CS584 Natural Language Processing

Recurrent Neural Networks

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **<u>24 hours</u>**.
- **40%** penalty for late submissions within **<u>24-48 hours</u>**.
- After 48 hours, you get **NO** points on the assignment.

Submission Policy

- It is your responsibility to make sure your submission file is correct and has no problem.
- You need to download the zip file to double-check if the file includes your solution after you upload it.
- You will get **0** points on the assignment if we cannot find your solution file.

Language modeling (motivates RNNs)

- Language Modeling is the task of predicting what word comes next. e.g. “the students opened their [blank]” (books? laptops? minds? exams?)

- Formally, given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$p(x^{(t+1)} = w_j | x^{(t)}, \ldots, x^{(1)})$$

where w_j is a word in the vocabulary $V = \{w_1, \ldots, w_{|V|}\}$.

- A system that does this is called a language model.

Language models everyday

Language models everyday

Google

how to maintain a

how to maintain a **beard**
how to maintain a **pool**
how to maintain a **healthy diet**
how to maintain a **hot tub**
how to maintain a **healthy relationship**
how to maintain a **car**
how to maintain a **long distance relationship**
how to maintain a **saltwater pool**
how to maintain a **septic tank**
how to maintain a **healthy lifestyle**

Google Search | I'm Feeling Lucky

Report inappropriate predictions

N-gram language models

- Question: how to learn a language model?
- Answer (pre-deep learning): learn an n-gram language model.

- Definition: An n-gram is a chunk of n consecutive words.
  - Uni-grams: “the”, “students”, “opened”, “their”
  - Bi-grams: “the students”, “students opened”, “opened their”
  - Tri-grams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-gram language models

- We make a simplifying assumption: the next word only depends on the preceding n-1 words
- Thus, the probability can be written as:

Prob. of a n-gram

$$P(x^{(t+1)}|x^{(t)},...,x^{(t-n+2)}) = \frac{P(x^{(t+1)},x^{(t)},...,x^{(t-n+2)})}{P(x^{(t)},...,x^{(t-n+2)})}$$

Prob. of a (n-1)-gram

- Q: How do we get these n-gram and (n-1)-gram probabilities?

N-gram language models

- Statistical approximation: By counting them in large text corpus!

$$P(x^{(t+1)}|x^{(t)},...,x^{(t-n+2)}) = \frac{P(x^{(t+1)},x^{(t)},...,x^{(t-n+2)})}{P(x^{(t)},...,x^{(t-n+2)})}$$

- Assuming 4-gram language model:

$$p(w_j|\text{“students opened their”}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}$$

For example, suppose that in this corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
- -> P(“books”| “students opened their”) = 0.4
- “students opened their exams” occurred 100 times
- -> P(“exams”| “students opened their”) = 0.1

Problems with N-gram language models

- Sparsity problem 1: what if w_j never occurred in the text, $p(w_j \mid$"students opened their"$) = 0$
  - (partial) solution: add small σ to count for every $w_j \in V$, this is called smoothing
- Sparsity problem 2: what if "students opened their" never occurred in the text?
  - (partial) solution: just condition on "opened their", this is called backoff
- Note: increasing n making sparsity problems worse. Typically we cannot have n bigger than 5
- Storage: need to store count for all possible n-grams. So model size is O(exp(n))
- Increasing n makes model size huge!

n-gram language models in practice

- You can build a simple tri-gram language model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*
- today the ___

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* https://nlpforhackers.io/language-models/

Generating text with a n-gram language model

- You can also use a language model to generate text.

today the ___

Condition on this

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039

sample

* https://nlpforhackers.io/language-models/

Generating text with a n-gram language model

- You can also use a language model to generate text.

today the price ____

Condition on this

of	0.308
for	0.050
it	0.146
to	0.046
is	0.031

...

sample

* https://nlpforhackers.io/language-models/

Generating text with a n-gram language model

- You can also use a language model to generate text.

today the price of ____

Condition on this

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
...	

sample

* https://nlpforhackers.io/language-models/

Generating text with a n-gram language model

- You can use a language model to generate text: today the price of gold___
  - Result: *today the price of gold per ton, while production of shoe lasts and shoe industry, the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks, sept 30 end primary 76 cts a share*
- Surprisingly grammatical!

- Incoherent!: we need to consider more than 3 words at a time if we want to generate good text.
- But increasing n worsens sparsity problem, and increase model size...

How to build a neural language model?

- Recall the language model task:
  - input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - output: prob dist of the next word
    - $P(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)})$
- How about a window-based neural model?

~~as the proctor started the clock~~ the students opened their _____

discard

fixed window

A fixed-window neural language model

- output distribution

$$\hat{y} = \text{softmax}(Uh + b_2) \in R^{|V|}$$

- hidden layer

$$h = f(We + b_1)$$

- concatenated word embeddings

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

- words/one-hot vectors:

$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$

A fixed-window neural language model

- Improvements over n-gram LM:
  - No sparsity problem
  - Model size is O(n) not O(exp(n))
- Remaining problems:
  - Fixed window is too small
  - Enlarging window enlarges W
  - Window can never be large enough!
  - Each x(i) uses different rows of W. We don't share weights across window.

We need a neural architecture that can process any length input

Recurrent Neural Networks (RNN)

A family of neural architectures

- Core idea: apply the same weights W repeatedly:

Recurrent Neural Networks (RNN)

A family of neural architectures

- Core idea: apply the same weights W repeatedly:

A RNN language model

- output distribution

$$\hat{y}^{(t)} = \text{softmax}(Uh^{(t)} + b_2) \in R^{|V|}$$

- hidden layer

$$h^{(t)} = \tanh(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

- word embeddings:

$$e^{(t)} = Ex^{(t)}$$

- words/one-hot vectors:

$$x^{(t)} \in R^{|V|}$$

Note: this input sequence could be much longer

A RNN language model

- Advantages:
  - Can process any length input
  - Model size doesn't increase for longer input
  - Computation for step t can use information from many steps back (in theory)
  - Weights are shared across time stamps: representation are shared
- Disadvantages:
  - recurrent computation is slow
  - in practice, difficult to access information from multiple steps back

Training A RNN language model

- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
- Loss function on step t is usual cross-entropy between out predicted probability $\hat{y}^{(t)}$ and the true next word $y^{(t)} = x^{(t+1)}$:

$$J^{(t)}(\theta) = \mathrm{CE}(y^{(t)}, \hat{y}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

- Average this to get overall loss for entire training set

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

Training A RNN language model

Training A RNN language model

Training A RNN language model

Training A RNN language model

Training A RNN language model

Training A RNN language model

- Computing loss and gradients across entire corpus is too expensive!
- Recall: Stochastic Gradient Descent allows us to compute loss and gradient for small set of data and update
- In practice, consider x(1),x(2),...,x(t) as a sentence:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

- Compute loss J(θ) for a sentence, compute gradients and update weights. Repeat.

Backpropagation for RNNs

- What is the derivative of $J(\theta)^{(t)}$ w.r.t the repeated weight matrix W_h?
- Answer:

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \mathbf{W}_h}\Big|_{(i)}$$

- Backpropagate over time steps i=t,...,0,summing gradients as you go. This algorithm is called "backpropagation through time"

"The gradient w.r.t a repeated weight is the sum of the gradient w.r.t each time it appears"
WHY?

Multivariable chain rule

- Given a multivariable function f(x,y), and two single variable functions x(t) and y(t), here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs: proof sketch

- Given a multivariable function f(x,y), and two single variable functions x(t) and y(t), here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

In our example:

Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(i)} \underbrace{\boxed{\frac{\partial W_h|_{(i)}}{\partial W_h}}}_{=1}$$

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(i)}$$

Backpropagation for RNNs

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \boxed{\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \mathbf{W}_h}\Big|_{(i)}}$$

Q: how do we calculate this?

Answer: backpropagate over time steps i = t, … ,0, summing gradients as you go.
This is called “backpropagation through time”

Generating text with a RNN language model

- Just like a n-gram LM, we can use a RNN LM to generate text by repeated sampling. Samplied output is next step's input.

Generating text with a RNN language model

RNN-LM trained on Obama speeches :

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. ...

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN language model

RNN-LM trained on *Harry Potter*:

"Sorry", Harry shouted, panicking -- "I'll lease those brooms in London, are they?"
"No idea", said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle of Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed, He reached the teams too.

https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN language model

RNN-LM trained on recipes:

```
MMMMM----- Recipe via Meal-Master (tm) v8.05

     Title: CARAMEL CORN GARLIC BEEF
 Categories: Soups, Desserts
     Yield: 10 Servings

     2 tb Parmesan cheese, ground
   1/4 ts Ground cloves
          -- diced
     1 ts Cayenne pepper

  Cook it with the batter. Set aside to cool. Remove the peanut oil in a small saucepan and
  pour into the margarine until they are soft. Stir in a
  a mixer (dough). Add the chestnuts, beaten egg whites, oil, and salt
  and brown sugar and sugar; stir onto the boqtly brown it.

  The recipe from an oiled by fried and can. Beans, by Judil Cookbook, Source: Pintore,
  October, by Chocolates, Breammons of Jozen, Empt.com
```

https://gist.github.com/nylki/1efbaa36635956d35bcc

Generating text with a RNN language model

RNN-LM trained on paint color names:

https://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

Evaluating language models

- The standard evaluation metric for language models is perplexity

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{LM}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \ldots, \mathbf{x}^{(1)})} \right)^{1/T}}_{\text{Inverse probability of corpus, according to LM}}$$

Normalized by number of words

- This is equal to the exponential of the cross-entropy loss J(θ)

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{y}_{x_{t+1}}^{(t+1)}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

Lower perplexity is better

RNNs have greatly improved perplexity

n-gram model

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Why should we care about LM?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Summary

- Language Model: A system that predicts the next word
- Recurrent Neural Network: a family of neural networks that
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step
- RNNs $\not\Leftrightarrow$ Language Model
- We've shown that RNNs are a great way to build a LM
- but RNNs are useful for much more!

RNNs can be used for tagging

- e.g part-of-speech tagging, named entity recognition

RNNs can be used for sentence classification

- e.g sentiment classification

How to compute sentence encoding?

RNNs can be used for sentence classification

- e.g sentiment classification

How to compute sentence encoding?

<u>Basic way:</u>
use final hidden state

RNNs can be used for sentence classification

- e.g sentiment classification

How to compute sentence encoding?

<u>Usually better:</u>
take element-wise max or mean of all hidden states

RNNs can be used for encoder module

- e.g question answering, machine translation, many other tasks!

RNNs can be used to generate text

- e.g speech recognition, machine translation, summarization

This is an example of a conditional language model.
See Machine Translation in much more detail later.

A note on terminology

- RNN described in this lecture = “vanilla RNN”

- Next lecture: learn about other RNN flavors like GRU and LSTM and multi-layer RNNs

- By the end of the course, you will understand phrases like
  “stacked bidirectional LSTM with residual connections and self-attention”

Useful links

- Recurrent neural networks cheatsheet:
  - https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

- Implementing RNN from scratch: https://github.com/pangolulu/rnn-from-scratch

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You