6. Recurrent Neural Networks

היתרון של שכבות קונבולוציה על פני FC הוא ניצול הקשר המרחבי שיש בין איברים שונים בדאטה, כמו למשל פיקסלים בתמונה. יש סוגי דאטה בהם האיברים השונים יוצרים סדרה שיש לסדר האיברים חשיבות, כמו למשל טקסט, גלי קול, רצף DNA ועוד. כמובן שדאטה מהסוג הזה דורש מודל הנותן חשיבות לסדר של האיברים, מה שלא קיים ברשתות קונבולוציה. בנוסף, הרבה פעמים הממד של הקלט לא ידוע או משתנה, כמו למשל מספר המילים במשפט, וגם לכך יש לתת את הדעת. כדי להתמודד עם אתגרים אלו יש לבנות ארכיטקטורה שמקבלת סדרה של וקטורים ומוציאה וקטור יחיד, כאשר הווקטור היחיד מקודד קשרים על הדאטה המקורי שנכנס אליו. את וקטור המוצא ניתן להעביר בשכבת FC או בכל מסווג אחר, תלוי באופי המשימה.

6.1 Sequence Models

6.1.1 Vanilla Recurrent Neural Networks

רשתות רקורסיביות הן הכללה של רשתות נוירונים עמוקות, כאשר הן מכילות משקולות המאפשרות להן לתת משמעות לסדר של איברי הכניסה. ניתן להסתכל על משקולות אלה כרכיב זיכרון פנימי, כאשר כל איבר שנכנס משוקלל ביחס לפונקציה קבועה בתוספת רכיב משתנה שתלוי בערכי העבר. כאשר נכנס וקטור x, הוא מוכפל במשקל w_{xh} ונכנס לרכיב זיכרון h_t, כאשר h_t הוא פונקציה של x_t, h_{t-1}:

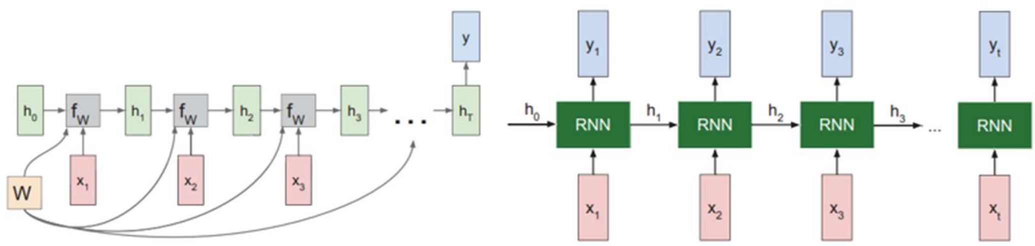
$$h_t = f(h_{t-1}, x_t)$$

מלבד המשקלים הפועלים על וקטור הכניסה, יש גם משקלים שפועלים על המשקולות הפנימיות (רכיב הזיכרון) – w_{hh}, ומשקלים הפועלים על המוצא של רכיב זה – w_{hy}. המשקלים w_{hx}, w_{hh}, w_{hy} זהים לכל השלבים, והם מתעדכנים ביחד. כמו כן, הפונקציה f_w היא קבועה לכל האיברים, למשל tanh, sigmoid או ReLU. באופן פורמלי התהליך נראה כך:

$$h_t = f_W(w_{hh}h_{t-1} + w_{xh}x_t), f_w = tanh/ReLU/sigmoid$$

$$y_t = w_{hy}h_t$$

ובאופן סכמתי ניתן לתאר את התהליך באופן הבא:



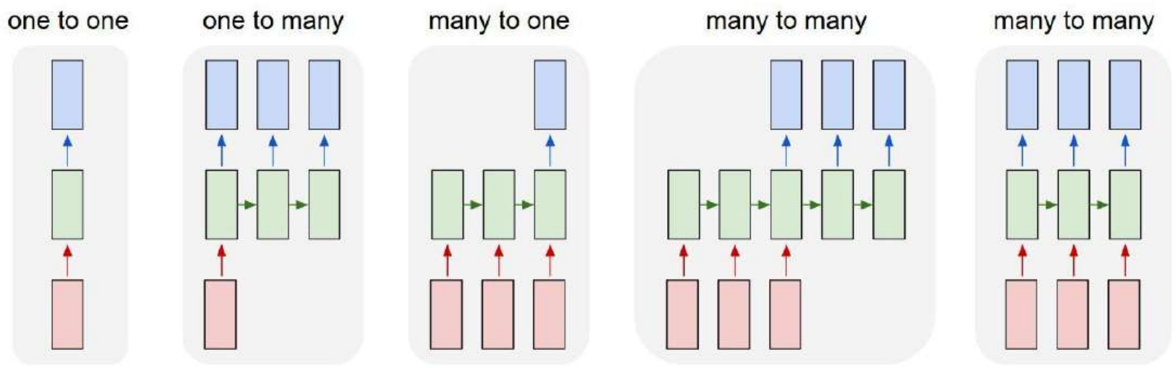
איור 6.1 ארכיטקטורות RNN בסיסיות: Many to Many (מימין) ו-Many to One (משמאל). על כל חץ יש משקל מתאים עליו מתבצעת הלמידה.

כמובן שניתן גם לשרשר שכבות חבויות ולקבל רשת עמוקה, כאשר פלט של שכבה מסוימת הופך להיות הקלט של השכבה הבאה. ישנם מודלים שונים של RNN, המתאימים לבעיות שונות:

One to many – יש קלט יחיד ורוצים להוציא סדרת פלטים, למשל מכניסים תמונה לרשת ורוצים משפט שיתאר אותה (Image captioning).

Many to one – רוצים לקבוע משהו יחיד עבור קלט סדרתי, למשל מקבלים משפט ורוצים לסווג את הסנטימנט שלו – האם הוא חיובי או שלילי.

Many to many – עבור כל סדרת קלט יש סדרת פלט, למשל תרגום משפה אחת לשפה אחרת – מקבלים משפט ומוציאים משפט.



איור 6.2 מודלים שונים של RNN.

6.1.2 Learning Parameters

הלמידה של הרשת נעשית בצורה דומה לרשתות שבפרקים הקודמים. עבור דאטה $x = (x_1, \dots, x_n), (y_1, \dots, y_n)$ נגדיר את פונקציית המחיר:

$$L(\theta) = \frac{1}{n}\sum_i L(\hat{y}_i, y_i, \theta)$$

כאשר הפונקציה $L(\hat{y}_i, y_i, \theta)$ תותאם למשימה – עבור משימות סיווג נשתמש ב-cross entropy ועבור בעיות רגרסיה נשתמש בקריטריון MSE. האימון יתבצע בעזרת GD, אך לא ניתן להשתמש ב-backpropagation הרגיל כיוון שכל משקל מופיע מספר פעמים – למשל w_{hx} פועל על כל הכניסות ו-w_{hh} פועל על כל רכיבי הזיכרון. כדי לבצע את עדכון המשקלים משתמשים ב-backpropagation through time (BPTT) – מסתכלים על הרשת הנפרשת כרשת אחת גדולה, מחשבים את הגרדיאנט עבור כל משקל, ואז סוכמים או ממצעים את כל הגרדיאנטים. אם הדאטה בכניסה הוא בגודל n, כלומר יש n דגימות בזמן, אז יש n רכיבי זיכרון, ו-$n-1$ משקלים w_{hh}. לכן הגרדיאנט המשוקלל יהיה:

$$\frac{\partial L}{\partial w_{hh}} = \sum_{n-1}\frac{\partial L}{\partial w_{hh}(t)} \quad or \quad \frac{\partial L}{\partial w_{hh}} = \frac{1}{n-1}\sum_{n-1}\frac{\partial L}{\partial w_{hh}(t)}$$

כיוון שהמשקלים זהים לאורך כל הרשת, $w_{hh}(t) = w_{hh}$ והשינוי בזמן יהיה רק לאחר ביצוע ה-BPTT ויהיה רלוונטי רק לוקטור הבא.

הצורה הפשוטה של ה-BPTT יוצרת בעיה עם הגרדיאנט. נניח שרכיב הזיכרון מיוצג בעזרת הפונקציה הבאה:

$$h_t = f(z_t) = f(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

לפי כלל השרשרת:

$$\frac{\partial h_n}{\partial x_1} = \frac{\partial h_n}{\partial h_{n-1}} \times \frac{\partial h_{n-1}}{\partial h_{n-2}} \times \cdots \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial x_1}$$

כיוון ש-w_{hh} קבוע ביחס לזמן עבור וקטור כניסה יחיד, מתקבל:

$$\frac{\partial h_t}{\partial h_{t-1}} = f'(z_t) \cdot w_{hh}$$

אם נציב זאת בכלל השרשרת, נקבל שעבור חישוב הנגזרת $\frac{\partial h_n}{\partial x_1}$ מכפילים $n-1$ פעמים ב-w_{hh}. לכן אם מתקיים $||w_{hh}|| > 1$ אז הגרדיאנט יתבדר, ואם $||w_{hh}|| < 1$ הגרדיאנט יתאפס. בעיה זו, של התבדרות או התאפסות הגרדיאנט, יכולה להופיע גם ברשתות אחרות, אבל בגלל המבנה של RNN והלינאריות של ה-BPTT ברשתות רקורסיביות זה קורה כמעט תמיד.

עבור הבעיה של התבדרות הגרדיאנט ניתן לבצע clipping – אם הגרדיאנט גדול מקבוע מסוים, מנרמלים אותו:

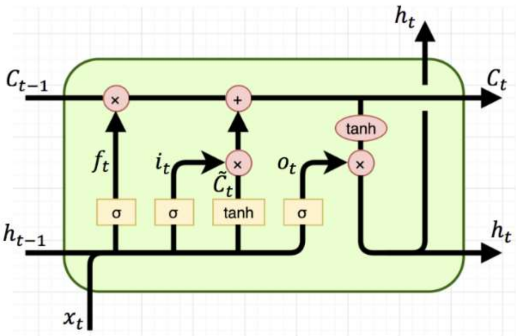
$$if\ \|g\| > c, then\ g = \frac{cg}{\|g\|}$$

הבעיה של התאפסות הגרדיאנט אמנם לא גורמת לחישובים של מספרים עצומים, אך היא בעצם מבטלת את ההשפעה של איברים שנמצאים רחוק אחד מהשני. אם למשל יש משפט ארוך, אז במקרה בו הגרדיאנט דועך במהלך ה-BPTT, כמעט ואין השפעה של המילה הראשונה על המילה האחרונה. במילים אחרות – התאפסות הגרדיאנט גוררת בעיה של Long-term, כלומר קשה ללמוד דאטה בעל תלות בטווח ארוך, כמו משפט ארוך או תופעות שמשתנות לאט. בגלל הבעיה הזו לא משתמשים ב-RNN הקלאסי (שנקרא גם Vanilla RNN), אלא מבצעים עליו שיפורים, כפי שיוסבר בפרק הבא.

6.2 RNN Architectures

6.2.1 Long Short-Term Memory (LSTM)

כדי להתגבר על בעיית דעיכת הגרדיאנט המונעת מהרשת להשתמש בזיכרון ארוך טווח, ניתן להוסיף אלמנטים לרכיב הזיכרון כך שהוא לא יכיל רק מידע על העבר, אלא יהיה גם בעל שליטה על איך וכמה להשתמש במידע. ב-RNN הפשוט לרכיב הזיכרון יש שתי כניסות – h_{t-1}, x_t, ובעזרתן מחשבים את המוצא על ידי שימוש בפונקציה $f_w(h_{t-1}, x_t)$. למעשה רכיב הזיכרון הוא קבוע והלמידה מתבצעת רק במשקלים. ב-LSTM יש שני שינויים עיקריים – מלבד הכניסות הרגילות יש עוד כניסה הנקראת memory cell state ומסומנת ב-c_{t-1}, ובנוסף לכך h_t מחושב בצורה מורכבת יותר. באופן הזה האלמנט c_t דואג לזיכרון ארוך טווח של דברים, ו-h_t אחראי על הזיכרון של הטווח הקצר. נתבונן בארכיטקטורה של תא הזיכרון:



איור 6.3 תא זיכרון בארכיטקטורת LSTM.

הצמד $[x_t, h_{t-1}]$ נכנס לתא ומוכפל במשקל w, ולאחר מכן עובר בנפרד דרך ארבעה שערים (יש לשים לב שלא מבצעים פעולה בין x_t ל-h_{t-1} אלא הם נשארים בנפרד ואת כל הפעולות עושים על כל איבר בנפרד). **השער הראשון** $f_t = [\sigma(x_t), \sigma(h_{t-1})]$ הוא שער שכחה והוא אחראי על מחיקת חלק מהזיכרון. אם למשל יש משפט ומופיע בו נושא חדש, אז שער זה אמור למחוק את הנושא שהיה שמור בזיכרון. **השער השני** i_t הוא שער זיכרון והוא אחראי על כמה יש לזכור את המידע החדש לטווח ארוך. אם לדוגמה אכן יש במשפט מסוים נושא חדש, אז השער יחליט שיש לזכור את המידע הזה. אם לעומת זאת המידע החדש הוא תיאור שלא רלוונטי להמשך אז אין טעם לזכור אותו. **השער הרביעי** o_t הוא שער מוצא והוא אחראי על כמה מהמידע רלוונטי לדאטה הנוכחי x_t, כלומר מה יהיה הפלט של התא בהינתן מידע העבר. שלושת השערים האלו נקראים מסכות (Masks), והם מקבלים ערכים בין 0 ל-1 כיוון שהם עוברים דרך סיגמואיד. יש **שער נוסף** $\tilde{c}_t$ (לפעמים מסומן באות g) שאחראי על השאלה כמה מהזיכרון להעביר לתא הבא. שער זה משקלל את המידע המתקבל יחד עם i_t שאומר עד כמה יש לזכור להמשך את המידע החדש.

באופן הזה מקבלים הן את h_t שאחראי על הזיכרון לטווח הקצר כמו ב-Vanilla RNN, והן את c_t שאחראי על זיכרון של כל העבר. ארכיטקטורת הרכיב מאפשרת להתייחס לאלמנטים נוספים הקשורים לזיכרון – ניתן לשכוח חלקים לא רלוונטיים של התא הקודם (f_t), להתייחס באופן סלקטיבי לכניסה (i_t) ולהוציא רק חלק מהמידע המשוקלל הקיים (o_t). באופן פורמלי ניתן לנסח את פעולת התא כך:

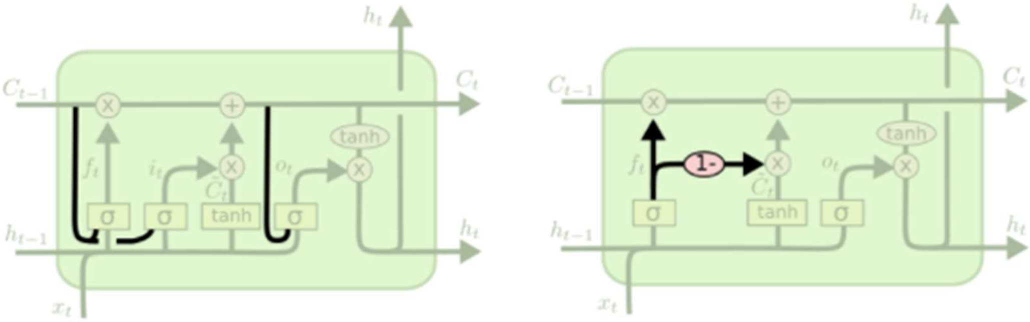
$$\begin{pmatrix} i \\ f \\ o \\ \tilde{c} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} = \begin{pmatrix} \sigma(w_i \cdot [x_t, h_{t-1}] + b_i) \\ \sigma(w_f \cdot [x_t, h_{t-1}] + b_f) \\ \sigma(w_o \cdot [x_t, h_{t-1}] + b_o) \\ tanh(w_{\tilde{c}} \cdot [x_t, h_{t-1}] + b_{\tilde{c}}) \end{pmatrix}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, h_t = o_t \odot \tanh(c_t)$$

כאשר האופרטור $\odot$ מסמל כפל איבר איבר (כיוון שלשערים נכנס הזוג $[x_t, h_{t-1}]$, אם במוצא מבצעים מכפלה מסוימת, יש לבצע אותה על כל אחד מהאיברים).

יש וריאציות שונות של רכיבי LSTM – ניתן למשל לחבר את c_{t-1} לא רק למוצא h_t אלא גם לשאר השערים. חיבור כזה נקרא peepholes, כיוון שהוא מאפשר לשערים להתבונן ב-c_{t-1} באופן ישיר. יש ארכיטקטורות שמחברות את c_{t-1} לכל השערים, ויש ארכיטקטורות שמחברות אותו רק לחלק מהשערים. חיבור כל השערים ל-c_{t-1} משנה כמובן את משוואות השערים. במקום $\sigma(w \cdot [x_t, h_{t-1}] + b)$, המשוואה החדשה תהיה $\sigma(w \cdot [\boldsymbol{c_{t-1}}, x_t, h_{t-1}] + b)$.

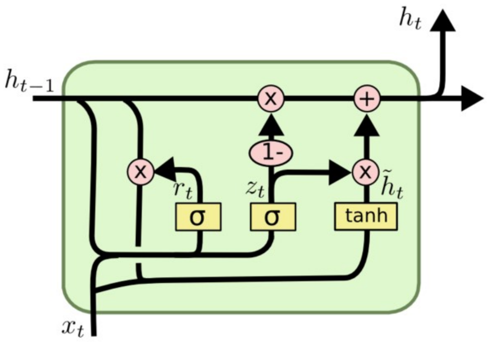
וריאציה אחרת של LSTM מאחדת בין שער השכחה f_t לבין שער הזיכרון i_t, וההחלטה עד כמה יש למחוק מידע מהזיכרון מתקבלת יחד עם ההחלטה כמה מידע חדש יש לכתוב. שינוי זה ישפיע על ה-memory cell, כאשר במקום $c_t = f_t \odot c_{t-1} + \boldsymbol{i_t} \odot \tilde{c}_t$, משוואת העדכון תהיה: $c_t = f_t \odot c_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) \odot \tilde{c}_t$.



איור 6.4 וריאציות של LSTM – peephole connections (ימין) ו-coupled forget and input gates (שמאל).

6.2.2 Gated Recurrent Units (GRU)

ישנה ארכיטקטורה נוספת של תא זיכרון הנקראת Gated Recurrent Units (GRU), והיא כוללת מספר שינויים ביחס ל-LSTM:



איור 6.5 תא זיכרון בארכיטקטורת GRU.

השינוי המשמעותי מ-LSTM הוא העובדה שאין memory cell state, וכל השערים מתבססים רק על הקלט והמוצא של התא הקודם. כדי לאפשר זיכרון הן לטווח ארוך והן לטווח קצר, יש שני שערים – Reset gate ו-Update gate, והם מחושבים על פי הנוסחאות הבאות:

$$\text{Update: } z_t = \sigma(w_z \cdot [x_t, h_{t-1}])$$

$$\text{Reset: } r_t = \sigma(w_r \cdot [x_t, h_{t-1}])$$

בעזרת שער ה-reset מחשבים Candidate hidden state:

$$\tilde{h}_t = \tanh(w \cdot [x_t, r_t \odot h_{t-1}]$$

ראשית יש לשים לב כי $r_t \in [0,1]$ כיוון שהוא תוצאה של סיגמואיד. כעת נתבונן על $\tilde{h}_t$ ביחס לרכיב זיכרון פשוט של Vanilla RNN: $h_t = f_W(w_{hh}h_{t-1} + w_{xh}x_t)$. כאשר r_t קרוב ל-1 מתקבל הביטוי:

$$\tilde{h}_t = \tanh(w \cdot [x_t, r_t \odot h_{t-1}] \approx \tanh(w[x_t, h_{t-1}]) = \tanh(w_{hx}x_t + w_{hh}h_{t-1}]$$

המשמעות היא שעבור $r_t \to 1$ מתקבל רכיב הזיכרון הקלאסי, השומר על זיכרון לטווח קצר. אם לעומת זאת $r_t \to 0$, אז מתקבל $\tilde{h}_t \approx \tanh(w \cdot [x_t, 0 \odot h_{t-1}] = \tanh(w_{xh}x_t)$, ולמעשה הזיכרון של הטווח הקצר מתאפס (reset).

לאחר החישוב של $\tilde{h}_t$ מחשבים את המוצא של המצב החבוי בעזרת z_t, שגם הוא מקבל ערכים בין 0 ל-1:

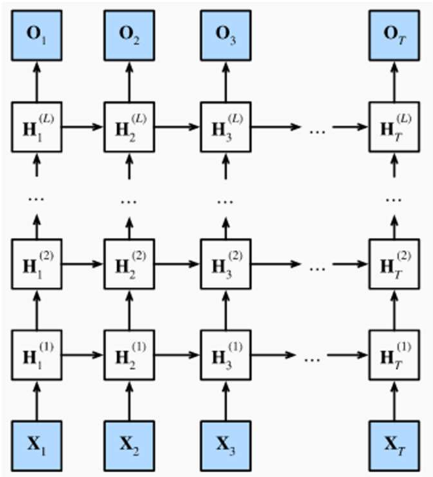
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

אם $z_t \to 0$, אז $h_t \approx h_{t-1}$, כלומר לא מתחשבים ב-$\tilde{h}_t$, ולמעשה מעבירים את המצב הקודם כמו שהוא. אם לעומת זאת $z_t \to 1$ אז $h_t \approx \tilde{h}_t$, כלומר מתעלמים מהמצב הקודם כמו שהוא ולוקחים את ה-Candidate hidden state, שהוא למעשה שקלול של המצב הקודם עם איבר הקלט הנוכחי. עבור כל ערך אחר של z_t, מקבלים שקלול של המצב החבוי הקודם וה-Candidate hidden state.

ארכיטקטורה זו מאפשרת גם לזכור דברים לאורך זמן, וגם מצליחה להתמודד עם בעיית הגרדיאנט. אם שער העדכון קרוב ל-1 כל הזמן, אז בעצם מעבירים את המצב החבוי כמו שהוא, ולמעשה הזיכרון נשמר לאורך זמן. בנוסף, אין בעיה של התבדרות הגרדיאנט, כיוון שאם השינוי בין תא לתא לא גדול, אז הגרדיאנט קרוב ל-1 כל הזמן ולא מתבדר.

6.2.3 Deep RNN

עד כה דובר על רכיבי זיכרון יחידים, שניתן לשרשר אותם יחד ולקבל שכבה שיכולה לנתח. ניתן להרחיב את המודל הפשוט לרשת בעל מספר שכבות עמוקות.



איור 6.6 ארכיטקטורת Deep RNN.

נתאר את הרשת באופן פורמלי. בכל נקודת זמן t יש וקטור כניסה $x_t \in \mathbb{R}^{n \times d}$ (וקטור בעל n איברים, כאשר כל איבר הוא ממימד d). איברי הסדרה נכנסים לרשת בעלת L שכבות ו-T איברים בכניסה, כלומר יש T נקודות זמן, ועבור כל אחת מהן יש L שכבות (או מצבים חבויים). כל שכבה מכילה h מצבים חבויים, כאשר השכבה ה-ℓ בנקודת זמן t מסומנת בתור $H_t^{(\ell)} \in \mathbb{R}^{n \times h}$. בכל נקודת זמן יש גם וקטור מוצא באורך n – $o_t \in \mathbb{R}^{n \times q}$. נסמן: $H_t^{(0)} = x_t$, ונניח שבין שכבה אחת לשנייה משתמשים באקטיבציה לא לינארית ϕ. בעזרת סימונים אלה נקבל את הנוסחה הבאה:

$$H_t^{(\ell)} = \phi_\ell\left(H_t^{(\ell-1)} w_{xh}^{(\ell)} + H_{t-1}^{(\ell)} w_{hh}^{(\ell)} + b_h^{(\ell)}\right)$$

כאשר $w_{xh}^{(\ell)} \in \mathbb{R}^{h \times h}, w_{hh}^{(\ell)} \in \mathbb{R}^{h \times h}, b_h^{(\ell)} \in \mathbb{R}^{1 \times h}$ הם הפרמטרים של השכבה החבויה ה-ℓ. הפלט o_t תלוי באופן ישיר רק בשכבה ה-L, והוא מחושב על ידי:

$$o_t = H_t^{(L)} w_{hq} + b_q^{(L)}$$

כאשר $w_{hq}^{(L)} \in \mathbb{R}^{h \times q}, b_q^{(L)} \in \mathbb{R}^{1 \times q}$ הם הפרמטרים של שכבת הפלט.

ניתן כמובן להשתמש במצבים החבויים ברכיבי זיכרון GRU או LSTM, וכך לקבל Deep Gated RNN.

6.2.4 Bidirectional RNN

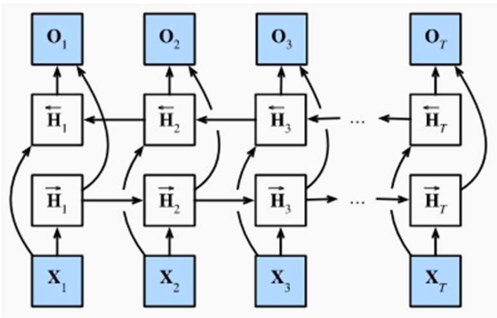
כל הרכיבים והרשתות שנידונו עד כה עסקו בסדרות סיבתיות, כלומר, סדרות בהן כל איבר מושפע מקודמיו אך לא מאלו הבאים אחריו. למשל, ערך מניה ביום מסוים קשור לערכה בימים הקודמים, אך הערך שלה ביום המחרת (שכלל עוד לא ידוע) לא משפיע בשום צורה על ערכה ביום הנוכחי. דוגמה נוספת – התפתחות של גל בזמן תלויה בערכי הקודמים של הגל אך אינה מושפעת ממצבי הגל בעתיד. זהו אמנם המצב היותר מצוי, אך ישנם מצבים בהם יש סדרה לאו דווקא סיבתית, כך שניתן לבחון את הקשר בין איבריה משני הכיוונים. ניקח לדוגמה את משימת ההשלמה הבאה:

I am __.

I am __ hungry.

I am __ hungry, and I can eat a big meal.

כעת נניח שבכל אחד מהמשפטים צריך לבחור את אחת מהמילים שבסט $\{happy, not, very\}$. כמובן שסוף הביטוי, במקרה וקיים, תורם מידע משמעותי על איזו מילה לבחור. מודל שאינו מסוגל לנצל את המידע לאחר המילה החסרה יכול לפספס מידע חשוב, ולרוב יכול לנחש מילה שאינה מסתדרת עם המשך המשפט הן מבחינה תחבירית והן מבחינת המשמעות. כדי לבנות מודל שמתייחס לכל חלקי המשפט, יש לתכנן ארכיטקטורה שמאפשרת לנתח סדרה משני הכיוונים שלה. ארכיטקטורה כזו נקראת Bidirectional RNN, והיא למעשה מבצעת ניתוח של סדרה משני הכיוונים שלה במקביל. באופן הזה כל מצב חבוי נקבע בו זמנית על ידי הנתונים של שני מצבים חבויים אחרים – זה שלפניו וזה שאחריו.



איור 6.6 ארכיטקטורת Bidirectional RNN.

עבור כל כניסה $x_t \in \mathbb{R}^{n\times d}$ נחשב במקביל שני מצבים חבויים – $\overleftarrow{H}_t \in \mathbb{R}^{n\times h}, \overrightarrow{H}_t \in \mathbb{R}^{n\times h}$, כאשר h זה מספר רכיבי הזיכרון בכל מצב חבוי. כל מצב מחושב באופן הבא:

$$\overrightarrow{H}_t = \phi\left(x_t w_{xh}^{(f)} + \overrightarrow{H}_{t-1} w_{hh}^{(f)} + b_h^{(f)}\right)$$

$$\overleftarrow{H}_t = \phi\left(x_t w_{xh}^{(b)} + \overleftarrow{H}_{t+1} w_{hh}^{(b)} + b_h^{(b)}\right)$$

כאשר $w_{xh}^{(f)} \in \mathbb{R}^{d\times h}, w_{hh}^{(f)} \in \mathbb{R}^{h\times h}, b_h^{(f)} \in \mathbb{R}^{1\times h}$ ו-$w_{xh}^{(b)} \in \mathbb{R}^{d\times h}, w_{hh}^{(b)} \in \mathbb{R}^{h\times h}, b_h^{(b)} \in \mathbb{R}^{1\times h}$ הם הפרמטרים של המודל. לאחר החישוב של $\overrightarrow{H}_t$ ו-$\overleftarrow{H}_t$ משרשרים אותם יחד ומקבלים את $H_t \in \mathbb{R}^{n\times 2h}$, ובעזרתו מחשבים את המוצא:

$$o_t = H_t w_{hq} + b_q$$

כאשר $w_{hq} \in \mathbb{R}^{2h\times q}, b_q \in \mathbb{R}^{1\times q}$ הם הפרמטרים של שכבת הפלט.

6. References

Vanilla:

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

LSTM, GRU:

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Deep RNN, Bidirectional RNN:

http://d2l.ai/chapter_recurrent-modern/index.html