ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

2 - Recurrent Neural Network

Giovanni Della Lunga
giovanni.dellalunga@gmail.com

Halloween Conference in Quantitative Finance

Bologna - October 25-26-27, 2023

Sequential Data

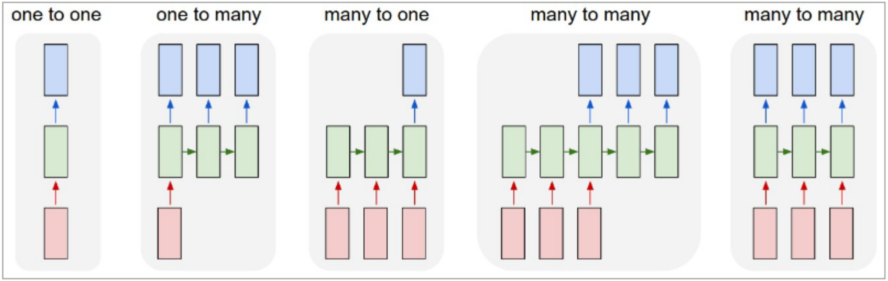
What are Sequential Data

- Sequential data refers to data that have a natural ordering, meaning that **each data point is dependent on the data that came before it**. In other words, **the sequence of the data matters**.
- Examples of sequential data include time series data such as stock prices, weather data, or audio signals, where each point in time is dependent on the previous point.
- Other examples include **text data** such as sentences or paragraphs, where **the meaning of each word is dependent on the context of the words that came before it**.
- Sequential data can be modeled and analyzed using various techniques, including Hidden Markov Models (HMMs), and autoregressive models such as ARIMA.

What are Sequential Data

- In the context of machine learning, sequential data presents unique challenges because the order of the data is important and traditional machine learning models such as decision trees and linear regression do not take this into account.
- Models such as RNNs (Recurrent Neural Network) are specifically designed to handle sequential data by processing the data one element at a time and retaining information about the previous elements to make predictions about future elements in the sequence.

What are Sequential Data

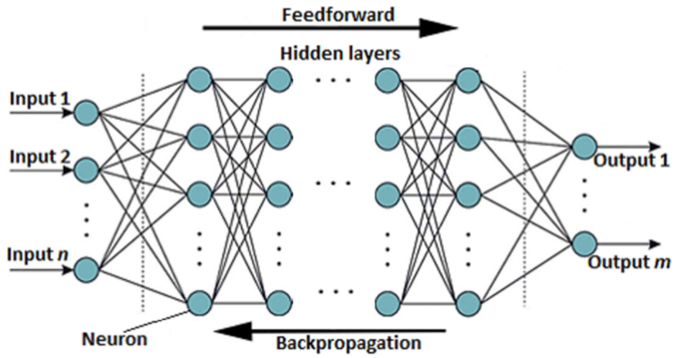


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

What are Recurrent Neural Networks

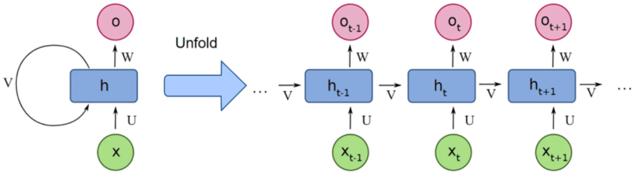
What are Recurrent Neural Networks

- A major characteristic of all neural networks you have seen so far, such as densely connected networks and convnets, is that they have no memory.
- Each input shown to them is processed independently, with no state kept in between inputs.



What are Recurrent Neural Networks

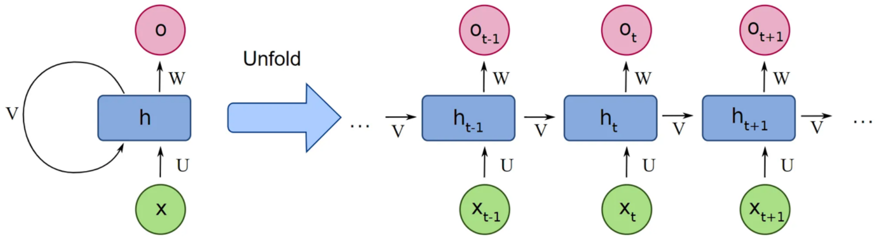
- A recurrent neural network (RNN) processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far.
- In effect, an RNN is a type of neural network that has an internal loop: **the network internally loops over sequence elements**.



What are Recurrent Neural Networks

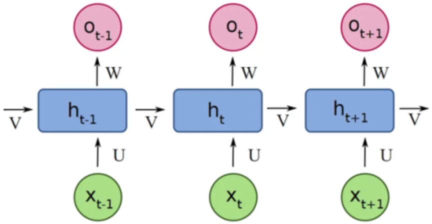
- These loops make recurrent neural networks seem kind of mysterious.
- However, if you think a bit more, it turns out that they aren‘t all that different than a normal neural network.
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
- Consider what happens if we unroll the loop...

What are Recurrent Neural Network



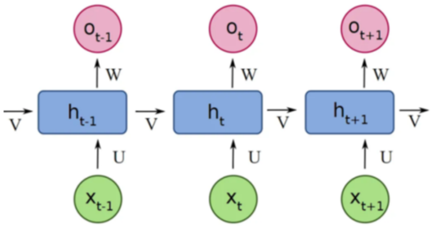
What are Recurrent Neural Network

- In the picture, there are some distinct components, from which the most important are:
- **x**: The input. It can be a word in a sentence or some other type of sequential data
- **O**: The output. For instance, what the network thinks the next word on a sentence should be given the previous words



What are Recurrent Neural Network

- In the picture, there are some distinct components, from which the most important are:
- **h**: The main block of the RNN. It contains the weights and the activation functions of the network
- **V**: Represents the communication from one time-step to the other.

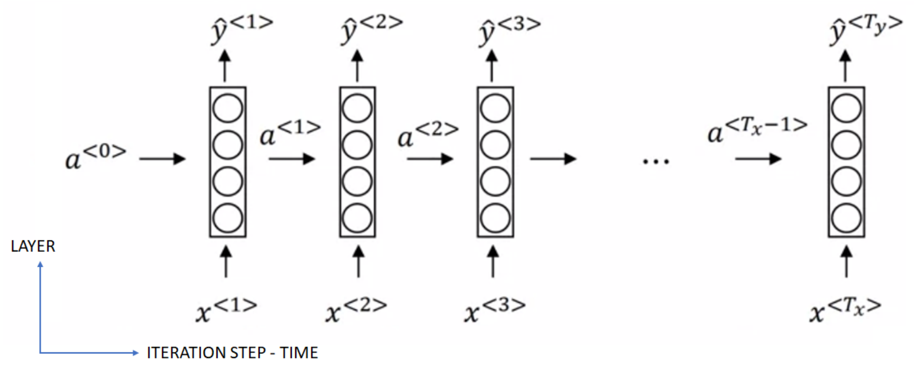


Forward Propagation in RNN

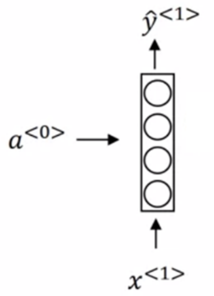
Forward Propagation in RNN

- In the forward propagation, the input data **is processed one time step at a time**.
- **Each time step corresponds to a moment in a sequence, and the RNN processes the sequence from the beginning to the end**.
- At each time step, **the RNN takes the current input** (e.g., a word in a sentence or a data point in a time series) and computes **an output and a hidden state**.
- **The hidden state is updated based on the current input and the previous hidden state, capturing the temporal dependencies**.

Forward Propagation in RNN



Forward Propagation in FNN

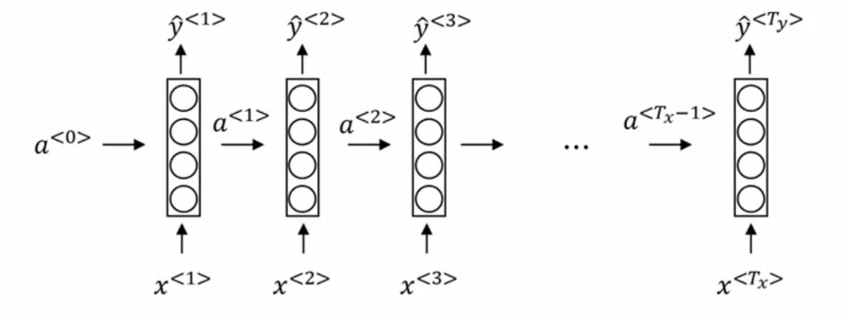


$a^{<0>} = 0$

$$a^{<1>} = g(w_{ax}x^{<1>} + b_a)$$

$$y^{<1>} = f(w_{ya}a^{<1>} + b_1)$$

Forward Propagation in RNN

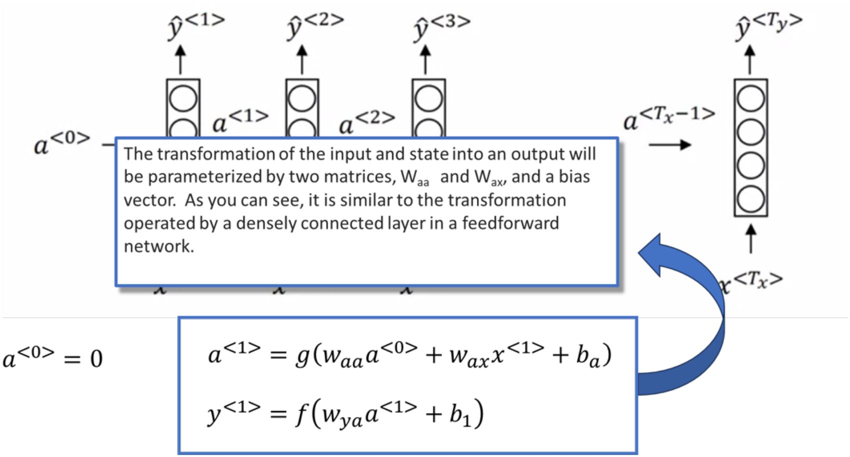


$a^{<0>} = 0$

$$a^{<1>} = g(w_{aa}a^{<0>} + w_{ax}x^{<1>} + b_a)$$

$$y^{<1>} = f\left(w_{ya}a^{<1>} + b_1\right)$$

Forward Propagation in RNN



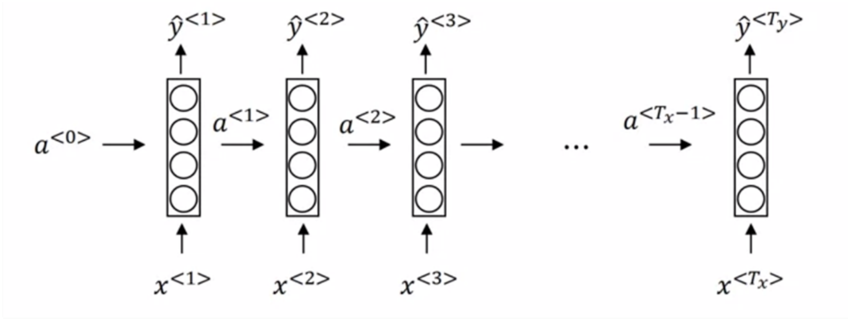
The transformation of the input and state into an output will be parameterized by two matrices, W_{aa} and W_{ax}, and a bias vector. As you can see, it is similar to the transformation operated by a densely connected layer in a feedforward network.

$$a^{<0>} = 0$$

$$a^{<1>} = g(w_{aa}a^{<0>} + w_{ax}x^{<1>} + b_a)$$

$$y^{<1>} = f\left(w_{ya}a^{<1>} + b_1\right)$$

Forward Propagation in RNN

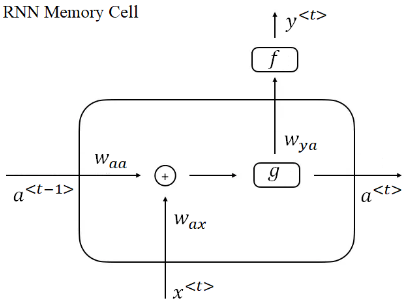


$$a^{<0>} = 0 \qquad a^{<t>} = g(w_{aa}a^{<t-1>} + w_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = f\left(w_{ya}a^{<t>} + b_1\right)$$

Memory Cell

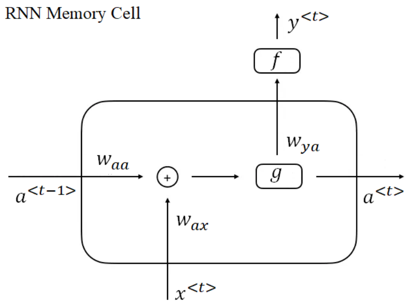
- Is very useful for educational purposes, introduce the idea of **Memory Cell**
- A memory cell refers to a fundamental component within an RNN unit that is designed to store and maintain information over time as the network processes sequential data.



$$a^{<t>} = g(w_{aa}a^{<t-1>} + w_{ax}x^{<t>} + b_a)$$
$$y^{<t>} = f\left(w_{ya}a^{<t>} + b_1\right)$$

Memory Cell

- Memory cell is simply a different representation of the fundamental equations.
- Can be useful in order to understand the behaviour of more complex forms of recurrent network as GRU and LSTM.



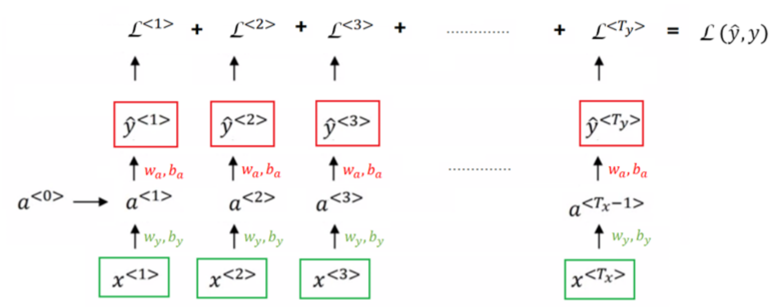
$$a^{<t>} = g(w_{aa}a^{<t-1>} + w_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = f\left(w_{ya}a^{<t>} + b_1\right)$$

Backpropagation Through Time

Backpropagation Through Time

- After processing the entire sequence, a loss is computed based on the predictions made by the RNN and the ground truth (target) values.
- This loss quantifies how well the RNN's predictions match the actual sequence data.



Backpropagation Through Time

- In the backpropagation phase, the gradients of the loss with respect to the model's parameters are computed.
- This involves **differentiating the loss function with respect to all weights and biases** in the RNN.
- The key idea in BPTT is to apply the chain rule of calculus to compute the gradients across time steps. It essentially "unrolls" the RNN in time, treating each time step as a layer in a deep feedforward neural network.

Backpropagation Through Time

- Starting from the last time step and moving backward in time, the gradients are computed iteratively.
- The gradient at each time step depends on the gradient from the next time step and the gradient from the current time step.
- The recurrent connections within the RNN are crucial in this process because they allow information to flow back in time, which is necessary for computing the gradients correctly.

Backpropagation Through Time

- After computing the gradients for all time steps, the model's parameters (weights and biases) are updated using an optimization algorithm like stochastic gradient descent (SGD) or one of its variants.
- The computed gradients are used to adjust the model's parameters in a way that minimizes the loss.
- Previous steps are repeated for a specified number of training iterations (epochs), or until convergence, to train the RNN.

Backpropagation Through Time

- Without going into the details of the calculation, let's just remember that the application of the chain derivation rule implies the multiplication of a greater number of derivatives the longer the sequence to be examined;
- For this reason, BPTT can lead to challenges such as the vanishing gradient problem, (where gradients become very small as they are propagated back in time) or the exploding gradient (where gradients grows without limits);
- Various techniques, such as using different RNN architectures (e.g., LSTMs and GRUs), gradient clipping, and more advanced optimization algorithms, have been developed to address these challenges and improve the training of RNNs.

Conclusions

Advantages of RNN

- Due to their shortterm memory RNNs can handle sequential data and identify patterns in the historical data.
- Moreover, RNNs are able to handle inputs of varying length.

Disadvantages of RNN

- The RNN suffers from the vanishing gradient descent.
- In this, the gradients that are used to update the weights during backpropagation become very small.
- Multiplying weights with a gradient that is close to zero prevents the network from learning new weights.
- This stopping of learning results in the RNN forgetting what is seen in longer sequences.
- The problem of vanishing gradient descent increases the more layers the network has.

Disadvantages of RNN

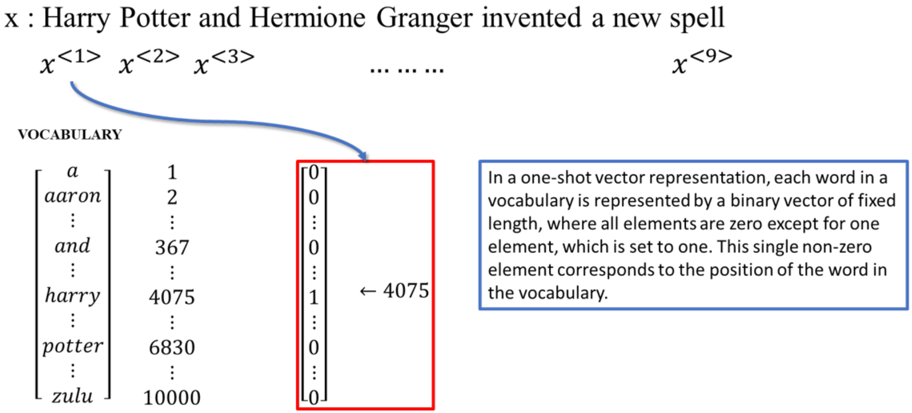
- As the RNN only keeps recent information, the model has problems to consider observations which lie far in the past.
- The RNN, thus, tends to loose information over long sequences as it only stores the latest information.
- Hence, the RNN has only a short-term but not a long-term memory.

Appendix

Representing Words

- Word embedding is a technique in natural language processing (NLP) and deep learning that represents words as continuous vector representations in a high-dimensional space.
- These vector representations, often referred to as word vectors or word embeddings, are designed to capture semantic and syntactic relationships between words, making them a fundamental component of various NLP applications.
- One popular type of word embedding is the one-shot vector representation, which is also known as "one-hot encoding" or "one-hot vectors."

Representing Words



x : Harry Potter and Hermione Granger invented a new spell

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... ... $x^{<9>}$

VOCABULARY

$$\begin{bmatrix} a \\ aaron \\ \vdots \\ and \\ \vdots \\ harry \\ \vdots \\ potter \\ \vdots \\ zulu \end{bmatrix} \quad \begin{matrix} 1 \\ 2 \\ \vdots \\ 367 \\ \vdots \\ 4075 \\ \vdots \\ 6830 \\ \vdots \\ 10000 \end{matrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \leftarrow 4075 \\ \\ \\ \\ \\ \end{matrix}$$

In a one-shot vector representation, each word in a vocabulary is represented by a binary vector of fixed length, where all elements are zero except for one element, which is set to one. This single non-zero element corresponds to the position of the word in the vocabulary.

Representing Words

x : Harry Potter and Hermione Granger invented a new spell

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … … … $x^{<9>}$

VOCABULARY

$$\begin{bmatrix} a \\ aaron \\ \vdots \\ and \\ \vdots \\ harry \\ \vdots \\ potter \\ \vdots \\ zulu \end{bmatrix} \begin{matrix} 1 \\ 2 \\ \vdots \\ 367 \\ \vdots \\ 4075 \\ \vdots \\ 6830 \\ \vdots \\ 10000 \end{matrix} \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \leftarrow 4075 \\ \\ \\ \\ \\ \end{matrix} \qquad \boxed{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \leftarrow 6830 \\ \\ \\ \end{matrix}}$$

Representing Words

x : Harry Potter and Hermione Granger invented a new spell

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ……… $x^{<9>}$

VOCABULARY

$$\begin{bmatrix} a \\ aaron \\ \vdots \\ and \\ \vdots \\ harry \\ \vdots \\ potter \\ \vdots \\ zulu \end{bmatrix} \quad \begin{matrix} 1 \\ 2 \\ \vdots \\ 367 \\ \vdots \\ 4075 \\ \vdots \\ 6830 \\ \vdots \\ 10000 \end{matrix} \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \leftarrow 4075 \\ \\ \\ \\ \\ \end{matrix} \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \leftarrow 6830 \\ \\ \\ \end{matrix} \qquad \boxed{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow 367 \\ \\ \\ \\ \\ \\ \\ \end{matrix}}$$

Representing Words

x : Harry Potter and Hermione Granger invented a new spell

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … … … $x^{<9>}$

VOCABULARY

$$\begin{bmatrix} a \\ aaron \\ \vdots \\ and \\ \vdots \\ harry \\ \vdots \\ potter \\ \vdots \\ zulu \end{bmatrix} \quad \begin{matrix} 1 \\ 2 \\ \vdots \\ 367 \\ \vdots \\ 4075 \\ \vdots \\ 6830 \\ \vdots \\ 10000 \end{matrix} \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 4075 \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 6830 \qquad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 367 \qquad \dots\dots\dots$$

Representing Words

- One-shot vectors are sparse because most elements are zero, and they have a dimensionality equal to the size of the vocabulary.
- As the vocabulary grows, so does the dimensionality of the vectors.
- More advanced word embeddings techniques, on the other hand, represent words as dense vectors of real numbers in a continuous vector space.
- Unlike one-shot vectors, word embeddings are not binary and can take on a range of real values.
- We'll talk a lot about word embeddings in the next chapters ...