CS-618
Deep Learning
LateSpring 2023

Lecture 5

Agenda

Recurrent neural networks (**RNNs**)

- Introducing sequential data
- RNNs for modeling sequences
- Long short-term memory (LSTM)
- Truncated backpropagation through time (TBPTT)
- Implementing a multilayer RNN for sequence modeling in TensorFlow
- Project one: RNN sentiment analysis of the IMDb movie review dataset
- Project two: RNN character-level language modeling with LSTM cells,
- using text data from Jules Verne's *The Mysterious Island*
- Using gradient clipping to avoid exploding gradients
- Introducing the *Transformer* model and understanding the *self-attention mechanism*

What are recurrent neural networks

- A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.
- Unlike feedforward neural networks, which process data in a single pass, RNNs have a feedback mechanism that allows them to maintain an internal state or memory.
- This memory enables RNNs to process sequences of variable length and learn dependencies over time.

https://www.ibm.com/topics/recurrent-neural-networks

Building block

- The basic building block of an RNN is the recurrent unit, which typically takes an input and the previous state as inputs and produces an output and a new state.
- The output can be fed back as input to the next step in the sequence, creating a loop that allows information to be propagated through time. This recurrent structure makes RNNs well-suited for tasks such as language modeling, speech recognition, machine translation, and time series analysis.

- These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning.
- They are incorporated into popular applications such as Siri, voice search, and Google Translate.
- Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn.
- They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output.

- While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence.
- While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions.
- Let’s take an idiom, such as “feeling under the weather”, which is commonly used when someone is ill, to aid us in the explanation of RNNs.
- In order for the idiom to make sense, it needs to be expressed in that specific order.
- As a result, recurrent networks need to account for the position of each word in the idiom and they use that information to predict the next word in the sequence.

characteristic of recurrent networks

- they share parameters across each layer of the network.
- While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network.
- these weights are still adjusted in the through the processes of backpropagation and gradient descent to facilitate reinforcement learning.

- Recurrent neural networks leverage backpropagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data.
- The principles of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer.
- These calculations allow us to adjust and fit the parameters of the model appropriately.
- BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

Here are some of the applications of RNNs:

- Natural language processing: RNNs can be used to solve a variety of natural language processing tasks, such as text classification, sentiment analysis, and machine translation.
- Speech recognition: RNNs can be used to recognize spoken words.
- Music generation: RNNs can be used to generate music.
- Image captioning: RNNs can be used to generate captions for images.
- Time series forecasting: RNNs can be used to forecast future values of time series data.

Introducing sequential data

- RNNs by looking at the nature of sequential data
- which is more commonly known as sequence data or **sequences**
- Unique properties of sequences make them different to other kinds of data
- The relationship between RNNs and sequences

Sequential data in many problem domains

- Image captioning
- Speech synthesis
- Music generation
- Playing video games
- Language modeling
- Character-level text generation models

In other domains, we need a sequence of input vectors:

- Time-series prediction
- Video analysis
- Music information retrieval

And then we have domains that require both a series of input and output vectors:

- Natural language translation
- Engaging in dialogue
- Robotic control

Sequential data versus time-series data

- Time-series data is a special type of sequential data
- where each example is associated with a dimension for time.
- In time-series data, samples are taken at successive time stamps
- the time dimension determines the order among the data points.
- For example, stock prices and voice or speech records are timeseries data.
- On the other hand, not all sequential data has the time dimension
- for example, text data or DNA sequences, where the examples are ordered but they do not qualify as time-series data.

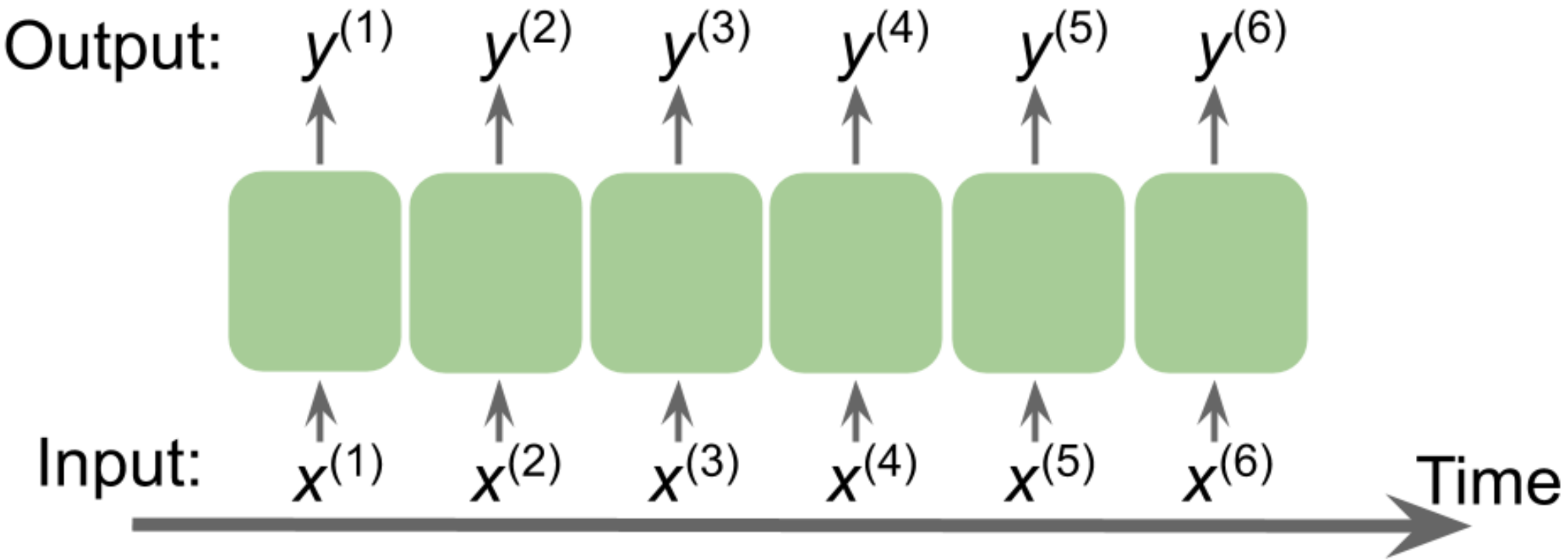
The most common type of RNN

The Long Short-Term Memory (LSTM) network, which was introduced to address the vanishing gradient problem that affects traditional RNNs. The vanishing gradient problem occurs when gradients diminish exponentially as they are backpropagated through many time steps, making it difficult for the network to learn long-term dependencies. LSTMs use a gating mechanism to selectively update and forget information in the memory, allowing them to better capture long-range dependencies.

Another variant of RNNs

- the Gated Recurrent Unit (GRU), which is similar to LSTM but has a simplified architecture with fewer gates. GRUs also address the vanishing gradient problem and have been shown to perform well in various sequence modeling tasks.

Representing sequences



The time-step aspect of Recurrent Neural Network input

Vector columns

	Values
albumin	0.0
alp	1.0
alt	0.5
ast	0.0
...	

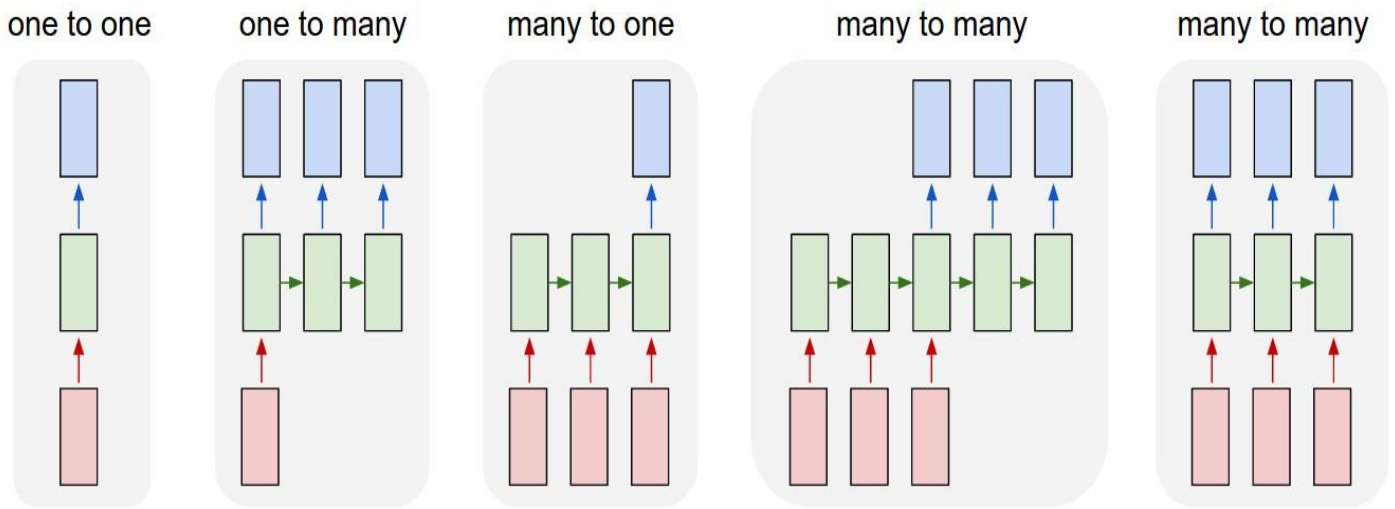
Time-steps

Vector columns

	0	1	2	3	4	...
albumin	0.0	0.0	0.5	0.0	0.0	
alp	0.0	0.1	0.0	0.0	0.2	
alt	0.0	0.0	0.0	0.9	0.0	
ast	0.0	0.0	0.0	0.0	0.4	
...						

The different categories of sequence modeling

- Sequence modeling has many fascinating applications, such as language translation (for example, translating text from English to German), image captioning, and text generation.
- However, in order to choose an appropriate architecture and approach,
- we have to understand and be able to distinguish between these different sequence modeling tasks.



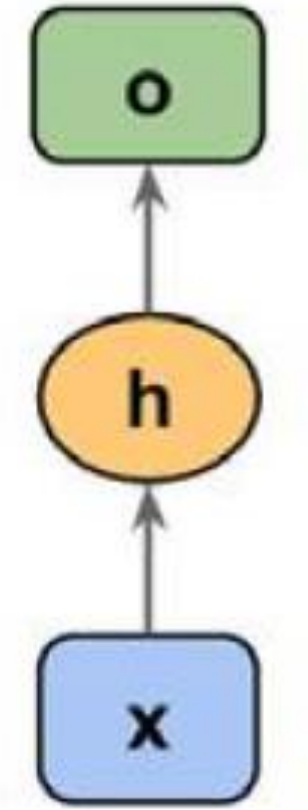
https://karpathy.github.io/2015/05/21/rnn-effectiveness/

categories of sequence modeling

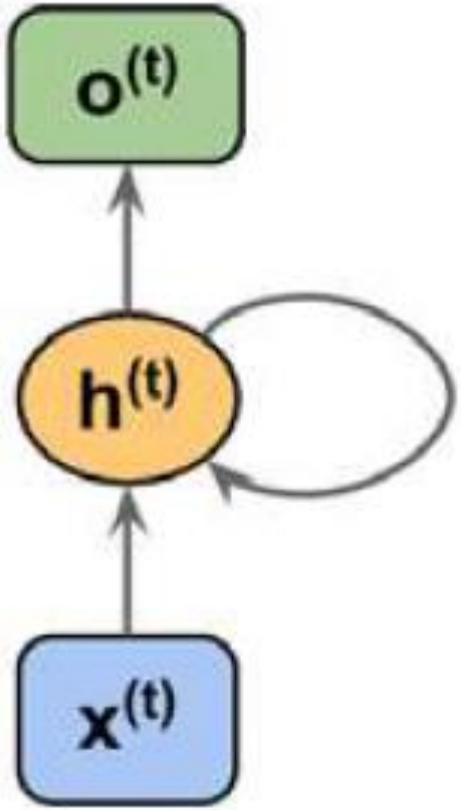
- **Many-to-one**: The input data is a sequence, but the output is a fixed-size vector or scalar, not a sequence. For example, in sentiment analysis, the input is text-based (for example, a movie review) and the output is a class label (for example, a label denoting whether a reviewer liked the movie).
- **One-to-many**: The input data is in standard format and not a sequence, but the output is a sequence. An example of this category is image captioning— the input is an image and the output is an English phrase summarizing the content of that image.
- **Many-to-many**: Both the input and output arrays are sequences. This category can be further divided based on whether the input and output are synchronized. An example of a synchronized many-to-many modeling task is video classification, where each frame in a video is labeled. An example of a *delayed* many-to-many modeling task would be translating one language into another. For instance, an entire English sentence must be read and processed by a machine before its translation into German is produced.

the RNN looping mechanism

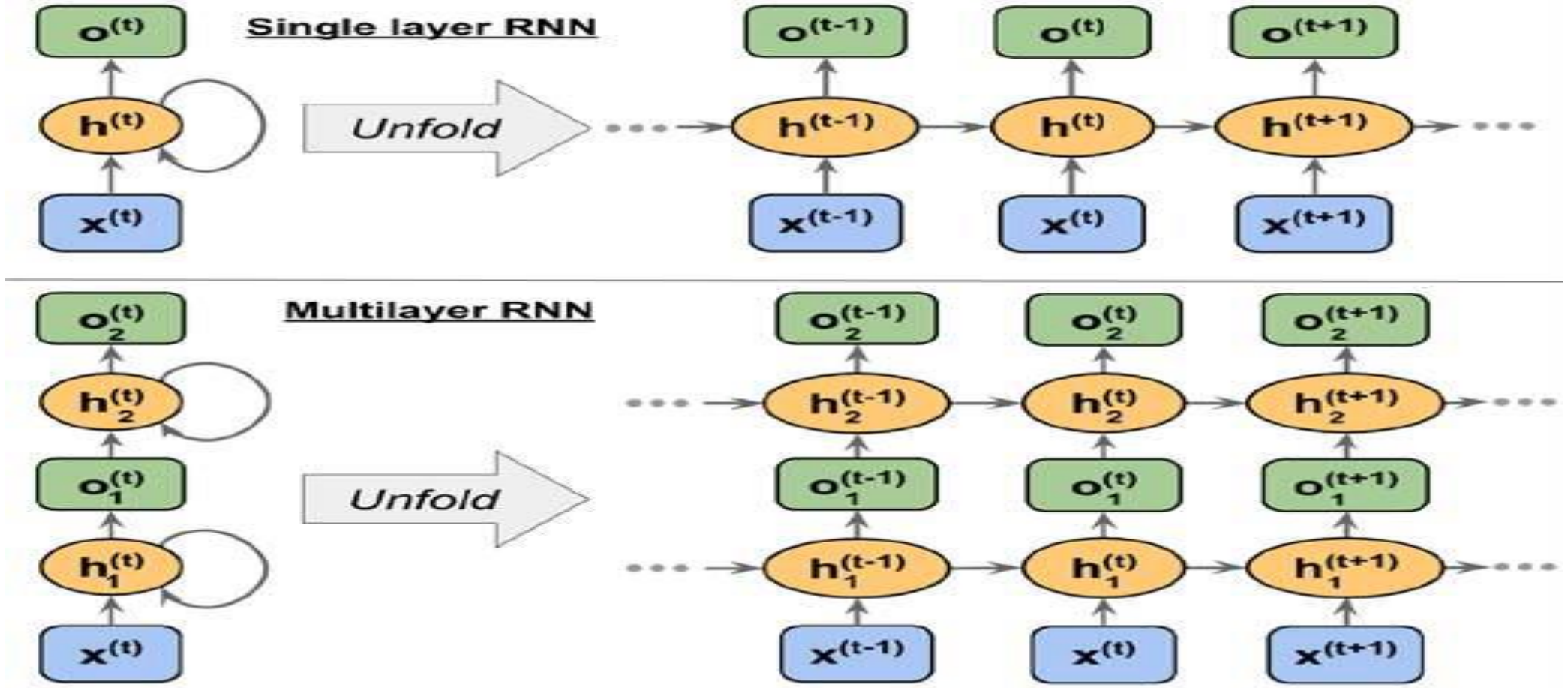
A standard feedforward network



Recurrent neural network

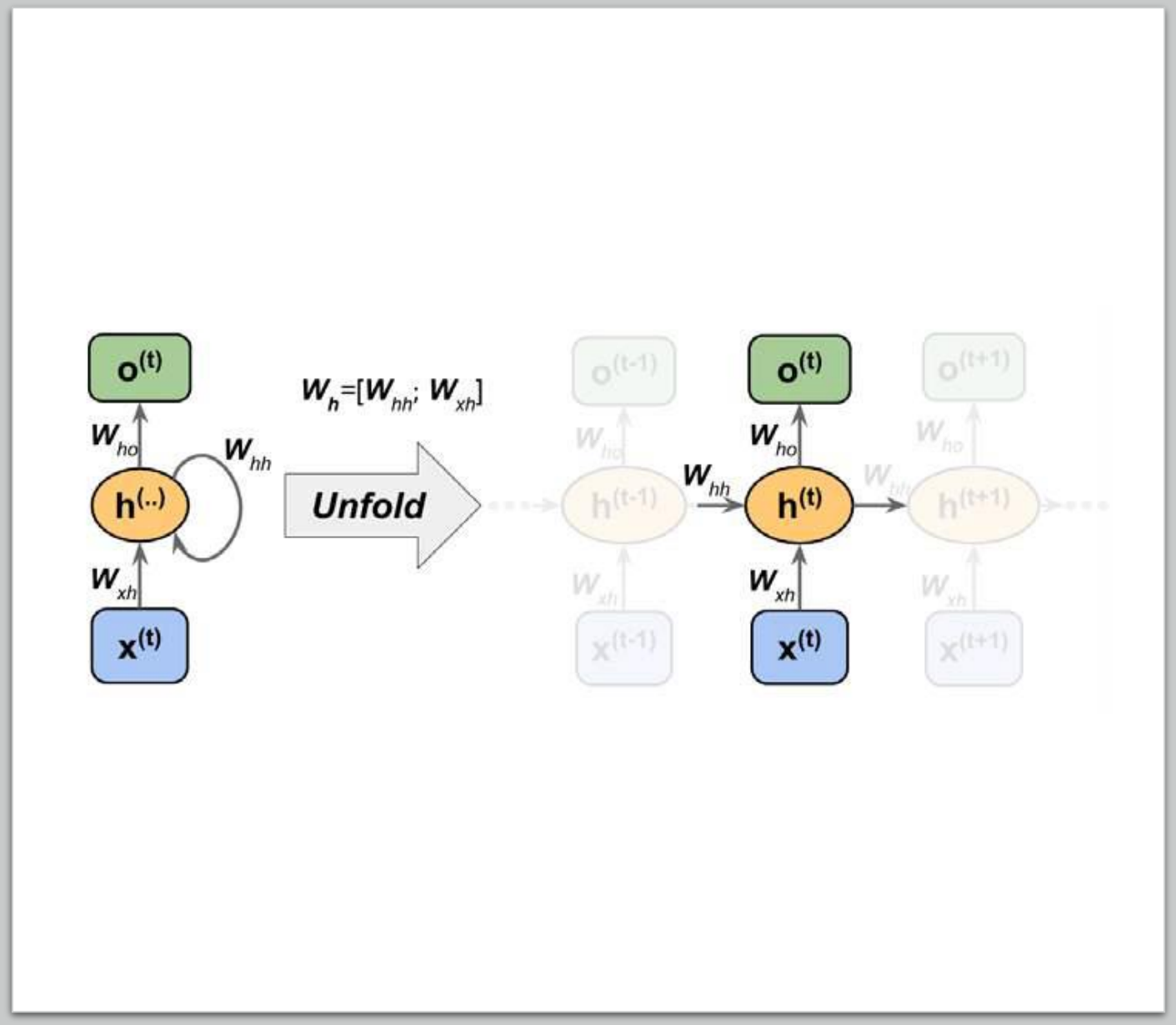


Multilayer RNN



Computing activations in an RNN

- The representation of an RNN that we just looked at is associated with a weight matrix.
- Those weights do not depend on time, t; therefore, they are shared across the time axis.
- The different weight matrices in a single-layer RNN are as follows:
  - $\boldsymbol{W}xh$: The weight matrix between the input, $\boldsymbol{x}(t)$, and the hidden layer, $\boldsymbol{h}$
  - $\boldsymbol{W}h$: The weight matrix associated with the recurrent edge
  - $\boldsymbol{W}ho$: The weight matrix between the hidden layer and output layer



Computing

the sum of the multiplications of the weight matrices with the corresponding vectors and add the bias unit:

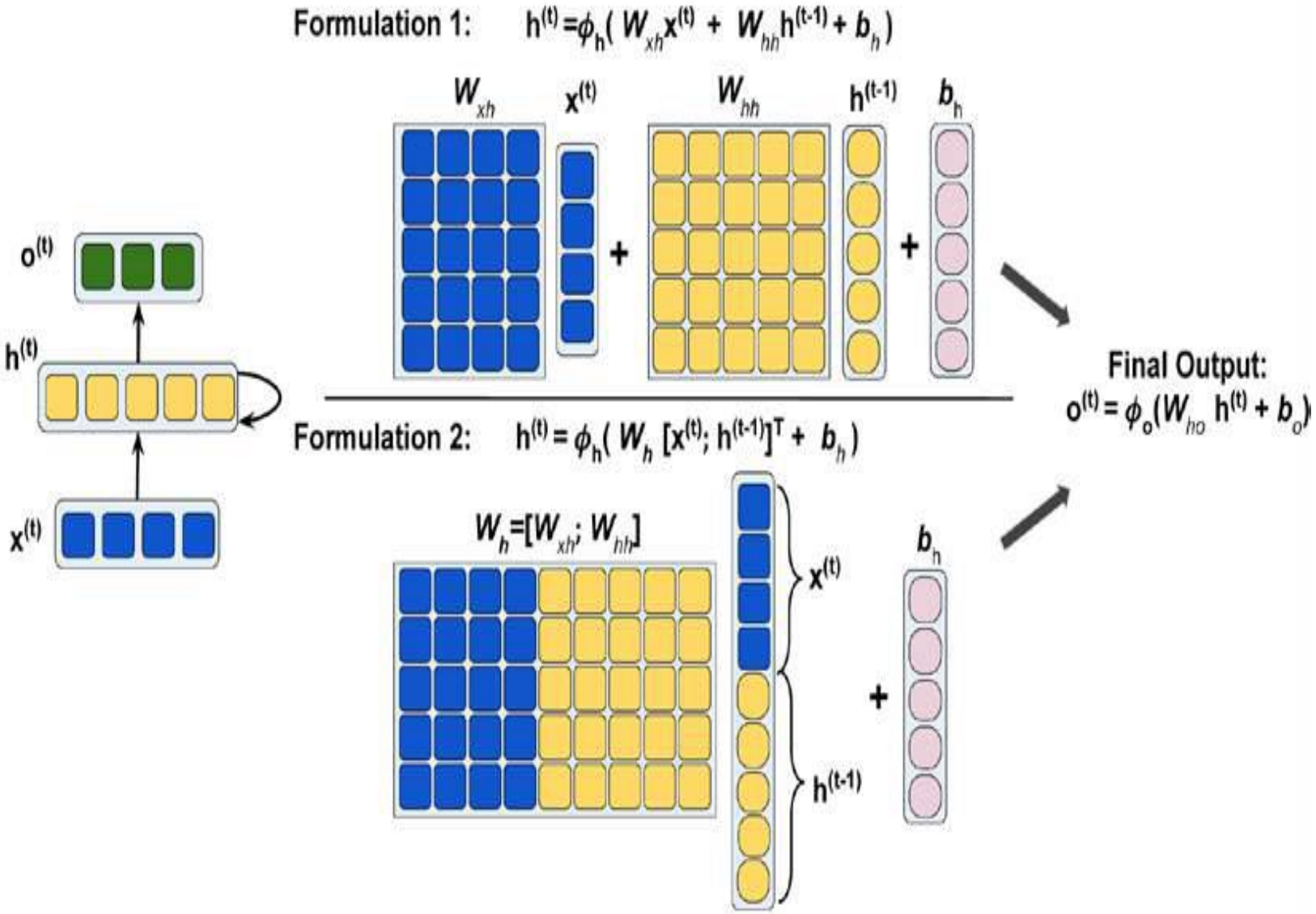
$$\boldsymbol{z}_h^{(t)} = \boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h$$

the activations of the hidden units at the time step, *t*, are calculated as:

$$\boldsymbol{h}^{(t)} = \phi_h\left(\boldsymbol{z}_h^{(t)}\right) = \phi_h\left(\boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h\right)$$

$\boldsymbol{b}_h$ is the bias vector for the hidden units and $\phi_h(\cdot)$ is the activation function of the hidden layer

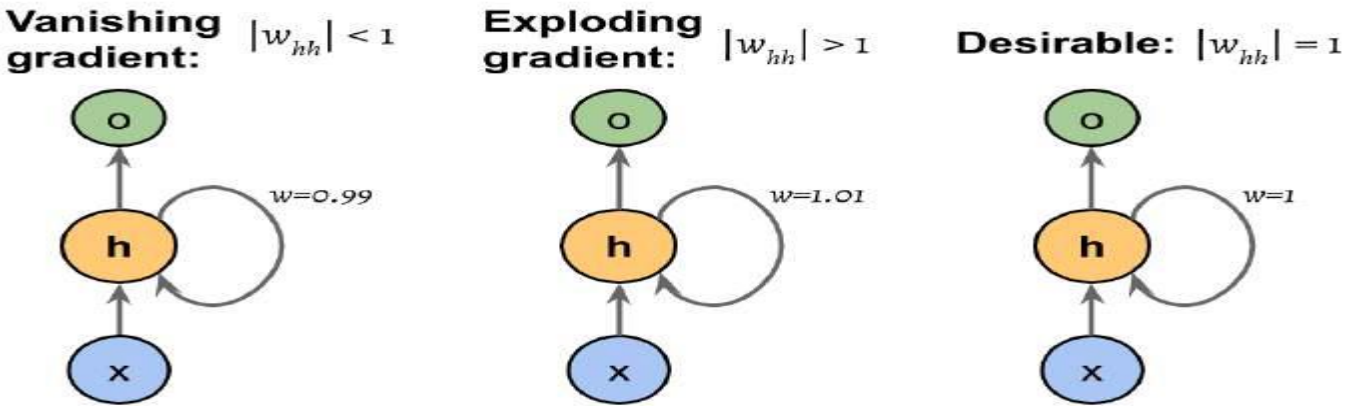
The process of computing activations with both formulations



Different RNN Architectures

- **Bidirectional Recurrent Neural Networks (BRNN)**
  - inputs from future time steps are used to improve the accuracy of the network. It is like knowing the first and last words of a sentence to predict the middle words.
- **Gated Recurrent Units (GRU)**
  - These networks are designed to handle the vanishing gradient problem. They have a reset and update gate. These gates determine which information is to be retained for future predictions.
- **Long Short Term Memory (LSTM)**
  - LSTM designed to address the vanishing gradient problem in RNNs. LSTMs use three gates called input, output, and forget gate. Similar to GRU, these gates determine which information to retain.

The challenges of learning long-range interactions



Vanishing

- As the backpropagation algorithm advances downwards(or backward) from the output layer towards the input layer, the gradients often get smaller and smaller and approach zero which eventually leaves the weights of the initial or lower layers nearly unchanged. As a result, the gradient descent never converges to the optimum.

Exploding

- On the contrary, in some cases, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge.

Convolutional vs Recurrent Neural Networks

RNN

- perform well when the input data is interdependent in a sequential pattern
- correlation between previous input to the next input
- introduce bias based on your previous output

CNN/FF-Nets

- all the outputs are self dependent
- *Feed-forward nets don’t remember historic input data at test time unlike recurrent networks.*

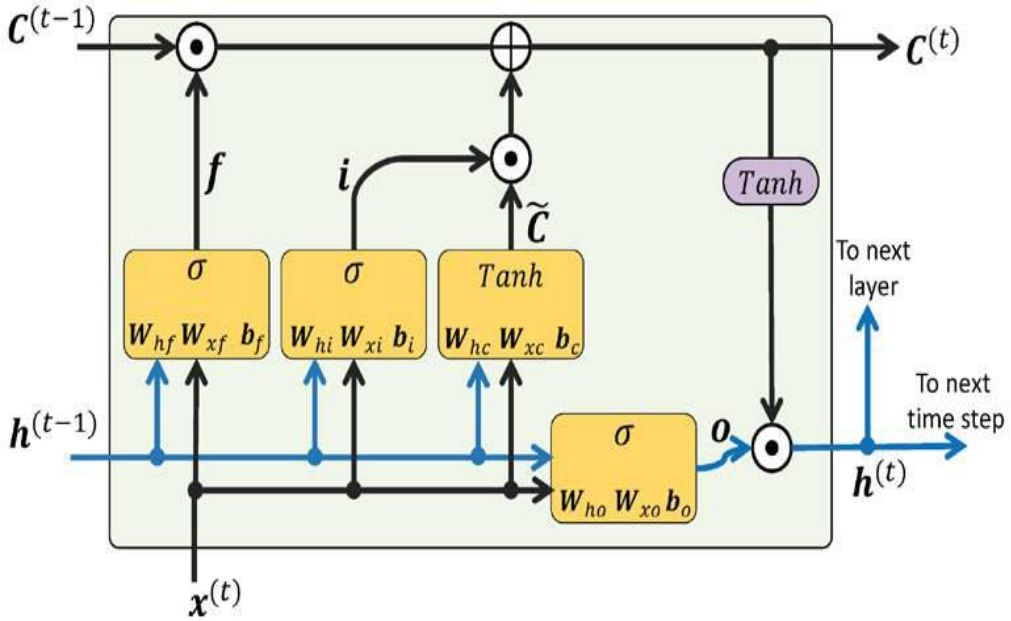
Solution

There are at least three solutions to this problem:

- Gradient clipping
- TBPTT
- LSTM

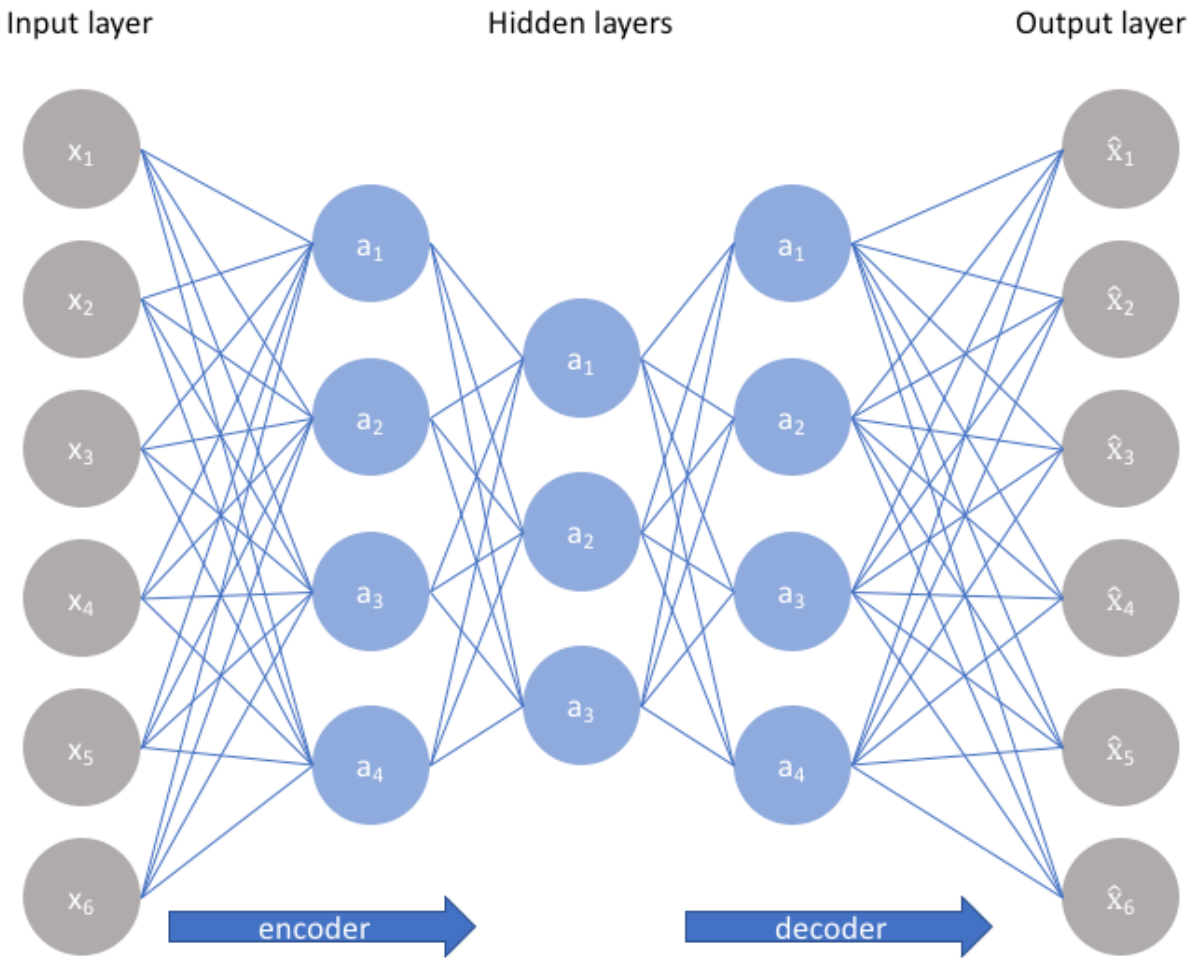
LSTM

- Long Short-Term Memory
- Capable of carrying information across many timesteps
- Saves information for later to prevent older signals from gradually vanishing during processing
- LSTMs provide a basic approach for modeling long-range dependencies in sequences.



Autoencoders

- DNNs for unsupervised learning
- Great for:
  - Dimensionality Reduction
  - Anomaly detection
  - Image reduction
- A neural network that learns to copy its input to its output
- Removes noise



Types of gates

- In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose

$$\Gamma = \sigma(W x^{<t>} + U a^{<t-1>} + b)$$

where *W*,*U*,*b* are coefficients specific to the gate and *σ* is the sigmoid function.
The main ones are summed up in the table below:

Type of gate	Role	Used in
Update gate Γ*u*	How much past should matter now?	GRU, LSTM
Relevance gate Γ*r*	Drop previous information?	GRU, LSTM
Forget gate Γ*f*	Erase a cell or not?	LSTM
Output gate Γ*o*	How much to reveal of a cell?	LSTM

The pros and cons of a typical RNN architecture

Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Project one – predicting the sentiment of IMDb movie reviews

- we will implement a many-to-many RNN for an application of language modeling. While the chosen examples are purposefully simple to introduce the main concepts of RNNs, language modeling has a wide range of interesting applications, such as building chatbots—giving computers the ability to directly talk and interact with humans.

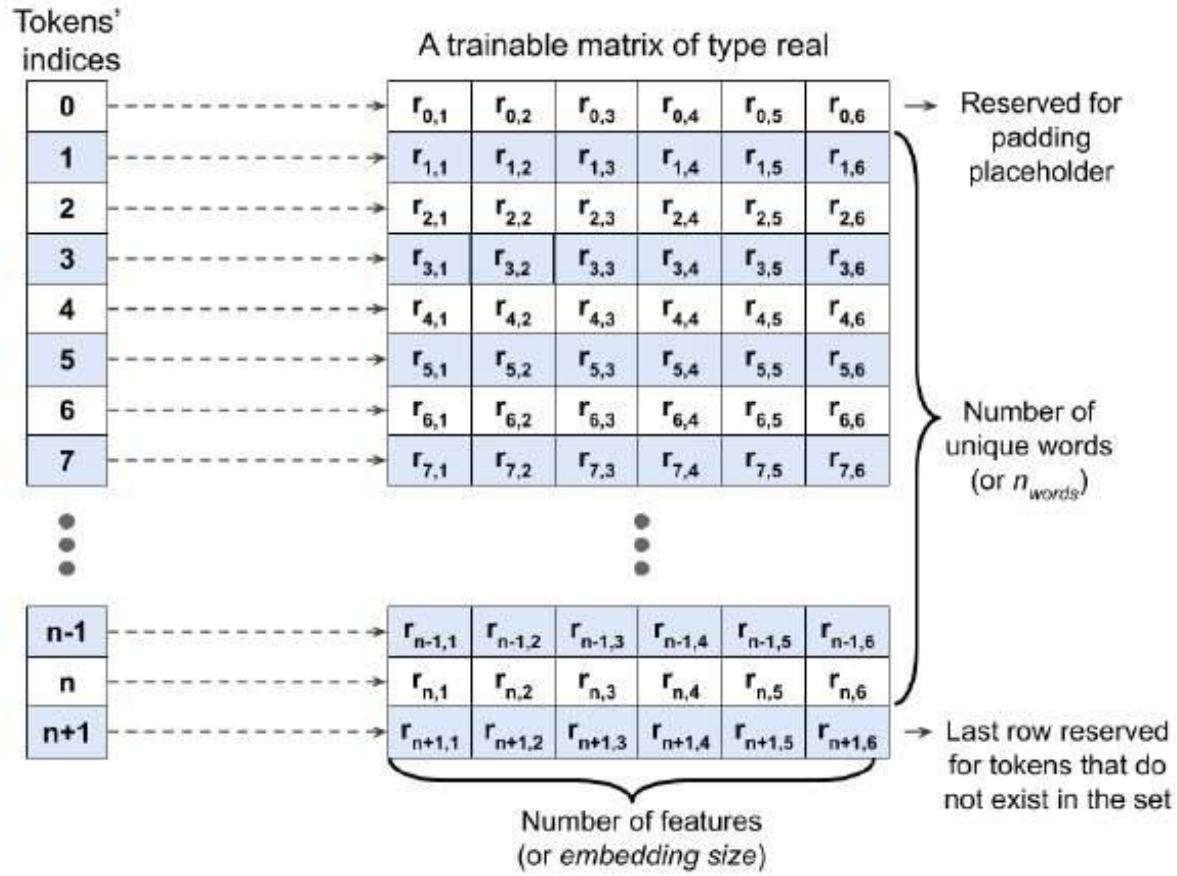
1. Create a TensorFlow dataset object and split it into separate training, testing, and validation partitions.

2. Identify the unique words in the training dataset.

3. Map each unique word to a unique integer and encode the review text into encoded integers (an index of each unique word).

4. Divide the dataset into mini-batches as input to the model.

The following schematic representation shows how embedding works by mapping token indices to a trainable embedding matrix:



Thank You