Machine Learning Engineer Course
Day 29

- Recurrent Neural Network -

DIVE INTO CODE

Thursday November 4, 2021

DIOP Mouhamed

Agenda

1. Check-in
2. Quick Review
3. Recurrent Neural Network (RNN)
4. Sample code
5. To do by next class
6. Check-out

Check-in

3 minutes Please post the following point to Zoom chat.

Q. What did you learn in the previous week?
(Anything is fine.)

Quick Review (NLP)

- Introduction to NLP
- Local vs Distributed Distribution
- BoW TF-IDF
- Problems with Distributed Representation

RNN (Recurrent Neural Network)

What was BoW?

The representation by BoW (including TF-IDF) was a matrix consisting of documents and words. And because of the relation that one dimension corresponds to one word, it can be a huge sparse matrix.

The scikit-learn article "Feature extraction /4.2.3.2. Sparsity" describes it as follows

"For most documents, we are dealing with a very small set of words in the corpus (a sequence of BoWs), so that more than 99% of the resulting BoWs will be zero. For example, if we consider a set of 10,000 short documents, it consists of 100,000 words, while the number of unique, non-overlapping words used in a single document is about 100 to 1,000.
To handle this sparse matrix, we use a sparse representation such as scipy.sparse.

https://scikit-learn.org/0.16/modules/feature_extraction.html#sparsity

	a	bad	film	good	is	movie	this	very
0	0	0	0	1	1	1	1	1
1	1	0	1	1	1	0	1	0
2	0	2	0	0	0	0	0	3

```
  (0, 3)        1
  (0, 7)        1
  (0, 4)        1
  (0, 5)        1
  (0, 6)        1
  (1, 0)        1
  (1, 2)        1
  (1, 3)        1
  (1, 4)        1
  (1, 6)        1
  (2, 1)        2
  (2, 7)        3
```

RNN

Local and distributed representations

The BoW-like representation described above was called local representation. This was a method of representing a symbol (word) by a vector consisting of one or very few non-zero elements.
In contrast, the representation extracted from Word2Vec is called a distributed representation, where multiple dimensions are used to construct a single word, and the same single dimension is used to construct multiple words. When describing an event, distributed representation is done using a collection of diverse features that share a concept with other events.

On the other hand, a local representation can be said to indicate **a few or a single characteristic element of the event.**

The term "distributed representation" is said to have originated in the 1980s when Hinton et al. proposed it as an application from the research fields of cognitive psychology and neuroscience, as well as neural networks [1].
1] It emerged as a way to represent discrete objects, such as events and concepts, as multiple continuous-valued features (vectors) when considering the modeling of the brain.

[1] Geoffrey E. Hinton, Learning distributed representations of concepts. 1989
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.408.7684&rep=rep1&type=pdf

Word2Vec = Word to Vector

A pre-processing neural network for generating distributed representations: Word2Vec.

How does Word2Vec create a distributed representation?

First, suppose you have a model that uses 100,000 or so one-hot (local) representations as input and predicts the next word through 200 or so hidden layers, and you are able to predict the next word reasonably well.
At this point, the information of 100,000 words can be considered to have been mapped to 200 hidden layers, or a space of dimensionally compressed latent variables.
In other words, the 200 parameters here can be thought of as substituting for the information of the 100,000 words.
Hence, the idea of Word2Vec is that these 200 parameters can be used as a distributed representation.

This distributed representation is the input data for RNNs, which are the representative language models of deep learning.
In the following sections, we will explain about these RNNs (various types of RNNs).

Deep Learning Language Models: What are RNNs?

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural network designed to process serial data such as **time series**, and research on them began in the late 1980s.

This differs from conventional feed-forward neural networks in that the network has a cyclical structure.
RNNs are still being used in the field of natural language processing.
Recurrent means "to circulate" in English.

What kind of structure is a circulating structure?

In the following, we will introduce the cyclic structure of mutual coupling and regression coupling as representative of RNNs.

RNNs: Interconnected type

fully interconnected network

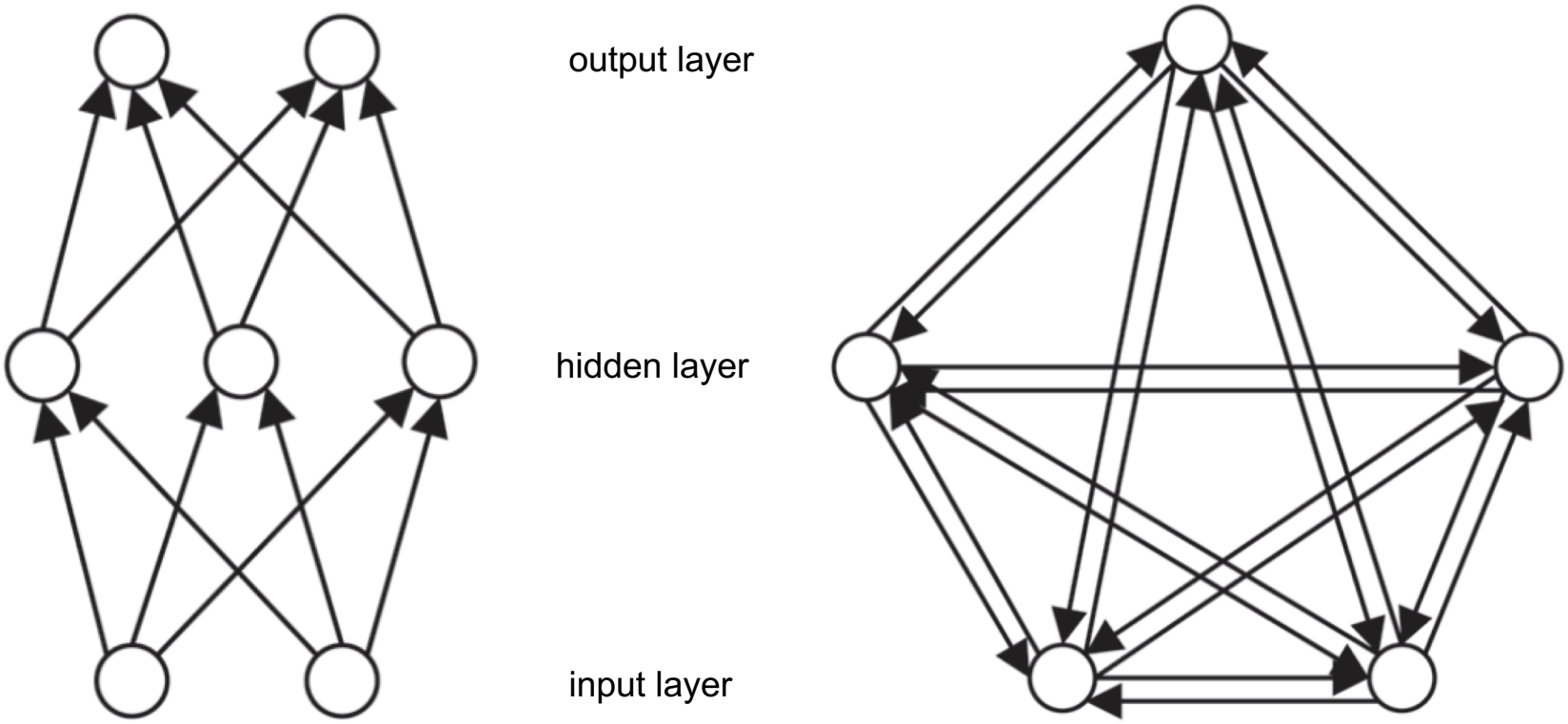
The Hopfield network [2], proposed in 1982, has a circular structure and is considered to be the earliest RNNs. Its structure is called fully interconnected, meaning that it is interconnected with all nodes except itself.

Hopfield can retain multiple storage patterns through learning (the number of patterns that can be stored is about 0.15 times the number of nodes). When estimating after learning, Hopfield can store the most similar patterns from the pre-stored ones against the noise in the unknown input.
This mechanism is called an associative memory model, and it is known to be applicable to discrete (combinatorial) optimization problems.

[2] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. 1982

RNN

Example structure of a neural network [3].



a) hierarchical network

b) Fully interconnected network

[3] Hirokazu IWASE The Influence of Parameter Settings on the Hopfield Neural Network. 2017

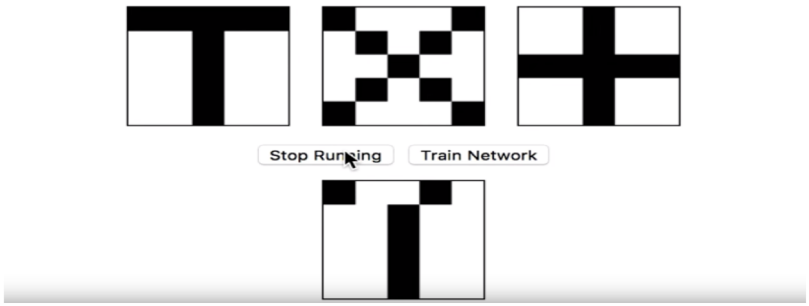
RNNs: Interconnected type

fully interconnected network

However, Hopfield networks are not suitable for time series data because their connections are **undirected** (undirected graph).

In order to handle time series, we need a network with temporal orientation, where we can infer future information from past information, and then use that information to further infer the next future information.

https://www.youtube.com/watch?v=HOxSKBxUVpg

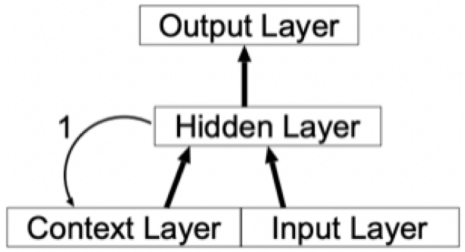


RNNs: Regression coupled type

simple recurrent networks

Simple recurrent networks [4], invented by Elman in 1990, establish a cyclic structure by means of unidirectional regressive connections (directed graphs).
Unlike the Hopfield network, Elman's network can be applied to time series data because the shape of the connections is directional.
Unlike the Hopfield network, Elman's network can be applied to time series data because it is unidirectional, meaning that information from the past can be received from the regression bonds.
The basic structure is a hierarchical network (with connections from the input layer to the middle layer, and from the middle layer to the output layer), the same as a conventional feed-forward neural network, with the addition of a context layer that retains (copies) the state of the previous middle layer. The context layer copies the information at one point in time and passes it to the intermediate layer (and passes the same information to the output layer), and at the next point in time, it passes the information to the intermediate layer along with the information from the input layer, thus realizing the cyclic structure of having a connection from the previous time itself (the intermediate layer).

Elmanet [5]



[4] Jeffrey L.Elman. Finding structure in time. 1990
[5] Shin-ichi Asakawa Applicability of Elmanets 2009

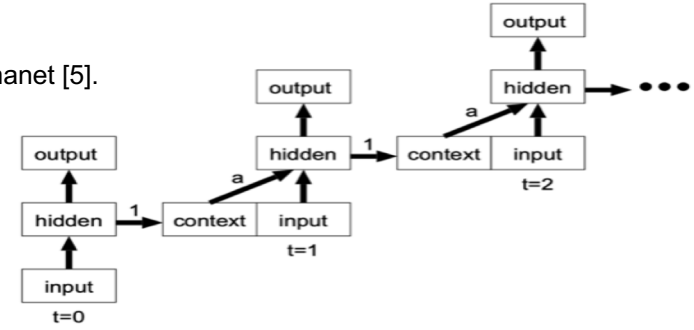
RNNs: Regression coupled type

simple recurrent networks

The content that occurs at a point t is the simultaneous processing of the input at that point and the state of the network that has been processed up to the earlier point t-1.

Since the middle layer (context layer) can be interpreted as **retaining the past state up to point t-1**, we can say that the state of the network at point t is estimated by the entirety of the current input and the input history from the past.
When the weighting of the connections from the context layer is less than 1, the influence of the input history from the past becomes exponentially smaller, and when it is greater than 1, the influence of the input history from the past becomes exponentially larger.

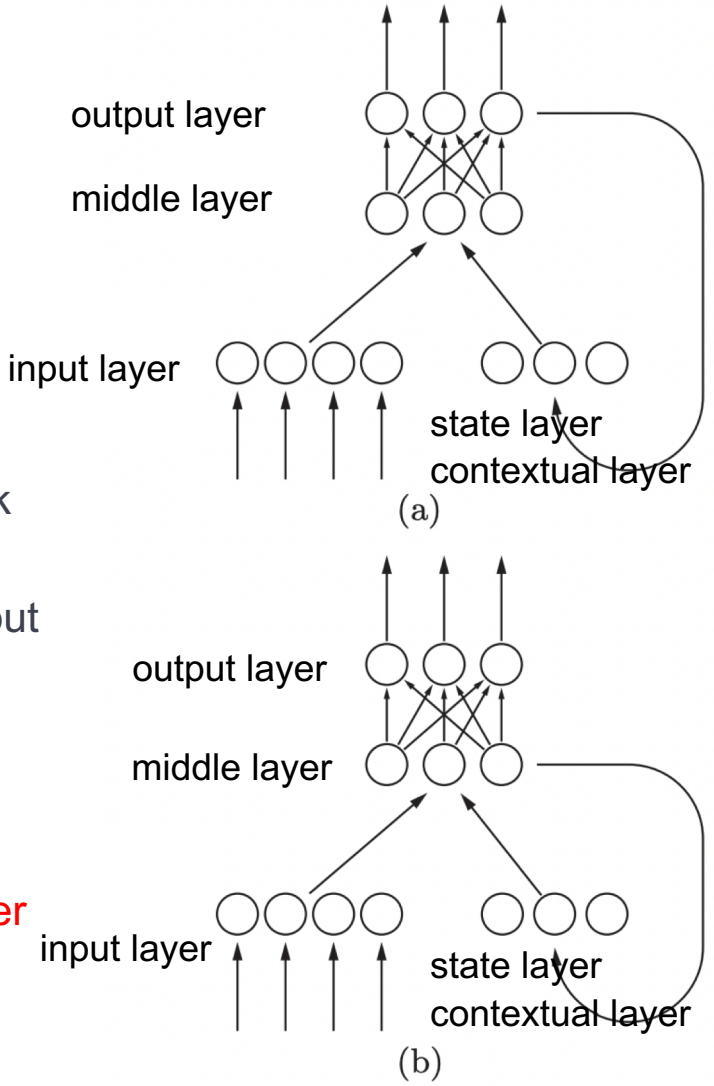
Time evolution of Elmanet [5].



RNNs: Regression coupled type

simple recurrent networks

Another network with the same simple regression coupling as the Elman network is the Jordan network [6], published in 1986, which differs from the Elman network in that the past history is input from the output layer to the context layer (called the state layer in Jordan) instead of the middle layer. The difference with the Elman network is that instead of an intermediate layer, the past history is input from the output layer to the context layer (called the state layer in Jordan).



(a) Jordan network
(b) Elman network

RNNs: Regression coupled type

simple recurrent networks

Since the output layer at time t-1 is trained to match the target of the series data (information at time t), the information in the output layer is considered to be information about the prediction at time t rather than information at time t-1.

For example, in a natural language processing task, the context layer would like to receive the past history as input, but in the case of the Jordan network, it does not have enough past history (at time t) because the copy that is handed over is that of the output layer.

For this reason, the Elman network with its regression-like connections in the middle layer is more suitable for natural language processing than the Jordan network.

On the other hand, using the information in the output layer at time t is useful for controlling robot arms, posture, motion, etc. Therefore, the Jordan network is more suitable for motion control applications.

Recurrent Hidden Units

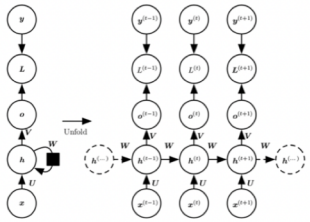


Figure 10.3

Recurrence through only the Output

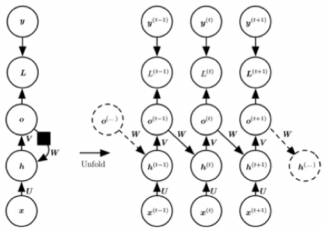


Figure 10.4

RNNs: gradient calculation

A famous learning rule for hierarchical networks such as feed-forward neural networks is the **error Back Propagation (BP)** method, also known as backward auto-differentiation.

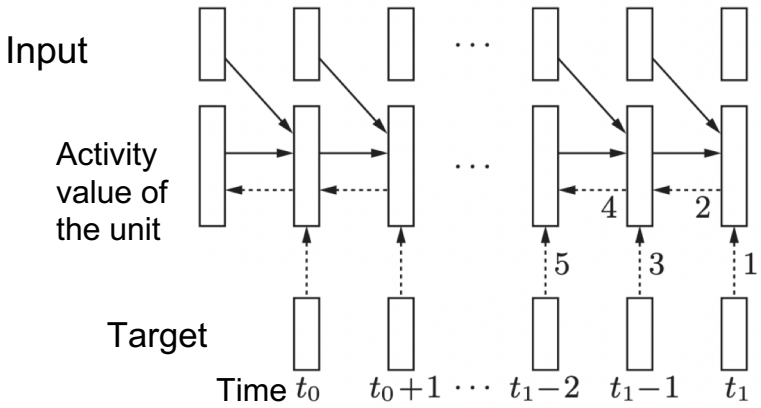
In the paper in which the BP method was presented, it was suggested that a simple regression-coupled network can be trained by a generalized error back propagation method, since it does not differ from a normal feed-forward neural network without regression coupling, except that the middle layer has more input information [7].

It is based on the idea that hierarchization is possible by fixing the coupling load and expanding the propagation of the output in the time direction (if the time step of the entire network is T, it can be regarded as a T-layer neural network consisting of nodes at each point in time).

[7] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. Learning representations by back-propagating errors. 1986

RNNs: gradient calculation

As a generalization, **BPTT** (back-propagation through time) and **RTRL** (real time recurrent learning) were proposed as algorithms that retain the series information for a certain time window and learn about the information within that time window.
In the usual BP method, the error is calculated only at the top layer, but in the BPTT method, the error is calculated at all times. This means that in addition to propagating the error in the top layer, the error in the top layer must be added to the error in the layer at each point in time, so if the number of nodes in the middle layer is N and the time step of the entire network is T, the order of computation is O(TN^2) and the order of memory is O(TN) This means that the order of computation is O(TN^2) and the order of memory is O(TN).
The RTRL method, on the other hand, does not need to go back in time and can be trained in real time (the weights at the next time t+1 are updated by the error generated at time t), but the order of computational complexity is O(TN^4) and the order of memory is O(TN^3), which is even larger. Proposed methods to reduce such computational complexity have also been studied.



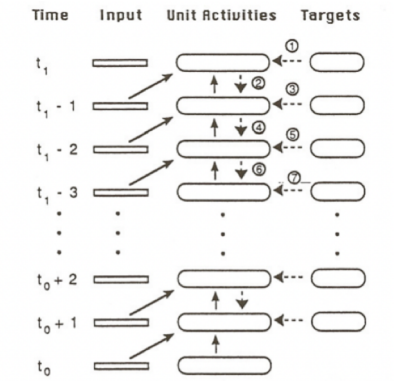


Figure 4. A schematic representation of the storage and processing required for epochwise BPTT. All input, unit output, and target values for every time step from t_0 and t_1 are stored in the history buffer. The solid arrows indicate how each set of unit output values is determined from the input and unit outputs on the previous time step. After the entire epoch is complete, the backward pass is performed as indicated by the dashed arrows. Each even-numbered step determines the virtual error from later time steps, while each odd-numbered step corresponds to the injection of external error. Once the backward pass has been performed to determine separate δ values for each unit and for each time step back to $t_0 + 1$, the partial derivative of the negative error with respect to each weight can then be computed.

https://web.stanford.edu/class/psych209a/ReadingsByDate/02_25/Williams%20Zipser95RecNets.pdf

RNNs: Learning Rules

The learning rule used in interconnected Hopfield networks is called the Hebbian rule (also called the outer product rule) [8], which is based on a rule that mimics the reversibility of synapses in the brain.

It realizes a mechanism in which the firing of successive front and rear nodes increases the value of the joint load connecting them, thereby increasing the transfer efficiency, while the lack of firing attenuates the transfer efficiency. In contrast, the BP method, which is a generalization of the delta rule, is a useful engineering method, but whether such a learning mechanism is actually possible in the brain is still in the research stage.

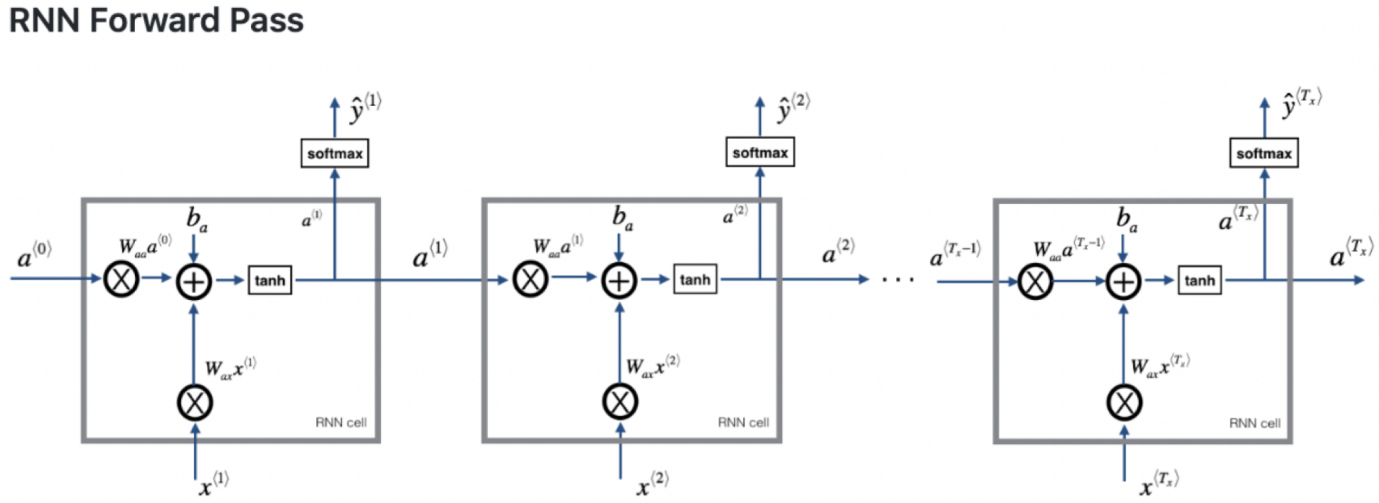
[8] Hebb, D. O. The organization of behavior. 1949

simple recurrent networks

: Network diagram

The feed-forward of simple reccurent networks (Elman-based), currently representing RNNs, can be represented as shown in the figure on the right.

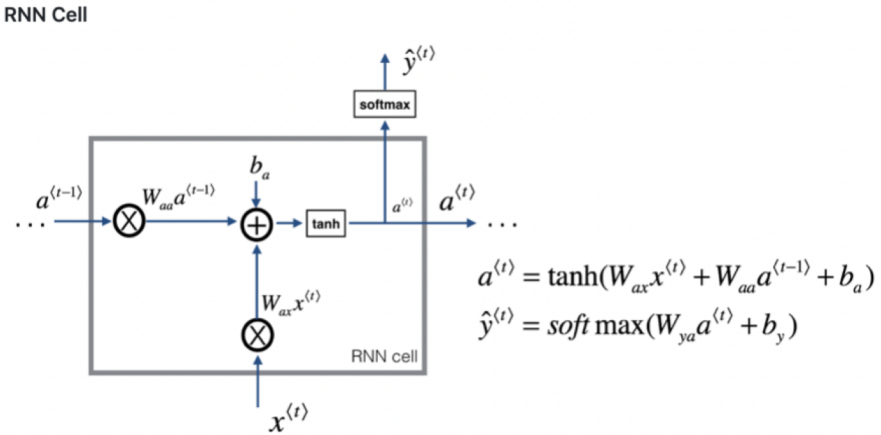
The state at t-1 is passed from the front quadrangle (at t-1) to the back quadrangle (at t).



simple recurrent networks

: Network diagram

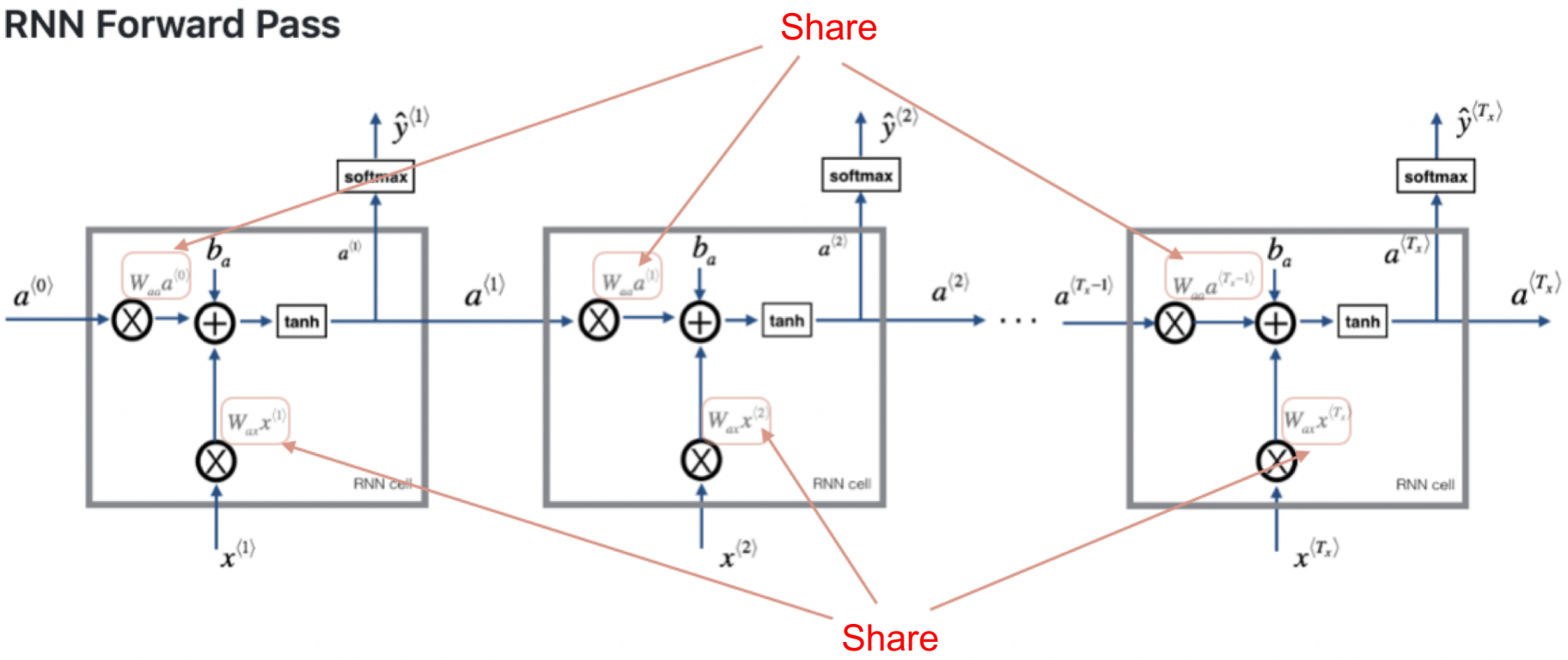
The sequence of rectangular ranges is an expansion of the same network (called a cell) in the direction of the time axis.
The output (a (t-1)) computed in the same network is input (a (t)) into itself, and the computation is repeated again.
It is called a regressive network because it has such a circular loop.
Two types of weights, W_ax and W_aa, are shared at each time.



RNN

simple recurrent networks

: Network diagram



Problem: Cannot learn long-term series

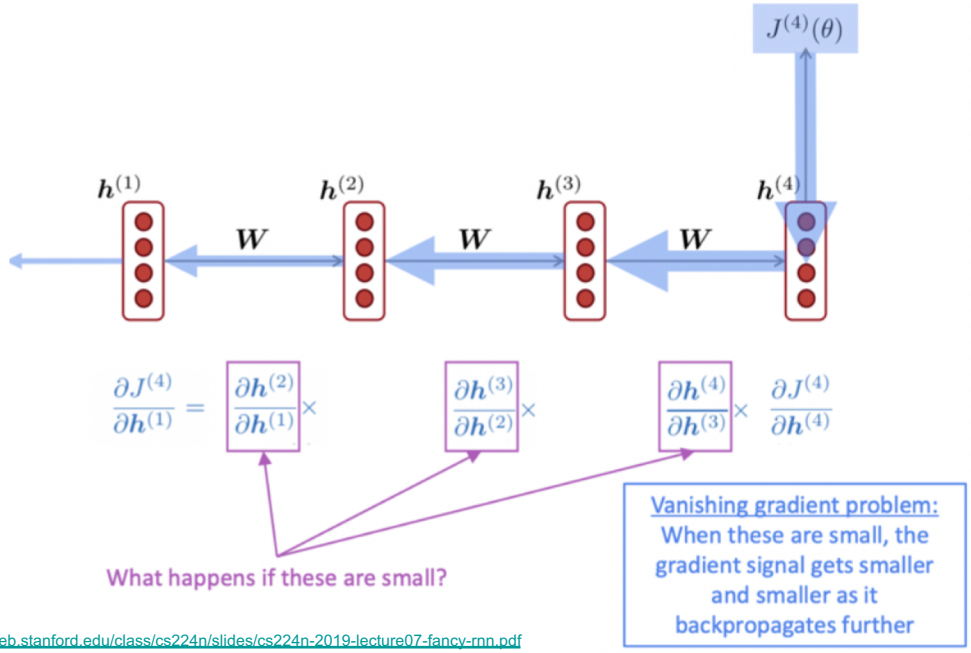
RNNs were theoretically thought to be capable of handling "long-term dependencies.

However, while it can handle short-term dependence of a few tens of steps, it is difficult to learn long-term series such as 1,000 steps.

Cause: Slope loss or slope explosion

Deep layered DNNs, not just RNNs, are prone to gradient loss or explosion in many situations. The reason why RNNs are prone to these is because all layers share the same weights [9].

[9] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, On the difficulty of training Recurrent Neural Networks. 2013



https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture07-fancy-rnn.pdf

Weight sharing: applying the same weight W multiple times in BPTT

After t steps, it becomes W. Let's think about what value this W will be.
For example, if we apply eigenvalue decomposition (decomposition into a diagonal matrix consisting of eigenvectors and eigenvalues) to the matrix W, we can decompose it as follows

diag(λ) is a diagonal matrix consisting of eigenvalues (a matrix in which all non-diagonal elements are zero). The eigenvalues, which are the diagonal elements, can be expressed as λi.
In this case, the t-squared of the matrix W is as follows

$$W^t = V \times diag(\lambda)^t \times V^{-1}$$

RNN

Weight sharing: applying the same weight W multiple times in BPTT

When the maximum eigenvalue of $W_t < 1$, the gradient shrinks exponentially

When the maximum eigenvalue of $W_t > 1$, the gradient diverges exponentially

Sample code

How to solve problems

“Recurrent Neural Network”

[Problem 1] Forward Propagation Implementation of SimpleRNN
[Problem 2] Experiment of Forward Propagation with Small Sequence
[Problem 3] (Advance Assignment) Implementation of Backpropagation

Sprint 22 – RNN

Explanation about this Sprint is given but please try it on your own first.

Sprint 22 – RNN

Please work on your own after class and submit your assignments on DIVER.

Sprint 22 – RNN

A Sample Code of this Sprint is given but please try it on your own.

Sprint 22 – RNN

Please work on your own after class and submit your assignments on DIVER.

ToDo by next class

Next class will be Zoom: Thursday 11 November 2021 19:30 ~ 20:30

🚨ToDo: LSTM
https://diver.diveintocode.jp/curriculums/2006

Check-out

3 minutes Please post the following point to Zoom chat.

Q. Current feelings and reflections
(joy, anger, sorrow, anticipation, nervousness, etc.)

Thank You For Your Attention