Introduction to Natural Language Processing

Goals, Main Problems

AUA, DS330 Deep Learning 2021, Hovhannes Tamoyan

Natural Language Processing

Natural Language Processing is concerned with tasks involving language data.

We will focus on text data NLP.

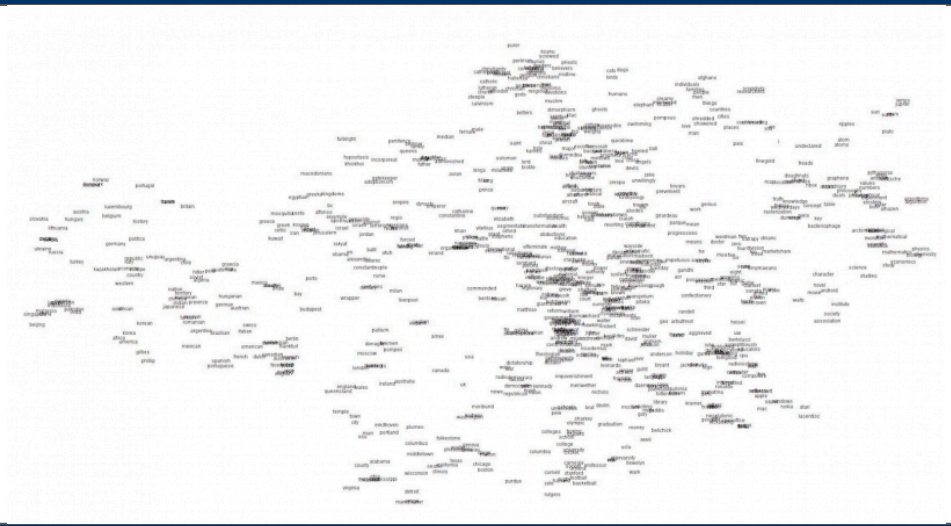
Speech recognition is also NLP, though it has its dedicated research community.

Main issue: text data is inherently high dimensional.

Main Idea

Central to Deep Learning and Natural Language is "word meaning," where a word and especially its meaning are represented as a vector of real numbers. With these vectors that represent words, we are placing words in a high-dimensional space.

The interesting thing about this is that the words, which are represented by vectors, will act as a semantic space. This simply means the words that are similar and have a similar meaning tend to cluster together in this high-dimensional vector space. You can see a visual representation of word meaning below:



Representation Learning

Commonly in NLP (Deep Learning) we learn both the W (word vectors parameters) - representation learning and the model parameters.

$$\theta = [W, x]$$

x - representations

Preprocessing - tokenization

Tokenize a long string to a list of token strings.

For many datasets this is already done for you.

Splitting into tokens based on spaces and separating punctuation is good enough in English or French.

Preprocessing - lemmatization

This is the process of putting words into their normal form: removing syntactic information, decapitalize etc.

The way of lemmatization will vary depending on the problem you are solving.

We can remove variations of words that are not relevant to the task at hand.

Preprocessing - vocabulary

Form vocabulary of words that maps lemmatized words to a unique ID.

Different criterion can be to use what words are in the vocabulary based on their frequency.

All words not in the vocabulary will be mapped to a special “out-of-vocab” ID.

Problems To Cover Today

Language Modeling (LM)

Sequence-to-Sequence Models (seq2seq)

Language Modeling

Why Language Modeling?

- 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

Language Modeling

Language Model is a probability distribution over sequences of words. The main goal is to assign probabilities to a given sequence of words.

$$P(x_1, \dots x_T) = P(x_1) \times P(x_2|x_1) \times \dots \times P(x_T, |x_{T-1} \dots x_1) =$$
$$= \prod_{t=1}^{T} P(x_t|x_{t-1}, \dots x_1)$$

Language Models (LM)

The intermediate objective is to predict what word comes next.

E.g. The students opened their ___.

More formally: given a sequence of words $x_1, x_2, \ldots, x_t$ compute the probability distribution of the next word x_{t+1}.

$$P(x_{t+1} | x_t \ldots x_1; \theta)$$

Where x_{t+1} can be any word in the vocabulary $V = \{w_t, \ldots, w_{|V|}\}$

n-gram Language Modeling

How to learn a Language Model?

Learn a n-gram Language Model!

A n-gram is a chunk of n consecutive words.

unigrams - “the”, “students”, “opened”, “their”

bigrams - “the students”, “students opened”, “opened their”

...

Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

n-gram Language Modeling

Firstly we make a simplified assumption x_{t+1} depends only on the preceding n-1 words.

$$P(x_{t+1}|x_t,\ldots,x_1) = P(x_{t+1}|x_t,\ldots,x_{t-n+2}) \quad \text{(assumption)}$$

$$\frac{P(x_{t+1},x_t,\ldots x_{t-n+2})}{P(x_t,\ldots,x_{t-n+2})} \quad \text{(definition of conditional probability)}$$

How to get these n-gram and n-1-gram probabilities?

By counting them in some large corpus of text!

$$\frac{count(x_{t+1},x_t,..,x_{t-n+2})}{count(x_t,..,x_{t-n+2})} \quad \text{(statistical approximation)}$$

n-gram Language Modeling: problems

Sparsity problem - What if the conditioned word never occurred in the data? Then the top of the fraction will have probability of 0. And what if the context never appeared in the data then we can’t calculate a probability for the conditioned word.

increasing n makes sparsity problem worse. Typically we can’t have n greater than 5.

Storage problem - Need to store the count of all n-grams we saw in the data.

Neural Language Model

Recall the LM problem:

Input: sequence of words $x_1, x_2, \ldots, x_t$.

Output: the prob dist of the next word $P(x_{t+1} | x_t \ldots x_1)$

Fixed-window Neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

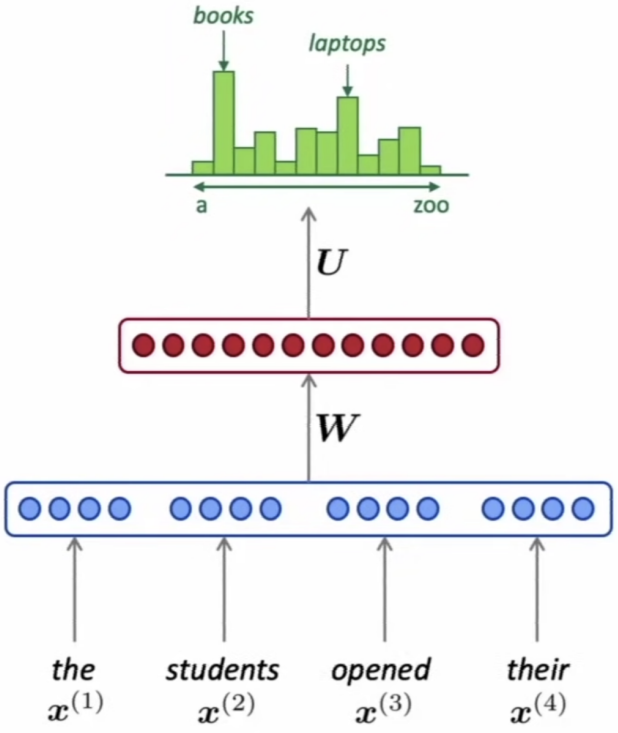
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



Neural Language Model

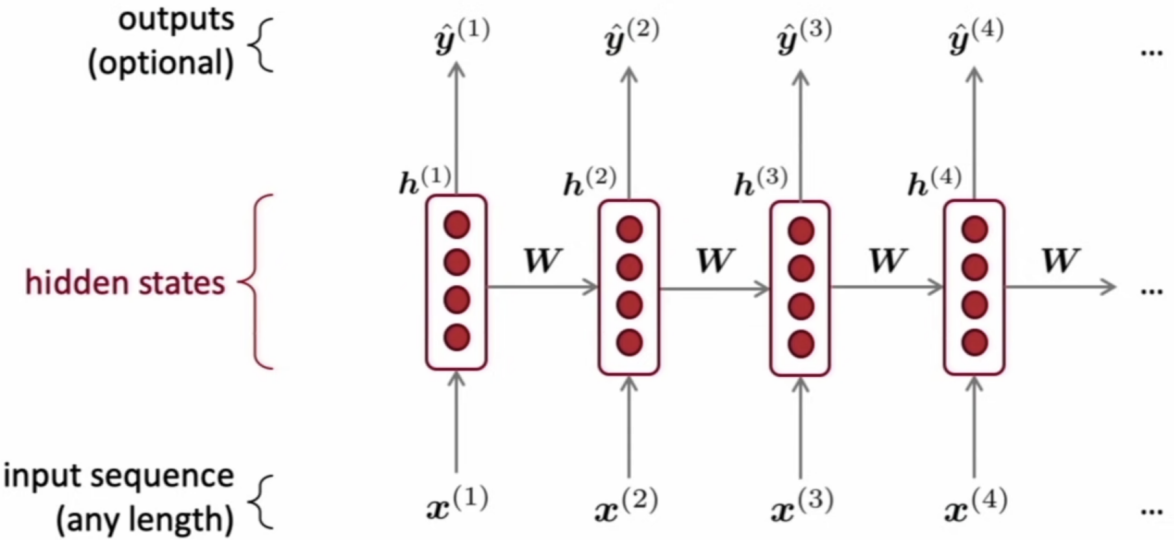
Improvements over n-gram LM:

- No sparsity problem,
- Don’t need to store all observed n-gram,

Remaining problems:

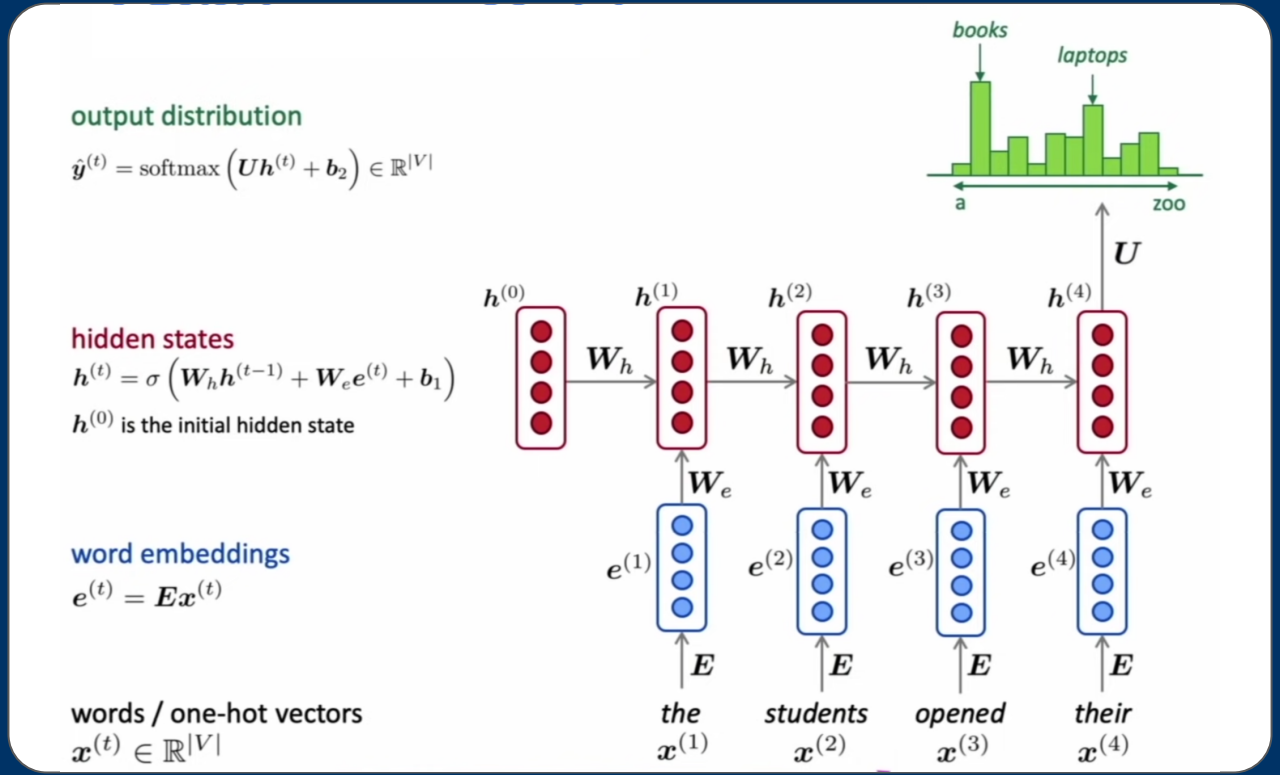
- Fixed window is too small,
- Enlarging window enlarges W
- Window can never be large enough
- No symmetry in how the inputs are processed. Xs are multiplied by completely different weights in W.

RNNs



Core idea: apply the same weights W repeatedly.

RNN Language Model



RNN Language Model

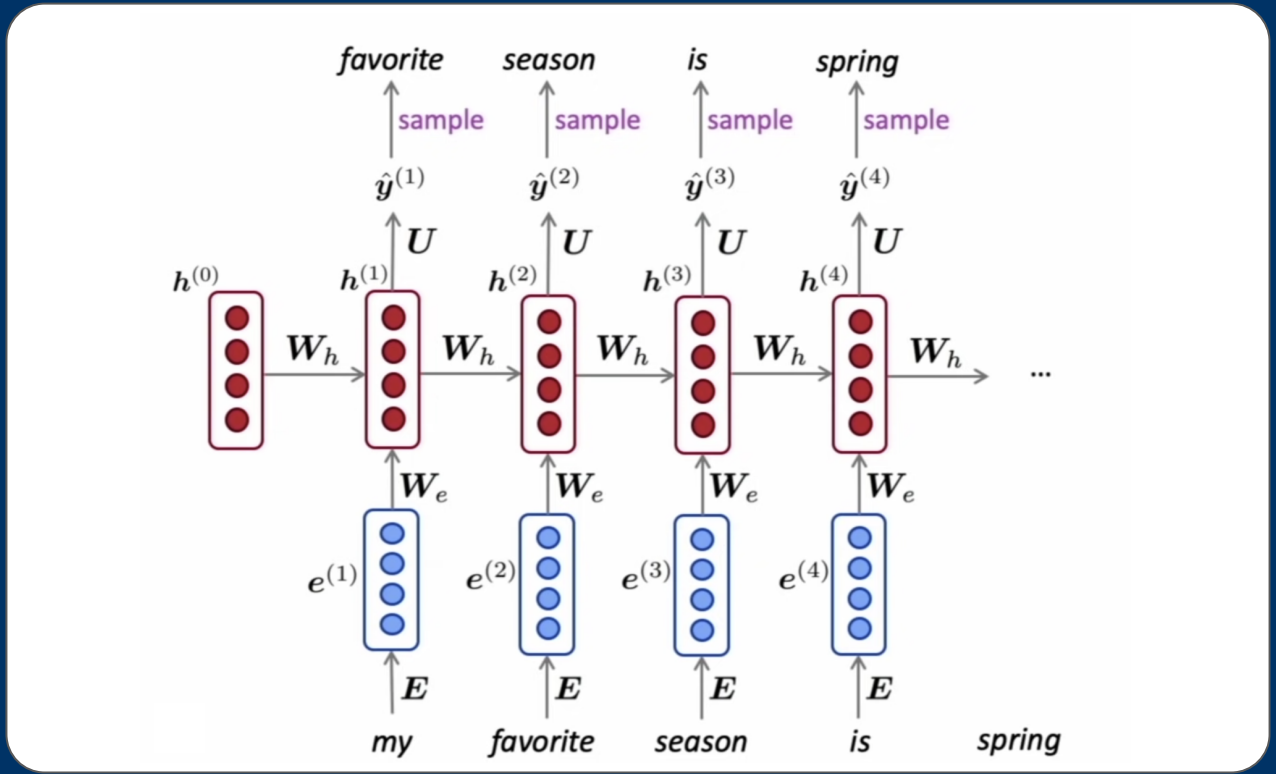
Advantages:

- RNN advantages,
- Model size doesn’t increase for longer inputs,
- Same weights applied on every timestamp, so there is symmetry in how inputs are processed.

Remaining problems:

- RNN disadvantages

Text generation with RNN LM



Training RNN

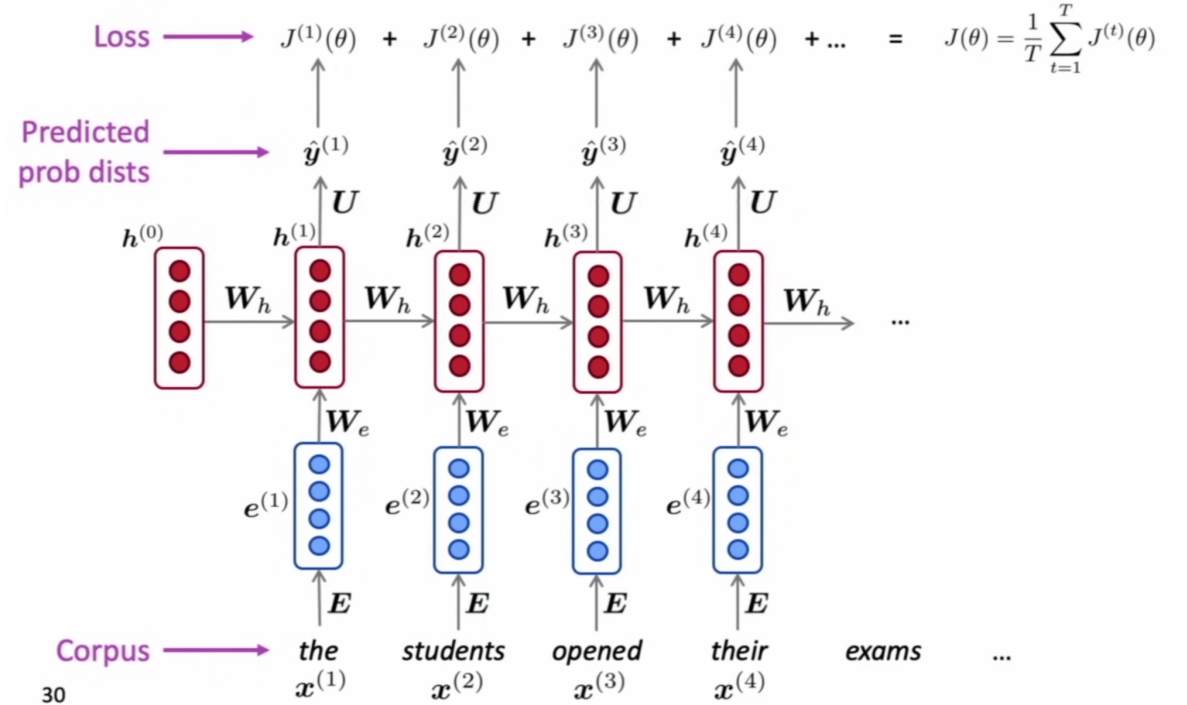
Loss function on step t is cross-entropy between predicted probability distribution and the true next word (one hot for x_{t+1}).

$$J(\theta) = -\sum_{\omega \in V} y_\omega log\hat{y}_\omega = -log\hat{y}_{x_{t+1}}$$

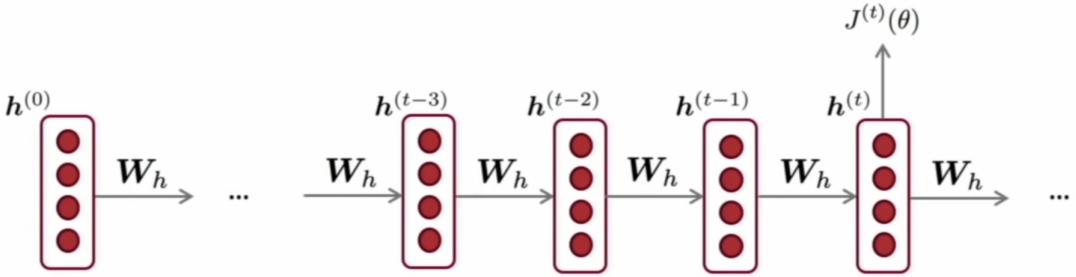
Average this to get the overall loss for entire training set.

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

Training RNN



Training RNN



Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

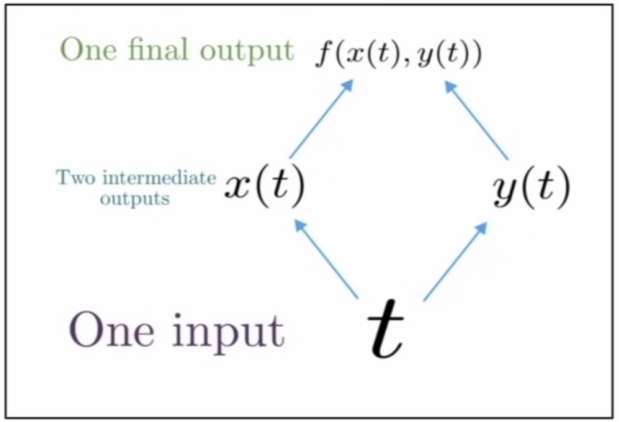
Answer: $$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}$$

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

Why it is true

- 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:

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

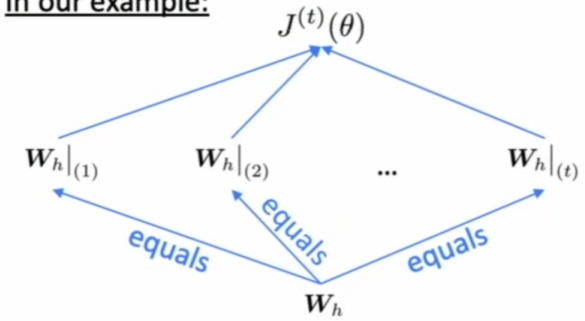


In case of RNN

- 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:

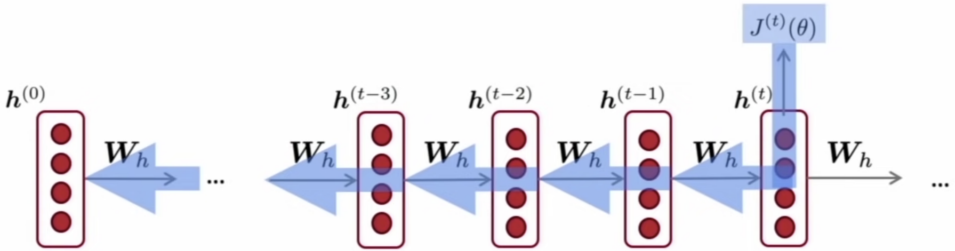


Apply the multivariable chain rule:

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

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

Backpropagation over time



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i=t,...,0*, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

Evaluating Language Models

The standard evaluation metric for Language Models is **perplexity**.

$$perplexity = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}\left(x^{(x+1)} \middle| x^{(t)}, \ldots, x^{(1)}\right)} \right)^{1/T}$$

Inverse probability of corpus, according to Language Model

Normalized by number of words

Evaluating Language Models

This (perplexity) is equal to the exponential of the cross-entropy loss.

$$perplexity = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}\left(x^{(x+1)} | x^{(t)}, \ldots, x^{(1)}\right)} \right)^{1/T} =$$

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

Low **perplexity** is good, because **perplexity** is the inverse of **probability**.

Well-Known Architectures/Models (from notes)

- BERT
- ELMo
- GPT
- RoBERTa
- mBERT
- XLM
- XLM-R
- XLNET
- ELECTRA
- T5

Thank You