Recurrent Neural Networks

อ. ปรัชญ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

Today

- Recurrent Neural Networks
- Backpropagation through time
- Types of RNN
- Vanishing Gradient Problem
- GRU
- LSTM
- Bidirectional RNN
- Deep RNN

Recurrent Neural Network

- เป็น Neural Network ชนิดหนึ่งที่เหมาะสำหรับใช้กับข้อมูลที่เป็นแบบ **sequential**
- ข้อมูลแบบ sequential คือข้อมูลแบบ array ที่มีลำดับ โดยค่าอาจเปลี่ยนไปตามเวลา t
  - sequential input X อยู่ในรูป $x^{<1>}, x^{<2>}, \dots, x^{<t>}$
  - sequential output $\hat{y}$ อยู่ในรูป $\hat{y}^{<1>}, \hat{y}^{<2>}, \dots, \hat{y}^{<t>}$

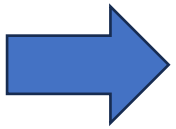
- **ตัวอย่าง:** X เป็นประโยคภาษาไทย
  - $x^{<1>} =$ ฉัน
  - $x^{<2>} =$ ชอบ
  - $x^{<3>} =$ สุนัข

Note: RNN อาจจะมี input ไม่ก็ output เป็นแบบ sequential, หรือจะเป็น sequential ทั้งคู่ก็ได้

Recurrent Neural Network

- โดยทั่วไป **RNN** มักจะรับ **input** X แบบ **sequential** ในรูป $x^{<1>}, x^{<2>}, \ldots, x^{<t>}$
- แล้ว **output** Y อาจเป็นได้หลายรูปแบบ เช่น
  - categorical เช่น $Y \in \{\text{spam}, \text{regular}, \text{important}\}$
  - numerical เช่น Y เป็นระดับความทุกข์ของผู้พูด
  - sequential เช่น $Y =$ “I like dogs”

- ตัวอย่าง: X เป็นประโยคภาษาไทย
  - $x^{<1>} =$ ฉัน
  - $x^{<2>} =$ ชอบ
  - $x^{<3>} =$ สุนัข



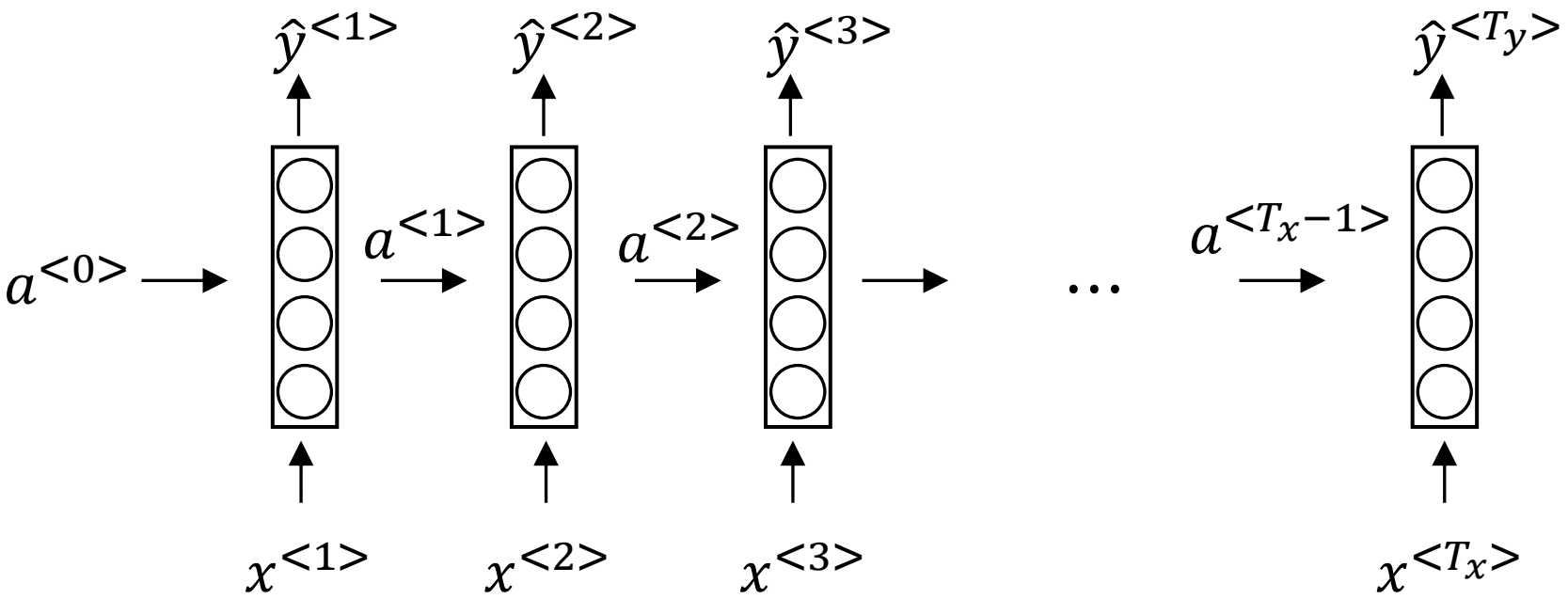
RNN Applications

- Natural Language Processing (NLP) tasks
  - Sentiment Analysis
  - Machine Translation
  - Speech Recognition
  - Name-Entity Recognition
  - Question & Answering
- Time Series (e.g., stock market prediction)
- Video Classification
- Music Generation
- ...และอีกมากมาย

Why not feedforward NN?

- ทำไม Feedforward NN จึงไม่เหมาะกับ Sequential Data
- **Problems**
  - ขนาดของ input/output แต่ละ instance อาจจะไม่เท่ากันได้
    - “I love dogs”, “I like to pet dogs”, “Please let me hug the dog”
  - ขาดการแชร์ features ที่เรียนรู้มาระหว่าง $x^{<t>}$ ในแต่ละตำแหน่ง
  - ต้องใช้ parameter จำนวนมหาศาล เนื่องจาก X อาจมีขนาดใหญ่มากได้
    - เช่น หากเราเก็บแต่ละคำ (word) ในรูปแบบ one-hot vector
    - จะได้ว่าขนาดของ X = dict_size * max_sentence_length

RNN Model



https://www.coursera.org/learn/nlp-sequence-models/

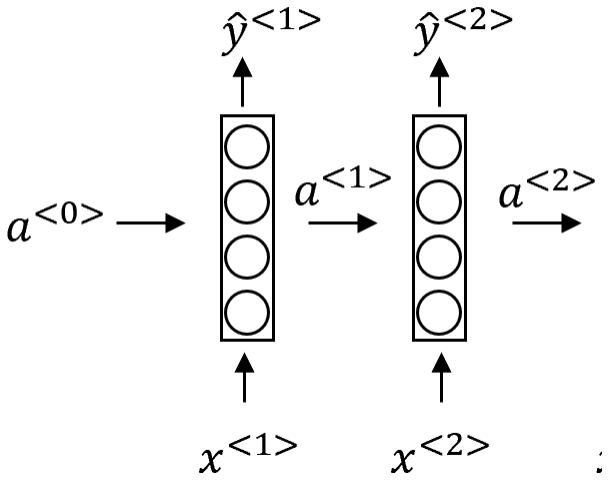
RNN Model Notations & Explanation

- Activation $a^{<t>}$ เป็น hidden state/memory ของ RNN
  - $a^{<t>}$ จะถูกส่งต่อไปใน step ถัดไปเรื่อยๆ (เกิดเป็น $a^{<t+1>}, a^{<t+2>}, \ldots$)
  - ทำให้ RNN สามารถหาคำตอบ $\hat{y}^{<t>}$ โดยอาศัยข้อมูลที่จำมาจาก step ก่อนๆ ได้

- Weight matrices
  - W_{ax} คือ weight ที่นำไปคูณกับ input $x^{<t>}$ เพื่อคำนวณหาค่า activation $a^{<t>}$
  - W_{aa} คือ weight ที่นำไปคูณกับ activation $a^{<t-1>}$ เพื่อคำนวณหาค่า activation $a^{<t>}$
    - จะจำมากน้อยแค่ไหน
  - W_{ya} คือ weight ที่นำไปคูณกับ activation $a^{<t>}$ เพื่อคำนวณหาค่า output $y^{<t>}$

RNN Equations

- เราสามารถเขียนการคำนวณ step แรก $(t = 1)$ ของ RNN model ในรูปสมการ ได้ดังนี้
  - เริ่มจาก $\boldsymbol{a}^{<0>} = \vec{0}$
  - หาการรับ memory จาก step ก่อนหน้า:
    - $\boldsymbol{a}^{<1>} = g_1(W_{aa}\boldsymbol{a}^{<0>} + W_{ax}\boldsymbol{x}^{<1>} + b_a)$
  - หา output:
    - $\hat{\boldsymbol{y}}^{<1>} = g_2(W_{ya}\boldsymbol{a}^{<1>} + b_y)$



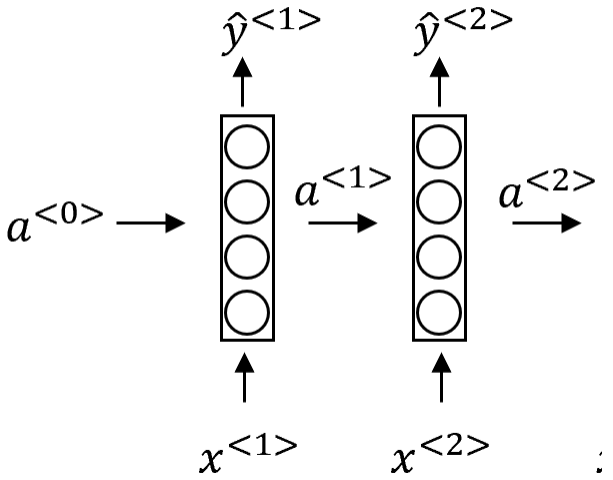
- ดังนั้น สมการสำหรับ step t ใดๆ คือ
  - $\boldsymbol{a}^{<t>} = g_1(W_{aa}\boldsymbol{a}^{<t-1>} + W_{ax}\boldsymbol{x}^{<t>} + b_a)$
  - $\hat{\boldsymbol{y}}^{<t>} = g_2(W_{ya}\boldsymbol{a}^{<t>} + b_y)$

Note/ทบทวน:
Activation Function $g(\,)$ ใช้สำหรับใส่ความ non-linear ให้กับ model (เหมือนใน Feedforward NN)

RNN Equations

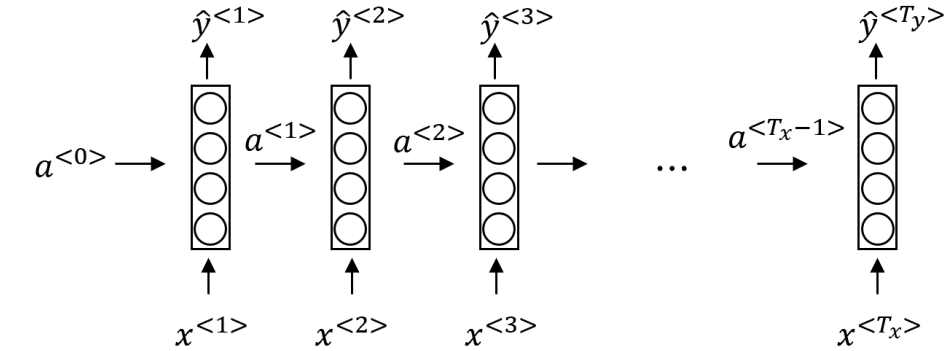
สำคัญ: สังเกตว่า ทุก step ใช้ W เดียวกัน

- เราสามารถเขียนการคำนวณ step แรก $(t = 1)$ ของ RNN model ในรูปสมการ ได้ดังนี้
  - เริ่มจาก $\boldsymbol{a}^{<0>} = \vec{0}$
  - หาการรับ memory จาก step ก่อนหน้า:
    - $\boldsymbol{a}^{<1>} = g_1(W_{aa}\boldsymbol{a}^{<0>} + W_{ax}\boldsymbol{x}^{<1>} + b_a)$
  - หา output:
    - $\hat{\boldsymbol{y}}^{<1>} = g_2(W_{ya}\boldsymbol{a}^{<1>} + b_y)$



- ดังนั้น สมการสำหรับ step t ใดๆ คือ
  - $\boldsymbol{a}^{<t>} = g_1(W_{aa}\boldsymbol{a}^{<t-1>} + W_{ax}\boldsymbol{x}^{<t>} + b_a)$
  - $\hat{\boldsymbol{y}}^{<t>} = g_2(W_{ya}\boldsymbol{a}^{<t>} + b_y)$

How to train RNN?

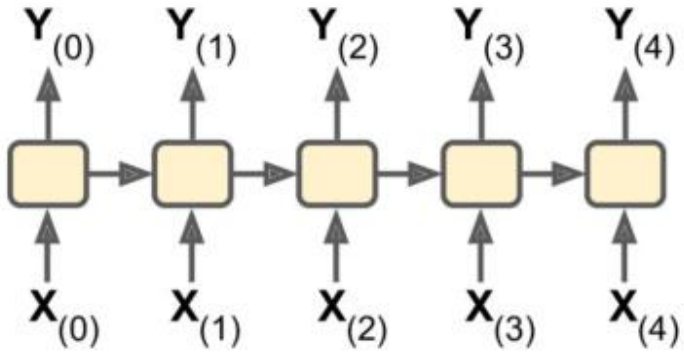


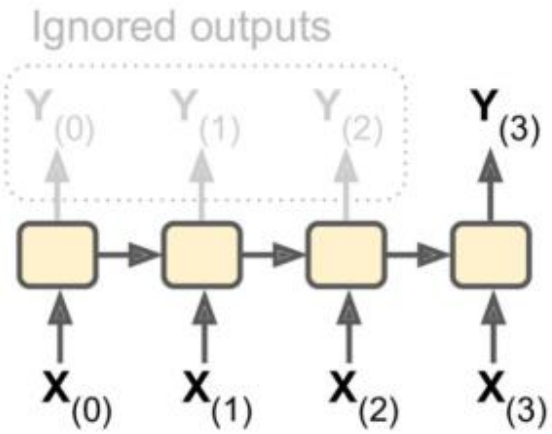
- ในการ train RNN เราจะใช้วิธี backpropagation เหมือนเดิม
  - forward pass: คำนวณตามสไลด์ก่อนหน้า นำ output $\hat{y}$ ไปเทียบกับเฉลย y เพื่อหา loss
  - backward pass: ส่ง gradient ของ loss ย้อนกลับไป update W ทุกตัว

- ต่างจาก backprop ใน feedforward NN อย่างไร?
  - ค่า total loss จะต้องรวม loss ที่เกิดจากทุก time step
    - $L(\hat{y}, y) = \sum_{t=1}^{T} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$
    - note: โดยที่ $L^{<t>}$ อาจจะเป็น logloss หรือ cross-entropy ตามปกติ
  - gradient ของ total loss จะถูกส่งย้อนกลับไป จากขวา -> ซ้ายใน unrolled network
  - ด้วยเหตุนี้จึงเรียกว่า backprop “through time” (BPTT)

Types of RNN

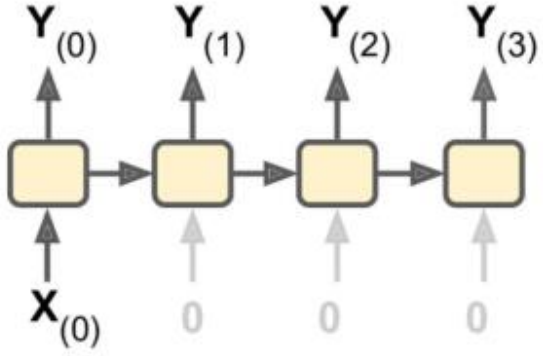
Name-Entity Recognition

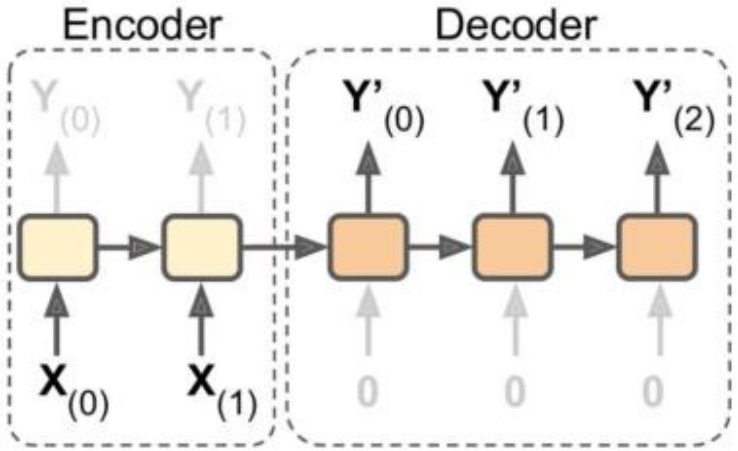




Sentiment Analysis

Image Captioning
Music Generation





Machine Translation

Hands-On Machine Learning with Scikit-Learn and TensorFlow, Geron

Vanishing Gradient Problem

- **ทบทวน:** ปัญหา vanishing gradient ใน feedforward NN
  - เกิดใน backward pass
  - **ปัญหา:** ค่า gradient ลดลงอย่างรวดเร็วระหว่างถูกส่งย้อนกลับไปใน layer แรกๆ
  - **ผลเสีย:** layer แรกๆ การเรียนรู้หยุดชะงัก
  - **สาเหตุ:**
    - NN มีจำนวน layer สูง (deep มากๆ)
    - ใช้ activation function ที่มีการบีบค่า (เช่น sigmoid)

Vanishing Gradient Problem in RNN

- ภาษามักมี long-term dependency ในข้อมูล เช่น
  - The **cat**, which sat on the sofa next to the kitchen, **was** full.
  - The **cats**, which sat on the sofa next to the kitchen, **were** full.

- และเนื่องจากใน BPTT นั้น gradient จะต้องถูกส่งย้อนกลับไปหลาย time step
  - จึงทำให้เกิดปัญหา vanishing gradient ได้ ไม่ต่างกับ deep feedforward NN
  - input “cat/cats” ในอดีตไกลๆ ก็จะไม่สามารถส่งอิทธิพลต่อ step ปัจจุบันได้
  - พูดง่ายๆ คือ model เกิดการ “ลืม” ได้ง่าย

Gated Recurrent Unit (GRU)

- Proposed by Cho et al., 2014
- ให้ $c^{<t>}$ เป็น memory cell โดยที่ $c^{<t>} = a^{<t>}$
- และ $\tilde{c}^{<t>}$ คือผลจากการนำ $c^{<t-1>}$ และ $x^{<t>}$ มาสร้าง memory ใหม่ (เป็น candidate)
  - $\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$
  - เช่น $\tilde{c}^{<t>}$ อาจจะใช้จำว่าประโยคนี้มีประธานเป็น plural หรือ singular

- main idea: เพิ่ม update gate Γ_u สำหรับควบคุมการจำ/ลืมข้อมูล
  - $\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$

- สุดท้ายจึง update memory cell: $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$

Full GRU Model

- $\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$
- $\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$
- $\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$ << เพิ่ม reset gate
- $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$

Why add Γ_u and Γ_r?

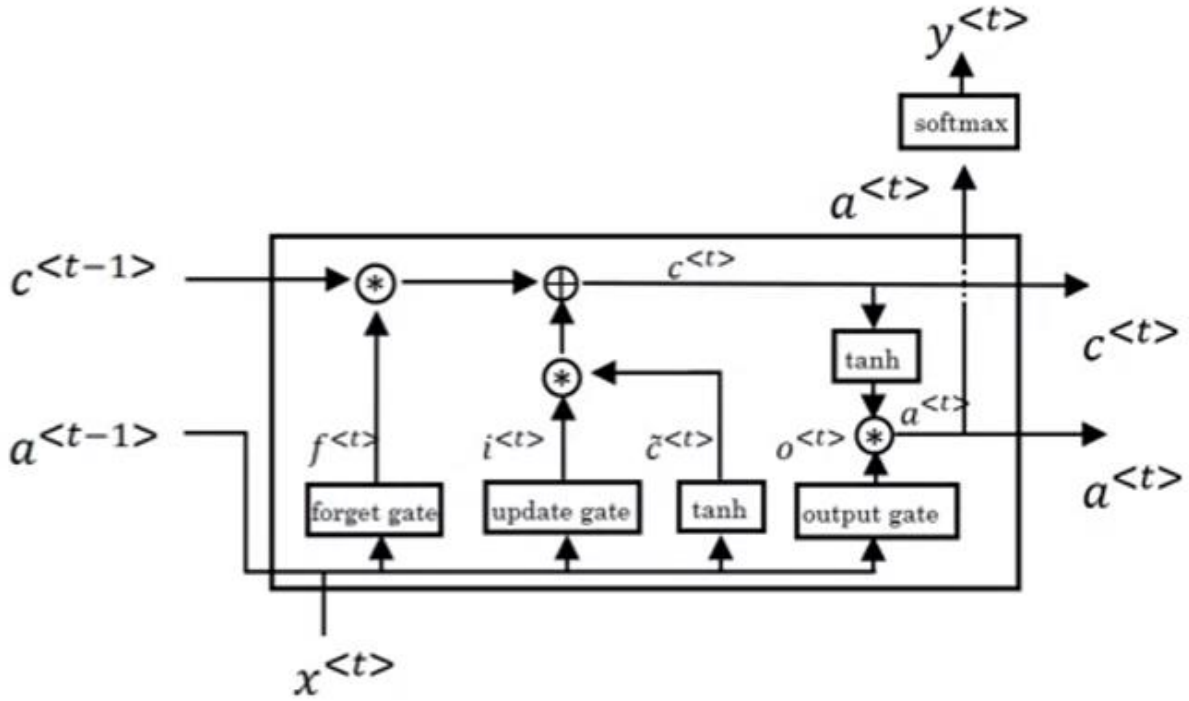
- ช่วยให้ model สามารถเลือกจำข้อมูลในอดีตเท่าที่จำเป็นต่อการทำนาย ทำให้สามารถรับมือกับข้อมูลที่มี long-term dependency ได้
- ช่วยลดการหายของ gradient ใน backprop ได้

Long-Short Term Memory (LSTM)

- Hochreiter & Schmidhuber, 1997
- เป็น model ที่เป็นที่นิยมมากในยุคก่อน Transformer
- มีความซับซ้อนกว่า GRU (ถึงแม้ว่าเก่ากว่า)
  - เพิ่ม forget gate Γ_f แทนการใช้ $1 - \Gamma_u$
  - เพิ่ม output gate Γ_o สำหรับแปลง memory $c^{<t>}$ เป็น activation $a^{<t>}$
    - ต่างจากใน GRU ที่ให้ $c^{<t>} = a^{<t>}$

Long-Short Term Memory (LSTM)

- $\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$
- $\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$
- $\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$
- $\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$
- $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
- $a^{<t>} = \Gamma_o * \tanh(c^{<t>})$



Bi-directional RNN

- WIP

Deep RNN

- WIP