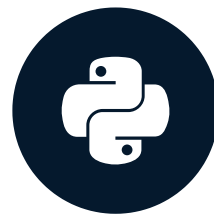


# Sequence to Sequence Models

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



**David Cecchini**  
Instructor

# Sequence to sequence

Possible architectures:

- Many inputs with one output
  - Sentiment analysis
  - Classification
- Many inputs to many outputs
  - Text generation
  - Neural Machine Translation (NMT)

# Text generation: example

Text generation: example

```
# Pre-trained model  
model.generate_sheldon_phrase()
```

```
'knock knock. penny. do you have an epost is part in your expert,  
too bealie to play the tariment with last night.'
```

# Text generation: modeling

How to build text generation models:

- Decide if a token will be characters or words
  - Words demands very large datasets (hundred of millions sentences)
  - Chars can be trained faster, but can generate typos
- Prepare the data
  - Build training sample with (past tokens, next token) examples
- Design the model architecture
  - Embedding layer, number of layers, etc.
- Train and experiment

# NMT: example

Neural Machine Translation: example

```
# Pre-trained model  
model.translate("Vamos jugar futbol?")
```

```
'Let's go play soccer?'
```

# NMT: modeling

How to build NMT models:

- Get a sample of translated sentences
  - For example, the **Anki project**
- Prepare the data
  - Tokenize input language sentences
  - Tokenize output language sentences
- Design the model architecture
  - Encoder and decoder
- Train and experiment

# Chapter outline

In this chapter:

- Text Generation
  - Use pre-trained model to generate a sentence
  - Learn to prepare the data and build the model
- Neural Machine Translation (NMT)
  - All-in-one NMT model

# Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



# The Text Generating Function

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



**David Cecchini**  
Data Scientist

# Generating sentences

- Sentence is determined by punctuation. For example, . (period), ! (exclamation) or ? (question).
  - The punctuation marks need to be in the vocabulary.
- There is a sentence token, e.g. <SENT> and </SENT>, that determines when a sentence begins and ends.
  - Need to pre-process the data to insert the labels.

# Generating sentences

```
sentence = ''  
# Loop until end of sentence  
while next_char != '.':  
    # Predict next char: Get pred array in position 0  
    pred = model.predict(X)[0]  
    char_index = np.argmax(pred)  
    next_char = index_to_char(char_index)  
    # Concatenate to sentence  
    sentence = sentence + next_char
```

# Probability scaling

Scale the probability distribution.

- **Temperature:** name from physics
  - *Small values:* makes prediction more confident
  - *Value equal to one:* no scaling
  - *higher values:* makes prediction more creative
  - Hyper-parameter: Try different values to fit the predictions to your need

# Probability scaling

```
def scale_softmax(softmax_pred, temperature=1.0):  
    # Take the logarithm  
    scaled_pred = np.log(softmax_pred) / temperature  
    # Re-apply the exponential  
    scaled_pred = np.exp(scaled_pred)  
    # Build probability distribution  
    scaled_pred = scaled_pred / np.sum(scaled_pred)  
    # Simulate multinomial  
    scaled_pred = np.random.multinomial(1, scaled_pred, 1)  
    # Return simulated class  
    return np.argmax(scaled_pred)
```

# Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

# Text Generation Models

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



**David Cecchini**  
Data Scientist

# Similar to a classification model

The Text Generation Model:

- Uses the vocabulary as classes
- The last layer applies a softmax with vocabulary size units
- Uses `categorical_crossentropy` as loss function



# Example model using keras

```
model = Sequential()
model.add(LSTM(units, input_shape=(chars_window, n_vocab),
              dropout=0.15, recurrent_dropout=0.15, return_sequences=True))
model.add(LSTM(units, dropout=dropout, recurrent_dropout=0.15,
              return_sequences=False))
model.add(Dense(n_vocab, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

# But not really classification model

Difference to classification:

- Computes loss, but not performance metrics (accuracy)
  - Humans see results and evaluate performance.
  - If not good, train more epochs or add complexity to the model (add more memory cells, add layers, etc.).
- Used with generation rules according to task
  - Generate next char
  - Generate one word
  - Generate one sentence
  - Generate one paragraph

# Other applications

- Name creation
  - Baby names
  - New star names, etc.
- Generate marked text
  - LaTeX
  - Markdown
  - XML, etc.
  - Programming code
- News articles
- Chatbots

# Data prep

I am not insane,  
my mother had  
me tested

step = 1



| Sentences | Next char |
|-----------|-----------|
| I         | \b        |
| I\b       | a         |
| I a       | m         |
| I am      | \b        |



chars\_window = 10  
vocabulary = 5

| X                     | Y           |
|-----------------------|-------------|
| [0 0 0 0 1 0 0 0 0 0] | [0 1 0 0 0] |
| [0 1 0 0 1 0 0 0 0 0] | [1 0 0 0 0] |
| [0 1 1 0 1 0 0 0 0 0] | [0 0 0 0 1] |
| [0 1 1 0 1 0 0 1 0 0] | [0 1 0 0 0] |

# Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

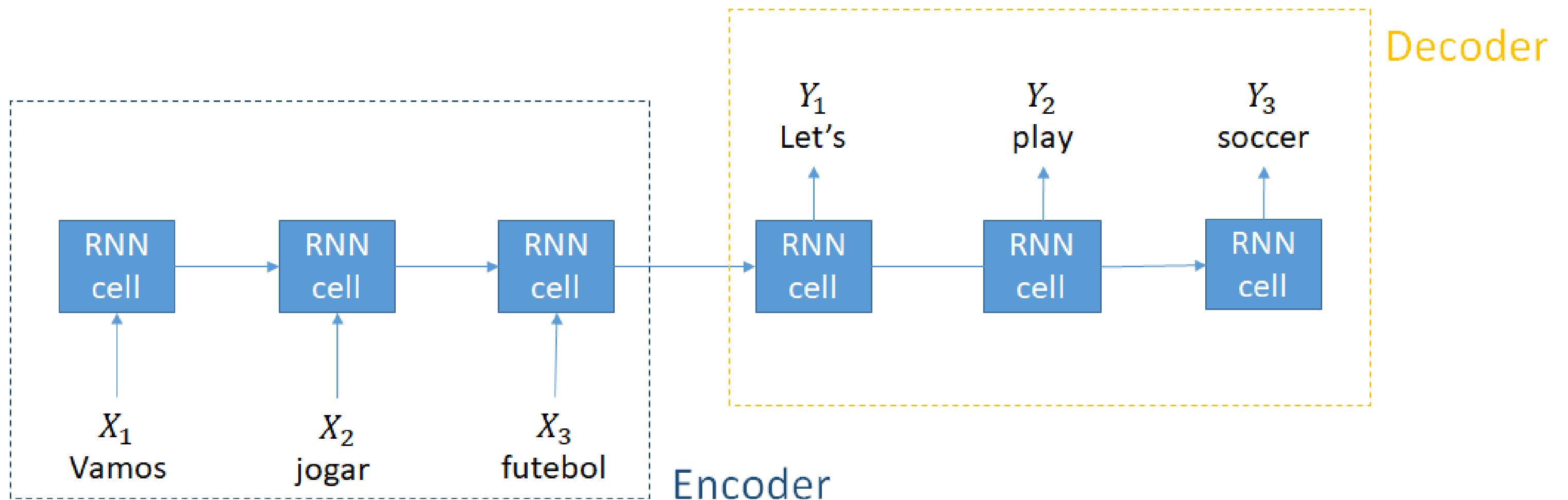
# Neural Machine Translation

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



**David Cecchini**  
Data Scientist

# Encoder and decoders



# Encoder example

```
# Instantiate the model
model = Sequential()

# Embedding layer for input language
model.add(Embedding(input_language_size, input_wordvec_dim,
                    input_length=input_language_len, mask_zero=True))

# Add LSTM layer
model.add(LSTM(128))

# Repeat the last vector
model.add(RepeatVector(output_language_len))
```



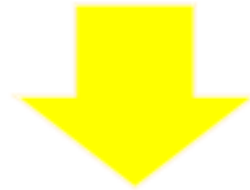
# Decoder example

```
# Right after the encoder
model.add(LSTM(128, return_sequences=True))

# Add Time Distributed
model.add(TimeDistributed(Dense(eng_vocab_size, activation='softmax')))
```

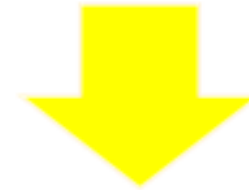
# Data prep

Vamos jogar futebol esse domingo



[2, 5, 12, 10, 15]

Let's play soccer this Sunday



[11, 3, 15, 10, 7]

Vocab size  
= 15



|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Data preparation for the input language

```
# Import modules
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
# Use the Tokenizer class
tokenizer = Tokenizer()
tokenizer.fit_on_texts(input_texts_list)
# Text to sequence of numerical indexes
X = tokenizer.texts_to_sequences(input_texts_list)
# Pad sequences
X = pad_sequences(X, maxlen=length, padding='post')
```

# Tokenize the output language

```
# Use the Tokenizer class
tokenizer = Tokenizer()
tokenizer.fit_on_texts(output_texts_list)
# Text to sequence of numerical indexes
Y = tokenizer.texts_to_sequences(output_texts_list)
# Pad sequences
Y = pad_sequences(Y, maxlen=length, padding='post')
```

# One-hot encode the output language

```
# Instantiate a temporary variable
ylist = list()

# Loop over the sequence of numerical indexes
for sequence in Y:
    # One-hot encode each index on current sentence
    encoded = to_categorical(sequence, num_classes=vocab_size)
    # Append one-hot encoded values to the list
    ylist.append(encoded)

# Transform to np.array and reshape
Y = np.array(ylist).reshape(Y.shape[0], Y.shape[1], vocab_size)
```

# Note on training and evaluating

Training the model:

```
model.fit(X, Y, epochs=N)
```

Evaluating:

- Use BLEU
  - `nltk.translate.bleu_score`

# Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

# Congratulations!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS



**David Cecchini**  
Data Scientist



# Wrap-up

- Introduction to language tasks:
  - Sentiment classification
  - Multi-class classification
  - Text Generation
  - Neural Machine Translation
- Sequence to sequence models
- Implementation in Keras

# RNN pitfalls and different cell types

- Vanishing and exploding gradient problems
- GRU and LSTM cells
- Word vectors and the Embedding layer
- Better sentiment analysis

# Multi-class classification

- Data preparation
- Transfer learning
- Keras models
- Model performance

# Text generation and NMT

- Text Generation
  - Chars as token
  - Data preparation
  - Generate sentences mimicking Sheldon
- Neural Machine Translation
  - Words as tokens
  - Data preparation: encoders and decoders
  - Translate Portuguese to English

# Congratulations!!!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS