecimento de textos falados
- Transcrição de texto manuscrito
- Tradução automática de textos
- Respostas em linguagem natural / assistentes digitais
- Jogos (e.g. Go)
- Retrossíntese química
- Classificação de sequências de proteínas e DNA

Deep learning: fatores determinantes

Tal como qualquer tecnologia, o DL não resolve todos os problemas e não será sempre a melhor opção para tarefas de aprendizagem

Um fator determinante para o sucesso de DL é a disponibilidade de dados em larga escala; para problemas com poucos dados, outros modelos podem dar resultados mais consistentes com menor esforço computacional

Em termos de hardware, o uso de processadores gráficos (GPU) traz vantagens no treino de modelos de DL acelerando o processo por fatores de mais do que 10x

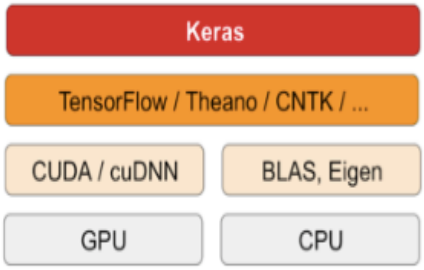
Melhorias em relação às redes neuronais “shallow”: funções ativação (RelU), inicialização dos pesos, algoritmos de otimização (RMSprop, Adam), métodos para abordar overfitting, pré-treino e treino “por camadas”

Implementação em Python: keras e tensorflow

Para implementar os modelos de DL em python iremos usar o package Keras: este permite criar, treinar e aplicar diversos modelos distintos de DL

O **keras** contém interfaces que lhe permitem usar diferentes "backends" implementando as diversas arquitecturas de DL e os seus algoritmos de treino e previsão; permite correr em CPU ou GPU dependendo do h/w da máquina

Por omissão, o keras usa o **tensorflow** para representar e treinar os modelos de DL e será este o "backend engine" que iremos utilizar.



Conjunto de dados MNIST

Como primeiro exemplo, vamos usar o conjunto de dados de reconhecimento de dígitos (MNIST) já usado noutras aulas

Inputs: imagens de 28 x 28 pixels

Output: classe representando o dígito (10 classes, dígitos 0-9)

Disponível como dataset do keras

60k imagens de treino e 10k imagens de teste

Conjunto de dados MNIST

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
        mnist.load_data()
print(train_images.shape, test_images.shape)
print(len(train_labels), len(test_labels))
```

Carregar os dados

Verificar dimensões

```
from keras.utils import to_categorical
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Ajustar as dimensões para tornar as entradas num vetor 1D por cada exemplo
Standardizar valores dividindo por 255
Converter outputs em variáveis categóricas

Deep neural networks (DNNs)

As DNNs são modelos de DL supervisionados, constando de redes neuronais **feedforward** podendo ter diversas camadas intermédias

Neurónios tipicamente usam função de ativação RelU ou sigmoid

Algoritmos de treino usados baseados em gradiente descendente: stochastic GD, RMSProp, Adam, etc

Treino feito em lotes (batches)

DNNs para o MNIST

Definir a arquitetura da rede; feed forward

```
from keras import models
from keras import layers
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Uma camada intermédia com 512 neurónios (RelU) e uma de saída com 10 neurónios (*softmax* - 1 neurónio sigmoid para cada saída; one-hot encoding – 1-of-C)

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Definir algoritmo de treino (RMSprop), *loss function* (*cross entropy*) e métricas de erro (*accuracy*)
Loss function – semelhante à que definimos para a regressão logística para várias classes

DNNs para o MNIST

Treinar o modelo (fit):
Define tamanho de cada batch e nº de iterações (epochs)

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Avaliação no conjunto de dados de teste

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

Conjunto de dados IMDB

Conjunto de dados constando de textos de reviews do IMDB sobre filmes classificados em duas classes: positivos ou negativos

25k reviews de treino + 25k para teste; balanceado – 50% de exemplos positivos e negativos

Incluído no keras

Apenas consideradas as 10k palavras mais comuns

```
(train_data, train_labels), (test_data, test_labels) =
        imdb.load_data(num_words=10000)

print(train_data.shape, test_data.shape)
print(len(train_labels), len(test_labels))
```

Carregar os dados

Verificar as dimensões

Conjunto de dados IMDB

Exploração dos dados

```
print(train_data[0])
print(train_labels[0])

print(max([max(sequence) for sequence in train_data]))

word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[2]])
print(decoded_review)
```

Apenas são consideradas 10000 palavras

Forma de aceder ao texto original

Conjunto de dados IMDB: pré-processamento

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

One-hot encoding dos inputs: Cada review representado pelas palavras que contém ignorando a ordem

DNN para IMDB

DNN – 2 camadas intermédias 16 neurónios em cada; RelU
Camada de saída sigmoid

```
from keras import models, layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Loss function – a que definimos para a reg. logistica

DNN para IMDB

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Definir um validation set

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

Treino da rede

```
results = model.evaluate(x_test, y_test)
print(results)
print(model.predict(x_test[0:3]))
```

Avaliar no test set
Prever para um novo caso

DNN para IMDB

```
history_dict = history.history
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
epochs = range(1, len(acc_values) + 1)
```

Plot dos erros (accuracy)
Também se pode fazer para loss

```
import matplotlib.pyplot as plt
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Um exemplo de regressão: boston housing

Previsão do preço médio de casas em Boston (anos 1970)

Inputs: várias variáveis caracterizando o imóvel

404 casos de treino; 102 de teste

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) =
        boston_housing.load_data()

print(train_data.shape)
print(test_data.shape)
print(max(train_targets), min(train_targets))
```

Boston housing: preparação dos dados

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

Standardizar inputs

Boston housing: DNN

```
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

Função para criar o modelo
Duas camadas intermédias com 64 neurónios; RelU
Camada de saída com um neurónio; ativação linear
Loss function: média do quadrado dos erros
Métrica: média dos erros absolutos

Boston housing: validação cruzada

```
import numpy as np
k = 5
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i*num_val_samples: (i+1)*num_val_samples]
    val_targets = train_targets[i*num_val_samples: (i+1)*num_val_samples]
    partial_train_data = np.concatenate( [train_data[:i*num_val_samples],
                                          train_data[(i+1)*num_val_samples:]], axis=0)
    partial_train_targets = np.concatenate( [train_targets[:i*num_val_samples],
                                          train_targets[(i+1)*num_val_samples:]], axis=0)
    model = build_model()
    model.fit(partial_train_data, partial_train_targets, epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
print(all_scores)
print(np.mean(all_scores))
```

Boston housing: modelo final

Construir o modelo final com todos os dados

```
model = build_model()
model.fit(train_data, train_targets, epochs=80, batch_size=16, verbose=1)


test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
print(test_mse_score, test_mae_score)
```

Avaliar no test set

Overfitting em modelos de DL

Tal como acontece com todas as abordagens de aprendizagem supervisionada, os modelos de DL sofrem de problemas de overfitting; se houver muitos dados, estes problemas podem ser amenizados

O overfitting pode ser resolvido recorrendo a diversas técnicas, como a **regularização** (L1 ou L2, semelhante à regressão linear/logística, chamado de decay nas DNNs), ou o controlo explícito da **capacidade** dos modelos (e.g. número de camadas intermédias e número de neurónios em cada)

Uma técnica específica desenvolvida para modelos de DL é o **dropout**: em cada iteração do treino a saída de alguns neurónios da camada, selecionados aleatoriamente, é colocada a zero. A proporção de neurónios anulados em cada iteração é um parâmetro definido por cada camada onde se aplicar o dropout

Uma outra técnica consta de parar o treino (*early stopping*) com base no erro num conjunto de validação (não usado no cálculo do gradiente)

Overfitting: exemplo IMDB

```
hidden = 4  ## testar com 4, 16, 64
model = models.Sequential()
model.add(layers.Dense(hidden, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(hidden, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Controlar capacidade do modelo

Overfitting: exemplo IMDB

Regularização

```
from keras import regularizers
model.add(layers.Dense(hidden, activation='relu', kernel_regularizer=regularizers.l2(0.001),
                       input_shape=(10000,)))
model.add(layers.Dense(hidden, activation='relu', kernel_regularizer=regularizers.l2(0.001),
                       input_shape=(10000,)))
model.add(layers.Dense(1, activation='sigmoid'))
```

Alterar para L1 ou L1 e L2:
regularizers.l1(0.001)
regularizers.l1_l2(l1=0.001, l2=0.001)

```
model = models.Sequential()
model.add(layers.Dense(hidden, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(hidden, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation='sigmoid'))
```

Dropout

Overfitting: exemplo IMDB

Early stopping

```
from keras.callbacks import EarlyStopping

early = EarlyStopping(monitor='val_loss', min_delta=0,
                      patience= 5,
                      verbose= True, mode='auto')

callbacks = [early]
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val),
                    callbacks = callbacks)
```

Patience – indica nº de epochs em que erro na validação pode não melhorar

Monitor – indica a métrica a usar

Otimização de hiper-parâmetros

Tal como acontece com todas as abordagens de aprendizagem supervisionada, os modelos de DL deverão ser alvo de um processo de otimização de hiper-parâmetros para garantir o seu melhor desempenho

Neste caso, os hiper-parâmetros mais relevantes incluem:

- Número de camadas
- Nº de neurónios em cada camada
- Funções de ativação a usar
- Algoritmo de treino e seus parâmetros
- Uso de dropout e sua taxa
- Uso de regularização e parâmetros associados
- Uso de paragem antecipada do treino
- …

Exercício

Implementar um processo de otimização de hiper-parâmetros por grid search. A função deve receber um dicionário onde as chaves sejam os nomes dos parâmetros a otimizar (e.g. “train_algorithm”, “activation_function”, “topology”, etc), bem como o conjunto de dados. Este deve ser dividido em duas partes: treino e validação.

Para cada combinação possível de valores para os parâmetros, deverá treinar a rede e calcular a métrica de erro nos exemplos de validação.

Poderá retornar o melhor modelo e uma tabela com os erros associados aos modelos testados