Lecture 7. Recurrent Neural Network

RNN and LSTM

Zhonghai Lu

KTH Royal Institute of Technology

April 16, 2024

Outline

1. Why Recurrent Neural Network (RNN)?

2. Vanilla RNN

3. Long Short Term Memory network (LSTM)

4. Various RNNs

5. Example of RNN computation and sentence completion

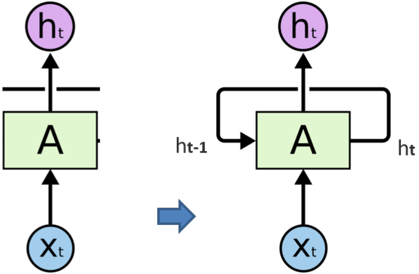
Feedforward NN

- Representation strength
  - Can use wide or deep layers to model arbitrarily complex non-linear functions between input vector and output vector.
- Representation weakness
  - Memory-less architecture, Output = F(Input).
  - Fixed-size input vector (e.g. image) mapped to fixed-size output vector (e.g. probabilities of different classes).
- Application limitation
  - Can be difficult to deal with sequence based applications such as language translation, voice recognition, text-to-voice interpretation, object tracking etc.
    - input/output sequences with possibly variable length
    - have to be aware of context (past and future data)

Basic concepts of RNN

To address sequence problems, why not introducing memory/state into NN? How?

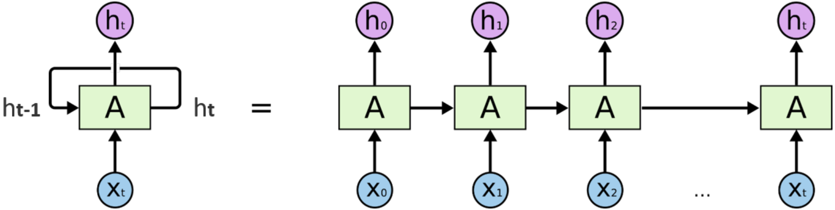
- Delayed feedback
- As history (state) information, current output goes back to the input at the next time step.



A neural network (NN) or fully connected NN layer A (in square box) takes in some input Xt and outputs a value or state ht.
A loop allows the state to be passed from one time step of network A to the next.

Unrolling RNN

Unrolling the loop gives a view as a normal NN architecture in a chain structure



By unfolding, the network is expanded in the time domain for the complete sequence.
An example in machine translation: If the sequence is a sentence of 4 words, the network operates as if being unrolled into a 4-layer neural network, one layer for each word.

RNN sharing layer over time

- Key point: Re-use the same NN cell with the same set of weights/biases at all time steps.
- Unlike a traditional deep NN, which uses different parameters at each layer, an RNN shares the same parameters across all time steps.

- The same function f is applied to transit the system state in different time steps.

$$h^t = f(h^{t-1}, x^t; \theta)$$

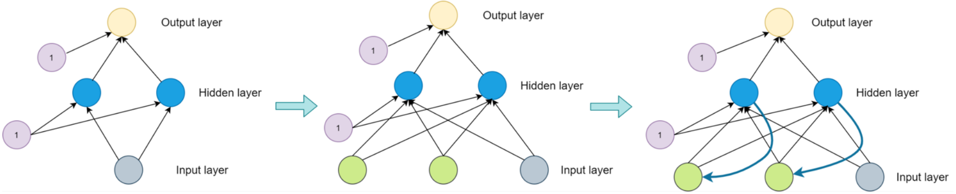
The length of sequence

- RNN usually has a maximum length of sequence.
    - In theory, RNN supports a sequence of arbitrary length. In practice, this is problematic.
    - If the sequence is too long, it can lead to the gradient vanishing or explosion problem.
- A too long sequence will take over-sized memory space.

MLP vs. RNN

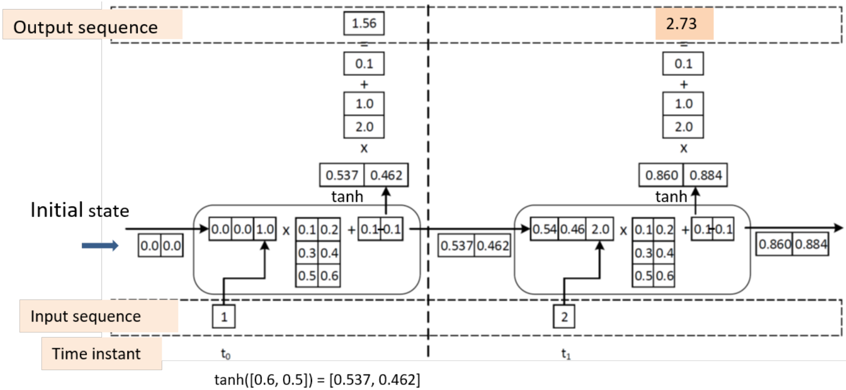
How is an RNN in contrast to MLP? Input-Hidden-Output

(a) 1-2-1 MLP. (b) 3-2-1 MLP. (c) 1-2-1 RNN.

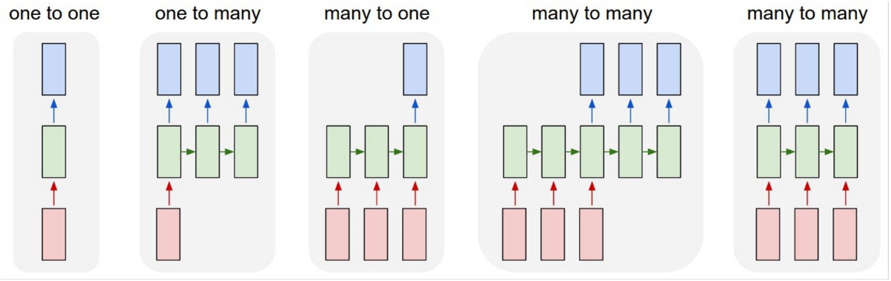


RNN inference

- The input is a concatenation of the current input (t) and the previous (t-1) state. The first input (t0) uses the initial state.
- RNN may generate output at each time instant. This means that the loss function shall be the sum of losses for all or partial times, depending on the application.



RNN dealing with sequences



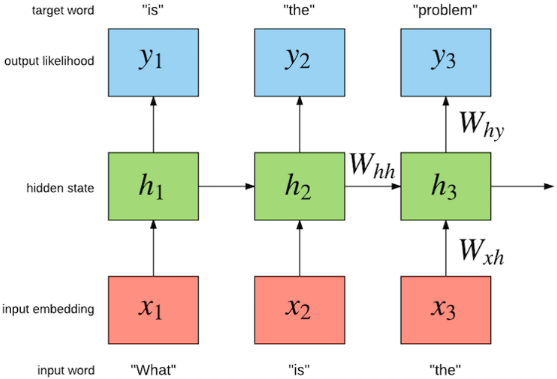
Each rectangle is a vector and arrows represent functions (e.g. matrix multiply).
Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state.

(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, no pre-specified constraints on the lengths of sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Andrej Karpathy. “The Unreasonable Effectiveness of Recurrent Neural Networks”
http://karpathy.github.io/2015/05/21/rnn-effectiveness

Word/Character predictor

- Task: Given a few characters or words as input, predict the next character or word?



Example from nndl slides. https://nndl.github.io/

Character predictor

Given a sequence of previous characters, build an RNN to predict the probability distribution of the next character in the sequence.

- Suppose we only had a vocabulary of four possible letters: “h”, “e”, “l”, o”.
- We had a training sequence “hello”.

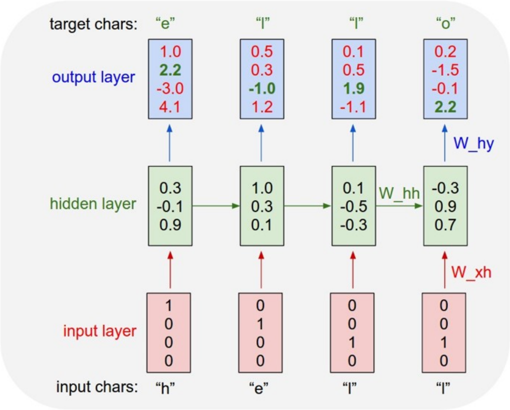
This training sequence can be split into 4 separate training instances.

- ”h” → ”e”: The probability of “e” should be likely given the context of “h”
- ”he” → ”l”: “l” should be likely in the context of “he”
- ”hel” → ”l”: “l” should also be likely given the context of “hel” (“l” followed by “l”)
- ”hell” → ”o”: “o” should be likely given the context of “hell” (“l” followed by “o”)

In summary, input sequence ”hell” → **target** output sequence: ”ello”

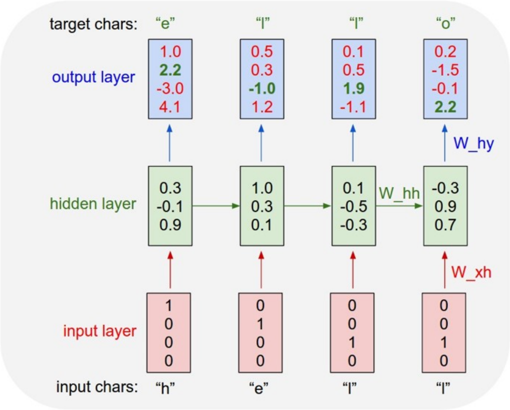
Character predictor

- Input embedding: Encode each character into a vector using 1-of-k encoding (i.e. all zero except a single one at the index of the character in the vocabulary). k is the number of characters.
- Training: Feed the embedded vectors for ”h”, ”e”, ”l”, ”l” into the RNN one at a time.
- Output interpretation: A 4-dimensional output vector, one dimension per character interpreted as the confidence the RNN predicts to appear.



Character predictor

- In the first time step, when the RNN sees “h”, it assigns confidence of 1.0 to the next letter being “h”, 2.2 to letter “e”, -3.0 to “l”, and 4.1 to “o”. ”h” → ”o”.
- Since in our training data “hello”, the next correct character is “e”, the RNN should increase its confidence (green one) and decrease the confidence of all other letters (red ones).
- Similarly, we have a target character at every one of the 4 time steps that we’d like the RNN to assign a greater confidence to.



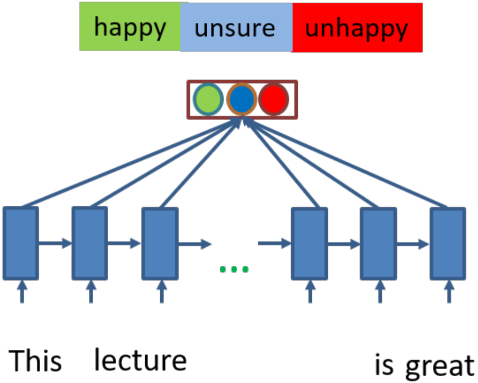
Character predictor

- Run the error back-propagation through time algorithm (BPTT) (a recursive application of the chain rule of derivative) to calculate error gradients with respect to weights.
- Weight update: adjusts every weight a small amount in the negative gradient direction to increase the scores of the correct targets (green bold numbers).

  Now, if we feed the same inputs to the RNN, the scores of the target characters (e.g. “e” in the first time step) would be higher (e.g. 2.4 instead of 2.2), and the scores of incorrect characters would be lower.
- Repeat this process over and over many times until the network converges and its predictions are eventually consistent with the training data.

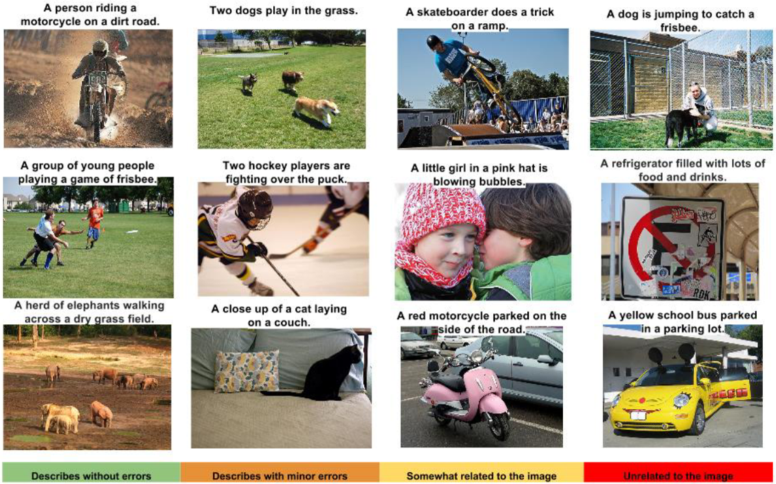
Many-to-one application

Many-to-one application: Sentiment analysis, musical genre analysis, bird song identification etc.



One-to-many application

One-to-many application: e.g. from image to text



A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A skateboarder does a trick on a ramp.

A dog is jumping to catch a frisbee.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A little girl in a pink hat is blowing bubbles.

A refrigerator filled with lots of food and drinks.

A herd of elephants walking across a dry grass field.

A close up of a cat laying on a couch.

A red motorcycle parked on the side of the road.

A yellow school bus parked in a parking lot.

🟩 Describes without errors

🟧 Describes with minor errors

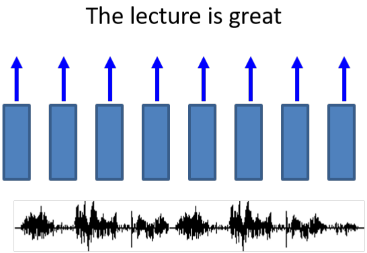
🟨 Somewhat related to the image

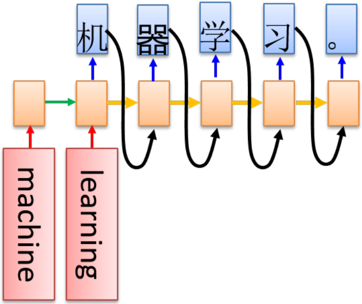
🟥 Unrelated to the image

Many-to-many application

From sequence to sequence:

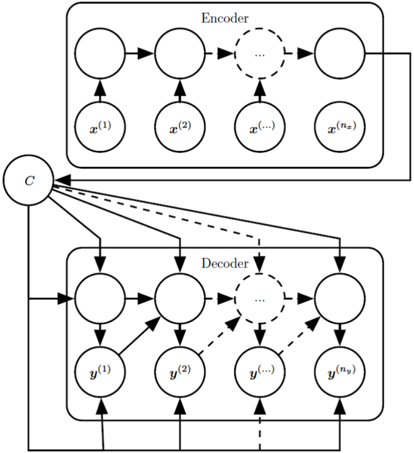
- Voice recognition (from voice to text)
- Machine translation (from e.g. English to Chinese)





Encoder-Decoder

- Sequence-to-sequence handled by an encoder-decoder architecture.
- A summary vector C is generated in between.



Short-term dependency

- Short range dependency works well in vanilla RNN.
- Example: Predict the last word in a sentence. ”Fish is in water”. Short range context is sufficient.

Long-term dependency

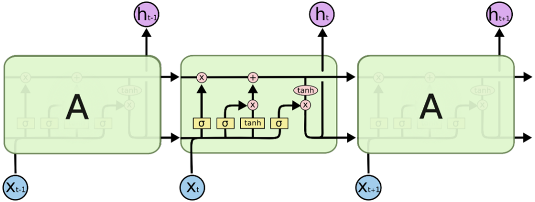
- Example: Predict a word in a text. ”He was born in China. He has done many great things. . . . . He speaks Chinese. ” – Need to capture a large context.
- It is possible but difficult to train a vanilla RNN to capture long-term dependency due to gradient vanishing/explosion (e.g. Multiplying a number of fractional numbers results in nearly 0).
- Gradient vanishing/exploding is a common problem for deep neural networks, but worse for RNNs due to sharing the same weights (in time).

LSTM

- Long Short Term Memory networks (LSTMs) are a special kind of RNN capable of learning long-term dependencies much better than RNNs.
- LSTM tackles the long-term dependency problem by introducing gated control of input, current state (forget or not), and output.

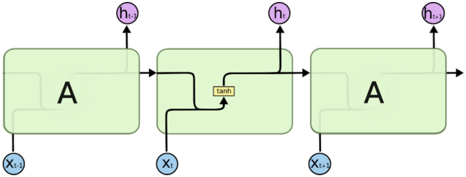
LSTM

- Long Short Term Memory networks (LSTMs) are a special kind of RNN capable of learning long-term dependencies much better than RNNs.
- LSTM tackles the long-term dependency problem by introducing gated control of input, current state (forget or not), and output.

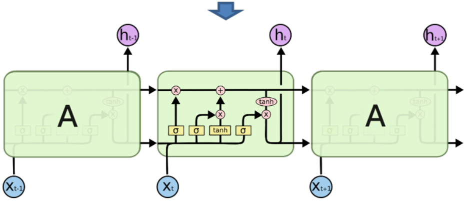


RNN vs. LSTM

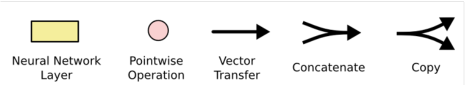
- LSTMs also have the chain-like structure, but the repeating module (often called cell) has a different structure.
- Instead of having a single neural network layer, there are four, interacting in a very special way.



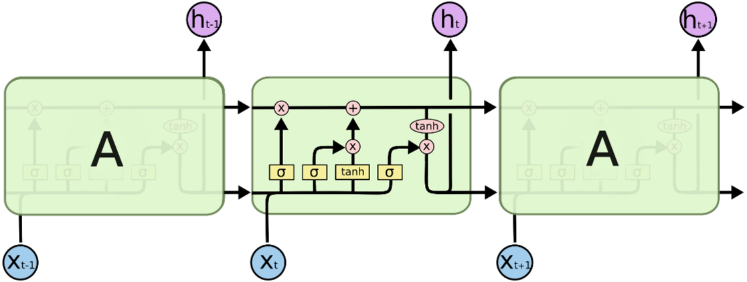
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.



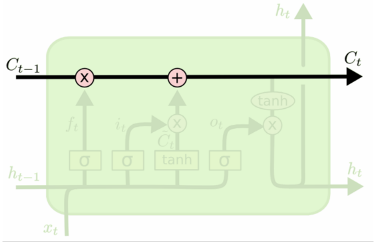
LSTM cell

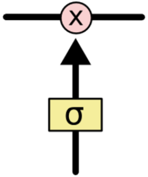


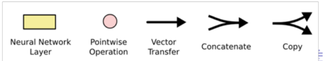
- At each time step, LSTM has 3 inputs: X_t, h_{t-1}, C_{t-1} and two outputs h_t, C_t.
- The state on the upper line (belt) C_t represents long-term memory, while the state on the lower line h_t records working memory or short-term memory.

Long-term memory representation and control

- The LSTM cell has a cell state for long time effect. It runs down the entire chain, with only some minor linear interactions.
- Gates control adding or removing information to the cell state.
  - A gate comprises a sigmoid network layer and a point-wise multiplication operation.
  - The sigmoid layer outputs numbers between zero and one, determining how much of each component should be let through.
    Zero for “let nothing through,” one “let everything through”
- An LSTM has three gates, to protect and control the cell state, sequentially.

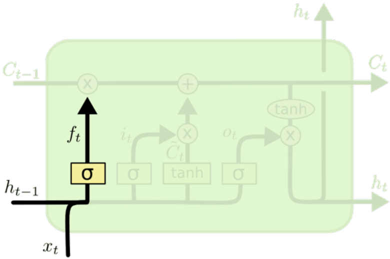




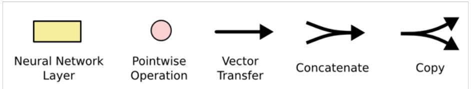


Forget gate control

Stage 1. The *forget gate layer* decides what information to throw away from the cell state.



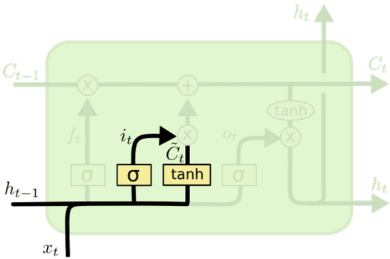
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



Input gate control (1)

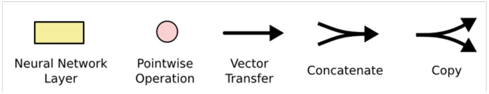
Stage 2. The *input gate layer* decides what new information to store in the cell state.

1. it first decides which values to update, and a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
2. then, combine these two to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

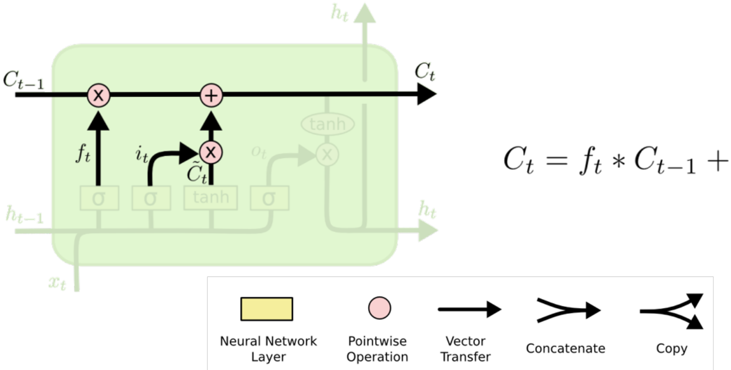
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



Input gate control (2)

Stage 2. The *input gate layer* decides what new information to store in the cell state.

1. it first decides which values to update, and a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
2. then, combine these two to create an update to the state.

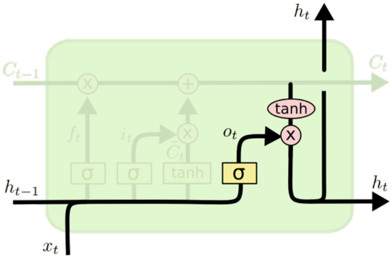


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate control

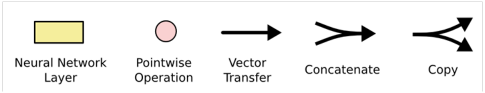
Stage 3. The *output gate layer* decides what to output, based on the cell state, but a filtered version.

1. Run the sigmoid gate layer deciding what parts of the cell state to output.
2. The cell state goes through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate.



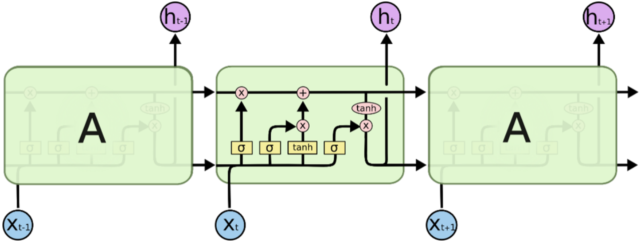
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



Information flow controlled by gates

- The cell state is implicit, which can be reset, kept, or modified at each time step.
- The cell state further controls the hidden state generation at each time step.



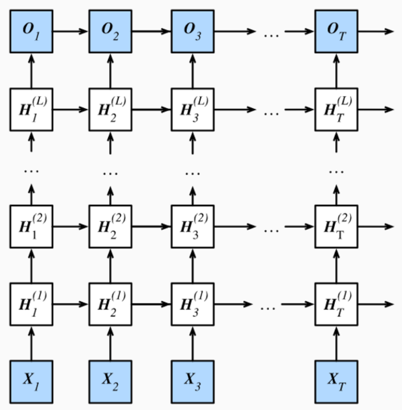
Action	Forget gate	Input gate	Output gate
Memory reset to 0	0	0	
Memory state kept	1	0	
Memory write	0	1	
switch on hidden state h_t			1
switch off hidden state h_t			0

Variants of RNN

- Deep RNN
- Bi-directional RNN
- Recursive Neural Network (RvNN)
- NN with explicit memory

Deep RNN

Each hidden state is continuously passed to both the next time step of the current layer and the current time step of the next layer.



$$\mathbf{H}_t^{(1)} = f_1\left(\mathbf{X}_t, \mathbf{H}_{t-1}^{(1)}\right)$$
$$\mathbf{H}_t^{(l)} = f_l\left(\mathbf{H}_t^{(l-1)}, \mathbf{H}_{t-1}^{(l)}\right)$$

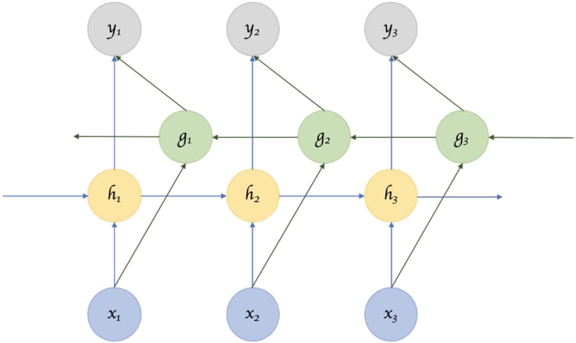
$$\mathbf{O}_t = g\left(\mathbf{H}_t^{(L)}\right)$$

$\mathbf{H}_t^{(\ell)}$ is the hidden state of hidden layer ℓ (ℓ=1,...,T)

Dive into Deep Learning. `https://d2l.ai/chapter_recurrent-neural-networks/`

Bi-directional RNN

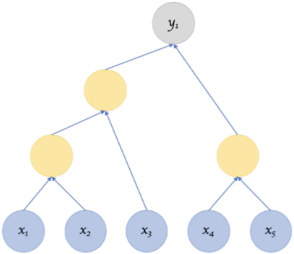
- Often we need information about both past and future to determine the present.
- Example: Fill a missing word (not necessarily the last one) in a sentence.
  He is Happy because he wins the competition.

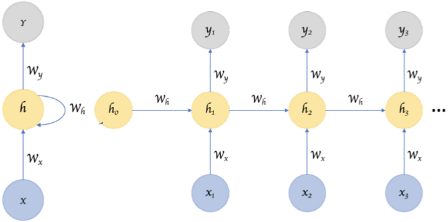


https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

Recursive RNN (RvNN) vs. RNN

- Recursive Neural Network (RvNN) applies the same set of weights recursively over a structured input, to produce output.
    - It can operate on any hierarchical tree structure, parsing through input nodes, combining child nodes into parent nodes and combining them with other child/parent nodes to create a tree-like structure.
- RNNs do the same, but the structure is strictly linear. i.e. weights are applied on the first input, then the second, third and so on.

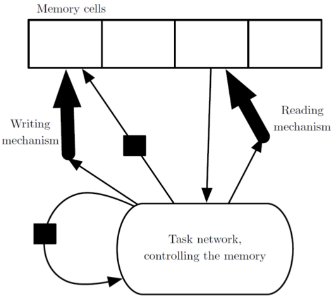


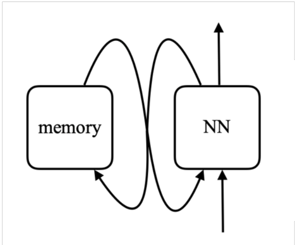


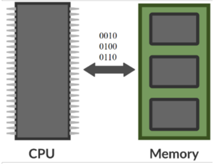
NN with explicit memory

From implicit memory to explicit memory:

- Some knowledge can be stored explicitly and fetched for use when needed, rather than computed each and every time when needed.
- Enable storage intelligence besides computational intelligence.







Demo

- RNN calculation example. `http://localhost:8888/notebooks/IL2233VT22/Lec7_rnn/RNN_calculation_example.ipynb`
- Character predictor `http://localhost:8888/notebooks/IL2233VT22/Lec7_rnn/RNN_walkthrough.ipynb`

Character predictor

Data set: {Hi How are you?}, {I am fine. Thank you!}, {How do you feel today?}
Tasks: Build a predictor to enable sentence completion.

- One-step predictor: Predict the next character, given one or more characters as input
  I ⇒ a, I a ⇒ m, I am ⇒ f
- Multi-step predictor: Use the one-step prediction result iteratively to predict the next characters
  I am ⇒ fine
  Four iterative steps to predict the next four characters:
  1. I am ⇒ f
  2. I am f ⇒ i
  3. I am fi⇒ n
  4. I am fin⇒ e

Character predictor: Data re-organization

Building an RNN/LSTM to realize the **one-step predictor**:

1. Re-organize the data samples to suit supervised learning.
   - Find the maximum length, Lmax, of the samples.
   - Organize each sample into input sequence and target output sequence (truth label) with equal length (Lmax-1) (1 is step length).
     For shorter samples, padding space, or zero to reach the equal length.
     RNNs are typically able to take in variably sized inputs. In order to do batch processing, we ensure that all sequences are of equal size.
   - Example: Here all input/output sequences will be aligned to 21 (22-1) characters.

Sample	Input sequence	Output sequence
Hi How are you? (15 characters)	Hi How are you	i How are you?
I am fine. Thank you! (21 characters)	I am fine. Thank you	am fine. Thank you!
How do you feel today? (22 characters)	How do you feel today	ow do you feel today?

Table: Create training samples from the data set

Character predictor: Input embedding

1. Input embedding: Map characters to numbers
    - Create a character library and count the number K of unique characters.
    - Encode each character using, e.g. one-hot encoding.
      Each character is represented by a K-dimensional vector with only one dimension being 1 and others being 0.

Character predictor: NN modeling

1. NN modeling: Build an RNN/LSTM model
   - Input layer: How many neurons?
     Since each time, only one character is entered into the model. the number of input neurons equals the number of dimensions of the encoded character, K.
   - Hidden layer: How many neurons? This is a design choice.
   - Output layer: How many neurons?
     Since each time one character is out, the number of output neurons equals to that of the input neurons.
2. Training: training epoch, learning rate, loss function, optimizer.
   One epoch means going through the entire training dataset once.
3. Inference: One-step or multi-step prediction

How good is the model? Perfect predictions, however just Overfitting!

Summary

- RNN processes sequence data by passing hidden states to the next time step of processing.
- RNN uses the same set of weights for all time steps. The length of time steps depends on the length of input sequence.
- RNNs can be trained by error back-propagation through time with gradient descent optimization.
- RNN is more difficult to capture long-range dependency due to the vanishing/exploding gradient problem.
- LSTM introduces the cell state and uses 3 gates to allow earlier state information to generate longer impact in time.
- The simplest vanilla form of RNNs has expanded to different directions, such as deep RNN, bidirectional RNN and Recursive NN.

References

- The Unreasonable Effectiveness of Recurrent Neural Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness`
- An Introduction to Recurrent Neural Networks. `https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912`
- Colah's blog. Understanding LSTM Networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`
- Good animation about RNN: `https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/`
- nndl and slides (in Chinese). `https://nndl.github.io/`
- Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola. “Dive into Deep Learning”. Release 0.7. 2019. `https://d2l.ai/`
- Goodfellow et al. Deep Learning. MIT press. 2017.
- Yann Lecun, `https://cds.nyu.edu/deep-learning/`
- RNN for character/word prediction `https://github.com/gabrielloye/RNN-walkthrough/blob/master/main.ipynb`