Recurrent Neural Networks

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Recurrent Neural Networks

- Capable of handling sequences of data
  - text, audio, video
- Unlike the previous architectures (MLP, CNN), the output of an RNN at time t is fed to the same network at time t+1 as part of the input

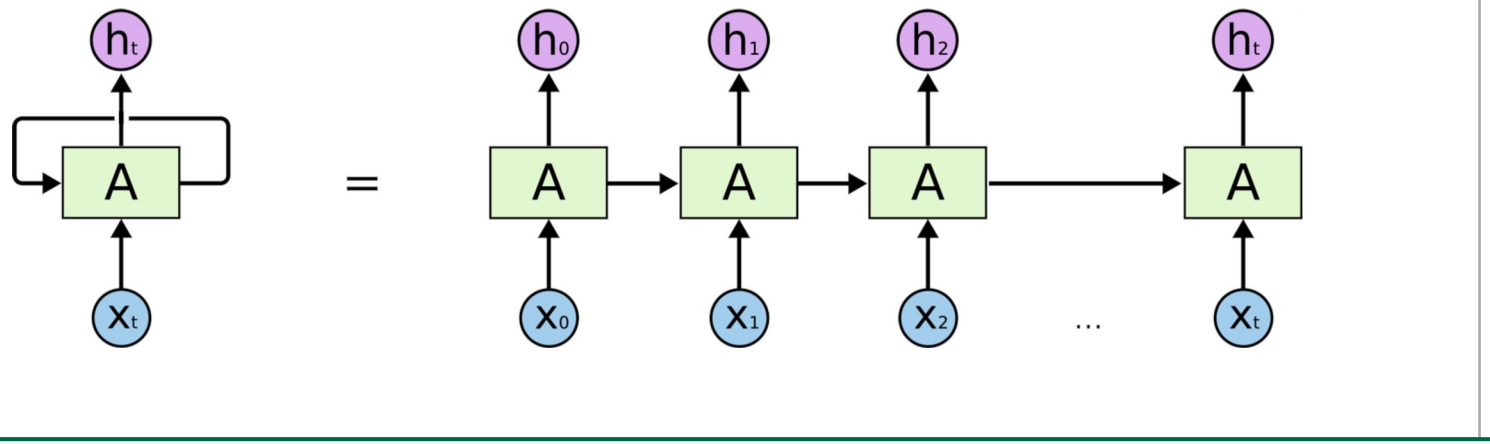


Figure 1. Illustration of a basic RNN architecture.

RNN Applications

One to one: object classification (image → label) **-- not an RNN**

- One to many: image captioning (image → sequence of words)
- Many to one: sentiment classification (sequence of words → label)
- Many to many: machine translation (sequence of words → sequence of words)

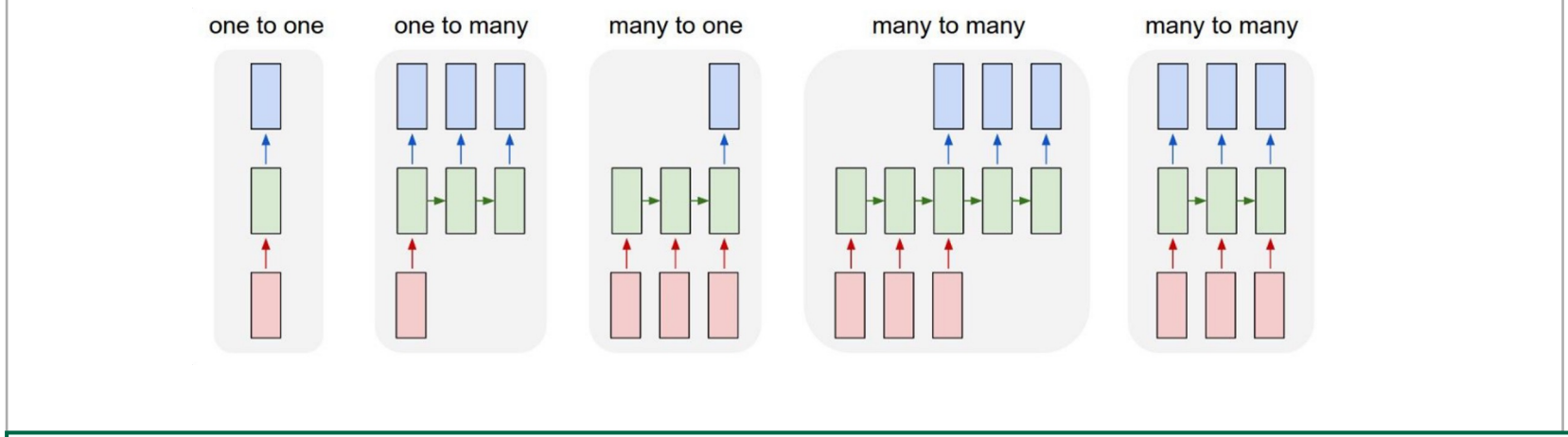


Figure 2. Different RNN applications.

Knowledge Check 1

Which of these tasks requires a many-to-many RNN?

A. Image description (input an image and output text description)

B. Sentiment classification (input a sentence and output sentiment label)

C. Speech recognition (input an audio clip and output a transcript)

D. Image classification (input an image and output class label)

How RNNs Work?

- The same layer is used over time
  - The same set of weights is used for every step of the sequence
- Maintain a hidden state to keep track of the content that was previously seen
  - The hidden state of the last step of the sequence encodes knowledge of the entire sequence
  - In a many-to-one application, this hidden state is then used to achieve the goal (e.g. classification).

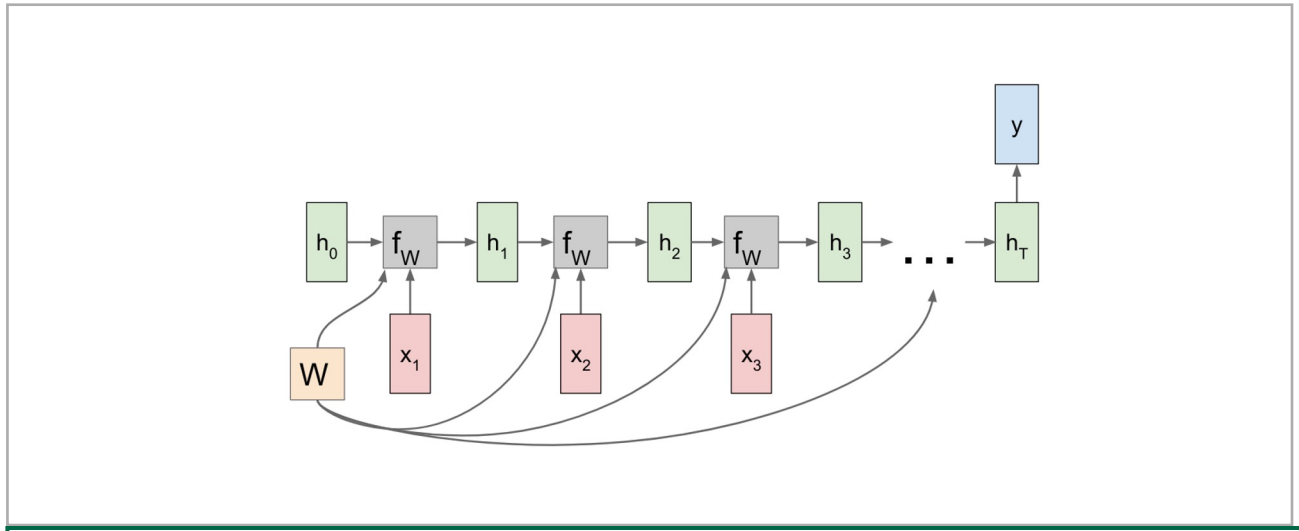
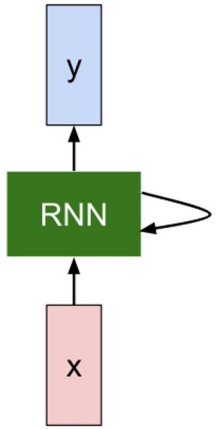


Figure 3. Basic RNN in detail.

How RNNs Work?

- We can model this hidden state update with two dense layers
  - One to decide how much information we should keep from the previous hidden state
  - Another to decide which parts from the current input should be saved



$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Figure 4. Implementation details of a basic RNN.

RNN in Keras

https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN

```
tf.keras.layers.SimpleRNN(
                    units,
                    activation='tanh',
                    use_bias=True,
                    kernel_initializer='glorot_uniform',
                    recurrent_initializer='orthogonal',
                    bias_initializer='zeros',
                    kernel_regularizer=None,
                    recurrent_regularizer=None,
                    bias_regularizer=None,
                    activity_regularizer=None,
                    kernel_constraint=None,
                    recurrent_constraint=None,
                    bias_constraint=None,
                    dropout=0.0,
                    recurrent_dropout=0.0,
                    return_sequences=False,
                    return_state=False,
                    go_backwards=False,
                    stateful=False,
                    unroll=False,
                    **kwargs
)
```

Args	
`units`	Positive integer, dimensionality of the output space.
`return_sequences`	Boolean. Whether to return the last output in the output sequence, or the full sequence. Default: `False`.

Call arguments	
`inputs`	A 3D tensor, with shape `[batch, timesteps, feature]`.
`mask`	Binary tensor of shape `[batch, timesteps]` indicating whether a given timestep should be masked. An individual `True` entry indicates that the corresponding timestep should be utilized, while a `False` entry indicates that the corresponding timestep should be ignored.
`training`	Python boolean indicating whether the layer should behave in training mode or in inference mode. This argument is passed to the cell when calling it. This is only relevant if `dropout` or `recurrent_dropout` is used.
`initial_state`	List of initial state tensors to be passed to the first call of the cell.

Figure 5. RNN in Keras.

How to Use Text with Neural Networks?

- Convert words to numbers
  - Preprocess words
  - Give a unique identification number to each word in a vocabulary

['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it',
'i', 'this', 'that', 'br', 'was', 'as', 'with', 'for', 'but', 'movie',
'film', 'on', 'not', 'his', 'you', 'are', 'have', 'be', 'he', 'one',
'its', 'at', 'all', 'by', 'they', 'an', 'from', 'who', 'so', 'like',
'her', 'or', 'just', 'if', 'about', 'out', 'has', 'what', 'some',
'there',
'good', 'more', 'very', 'when', 'my', 'she', 'no', 'even',
'would', 'up',
'time', 'which', 'really', 'only', 'had', 'were', 'see', 'their',
'can', 'story',
'me', 'than', 'get', 'much', 'we', 'into', 'been', 'first',
'also', 'will',
…]

- Convert sentences to sequences of numbers

"This was an absolutely terrible movie."

[11, 14, 35, 436, 389, 19]

TextVectorization in Keras

https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization

```
tf.keras.layers.TextVectorization(
        max_tokens=None,
        standardize='lower_and_strip_punctuation',
        split='whitespace',
        ngrams=None,
        output_mode='int',
        output_sequence_length=None,
        pad_to_max_tokens=False,
        vocabulary=None,
        idf_weights=None,
        sparse=False,
        ragged=False,
        **kwargs
)
```

Args

`max_tokens`	Maximum size of the vocabulary for this layer. This should only be specified when adapting a vocabulary or when setting `pad_to_max_tokens=True`. Note that this vocabulary contains 1 OOV token, so the effective number of tokens is `(max_tokens - 1 - (1 if output_mode == "int" else 0))`.
`standardize`	Optional specification for standardization to apply to the input text. Values can be: • `None`: No standardization. • `"lower_and_strip_punctuation"`: Text will be lowercased and all punctuation removed. • `"lower"`: Text will be lowercased. • `"strip_punctuation"`: All punctuation will be removed. • Callable: Inputs will passed to the callable function, which should standardized and returned.
`split`	Optional specification for splitting the input text. Values can be: • `None`: No splitting. • `"whitespace"`: Split on whitespace. • `"character"`: Split on each unicode character. • Callable: Standardized inputs will passed to the callable function, which should split and returned.
`ngrams`	Optional specification for ngrams to create from the possibly-split input text. Values can be None, an integer or tuple of integers; passing an integer will create ngrams up to that integer, and passing a tuple of integers will create ngrams for the specified values in the tuple. Passing None means that no ngrams will be created.

Figure 6. TextVectorization in Keras.

How to Use Text with Neural Networks?

- Neural networks don't work well with large numbers
  - Convert numbers to small vectors of small values

$$0 \rightarrow [0.15, 0.62, 0.31, 0.82, 0.53, 0.12, 0.83, 0.70]$$

$$1 \rightarrow [0.80, 0.92, 0.03, 0.20, 0.12, 0.60, 0.10, 0.58]$$

…

$$1{,}667{,}290 \rightarrow [0.22, 0.04, 0.82, 0.18, 0.74, 0.26, 0.04, 0.08]$$

…

Embedding in Keras

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

```
tf.keras.layers.Embedding(
        input_dim,
        output_dim,
        embeddings_initializer='uniform',
        embeddings_regularizer=None,
        activity_regularizer=None,
        embeddings_constraint=None,
        mask_zero=False,
        input_length=None,
        **kwargs
)
```

Args

`input_dim`	Integer. Size of the vocabulary, i.e. maximum integer index + 1.
`output_dim`	Integer. Dimension of the dense embedding.
`embeddings_initializer`	Initializer for the `embeddings` matrix (see `keras.initializers`).
`embeddings_regularizer`	Regularizer function applied to the `embeddings` matrix (see `keras.regularizers`).
`embeddings_constraint`	Constraint function applied to the `embeddings` matrix (see `keras.constraints`).
`mask_zero`	Boolean, whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers which may take variable length input. If this is `True`, then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to True, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).
`input_length`	Length of input sequences, when it is constant. This argument is required if you are going to connect `Flatten` then `Dense` layers upstream (without it, the shape of the dense outputs cannot be computed).

Input shape

2D tensor with shape: `(batch_size, input_length)`.

Output shape

3D tensor with shape: `(batch_size, input_length, output_dim)`.

Figure 7. Embedding in Keras.

Review Classification (Good vs. Bad) with RNNs

```
max_words = 1000
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=max_words)
vectorize_layer.adapt(X_train)

model = tf.keras.Sequential()
model.add(vectorize_layer)
model.add(tf.keras.layers.Embedding(input_dim=max_words, output_dim=64,
mask_zero=True))
model.add(tf.keras.layers.SimpleRNN(64))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

optimizer = tf.keras.optimizers.SGD(lr=0.001)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, validation_data=(X_val, y_val), epochs=512)
```

You have reached the end of the lecture.

Image/Figure References

Figure 1. Illustration of a basic RNN architecture. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 2. Different RNN applications. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 3. Basic RNN in detail. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 4. Implementation details of a basic RNN. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 5. RNN in Keras. Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN
Figure 6. TextVectorization in Keras. Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization
Figure 7. Embedding in Keras. Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding