Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 6-1: Language models and recurrent neural networks

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 4 月 1 日

Adapted in part from CS 224n and Dive into deep learning

Recap: a journey to sentences

- Sentence: tokens (words and subwords); parsing;
- Now it is time to combine tokens and the grammar structures to sequences (sentences).
- For a sentence, it can be tokenized and we can represent is as a vector $\mathbf{x} = (x_1, \cdots, x_L)$ where each x_i is from a vocabulary $\mathcal{V}$.

Recap: probability models

- For a sequence $\mathbf{x} = (x_1, \cdots, x_L)$, a language model assigns a probability $p(x_1, \cdots, x_L)$

Recap: next token prediction and auto-regressive models

- Xiamen University is located in ????.
- Xiamen (80%); Fujian (10%); China (5%); others
- The probability of the i-th token is based on the previous $i-1$ tokens

$$p(x_i|x_1, \cdots, x_{i-1}) \qquad \text{for} \qquad i = 1, 2, \cdots, L$$

- $$\sum_{v \in \mathcal{V}} p(v|x_1, \cdots, x_i) = 1$$

- $p(x_1, \cdots, x_L) = p(x_1)p(x_2|x_1) \cdots p(x_L|x_1, \cdots, x_{L-1})$.

Recap: evaluation of language models

- The standard evaluation metric for Language Models is perplexity.
- Perplexity:

$$\exp\left(\text{Perplexity}(p)\right) = \prod_{i=1}^{L} \left(p(x_i|x_1, \cdots, x_{i-1})\right)^{-1/L}.$$

- $$\text{Perplexity}(p) = -\frac{1}{L}\sum_{i=1}^{L} \log p(x_i|x_1, \cdots, x_{i-1})$$

- Lower perplexity is better.

n-gram language models

- Xiamen University is located in [ ].
- Unigrams: “Xiamen”, “University”, “is”, “located”, “in”
- Bigrams: “Xiamen University”, “University is”, “is located”, “located in”
- Trigrams: “Xiamen University is”, “University is located”, “is located in”
- 4-grams: “Xiamen University is located”, “University is located in”.

n-gram language models

- Unigram: $p(x_1, \cdots, x_L) = p(x_1) \cdots p(x_L)$ (independent)
- Bigram: $p(x_1, \cdots, x_L) = p(x_1)p(x_2|x_1) \cdots p(x_L|x_{L-1})$ (Markov chain).
- Trigram: $p(x_1, \cdots, x_L) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1) \cdots p(x_L|x_{L-1}, x_{L-2})$.
- n-gram:

$$\begin{aligned} p(x_i|x_1, \cdots, x_i) &= p(x_i|x_{i-n+1}, \cdots, x_{i-1}) \\ &= \frac{p(x_i, x_{i-1}, \cdots, x_{i-n+1})}{p(x_{i-1}, \cdots, x_{i-n+1})}. \end{aligned}$$

Word Frequency

- Consider the following example corpus:

I love you tons.
I love you 3000.
I am Iron Man.
You love me.

- $n(x_i, x_{i+1}, \cdots, x_{i+n})$: number of occurrences of consecutive word pairs $(x_i, x_{i+1}, \cdots, x_{i+n})$.
- For example, $n(\text{love}, \text{you}) = 2$

Estimate based on frequencies

- $p(\text{love}) = \frac{n(\text{love})}{\text{total number of words}} = 2/15.$
- Another way to estimate the prob of the first word is to count the number of sentences start with that word (since it is the first word of a sentence).
- For example $p(\text{I}) = 3/4$ rather than $3/15$.
- $p(\text{you}|\text{love}) = \frac{n(\text{love,you})}{n(\text{love})} = 2/3.$

Sparsity problems of n-gram models

- $n(x_i, x_{i+1}, \cdots, x_{i+n}) = 0$
- they love you [ ]
- $$p(w|\text{they love you}) = \frac{n(\text{they love you } w)}{n(\text{they love you})}.$$
- Smoothing (for both the numerator and denominator): $n(\text{they love you } w) + \delta$ for some $\delta > 0$
- Backoff (only works for the denominator): $n(\text{they love you}) \Rightarrow n(\text{love you})$

Laplace smoothing

- Let N be the total number of words and let M be the number of unique words in the corpus.
- $$p(x) = \frac{n(x) + \epsilon_1/M}{n + \epsilon_1}$$
- $$p(x'|x) = \frac{n(x, x') + \epsilon_2 p(x')}{n(x) + \epsilon_2}$$
- $$p(x''|x, x') = \frac{n(x, x', x'') + \epsilon_3 p(x'')}{n(x, x') + \epsilon_2}.$$
- $\epsilon_1, \epsilon_2, \epsilon_3$ are hyperparameters ($\epsilon_i = 0$ and $\epsilon_i = \infty$?)

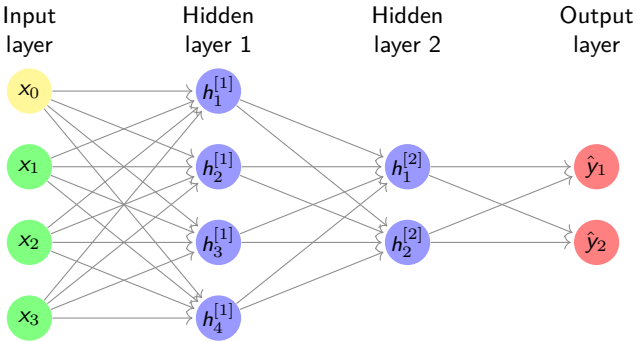
Train a *n*-gram model using "Reuters"

- n-gram.ipynb

Main weaknesses of *n*-gram models

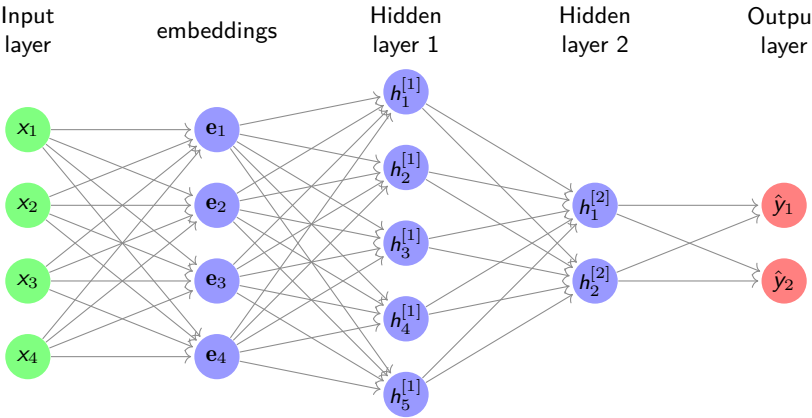
- Sparsity problems: Some *n*-grams may be rare and smoothing may not be suitable for language modeling.
- The sparsity problem is severe when n is large, usually $n \leq 5$.
- Storage Problems: need to store all counts of *n*-grams in the corpus.
- Ignores the meaning of the words.
- A neural network model wins again.

Recap: Neural network



- $\widehat{y} = \text{SoftMax}\ \sigma(W^{[2]}\sigma(W^{[1]}\mathbf{x} + \mathbf{b}))$.

Neural Language Models



- $\widehat{y} = \mathrm{SoftMax}\ \sigma(W^{[2]}\sigma(W^{[1]}\mathbf{e} + \mathbf{b}))$ with $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4)$ (each $\mathbf{e}_i$ is a d-dimensional vector.)
- Here $\mathbf{x} = (x_1, x_2, x_3)$ is a sequence of tokens or one-hot vectors.
- The embeddings, weights, bias are all trainable parameters.

Fixed-window neural Language model

- Xiamen Univeristy is located in _____.
- Input layer: $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5) =$ (Xiamen, Univeristy, is, located, in).
- Output: $p(\text{Xiamen}|\mathbf{x}) = \widehat{y}^{(1)}$, $p(\text{China}|\mathbf{x}) = \widehat{y}^{(2)}$, $p(\text{Fujian}|\mathbf{x}) = \widehat{y}^{(3)}$, ...
- $\widehat{\mathbf{y}} = (\widehat{y}^{(i)})_{i=1}^{|\mathcal{V}|}$.

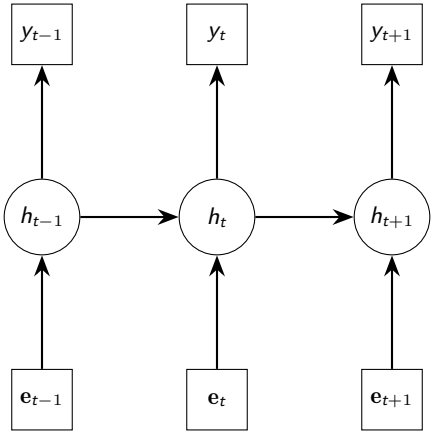
Window size

- What if the window size is too small?
  - $p(?|\text{University is located in})$
  - The prediction is not accurate.
- What if the window size is too large?
  - Too many trainable parameters; computationally inefficient.
- We need a model that can process sequences of any length!

Working with sequences

- For each sentence with L tokens, we have a sequence of vectors $\mathbf{e} = (\mathbf{e}_1, \cdots, \mathbf{e}_L)$.
- For previous neural networks, we consider an i.i.d. dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$.
- The input of a sequence is $\mathbf{e} = (\mathbf{e}_1, \cdots, \mathbf{e}_L)$ and the output can be a single $\mathbf{y}$ or a corresponding sequence $(y_1, \cdots, y_L)$.
- Some datasets consist of a single massive sequence (such as the FTSE 100 index)
- For language models, we often have a collection of sequences $\{(\mathbf{e}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^n$.
- Here each $\mathbf{e}^{(i)}$ is an input sequence (such as a sentence) and $\mathbf{y}^{(i)}$ is the corresponding output.

A neural network for sequences



Mathematical form

- $y_t = p(\mathbf{e}_{t+1}|h_t)$
- $h_t = g_t(h_{t-1}, \mathbf{e}_t)$ for some function g_t.
- $h_{t-1} = g_{t-1}(\mathbf{e}_{t-1}, h_{t-2})$.
- Overall, y_t depends on all previous inputs $\mathbf{e}_1, \cdots, \mathbf{e}_t$
- This model is also know as a latent autoregressive model since h_t is latent.

Embeddings, vocabularies, and sequence

- A vocabulary $\mathcal{V}$= {I, he, am, is, 30, 20, .}
- Embeddings: $\mathbf{e}_{\text{I}}, \mathbf{e}_{\text{he}}, \cdots, \mathbf{e}_{20}, \mathbf{e}_{.} \in \mathbb{R}^d$.
- A sentence $\mathbf{x} = (\text{I}, \text{am}, 30, .)$
- Embeddings: $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4)$ with $\mathbf{e}_1 = \mathbf{e}_{\text{I}}, \mathbf{e}_2 = \mathbf{e}_{\text{am}}, \mathbf{e}_3 = \mathbf{e}_{30}, \mathbf{e}_4 = \mathbf{e}_{.}$.

Embeddings, vocabularies, and sequence

- A vocabulary $\mathcal{V} = \{v_1, v_2, \cdots, v_{|\mathcal{V}|}\}$
- Embeddings: $\mathbf{e}_{v_1}, \mathbf{e}_{v_2}, \cdots, \mathbf{e}_{v_{|\mathcal{V}|}} \in \mathbb{R}^d$.
- A sentence $\mathbf{x} = (x_1, x_2, \cdots, x_L)$ with each $x_i \in \mathcal{V}$.
- Embeddings: $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_L)$ with $\mathbf{e}_i = \mathbf{e}_{x_i}$.

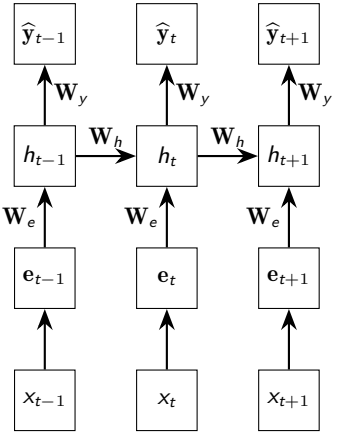
A lauguage model

- A language model (denoted as p) assigns a probability $p(\mathbf{x})$ to a sequence $\mathbf{x} = (x_1, \cdots, x_L)$.
- $$\sum_{L=1}^{\infty} \sum_{\mathbf{x} \in \mathcal{V}^L} p(\mathbf{x}) = 1.$$
- Why? There will be "<START>" and "<END>" tokens in the vocabulary $\mathcal{V}$ that decides the length of a sentence. The "<END>" token is assigned some probability.

Auto-regressive models and neural networks

- $p(\mathbf{x}) = p(x_1) \prod_{i=2}^{L} p(x_i|x_1, \cdots, x_{i-1})$ sample $x_i \sim p(\cdot|x_1, \cdots, x_{i-1})$.
- For each position t, we aim to find a vector $\widehat{\mathbf{y}}_t = (\widehat{y}_t^{(1)}, \cdots, \widehat{y}_t^{(|\mathcal{V}|)}) \in [0,1]^{|\mathcal{V}|}$ such that $\widehat{\mathbf{y}}_t \approx p(\cdot|x_1, \cdots, x_{t-1})$.
- RNN: $\widehat{\mathbf{y}}_t = f_t(x_t, h_t(x_1, \cdots, x_{t-1})) \approx p(\cdot|x_1, \cdots, x_t)$ for some function h_t.
- $h_t(x_1, \cdots, x_{t-1}) = g(x_{t-1}, h_{t-1}(x_1, \cdots, x_{t-2}))$ for some function g.

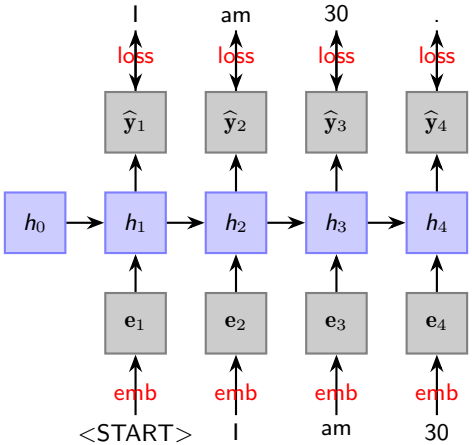
A vanilla recurrent neural network



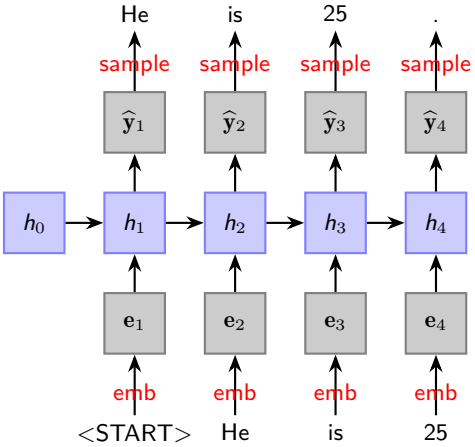
A vanilla recurrent neural network

- $h_t = \sigma(\mathbf{W}_e \mathbf{e}_t + \mathbf{b}_e + \mathbf{W}_h h_{t-1} + \mathbf{b}_h)$
- $\widehat{\mathbf{y}}_t = \mathrm{SoftMax}\left(\mathbf{W}_y h_t + \mathbf{b}_y\right) \in \mathbb{R}^{|\mathcal{V}|}$.
- The initialization h_0 is usually a 0 vector.
- The starter $\mathbf{e}_1$ is usually a seed token such as "<START>".
- σ can be a relu function or a tanh function.
- Consider an example of "<START> I am 30."

Forward propagation



Generating sequences



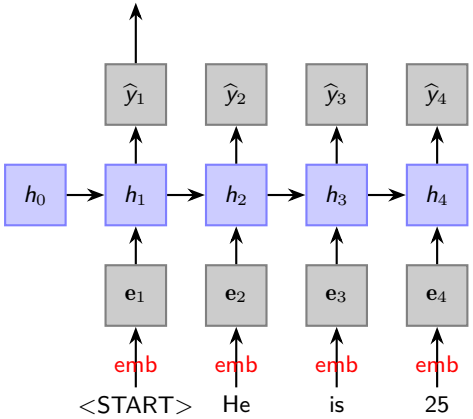
How to train?

- Recall $\widehat{\mathbf{y}}_t \in [0,1]^{|\mathcal{V}|}$ assigns a probability $p_\theta(\cdot|h_t) \approx p(v|x_1, \cdots, x_t)$ for all $v \in \mathcal{V}$.
- θ is a vector of all trainable parameters $(\mathbf{W}_y, \mathbf{b}_y, \mathbf{W}_h, \mathbf{b}_h, \{\mathbf{e}_{v_i}\}_{i=1}^{|\mathcal{V}|}.)$
- What is the true label?
- The true label y_t is the one-hot vector of the true next word.
- $\mathcal{V} = \{\text{Xiamen}, \text{Fujian}, \text{China}\}$.
- Xiamen University is located in _____
- Your predictor $\widehat{\mathbf{y}}_t = [0.8, 0.15.0.05]$
- True lable $\mathbf{y}_t = [1, 0, 0]$.

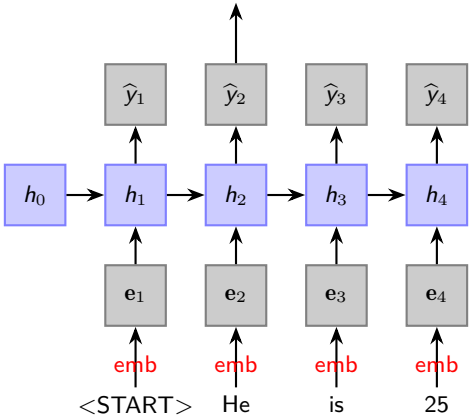
Cross entropy loss again!

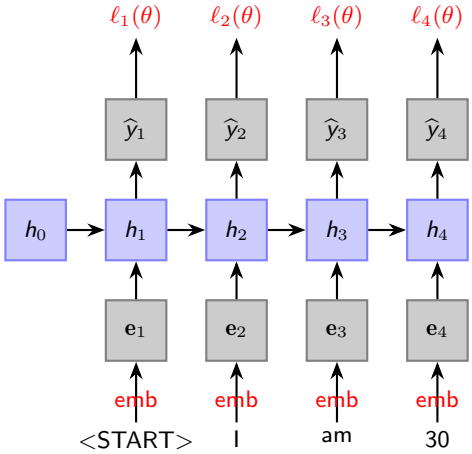
- The predicted probability of the $(t+1)$-th word $\widehat{\mathbf{y}}_t \in [0,1]^{|\mathcal{V}|}$ and true label $\mathbf{y}_t \in \{0,1\}^{|\mathcal{V}|}$ are both $|\mathcal{V}|$-dimensional vectors.
- $$\ell_t(\theta) = \mathrm{CE}(\mathbf{y}_t, \widehat{\mathbf{y}}_t) = -\sum_{i=1}^{|\mathcal{V}|} y_t^{(i)} \log \widehat{y}_t^{(i)}.$$
- E.g. $\mathrm{CE}([1,0,0],[0.8,0.15.0.05])$.

$\ell_1(\theta) = -\log$ predcited probability of "He"



$\ell_2(\theta) = -\log$ predcited probability of "is"




$\ell_1(\theta)$
$\ell_2(\theta)$
$\ell_3(\theta)$
$\ell_4(\theta)$
$\widehat{y}_1$
$\widehat{y}_2$
$\widehat{y}_3$
$\widehat{y}_4$
h_0
h_1
h_2
h_3
h_4
$\mathbf{e}_1$
$\mathbf{e}_2$
$\mathbf{e}_3$
$\mathbf{e}_4$
emb
emb
emb
emb
<START>
I
am
30

Train a language model

- Training data: A corpus containing a large amount of words which forms a sequence of length L.
- The overall loss is to average all words (tokens):

$$l(\theta) = \frac{1}{L}\sum_{t=1}^{L} \ell_t(\theta) = \frac{1}{L}\sum_{t=1}^{L} \mathrm{CE}(y_t, \widehat{y}_t).$$

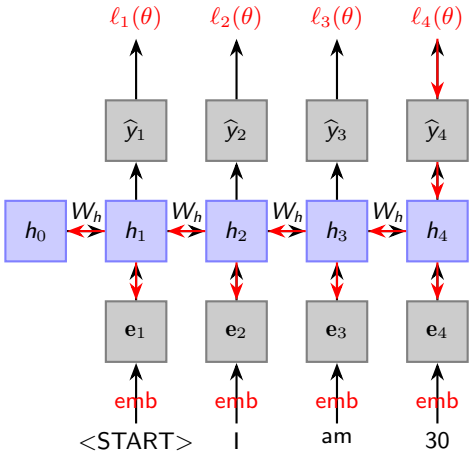
- This explains why perplexity is used to evaluate your model.
- Train using the backpropagation.

RNN in PyTorch

```
class SentimentRNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.RNN(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
```

- What is the embedding layer? (cf., `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html`)
- What is the activation function here? (cf., `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html`)

Backpropagation of RNN



Chain rule: interactive session

For $\mathbf{W}_h = [\mathbf{w}_{h1}, \cdots, \mathbf{w}_{hd}] \in \mathbb{R}^{d \times d}$, define $\mathbf{w}_h = \mathrm{Vec}(\mathbf{W}_h) = (\mathbf{w}_{h1}^T, \cdots, \mathbf{w}_{hd}^T)^T \in \mathbb{R}^{d^2}$. Calculate $\frac{\partial \ell_4}{\partial \mathbf{w}_h}$ and $\frac{\partial \ell_4}{\partial \mathbf{W}_h}$

Chain rule: interactive session

- $\widehat{\mathbf{y}}_t = g(\mathbf{W}_y, h_t)$ and $h_t = f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)$.
- $$\frac{\partial \ell_t}{\partial \mathbf{w}_h} = \frac{\partial \ell}{\partial \widehat{\mathbf{y}}_t} \frac{\partial g(\mathbf{w}_y, h_t)}{\partial h_t} \frac{\partial h_t}{\partial \mathbf{w}_h}.$$
- $$\begin{aligned} \frac{\partial h_t}{\partial \mathbf{w}_h} &= \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \mathbf{w}_h} \\ &= \mathrm{Diag}(\sigma') \left[h_{t-1} \otimes I + \mathbf{W}_h \frac{\partial h_{t-1}}{\partial \mathbf{w}_h} \right] \end{aligned}$$
- Help me to verify it.

Chain rule: interactive session

Chain rule: interactive session

The gradient may be a trouble maker

- $$\begin{aligned}\frac{\partial h_t}{\partial \mathbf{w}_h} &= \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \mathbf{w}_h} \\ &= \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(\mathbf{e}_j, h_{j-1}, \mathbf{w}_h)}{\partial h_{j-1}} \right) \frac{\partial f(\mathbf{e}_i, h_{i-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h}\end{aligned}$$

- Compute the full gradient is time-consuming and is not robust to changes in initial conditions.

Truncating time steps

$$\begin{aligned}\frac{\partial h_t}{\partial \mathbf{w}_h} &= \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \mathbf{w}_h} \\ &\approx \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \sum_{i=1}^{\tau-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(\mathbf{e}_j, h_{j-1}, \mathbf{w}_h)}{\partial h_{j-1}} \right) \frac{\partial f(\mathbf{e}_i, h_{i-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h}\end{aligned}$$

for some $\tau < t$.

Random truncation

- $$\frac{\partial h_t}{\partial \mathbf{w}_h} \approx \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial \mathbf{w}_h} + \pi_t \xi_t \frac{\partial f(\mathbf{e}_t, h_{t-1}, \mathbf{w}_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \mathbf{w}_h} =: g_t$$

- $\xi_t \in \{0, 1\}$ with $\mathbb{P}[\xi_t = 0] = 1 - \pi_t$ and $\mathbb{P}[\xi_t = 1] = \pi_t$.
- $\mathbb{E}_{\xi_t}[g_t] = \frac{\partial h_t}{\partial \mathbf{w}_t}$.

The gradient may be a trouble maker

- Consider a simple case $\sigma(t) = t$
- $h_t = \sigma(\mathbf{W}_h h_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}) = \mathbf{W}_h h_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}$
- $$\frac{\partial \ell_t(\theta)}{\partial h_j} = \frac{\partial \ell_t(\theta)}{\partial h_t} \prod_{j<m\leq t} \frac{\partial h_m}{\partial h_{m-1}} = \frac{\partial \ell_t(\theta)}{\partial h_t} \mathbf{W}_h^{t-j}.$$
- What are the eigenvalues λ_i of $\mathbf{W}_h^{t-j}$?
- What if $\lambda_i < 1$ for all i? (Vanishing gradient)
- What if there is an i such that $\lambda_i > 1$? (Exploding gradient)

The proof of of non-linear activation functions are similar, please refer to the paper On the difficulty of training recurrent neural networks

Exploding gradient

- Recall SGD:
$$\theta^{new} = \theta^{old} - \eta \nabla_\theta l(\theta^{old})$$
- $\nabla_\theta l(\theta^{old})$ might be too large (its length might be infinity).
- Clip the gradient: $g(\theta) = \nabla_\theta l(\theta)$ if $\|\nabla_\theta l(\theta)\| \leq C$ for some constant C; otherwise $g(\theta) = C\frac{\nabla_\theta l(\theta)}{\|\nabla_\theta l(\theta)\|}$.
-
$$\theta^{new} = \theta^{old} - \eta g(\theta^{old})$$

Vanishing gradient

- Explained as the machine may forget what they have learned after a long time.
- What is a good way to remember vocabularies?? (lol)
- Next time: Long Short-Term Memory (LSTM)
- 让神经网络长长记性