Deep learning

Redes recorrentes, processamento de dados sequenciais, embeddings

Conceitos básicos e exemplos de aplicação

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

Exemplos/ figuras adaptados de F. Chollet, "Deep Learning with Python" e Andrew Ng (deeplearning.ai)

Processamento de sequências e redes recorrentes

Uma área de sucesso do DL passa pela capacidade de processar dados sequenciais, em particular textuais e audio, bem como séries temporais, com bom desempenho

Os tipos de arquitetura mais usados nestas aplicações são as redes neuronais recorrentes com diversas configurações

Aplicações incluem:

- classificação de documentos; identificação documentos relevantes
- tradução automática
- análise de sentimentos
- identificação de entidades em textos (e.g. nomes próprios, genes, …) e de relações entre estas
- previsão de séries temporais

Processamento de textos

Tipicamente encaramos textos como sequências de palavras

Textos são convertidos em vetores numéricos, onde as palavras (ou outros tipos de *tokens*) são codificadas como índices de um vocabulário que tipicamente tem um tamanho máximo pré-definido

Representação mais usada para palavras: ***One-hot encoding***- cada palavra tem um índice inteiro; representação de uma palavra é um vetor com um valor de 1 nesse índice e zeros nos restantes

One-hot encoding: exemplo

V = [a, aaron, …, zulu, <UNK>] **Vocabulário**

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
O_{5391}	O_{9853}	O_{4914}	O_{7157}	O_{456}	O_{6257}

$|V| = 10000$

Andrew Ng

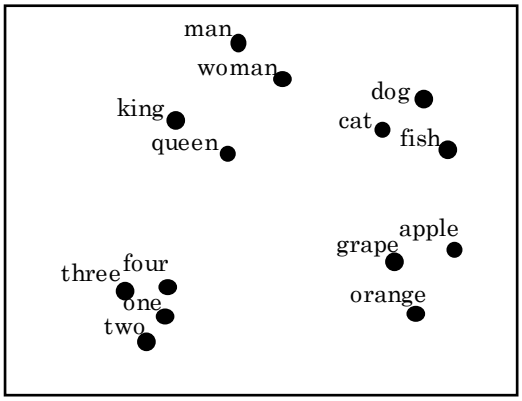
Word embeddings

Cada palavra é representada por um vetor numérico "denso" de dimensão relativamente pequena (e.g. 256-1024)

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1.0	1.0	-0.95	0.97	0.00	0.01
Royal	0.02	0.01	0.93	0.95	-0.01	0.00
Age	0.41	0.38	0.7	0.69	0.03	-0.02
Food	0.01	0.01	0.02	0.01	0.95	0.97

Andrew Ng

Visualização de word embeddings/ analogias



$$e_{man} - e_{woman} \approx e_{king} - e_?$$

MAX:

$$sim(e_w, e_{king} - e_{man} + e_{woman})$$

t-SNE Permite representar os embeddings em 2D mantendo distâncias

[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Andrew Ng

Aprendizagem dos embeddings

Embedding – matriz E:

- linhas: dimensões do embedding
- colunas: palavras do vocabulário

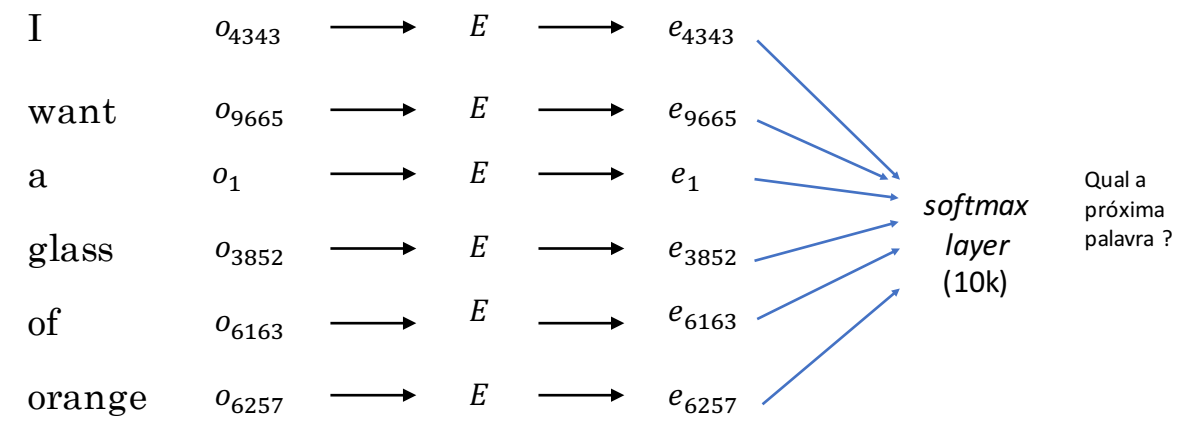
Note-se que ao multiplicar a representação de uma palavra como *one-hot encoding* por E temos a coluna que corresponde ao vetor do embedding dessa mesma palavra

Valores desta matriz E podem ser aprendidos como pesos em qualquer rede neuronal por gradiente descendente, sendo matriz E a primeira camada, tendo por inputs uma ou várias palavras e podendo ter qualquer tipo de tarefa nas camadas subsequentes

Aprendizagem dos embeddings: exemplo

I want a glass of orange _____.

4343 9665 1 3852 6163 6257



[Bengio et. al., 2003, A neural probabilistic language model]

Embeddings: transfer learning

Os *embeddings* podem ser aprendidos "de raiz" para cada problema (se houver dados suficientes)

Em muitos casos, é preferível re-utilizar embeddings pré-treinados em conjuntos de dados alargados com vocabulários grandes – *transfer learning*

Se tivermos dados suficientes, podemos sempre refinar os embeddings com os dados disponíveis

Embeddings mais usados: Word2Vec, Glove

Exemplo Embedding - IMDB

```
from keras.datasets import imdb
from keras import preprocessing

max_features = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train,  maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test,  maxlen=maxlen)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
print(x_train[0])
print(y_train[0])
```

Exemplo Embedding - IMDB

```
model = Sequential()
model.add(Embedding(10000,  8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1,  activation='sigmoid'))

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

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_2 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_82 (Dense)             (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
```

Matriz E:

- Vocabulário: 10 k
- Tamanho embedding: 8

Camada *embedding*:

- Input: samples x maxlen (20)
- Output: samples x maxlen x 8

Exemplo Embedding - IMDB

```
history = model.fit(x_train, y_train, epochs=10,
                    batch_size=32, validation_split=0.2)

results = model.evaluate(x_test, y_test)
print(results)
```

Este modelo não tem em consideração a sequência das palavras, tratando-as de forma independente !

Podem usar-se embeddings pré-treinados: por exemplo Word2vec ou Glove – ver exemplo: https://github.com/fchollet/deep-learning-with-python-notebooks - exemplo 6.1 - using word embeddings

Redes neuronais recorrentes

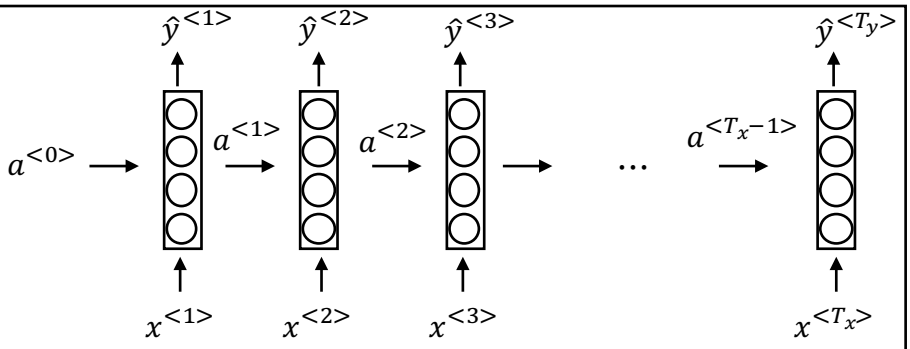
Ao contrário das redes *feedforward*, as redes recorrentes têm uma topologia que pode conter ciclos, havendo realimentação das saídas da rede nas entradas

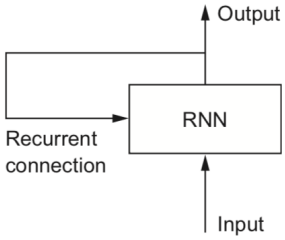
Assim, têm capacidade para processar dados sequenciais, podendo manter e podendo ter vários tipos de estado / "memória" (de "curto" e "longo" prazo)

Por exemplo, no caso do IMDB, cada review pode ser processado sequencialmente sendo mantido um estado relativo às palavras já vistas (ainda que cada review fosse analisado independentemente)

Algoritmos de treino baseados no gradiente descendente (e.g Backprop through time) propagam erros através da "chain rule" desde o último estado até ao estado inicial

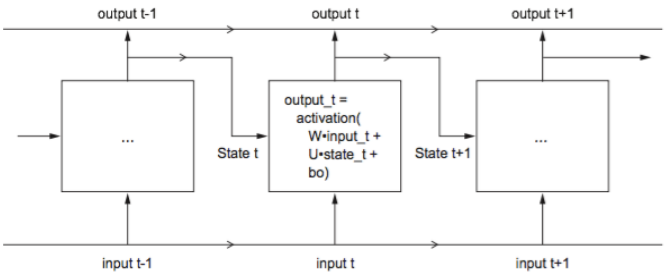
Redes neuronais recorrentes





Cálculo das saídas e dos estados →

← Propagação do erro



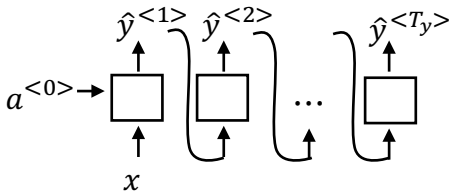
Exemplo de uma rede recorrente

Layer ***SimpleRNN*** no keras

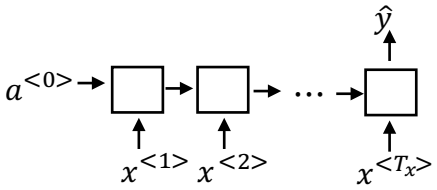
RNN em numpy

```
import numpy as np

timesteps = 100
input_features = 32
output_features = 64

inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features)) * 0.05
U = np.random.random((output_features, output_features)) * 0.05
b = np.random.random((output_features,)) * 0.05

successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t)
    state_t = output_t
final_output_sequence = np.concatenate(successive_outputs, axis=0)
print(final_output_sequence)
```

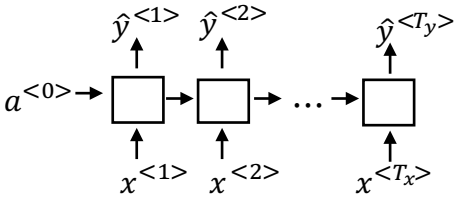
Vários tipos de RNNs



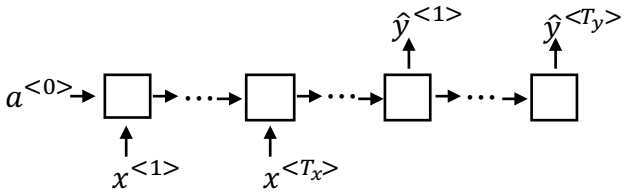
One to many



Many to one



Many to many



Many to many

Andrew Ng

Exemplo RNN- IMDB

```
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 10000
maxlen = 500
batch_size = 32

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences and ', len(input_test), 'test sequences')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

print(x_train.shape, x_test.shape)
```

Exemplo RNN- IMDB

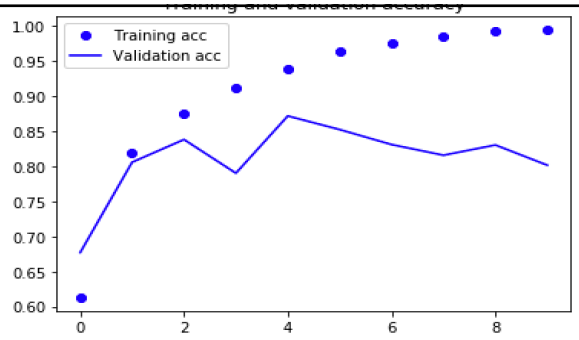
Layer Many to One
Só dá output no final
Flag "return_sequences"
Usada para Many to Many

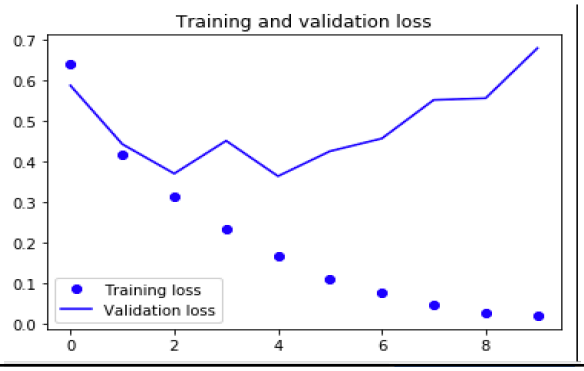
```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.suymmary()

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

history = model.fit(input_train, y_train,
                    epochs=10, batch_size=128,
                    validation_split=0.2)
```

Exemplo
RNN- IMDB

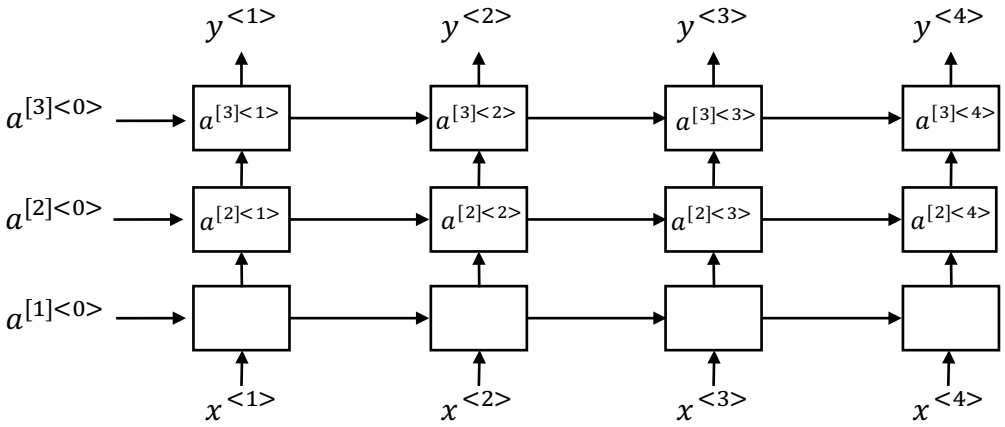
Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, None, 32)	320000
simple_rnn_2 (SimpleRNN)	(None, 32)	2080
dense_5 (Dense)	(None, 1)	33

Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0





RNNs com várias camadas



Andrew Ng

Exemplo RNN-IMDB Várias camadas

Flag "return_sequences" usada para passar outputs Para outras camadas

```
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32, return_sequences = True))
model.add(SimpleRNN(32, return_sequences = True))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.suymmary()

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

history = model.fit(input_train, y_train,
                    epochs=10, batch_size=128,
                    validation_split=0.2)
```

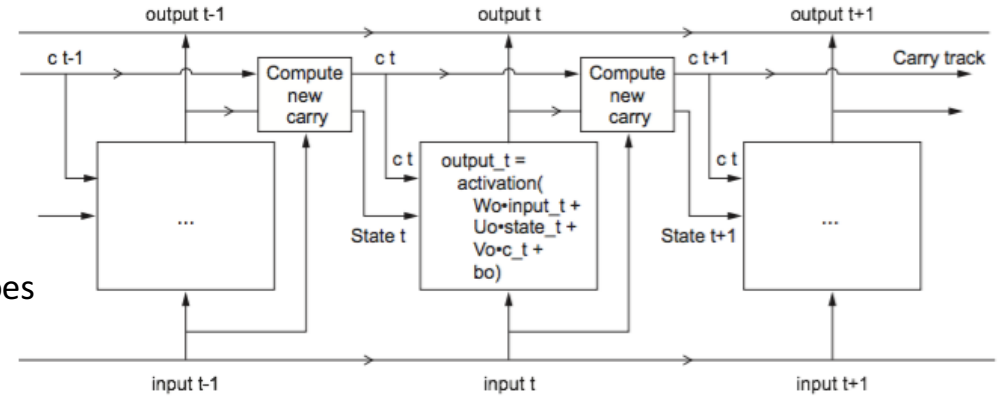
LSTMs

As redes LSTM – Long Short Term Memory – são redes recorrentes que se caracterizam por manterem "memória"

Foram introduzidas para resolver o chamado problemas dos "vanishing gradients"

c_t – *carry*

Calculado a a partir de 3 transformações estilo RNN



Exemplo LSTM - IMDB

```
from keras.datasets import imdb
from keras.preprocessing import sequence

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = 10000)
input_train = sequence.pad_sequences(x_train, maxlen=500)
input_test = sequence.pad_sequences(x_test, maxlen=500)

print(input_train.shape)
print(input_test.shape)
```

Exemplo LSTM - IMDB

```
model = models.Sequential()
model.add(Embedding(10000, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

model.summary()
```

Camada LSTM pode ser substituída por GRU:

from keras.layers import GRU

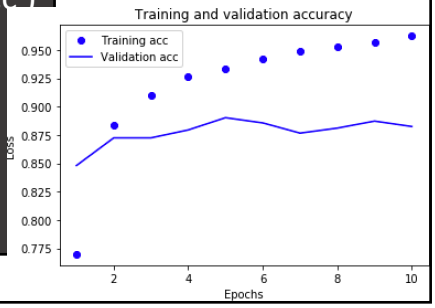
...
model.add(GRU(32))
...

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense_84 (Dense)             (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
```

Exemplo LSTM - IMDB

```
history = model.fit(input_train, y_train,
                    epochs=4,
                    batch_size=128,
                    validation_split=0.2)

res = model.evaluate(input_test, y_test)
print(res)
```

```
import matplotlib.pyplot as plt
plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
epochs = range(1, len(acc_values) + 1)
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('Loss')
plt.legend()
plt.show()
```



Redes recorrentes: notas complementares

As camadas LSTM podem ser substituídas por camadas GRU com funcionalidade similar (mais simples – menos pesos para treinar)

As camadas GRU ou LSTM podem ser empilhadas criando modelos com maior capacidade, da mesma forma que fizemos para as camadas SimpleRNN

Pode considerar-se o uso de Dropout nestas camadas para abordar o overfitting

Podem usar-se redes recorrentes bidirecionais nos casos onde quer a ordem natural das sequências, quer o seu inverso possam fazer sentido – layer Bidirectional do Keras

Redes recorrentes: outros exemplos

As redes LSTM são bastante utilizadas em cenários de previsão de séries temporais (ver notebook do exemplo 6.3)

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb

Uma alternativa às redes recorrentes para processar sequências são as redes convolucionais 1D (ver notebook do exemplo 6.4), podendo os dois modelos combinar-se

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.4-sequence-processing-with-convnets.ipynb