CIS 6930 Topics in Computing for Data Science
Week 6a: Recurrent Neural Networks (1)
RNN, LSTM, GRU

9/28/2021

Yoshihiko (Yoshi) Suhara

2pm-3:20pm (80min session!)

Teaching Style

- Fully online via Zoom
  - I'd like to make it interactive as much as possible
  - Please turn on cameras. This helps me feel the in-class atmosphere :)

- Theory & practice
  - i.e., lectures + hands-on session/assignments/term projects
  - My focus is to teach **Why? (Theory)** and **How? (Practice)**

python + colab PyTorch

Week 6!

- ~~Week 1: Deep Learning Basics (Thu 9/9)~~
- ~~Week 2: AutoEncoder (Tue 9/14)~~
- ~~Week 3: Convolutional Neural Networks (Thu 9/16)~~
- ~~Week 4: GAN (Tue 9/21)~~
- ~~Week 5: Word embeddings: Word2vec, GloVe (Thu 9/23)~~
- **Week 6: Recurrent Neural Networks (Tue 9/28, Thu 9/30)**
- Week 7: Review/Project pitch & Mid-term (Tue 10/5, Thu 10/7)
- Fall Break
- ...

Recap: NLP Basics & Word Embeddings

NLP Tasks and Applications

Practical Natural Language Processing | O'Reilly

Traditional NLP Pipeline

- Tokenization
- Sentence splitting
- Lemmatization
- POS tagging
- Chunking
- Dependency parsing

Shallow

Deep

https://stanfordnlp.github.io/CoreNLP/

What Makes Text Processing Difficult for Computers?

- Symbolic representations cannot be directly converted into vector/tensor representations
  - cf. Images

Images

"A cat is starting at me" → ???

Text

One-hot Vector Representation

- Assign one dimension to each word

"A cat and a dog"

$$
\begin{array}{cccccc}
\text{a} & \text{cat} & \text{and} & \text{dog} & & \\
\left[\begin{matrix} 1 \end{matrix}\right. & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} \dots \end{matrix} & \left.\begin{matrix} 0 \end{matrix}\right] \\
\left[\begin{matrix} 0 \end{matrix}\right. & \begin{matrix} 1 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} \dots \end{matrix} & \left.\begin{matrix} 0 \end{matrix}\right] \\
\left[\begin{matrix} 0 \end{matrix}\right. & \begin{matrix} 0 \end{matrix} & \begin{matrix} 1 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} \dots \end{matrix} & \left.\begin{matrix} 0 \end{matrix}\right] \\
\left[\begin{matrix} 1 \end{matrix}\right. & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} \dots \end{matrix} & \left.\begin{matrix} 0 \end{matrix}\right] \\
\left[\begin{matrix} 0 \end{matrix}\right. & \begin{matrix} 0 \end{matrix} & \begin{matrix} 0 \end{matrix} & \begin{matrix} 1 \end{matrix} & \begin{matrix} \dots \end{matrix} & \left.\begin{matrix} 0 \end{matrix}\right]
\end{array}
$$

Vocabulary size

Word Embeddings: *Dense* Vector Representations

"A cat and a dog"

$$\begin{bmatrix} 0.1 & 0.2 & 0.8 & 0.1 \ldots & 0.2 \\ 0.8 & 0.4 & 0.1 & 0.0 \ldots & 0.0 \\ 0.1 & 0.2 & 0.1 & 0.1 \ldots & 0.1 \\ 0.1 & 0.2 & 0.8 & 0.1 \ldots & 0.2 \\ 0.8 & 0.3 & 0.1 & 0.0 \ldots & 0.0 \end{bmatrix}$$

Dimension size
(e.g., 300, 500)

Dense vector representations

Distributional Hypothesis (in Linguistics)

- Words that occur in the same contexts tend to have similar meanings [Harris 1954]
- The idea “a word is characterized by the company it keeps” was popularized by [Firth 1957]

Zellig Harris

John Rupert Firth

Word2vec: Embedding Meanings into Dense Vectors

- Word2vec (2013) presents that we can **meaningfully** embed words into the vector space

Male-Female

Verb tense

Country-Capital

e.g., v("king") - v("man") + v("woman") =~ v("queen")

GloVe [Pennington et al. 2014]

- Learning **u** and **v** that **approximates the co-occurrence probability** based on word-word co-occurrence matrix

I like deep learning.
I like NLP.
I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij}) \left(u_i^T v_j - \log P_{ij} \right)^2$$

$$z_i = u_i + v_i$$

Out of Vocabulary (OOV) Problem

- The vocabulary must be constructed in advance
- No way to represent “unseen” words
  - known as Out-of-vocabulary (OOV) Problem

- Any workaround?
  - character-based representations
  - sub-token representations (common choice for pre-trained language models)
    - e.g., fastText

fastText: Subword Embeddings

- **N-gram embeddings** instead of “word” embeddings
  - eating → (<s>ea, eat, ati, tin, ing, ng</s>)
- CBoW/Skip-gram can be used for training

Is fixed n-gram the best? → Learned sub-tokenization (Week 8)

spaCy for word embeddings

```python
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for token in doc:
    print([token.text, token.lemma_, token.pos_, token.dep_, token.ent_iob_, token.ent_type_])
```

```python
token.vector
```

```
array([-1.535186  , -0.4060119 ,  2.9251206 ,  1.4862046 , -0.5711231 ,
        0.04101598,  2.2709746 , -2.7806776 , -0.5188239 , -1.9200461 ,
        0.6264886 ,  5.113931  , -4.4795375 , -3.3172297 , -2.591983  ,
        0.2952245 ,  3.0970662 , -2.1948025 , -1.0125057 , -1.9231814 ,
        3.3866777 , -1.1537448 , -1.3051802 ,  0.10659629, -2.4653244 ,
        4.1557007 , -4.0343957 ,  0.96222323, -0.5591469 , -2.192143  ,
       -0.194722  , -1.6126934 ,  1.8915527 ,  0.05074558, -1.1258242 ,
       -1.311002  , -0.22932756,  0.86395764,  2.1479762 , -0.04867677,
        1.0269833 , -1.6562294 , -3.2395341 , -0.8505454 , -0.7531296 ,
       -0.29259783, -0.24896976, -3.0125833 ,  4.453686  ,  2.874859  ,
        4.92746   , -3.9719796 , -1.8743998 , -0.7232732 ,  0.68904847,
        1.0461929 ,  1.0565916 ,  2.1449113 ,  2.593386  , -0.22828588,
        4.1223226 ,  0.56345993,  3.7295504 , -2.460603  ,  3.2813878 ,
       -3.3111894 ,  2.4160173 , -1.5893238 , -2.481103  ,  2.9629679 ,
        5.0079627 , -3.640224  ,  4.620039  , -0.13005406,  1.9256897 ,
       -1.8064855 ,  0.5464196 , -0.21171951, -1.3014529 , -4.0162244 ,
        0.98979425,  0.5273107 , -2.2770624 ,  2.0088112 ,  2.5486505 ,
        1.4092522 ,  2.6082263 , -1.2317483 , -2.1186576 , -2.004669  ,
       -0.01678723,  0.21361685,  0.64057314, -3.5370805 ,  0.56346744,
        2.3472533 ], dtype=float32)
```

```python
doc.vector
```

```
array([ 1.0188444 ,  0.03571144,  0.7764361 ,  1.2014427 ,  0.1412878 ,
        0.1682428 ,  1.1233734 ,  0.3487686 ,  0.6835691 ,  1.4795598 ,
        0.20042907,  0.9256677 , -1.5593272 , -0.6143042 , -1.557381  ,
       -0.9958308 , -0.5963076 , -1.6042081 , -0.39841834, -0.3468938 ,
        1.0979723 , -0.07698824,  0.5335008 ,  0.72681975, -1.0884286 ,
        0.7676121 , -1.7708863 ,  0.72250545,  0.20121218, -1.444205  ,
        0.9010805 , -0.15313278,  0.2421783 , -1.3908852 , -0.69058996,
       -0.9723627 ,  0.7619291 , -0.49309894, -1.1831714 , -0.02785109,
        0.76898694, -0.26471007, -0.5232663 , -1.2854791 , -0.9188446 ,
        0.2745516 ,  1.0659344 ,  0.31896058, -0.04838419,  0.8291477 ,
        0.8577619 , -1.0837296 , -0.01163011,  0.04729161, -1.3421645 ,
        1.395626  , -0.2455926 ,  0.6314798 ,  2.260128  , -0.14274798,
        1.7947469 , -0.4407996 ,  1.5280372 , -0.25356895,  1.2945919 ,
       -0.15937866,  0.67561346, -2.3464286 , -0.5416016 ,  1.6168556 ,
        1.8714851 , -1.9004959 ,  1.2033213 , -1.0751767 ,  1.1551958 ,
       -0.16097708,  1.0588483 , -0.35734576, -1.1385773 , -0.8543524 ,
        0.11442439, -1.3668202 , -0.05677014, -0.03700415,  2.2501202 ,
        1.0935678 , -0.8177019 ,  0.7018638 , -0.852125  , -0.05367121,
       -1.1944368 , -1.9318706 ,  0.39264107, -0.14523607,  0.8427848 ,
        1.0945965 ], dtype=float32)
```

Brainstorming: X Embeddings (X2vec)

- doc → ?
- token_id → ?

Any ideas?

Non-NLP Example 1:
Grouping Users by Mobility Patterns

- Document: User
- Word: WiFi spot

Non-NLP Example 2:
Grouping Users by Phone App Usage

- Document: User
- Word: Application

Example: Friends and Family Dataset

http://realitycommons.media.mit.edu/friendsdataset.html

Android OS based mobile phones are used as in-situ social sensors to map users activity features, proximity networks, media consumption, and behavior diffusion patterns.

Subjects complete surveys at regular intervals, combining web-based and on-phone surveys. Monthly surveys include questions about self-perception of relationships, group affiliation, interactions, and also standard scales like the Big-Five personality test.

Daily surveys include questions like mood, sleep, and other activity logging. Information on purchases is collected through receipts and credit card statements submitted at the participants' discretion

Friends and Family description

http://realitycommons.media.mit.edu/friendsdataset2.html

- Sensor data (every 6 minutes)
  - Geographical location and proximity with other members
  - Call/SMS log information
  - Activity (Walking, Jogging, In a vehicle etc.)
- Individual demographics survey
  - Income, lifestyle
  - Big-5 personality test
- Weekly survey
  - Times going to restaurant, ordering food, and cooking; dining with whom - alone, with friends, with family etc.
  - Watching movie, TV frequency and with whom and where
  - Expected and enjoyed events, with whom
  - Amount of time spend with kids
  - Involvement with activity groups

Questions?

Agenda

- Recurrent Neural Networks Basics
  - What are RNNs? Why do we need RNNs?
  - Backpropagation through time
  - RNN Patterns
- RNN in depth
  - RNN
  - LSTM
  - GRU
- Hands-on session
- Assignment

What are RNNs? Why?

MLP for Sequential Input?

- One option is to have **M input nodes** (M = max sequence length)

"Deep learning is great" → ["deep", "learning", "is", "great", None, None, None]
"Machine learning is so boring" → ["machine", "learning", "is", "so", "boring", None, None]

If we choose sufficiently large M, it seems to work. Really?

What about this example?

- Sequential data (e.g., words in a sentence) generally create meanings based on relations between adjacent tokens

"Deep learning is great" → ["deep", "learning", "is", "great", None, None, None]
"I believe deep learning is great" → ["I", "believe", "deep", "learning", "is", "great", None]

How should we model the functionality?

Convolutional Neural Networks for Text

- CNN can be a solution to incorporate the **sequence order** into the model

https://arxiv.org/abs/1408.5882

Recurrent Neural Networks for Sequential Input

- Inspired by Hopfield Network (1982)
- “Simple” Recurrent Neural Networks
  - Jordan Net (1986)
  - **Elman Net (1990)**

Hopfield Network

Jordan Net

Elman Net

Why Elman Net?

- Named after the author of the original paper
- Elman is known as a “connectionist” (**connectionism)**
  - Challenging the innateness hypothesis/generative grammar
  - i.e., vs Chomsky

Jeffrey L. Elman
(1948-2018)

Noam Chomsky
(1928-)

“Finding Structure in Time” [Elman 1990]

- This seminal paper claims a “Simple” RNN model, trained on the next word prediction task, learns **the grammatical structure** only from finite amount of examples

Clusters of hidden representations of the trained RNN

Terminology

- {Elman Net, LSTM, GRU} $\in$ RNN
- But, RNN often refers to Elman Net

Unfolding RNN (1/2)
Backpropagation Through Time (BPTT)

Unfolding RNN (2/2)
Backpropagation Through Time (BPTT)

RNNs in Depth

What Does Each Step in RNN Do?
What Does an RNN Cell Do?

RNN Cell for Sequential Data

RNN = Elman Net

- Input value + previous hidden state → Next hidden state

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

$$W_h[x_t; h_{(t-1)}] + b_h$$

Another notation

The figures are based on the following blog article (highly recommended!)

Illustrated Guide to LSTM's and GRU's: A step by step explanation

RNN Cell for Sequential Data

- Passing the hidden state to the next step
- Note that each cell in the figure is **the same RNN cell**

What's the cell?
What should the initial hidden state h_0 be?

Where to Attach Output Layer?

- Where?

Where to Attach Output Layer?

- Where?
- For classification

Text Classification Example

- Input: “Deep Learning is great”
- Label: positive

Any other options?

Where to Attach Output Layer?

- Where?

Any other options?
→ RNN for **sequential tagging**

Sequential Labeling Problem

- Input: Sequence of tokens
- Output: Sequence of labels (each label can be either binary or multi-class)

Named Entity Recognition (NER)

IOB (Inside-Outside-Beginning) Format

- Commonly used annotation format for chunking problems

```
Alex        B-PER
is          O
going       O
to          O
Los         B-LOC
Angeles     I-LOC
in          O
California  B-LOC
```

Quiz. RNN Parameters?

- How many parameters does the model have?

Quiz. RNN Parameters?

- How many parameters does the model have?

hidden_dim * output_dim

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

hidden_dim * hidden_dim
+ emb_dim * hidden_dim

RNN Application Patterns

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

RNN Application Patterns

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

Questions?

For Longer Sequence: LSTM and GRU

What’s the issue with RNNs?

- Not a very good design for **a long sequence**
  - **Later input values** have higher impact on the last hidden state
  - Gradient vanishing problem

For Longer Sequence: LSTM and GRU

Long-short Term Memory (LSTM)
[Hochreiter and Schmidhuber 1997]

LSTM Cell

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Key Idea: Cell State + Gating Mechanism

RNN Cell

LSTM Cell

Difference b/w RNN and LSTM

Forget Gate

- Decide what information should be kept or discarded

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$
$$\boxed{f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})}$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$
$$c_t = \boxed{f_t \odot c_{t-1}} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Input Gate

- Combine and process the previous hidden and the current input

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Cell State

- Forget gate value + Input gate value

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$
$$c_t = \boxed{f_t \odot c_{t-1}} + \boxed{i_t \odot g_t}$$
$$h_t = o_t \odot \tanh(c_t)$$

Output Gate

- Decide what the next hidden state should be

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$
$$\boxed{o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = \boxed{o_t} \odot \boxed{\tanh(c_t)}$$

Recap

Long-short Term Memory (LSTM) Cell
[Hochreiter and Schmidhuber 1997]

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

For Longer Sequence: LSTM and GRU

Gated Recurrent Unit (GRU) [Cho et al. 2014]

- What are major differences against LSTM?

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

Gated Recurrent Unit (GRU) [Cho et al. 2014]

- What are major differences against LSTM?

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

(1) No cell state (100% compatible with RNN cell!)
(2) Different gating functions

Reset Gate

- Decide how much to forget about the past information (i.e., Forget gate)

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

Update Gate

- Decide what information to pass to the next state

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$

$$\boxed{z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})}$$

$$\boxed{n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))}$$

$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

$$h_t = z_t \cdot n_t + (1 - z_t) h_{t-1}$$

??

Update Gate

- Decide what information to pass to the next state

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

$$h_t = z_t \cdot n_t + (1 - z_t) h_{t-1}$$

??

Notation discrepancy: https://github.com/pytorch/pytorch/issues/60645

PyTorch Code Reading

- https://github.com/pytorch/pytorch/blob/90ffab6e3713a40f47f4c9d3d3b16d56e83f97c4/aten/src/ATen/native/RNN.cpp#L444-L454

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$
$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

```
    const auto chunked_igates = pre_
        ? input.chunk(3, 1)
        : params.linear_ih(input).ch
    auto chunked_hgates = params.linear_hh(hidden).chunk(3, 1);
    const auto reset_gate =
        chunked_hgates[0].add_(chunked_igates[0]).sigmoid_();
    const auto input_gate =
        chunked_hgates[1].add_(chunked_igates[1]).sigmoid_();
    const auto new_gate =
        chunked_igates[2].add(chunked_hgates[2].mul_(reset_gate)).tanh_();
    return (hidden - new_gate).mul_(input_gate).add_(new_gate);
```

Wikipedia Article: Gated Recurrent Unit
https://en.wikipedia.org/wiki/Gated_recurrent_unit

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

$$
\begin{aligned}
r_t &= \sigma(W_{ir} x_t + b_{ir} + W_{hr} h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz} x_t + b_{iz} + W_{hz} h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in} x_t + b_{in} + r_t * (W_{hn} h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

PyTorch version

Original Paper

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho
Bart van Merriënboer **Caglar Gulcehre** Université de Montréal
firstname.lastname@umontreal.ca

Dzmitry Bahdanau Jacobs University, Germany
d.bahdanau@jacobs-university.de

Fethi Bougares **Holger Schwenk** Université du Maine, France
firstname.lastname@lium.univ-lemans.fr

Yoshua Bengio Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

2.3 Hidden Unit that Adaptively Remembers and Forgets

In addition to a novel model architecture, we also propose a new type of hidden unit (f in Eq. (1)) that has been motivated by the LSTM unit but is much simpler to compute and implement.[1] Fig. 2 shows the graphical depiction of the proposed hidden unit.

Let us describe how the activation of the j-th hidden unit is computed. First, the *reset* gate r_j is computed by

$$r_j = \sigma\left(\left[\mathbf{W}_r\mathbf{x}\right]_j + \left[\mathbf{U}_r\mathbf{h}_{\langle t-1\rangle}\right]_j\right), \quad (5)$$

where σ is the logistic sigmoid function, and $[\cdot]_j$ denotes the j-th element of a vector. $\mathbf{x}$ and $\mathbf{h}_{t-1}$ are the input and the previous hidden state, respectively. $\mathbf{W}_r$ and $\mathbf{U}_r$ are weight matrices which are learned.

Similarly, the *update* gate z_j is computed by

$$z_j = \sigma\left(\left[\mathbf{W}_z\mathbf{x}\right]_j + \left[\mathbf{U}_z\mathbf{h}_{\langle t-1\rangle}\right]_j\right). \quad (6)$$

The actual activation of the proposed unit h_j is then computed by

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + (1 - z_j)\tilde{h}_j^{\langle t\rangle}, \quad (7)$$

where

$$\tilde{h}_j^{\langle t\rangle} = \phi\left(\left[\mathbf{W}\mathbf{x}\right]_j + \left[\mathbf{U}\left(\mathbf{r}\odot\mathbf{h}_{\langle t-1\rangle}\right)\right]_j\right). \quad (8)$$

PyTorch Implementation Follows the Original Paper

- That said, the other design should work (it's just a linear interpolation)
- Online discussion https://github.com/pytorch/pytorch/issues/60645

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

Questions?

LSTM/GRU Extension

- LSTM/GRU still **may not be sufficient** to capture the long-distance dependency in the context

- Two types of extensions
  - (a) Stacked (N-layer) models
  - (b) Bidirectional models
  - (a) + (b)

(a) Stacked (Multi-layer) LSTM/GRU

- Use the hidden state of the previous layer as input

(a) Stacked (Multi-layer) LSTM/GRU + Output layer

(b) Bidirectional LSTM/GRU

- Two RNN chains for **forward/backward sequence**

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." Signal Processing, IEEE Transactions on 45.11 (1997):

(b) Bidirectional LSTM/GRU

(b) Bidirectional LSTM/GRU + Output Layer

(a) + (b) Stacked Bidirectional LSTM/GRU

Thank you PyTorch!

LSTM

CLASS `torch.nn.LSTM(*args, **kwargs)` [SOURCE]

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

20min more

Hands-on Session: RNN in Action!

- Google Colab

RNN Application Patterns

Single-input Generation?

Classification

Encoder-decoder model (aka seq2seq)

Sequential tagging

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

Sequential Labeling Problem

- Input: Sequence of tokens
- Output: Sequence of labels

Chunking / Named Entity Recognition (NER)

- Extracting **phrases** and assigning **type** information
- Technically, Chunking != NER, but NER is often formulated as Chunking

IOB (Inside-Outside-Beginning) Format

- Commonly used annotation format for chunking problems

```
Alex        B-PER
is          O
going       O
to          O
Los         B-LOC
Angeles     I-LOC
in          O
California  B-LOC
```

Assignment 3: Develop a BiLSTM Model for NER

- Google Colab

Summary

- Recurrent Neural Networks Basics
  - What are RNNs? Why do we need RNNs?
  - Backpropagation through time
  - RNN Patterns
- RNN in depth
  - RNN
  - LSTM
  - GRU

Key Takeaway 1: RNN Application Patterns

Key Takeaway 2: Three Cells

Questions?

- Next sessions
  - Thu: RNN (2): Encoder-decoder models
  - Next Tue: Review session (Discussion & Term project ideas)
  - Next Thu: mid-term! (written exam)