CS 589 Fall 2020

Language models

RNN/word embedding

Instructor: Susan Liu
TA: Huihui Liu

Stevens Institute of Technology

The majority of slides are adapted from Stanford CS224

Dense vector representation

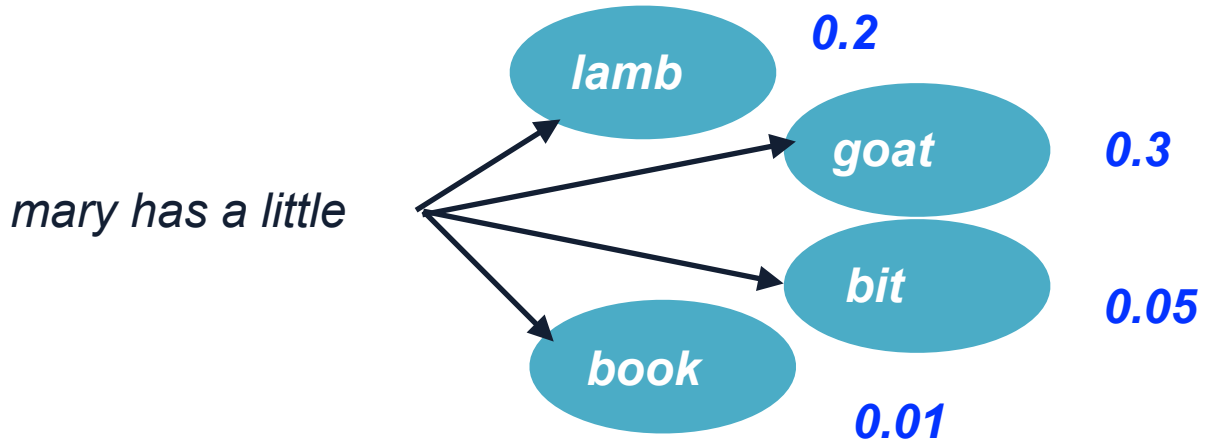
Today's lecture

- Language models
  - Word2vec

- Recurrent neural network
  - Vanilla recurrent neural network
  - Long short-term memory (LSTM)
  - Gated recurrent unit (GRU)

Language models

- Language modeling is the task of **predicting what word comes next**



- Given a sequence of words x1, x2, …, xt, compute the probability distribution for the next word xt+1

$$p(x_{t+1}|x_1, x_2, \cdots, x_t)$$

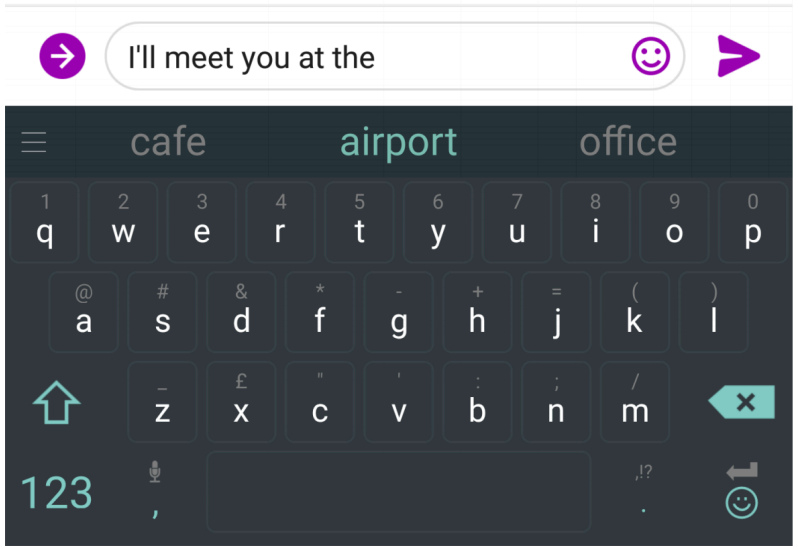
where xt+1 can be any word in the vocabulary $V = \{w_1, \cdots, w_{|V|}\}$

Language models

- Using the chain rule to estimate the probability of text:

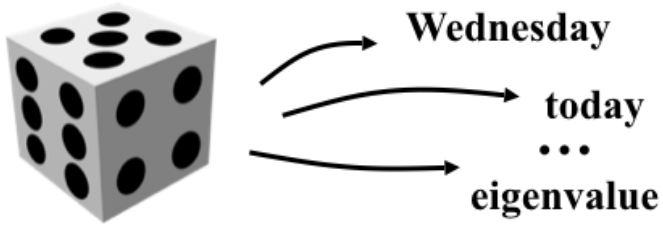
$$
\begin{aligned}
P(x_1, \ldots, x_T) &= P(x_1) \times P(x_2 \mid x_1) \times \cdots \times P(x_T \mid x_{T-1}, \ldots, x_1) \\
&= \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots, x_1)
\end{aligned}
$$

- Keyboard next word prediction:



Language models

- Collect statistics about how frequent different n-grams are, and use these to predict next word
  - Unigram language model using MLE:



$$p(\text{"}today\ is\ Wed\text{"}) = p(\text{"}today\text{"})p(\text{"}is\text{"})p(\text{"}Wed\text{"}) = 0.0002 \times 0.001 \times 0.000015$$

Table 1:	
Mary	30
little	100
lamb	30
Mary little	5
little Mary	10
little lamb	25
lamb little	0
Mary lamb	0
lamb Mary	0
little little	0
Mary Mary	0
lamb lamb	0

  - Bigram language model using MLE:

$$p(\text{"}Mary\ little\ lamb\text{"}) = p(\text{"}Mary\text{"}) \times p(\text{"}little\text{"}|\text{"}Mary\text{"}) \times p(\text{"}lamb\text{"}|\text{"}little\text{"})$$
$$= \frac{30}{160} \times \frac{5}{5} \times \frac{25}{35} = 0.133$$

Language models

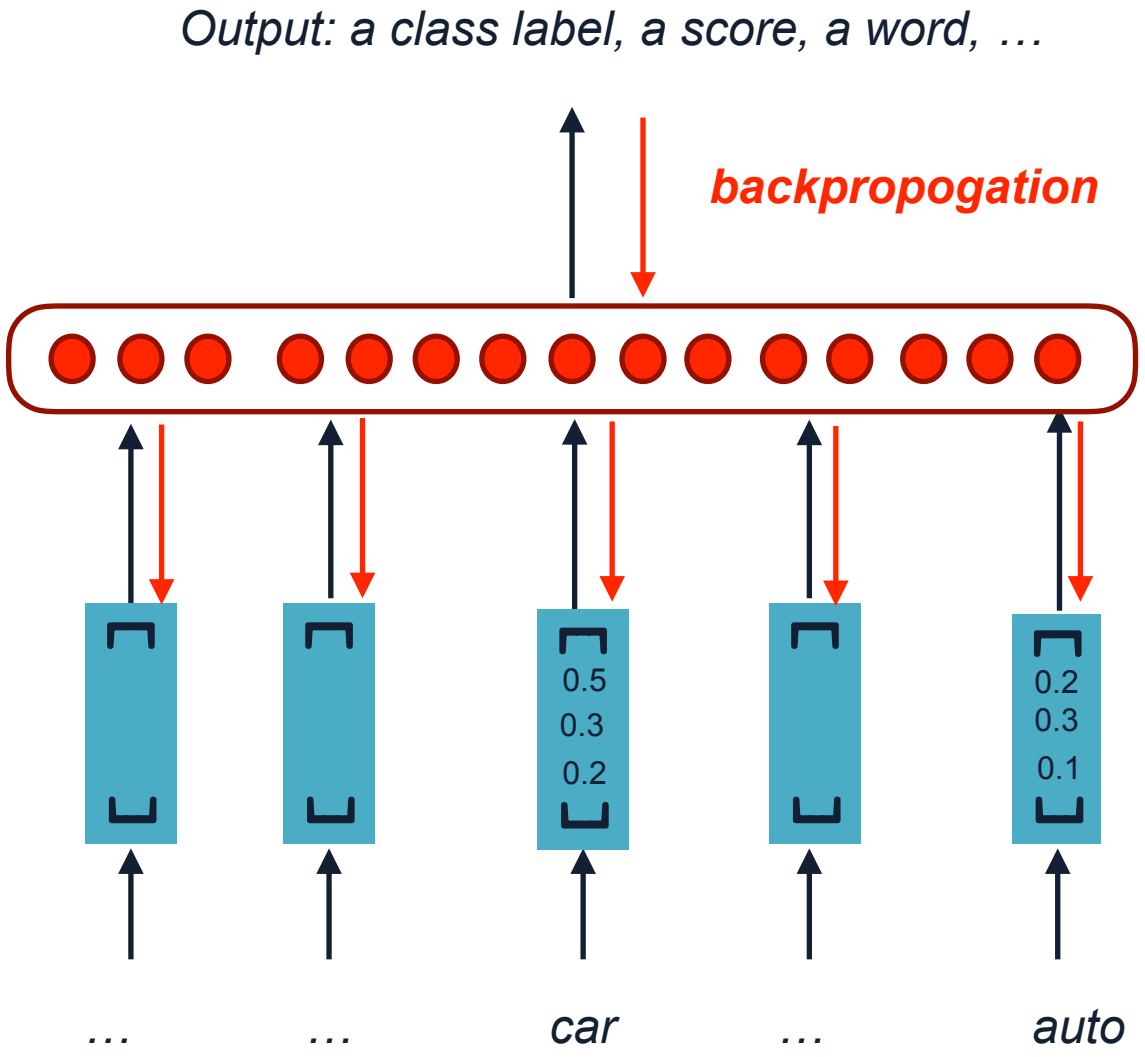
- Markov assumption: xt+1 depends only on the preceding k-1 words:

$$P\left(x_{t+1} \mid x_t, \ldots, x_1\right) = P\left(x_{t+1} \mid x_t, \ldots, x_{t-n+2}\right)$$
$$= \frac{P\left(x_{t+1}, x_t, \ldots, x_{t-n+2}\right)}{P\left(x_t, \ldots, x_{t-n+2}\right)}$$

- How do we get the n-gram statistics? By counting the n-grams in a large corpus of text

- Sparsity problem with n-gram LM: what if “mary has a little” never occurred?
  - Partial solution: just condition on “a little” instead
  - Trade off between granularity and sparsity

Neural network



output distribution

$$\hat{y} = softmax(Uh + b)$$

hidden layer

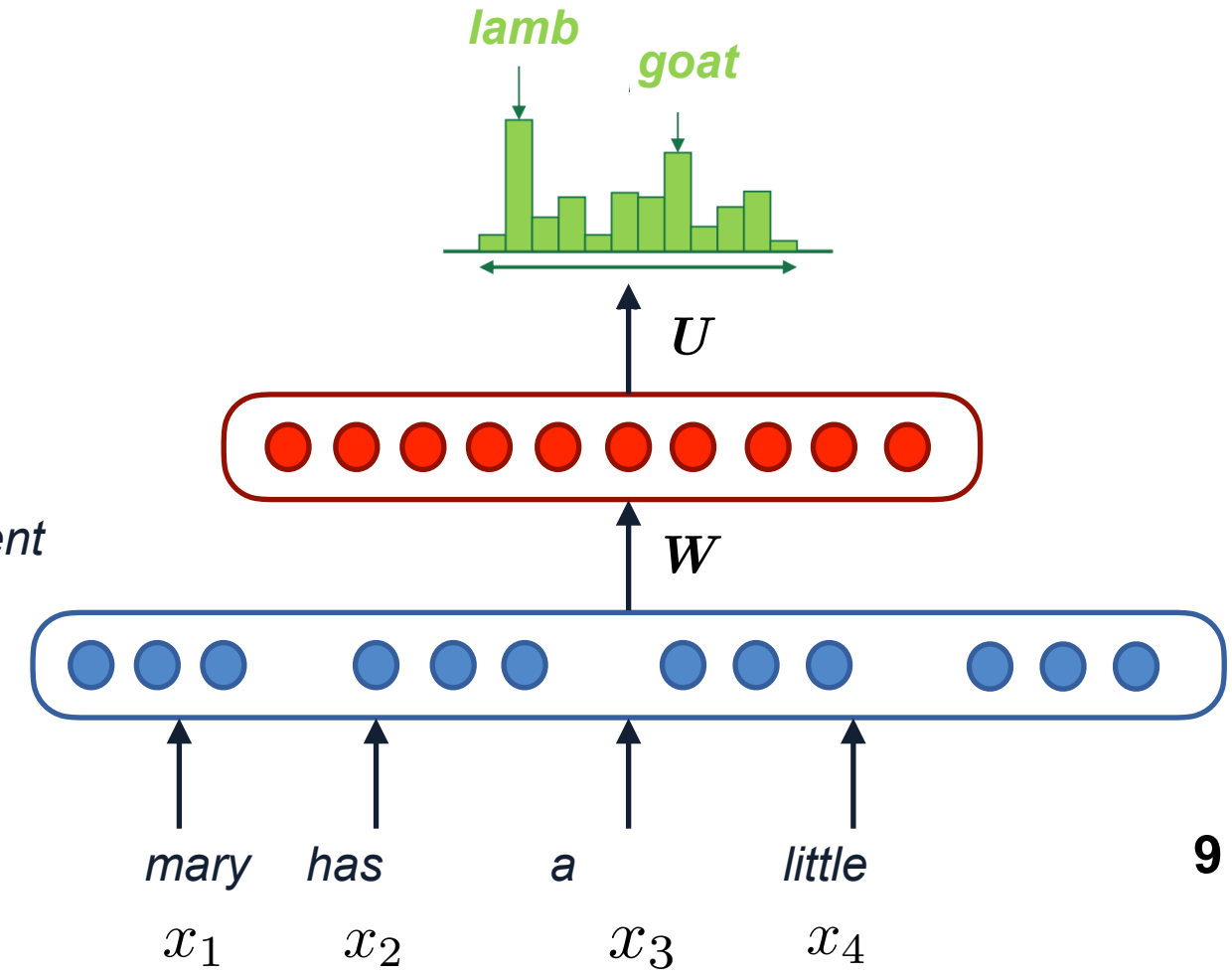
embedding layer:
1. *maps a 1-hot vector to low dim representation*
2. *randomly initialized*
3. *can be trained*

Neural language models

- Bengio et al. (2000/2003): a neural probabilisitic LM using fixed window:

Improvements over n-gram LM:
1. No sparsity problem
2. Low storage cost

Remaining problems:
1. Fixed window size too small
2. Words at different positions use different weights, ***no symmetry***



Recurrent neural network (RNN)

- Core idea: apply the same weights W repeatedly

output distribution

$$\hat{y}_t = \text{softmax}\left(Uh_t + b_2\right) \in \mathbb{R}^{|V|}$$

hidden state

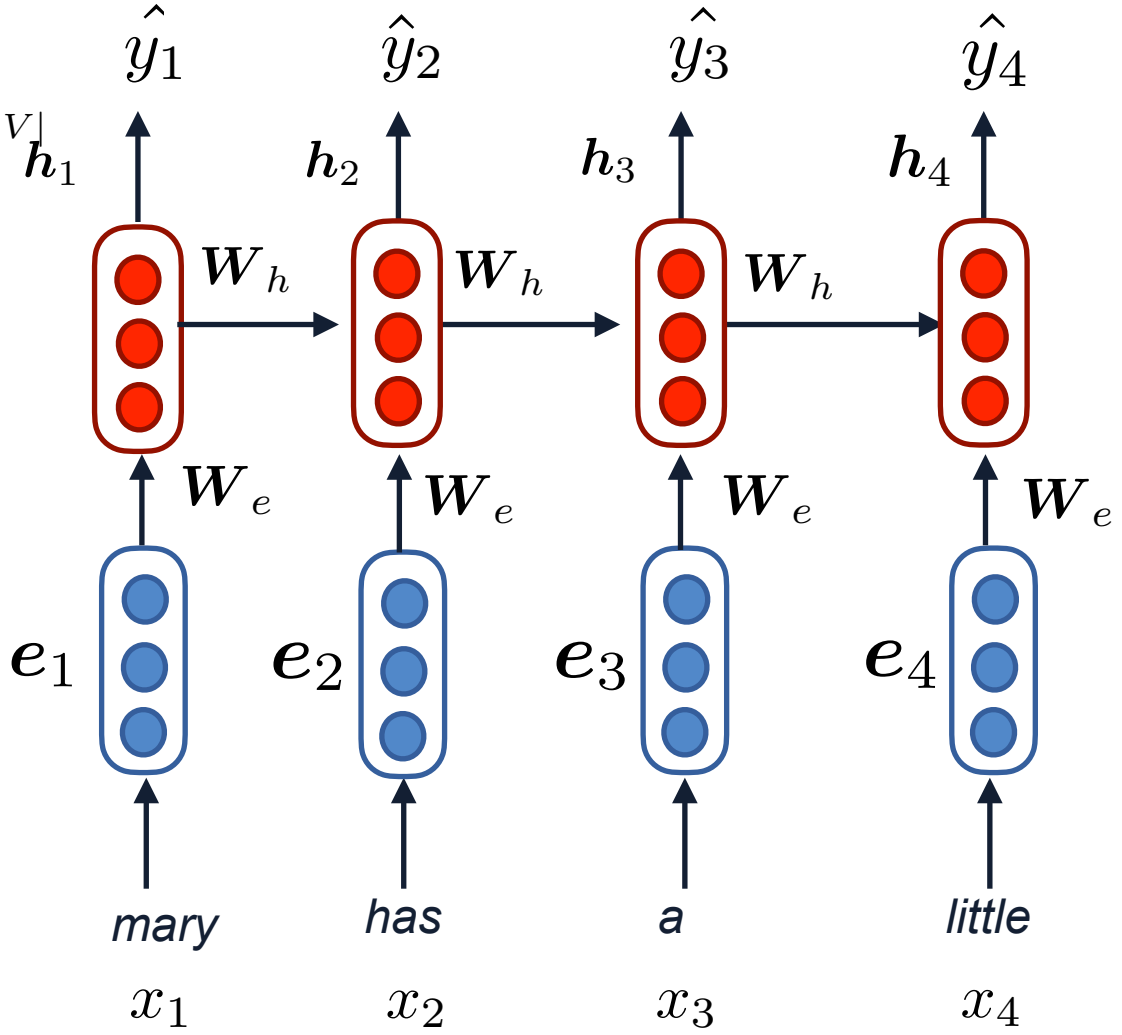
$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

word embeddings

$$e_t = \boldsymbol{E} x_t$$

word/one-hot vectors

$$x_t \in \mathbb{R}^{|V|}$$



Recurrent neural network (RNN)

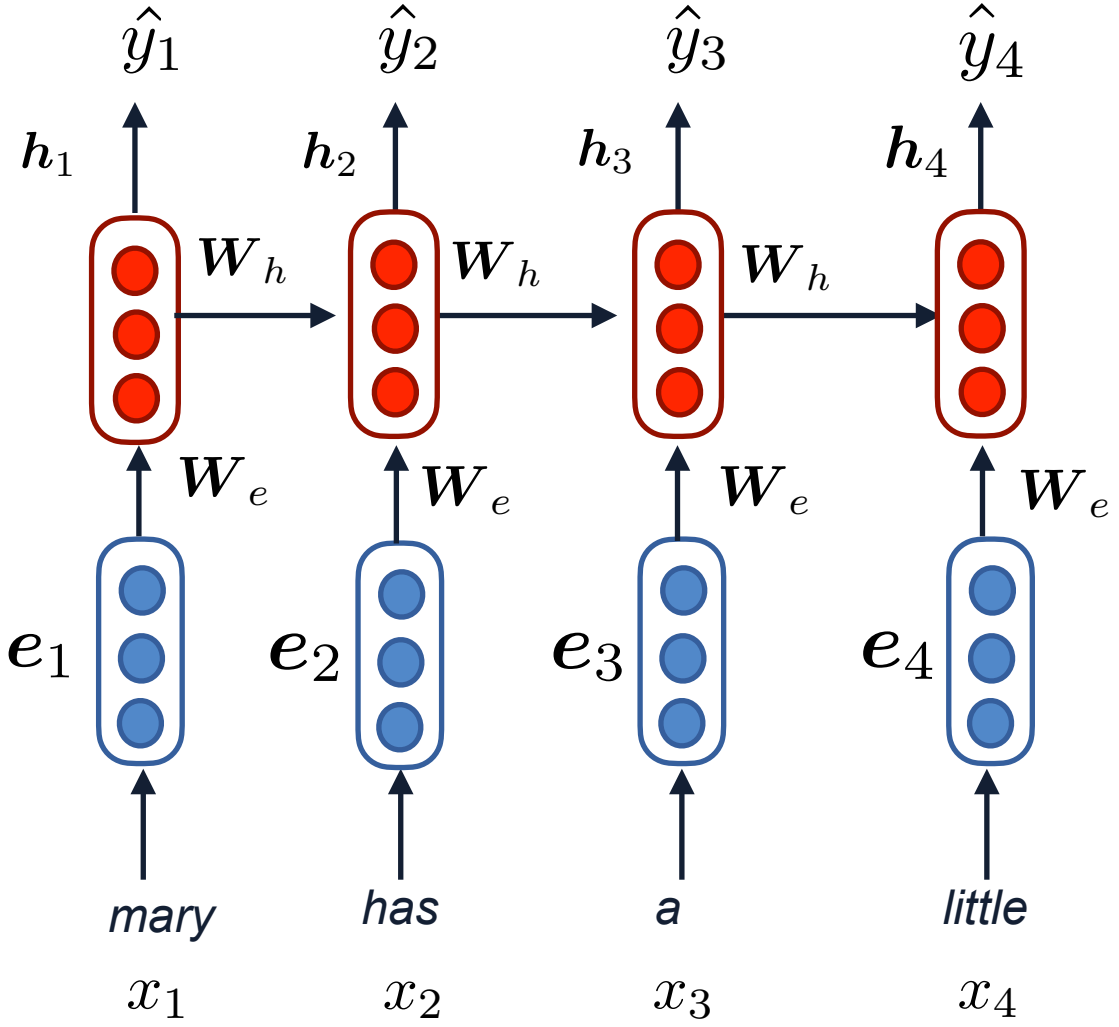
- Core idea: apply the same weights W repeatedly

RNN advantage:

- *Can process any length input*
- *Model size doesn't increase for longer input*
- *Symmetry because the same weight applied on every timestep*

RNN disadvantage:

- *Recurrent computation is slow*
- *Long distance problem*



Training an RNN language model

- Get a big corpus of text
- Feed into RNN-LM, compute output distribution yhat for every step t

- The loss function at step t is the cross entropy between the predicted probability distribution $\hat{y}_t$, and the true next word y_t or x_{t+1} :

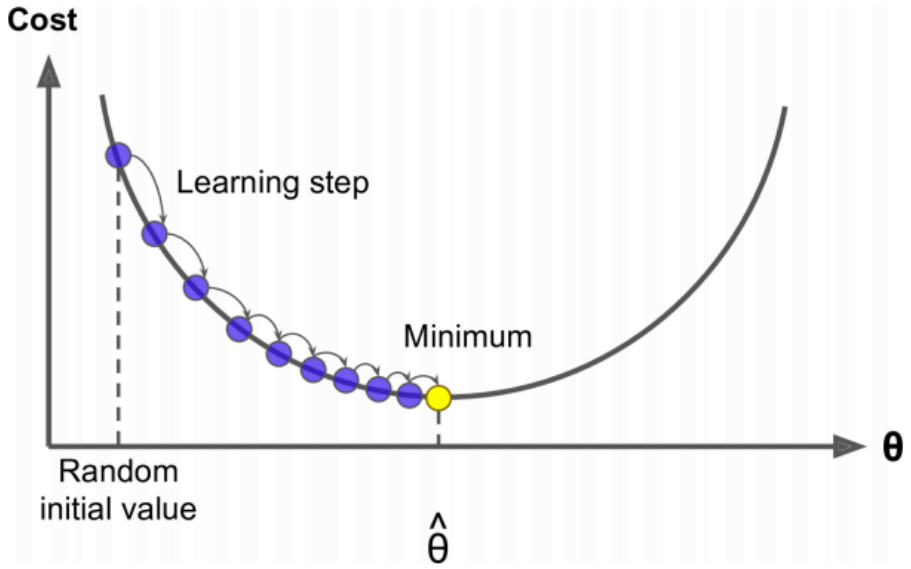
$$J_t(\theta) = CE\left(y_t, \hat{y}_t\right) = -\sum_{w \in V} y_{t,w} \log \hat{y}_{t,w} = -\log \hat{y}_{t,x_{t+1}}$$

- Minimize the average of the loss function over each word t:

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

Gradient descent

- We have a cost function $J(\theta)$ we want to minimize
- There are a list of optimization algorithm (for both convex and non-convex objectives), the most popular one is gradient descent (first-order optimization)
- At each iteration, GD calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat



Gradient descent

- We have a cost function $J(\theta)$ we want to minimize
- There are a list of optimization algorithm (for both convex and non-convex objectives), the most popular one is gradient descent (first-order optimization)
- At each iteration, GD calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

Training an RNN language model

- However, computing loss and gradients across entire corpus (i.e., gradient descent) is too expensive

- Stochastic gradient descent allows us to compute gradients of a batch of sentences and update weights
  - Batch size matters: smaller batch size takes less memory, larger batch size are memory consuming but converges faster

Generating text with an RNN language model

- You can train an RNN-LM on any kind of text, then generate in that style

- RNN trained on the first 4 Harry Porter books:

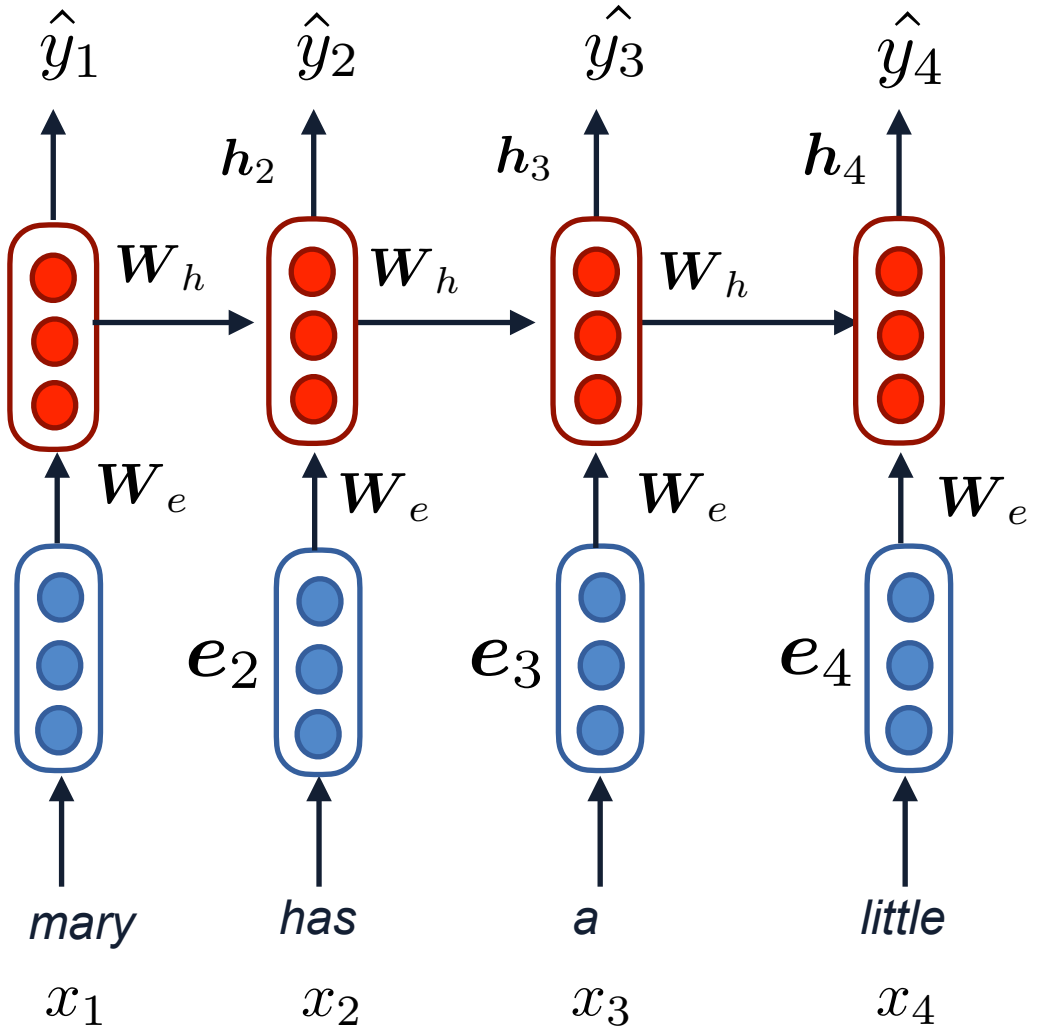
> “Sorry,” Harry shouted, panicking
> — “I’ll leave those brooms in London, are they?”
> “No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob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

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

RNN used in supervised learning

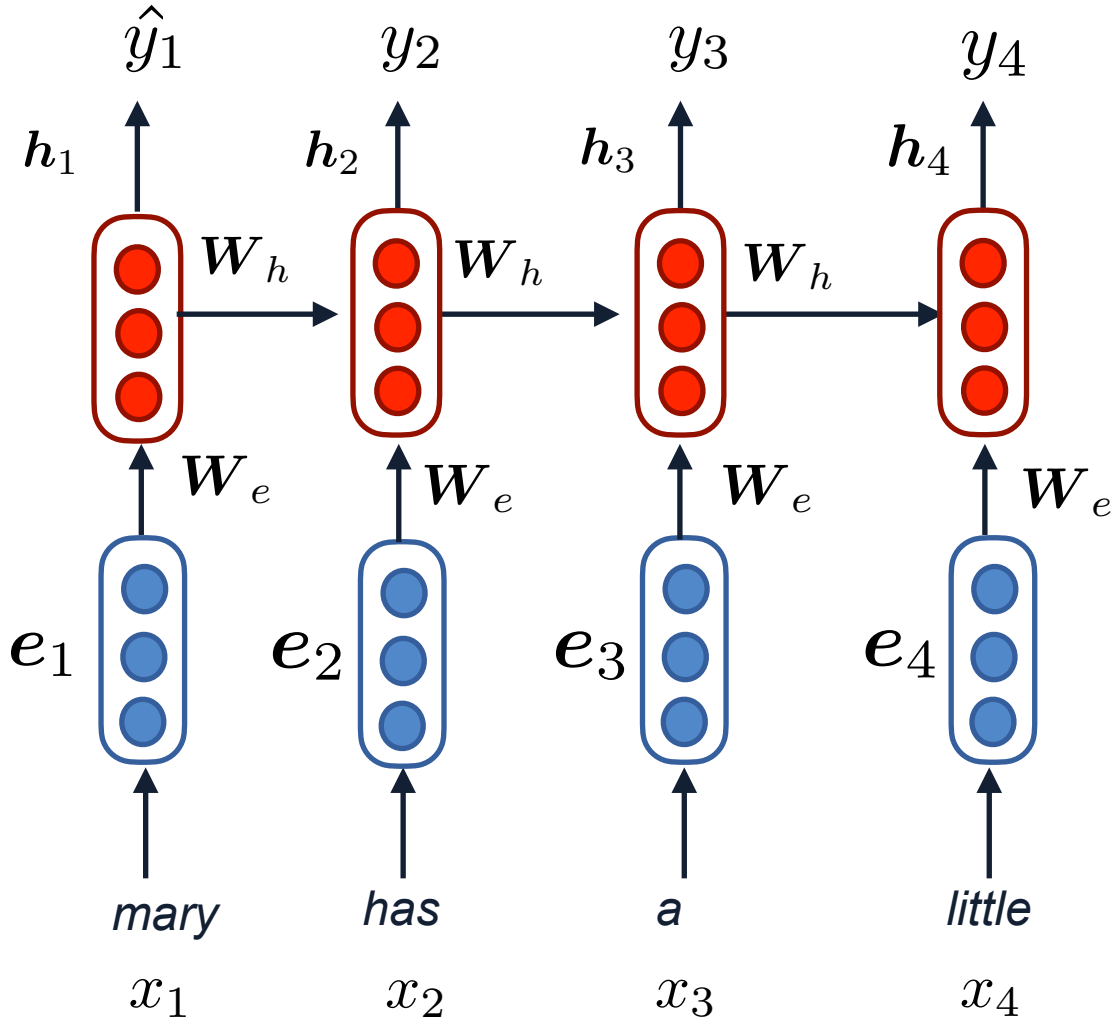
- Word tagging (use each y_t)
  - PoS tagging
  - NER tagging

- Sentence classification (use the last y_t or the average)
  - sentiment classification
  - text categorization



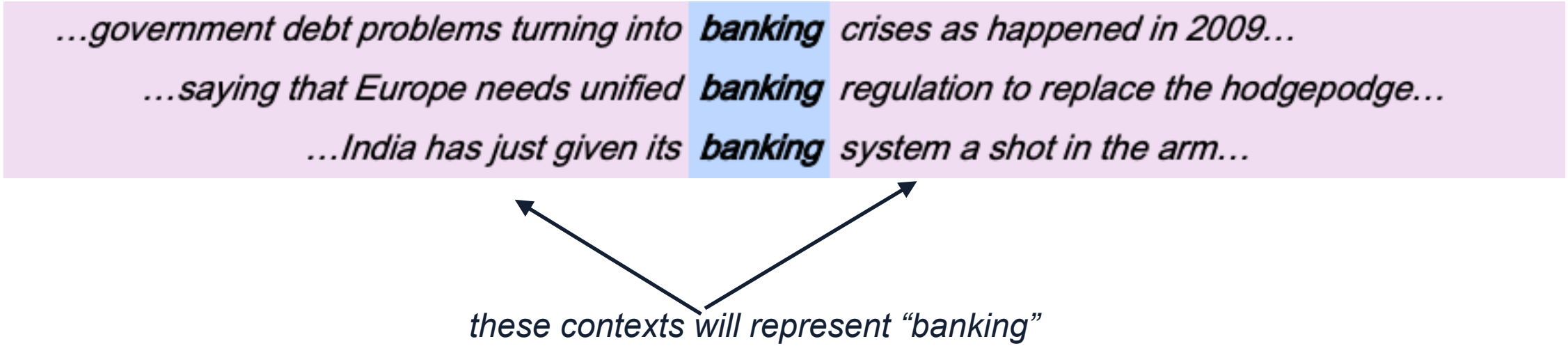
Word embedding

- In RNN, we randomly initialize the weight, use the back propagation to update this weight
- The random initialization may not give us a good starting point
- We can choose a better starting point with embedding vectors pertained in a large corpus to help the initialization [Mikolov et al. 13]



What makes a good word representation?

- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
  - "You shall know a word by the company it keeps" (J. R. Firth 1957)



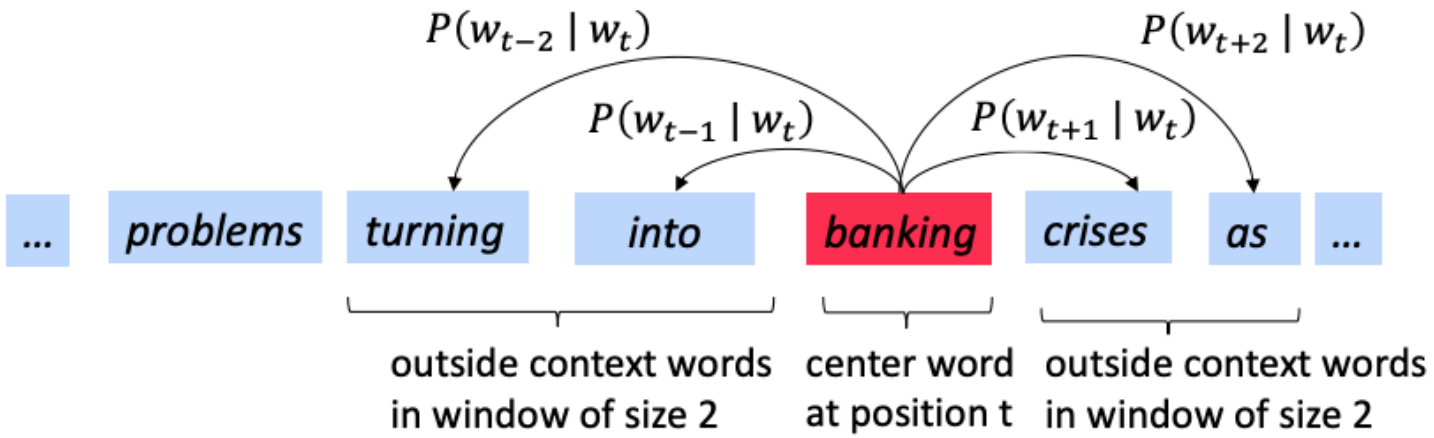
Word2vec Overview [Mikolov et al. 13]

- Maximizing the probability of the context words given w_t (Skipgram)

$$L(\theta) = \prod_{t=1}^{T} \prod_{m \le j \le m} P\left(w_{t+j} \mid w_t; \theta\right)$$

$$P(w_{t+j} \mid w_t) = \frac{\exp\left(u_{w_{t+j}}^T v_{w_t}\right)}{\sum_{w \in V} \exp\left(u_w^T v_{w_t}\right)}$$

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{m \le j \le m} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

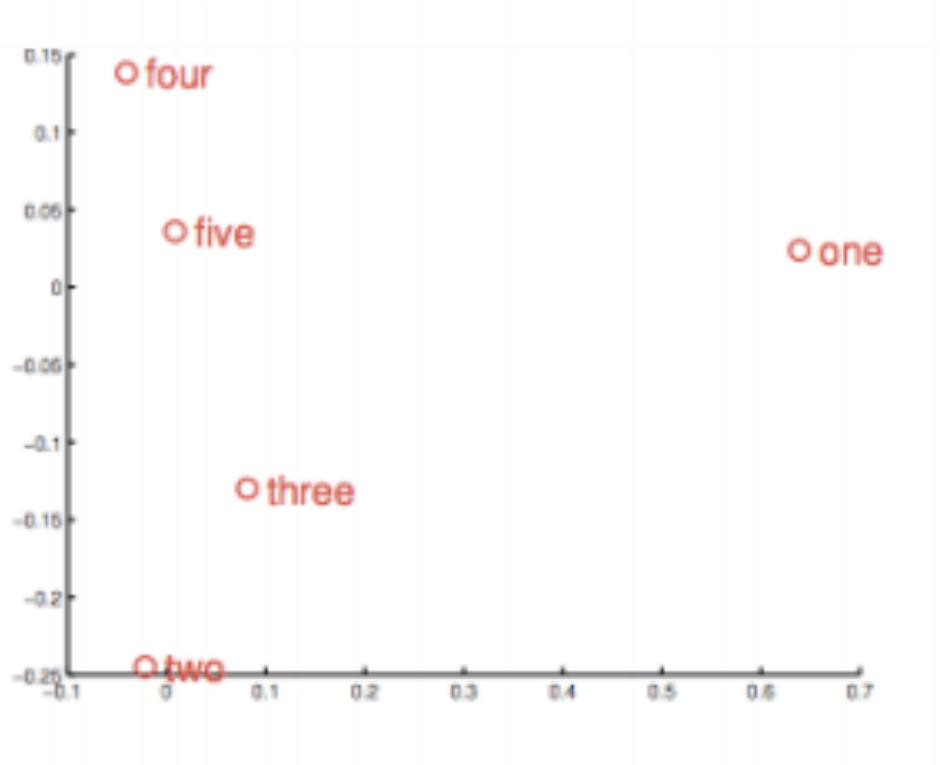


Word embedding results

- Maximizing the probability of the context words given w_t (Skipgram)

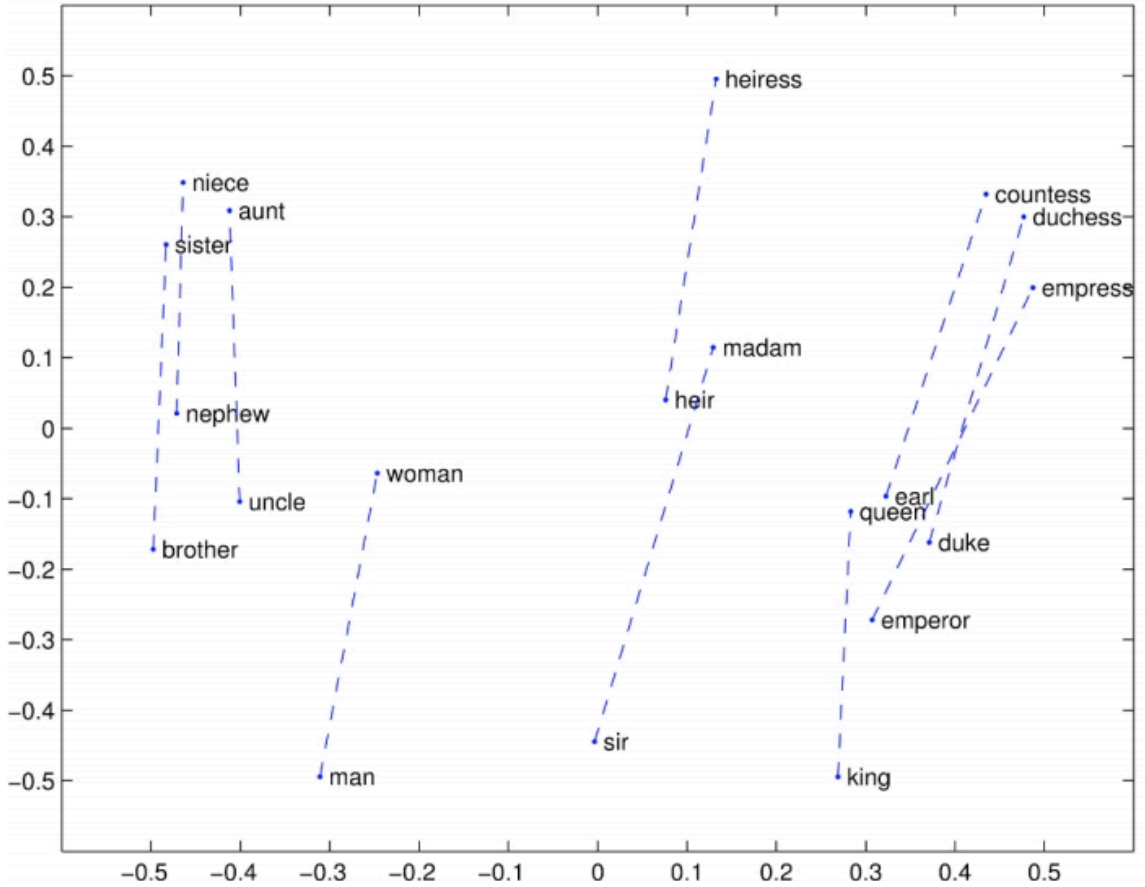
words closest to “sweden”

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408



Word embedding results

- Maximizing the probability of the context words given w_t (Skipgram)



Skipgram with negative sampling

- It is computationally expensive to use all words for normalization

$$P(w_{t+j} \mid w_t) = \frac{\exp\left(u_{w_{t+j}}^T v_{w_t}\right)}{\sum_{w \in V} \exp\left(u_w^T v_{w_t}\right)}$$

- In standard word2vec, implement negative sampling to improve the performance
- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)

$$J_t(\theta) = \log \sigma\left(u_{w_{j+t}}^T v_{w_t}\right) + \sum_{j=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_{w_t}\right)\right]$$

maximize prob that context word appears

minimize prob that random words appear

Negative sampling

- How to select negative samples?

- The probability of selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples

- Instead of using the raw frequency, in the original word2vec paper, each word is given a weight that's equal to it's frequency (word count) raised to the 3/4 power

$$P\left(w_i\right) = \frac{\text{freq}\left(w_i\right)^{3/4}}{\sum_{j=0}^{n}\left(\text{freq}\left(w_j\right)^{3/4}\right)}$$

Sub sampling frequent words

- Two problems with frequent words such as “the”:
  - When looking at word pairs that includes "the", e.g. ("fox", "the"), "the" doesn't tell us much about the meaning of "fox", since it appears in the context of pretty much every word.
  - We will have more than enough samples of ("the", "the other word for the word pair") than we need to learn a good vector for “the"
  - For every word in the training set, there is a probability we discard it, keep words with probability:

probability of keeping $w_i = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot 0.001 z(w_i)$

Detecting phrases

- Phrases such as “san francisco” are much more meaningful than individual words

- The rule used for detecting phrase is:

$$\frac{count(AB) - count_{min}}{count(A) \cdot count(B)} \cdot N > threshold$$

Sub sampling frequent words

- Two problems with frequent words such as “the”:
  - When looking at word pairs that includes "the", e.g. ("fox", "the"), "the" doesn't tell us much about the meaning of "fox", since it appears in the context of pretty much every word.
  - We will have more than enough samples of ("the", "the other word for the word pair") than we need to learn a good vector for “the"
  - For every word in the training set, there is a probability we discard it, keep words with probability:

probability of keeping $w_i = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot 0.001 z(w_i)$

Count vs. prediction for word embedding

- Count-based methods:
  - LSI
  - **Fast training**
  - **Making efficient usage of statistics**
  - **Capture mostly word similarity but not other patterns**
  - **Disproportion of word importance**
- Prediction-based methods:
  - Skipgram/CBOW
  - neural network probability language model
  - **Scale with corpus size**
  - **Inefficient usage of statistics**
  - **Improved performance in downstream tasks**
  - **Can capture pattern beyond word similarity**

Encoding meaning in vector differences

- Ratio of co-occurrence probability can encode meaning components

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

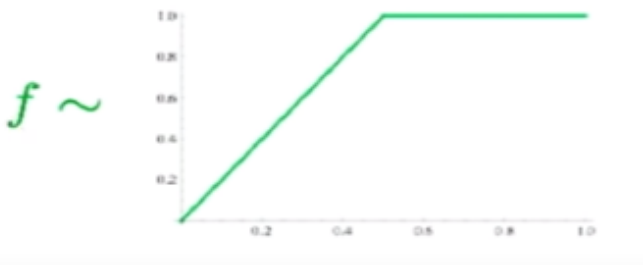
- Capturing the ratio using a bilinear model:

$$w_i \cdot w_j = \log P(i|j) \qquad w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

Encoding meaning in vector differences

- Ratio of co-occurrence probability can encode meaning components

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



reweighs the co-occurrence

make the inner product similar to co-occurrence

- Capturing the ratio using a bilinear model:

$$w_i \cdot w_j = \log P(i|j)$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

Encoding meaning in vector differences

- Ratio of co-occurrence probability can encode meaning components

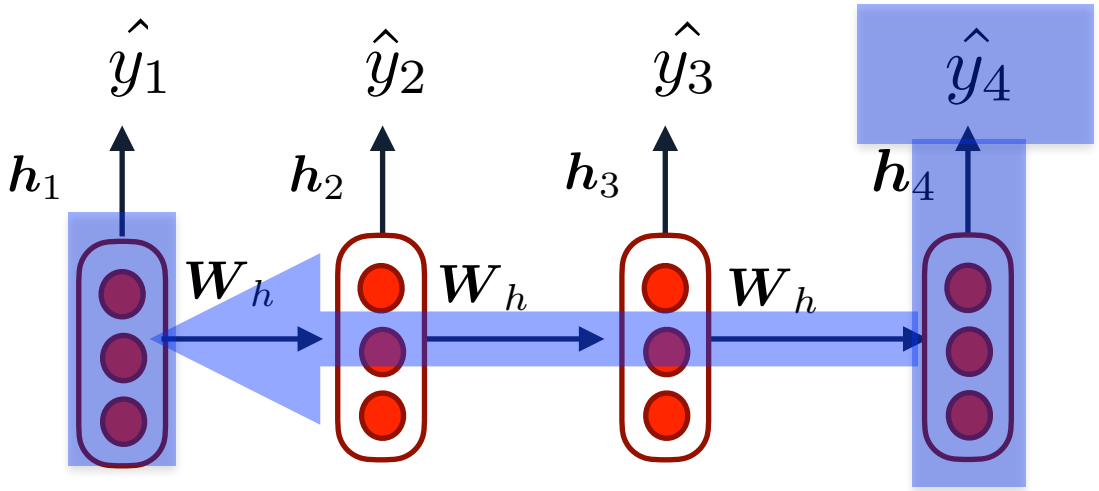
	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

-

Summary of word embedding

- Word embedding is designed for initializing the embedding layer of a neural network
- Skipgram and CBOW train word embedding vectors by predicting the probability of word generation from context words
- Training of word embedding vectors can be improved using negative sampling, subsampling of frequent words, and detecting phrases
- GloVe improves word embedding by capturing other patterns

Vanishing gradient problem in RNN



$$\frac{\partial J_4}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial J_4}{\partial h_2} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_4}{\partial h_3} \times \frac{\partial J_4}{h_4}$$

Vanishing gradient problem in RNN

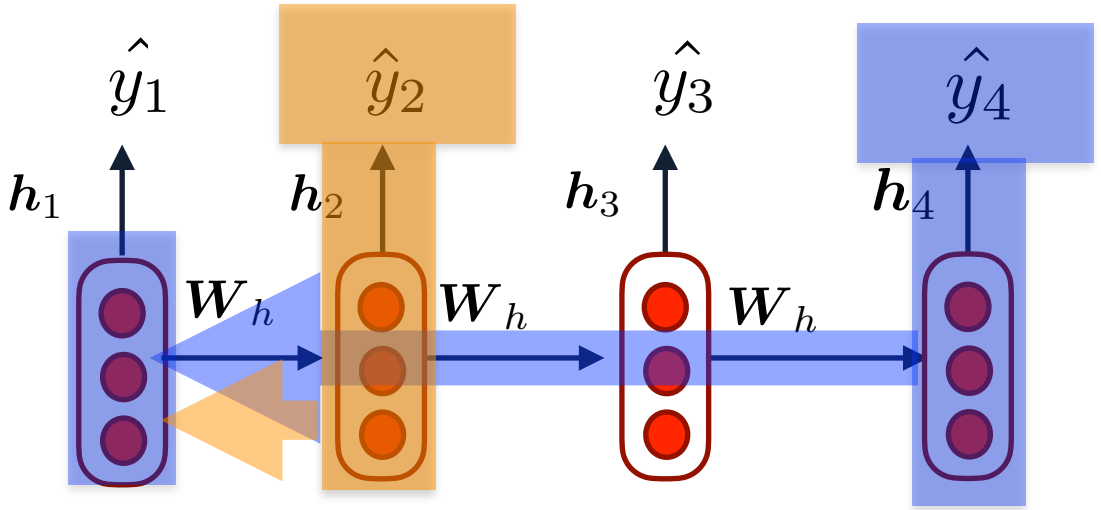
- Recall: $h_t = \sigma\left(W_h h_{t-1} + W_x x_t + b_1\right)$
- What if the sigma function is the identity function, i.e., $\sigma(x) = x$?

$$\begin{aligned}\frac{\partial h_t}{\partial h_{t-1}} &= \operatorname{diag}\left(\sigma'\left(W_h h_{t-1} + W_x x_t + b_1\right)\right) W_h \\ &= I W_h = W_h\end{aligned}$$

- Consider the gradient of loss in step i with respect to step j's hidden vector. If Wh is small (large), the gradient gets exponentially small (large), causing the vanishing (exploding) gradient problem:

$$\begin{aligned}\frac{\partial J^{(i)}(\theta)}{\partial h_j} &= \frac{\partial J^{(i)}(\theta)}{\partial h_i} \prod_{j<t\leq i} \frac{\partial h_t}{\partial h_{t-1}} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial h_i} \prod_{j<t\leq i} W_h = \frac{\partial J^{(i)}(\theta)}{\partial h_i} W_h^{\ell}\end{aligned}$$

Vanishing gradient problem in RNN



$$\frac{\partial J_4}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial J_4}{\partial h_2} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_4}{\partial h_3} \times \frac{\partial J_4}{h_4}$$

$$\frac{\partial J_2}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial J_2}{h_2}$$

gradient gets dominated by larger item

Example of vanishing gradient

- When she tried to print her **tickets**, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____
- RNN needs to model the dependency between the first “ticket” and the word to fill in the blank
- If the gradient is too small, RNN cannot learn this dependency, makes it difficult to tell between
  - There’s no dependency between step t and t+n
  - We have the wrong parameter to capture the dependency

Problem caused by exploding gradient

- Exploding gradient: the similar problem as the vanishing gradient problem

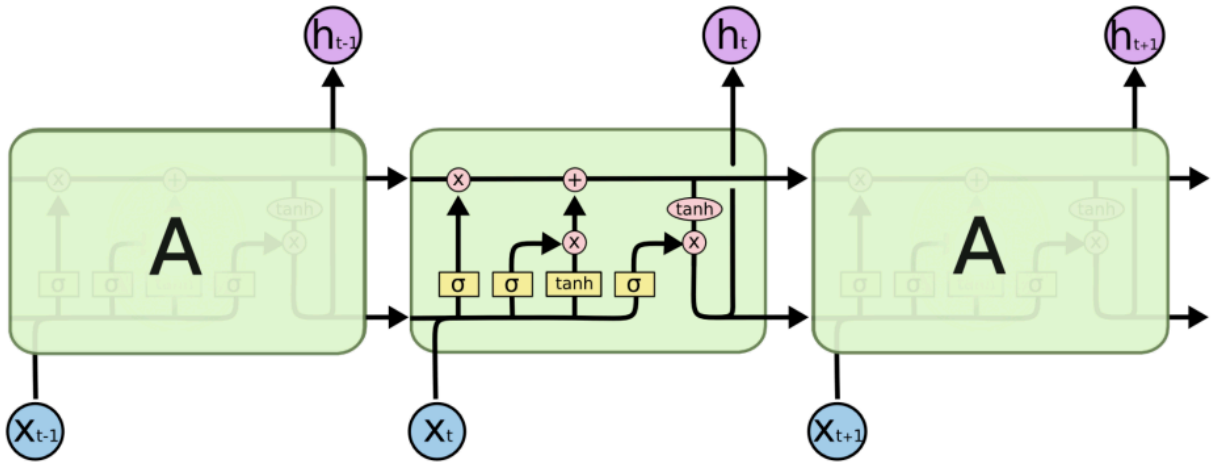
$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

- Problem 1: we take a large step and reach a bad parameter position
- Problem 2: resulting in Inf or NaN in your network (**anyone has experienced this problem?**)
- Solution 1: gradient clipping. Scale down the gradient if larger than a threshold

Long short term memory [Hochreiter & Schmidhuber 97]

- Solution to the vanishing gradient problem

- At time t, there is a hidden state and a cell state
  - Both vectors are of the same length
  - The cell state stores long-term information
  - LSTM can erase, read and write information from the cell



Long short term memory [Hochreiter & Schmidhuber 97]

$$\tilde{c}_t = \tanh\left(\boldsymbol{W}_c h_{t-1} + \boldsymbol{U}_c x_t + b_c\right)$$
$$c_t = \boldsymbol{f}_t \circ c_{t-1} + \boldsymbol{i}_t \circ \tilde{c}_t$$
$$h_t = \boldsymbol{o}_t \circ \tanh c_t$$

If i is 0 and f is 1, the information is preserved indefinitely

Controls what's forgotten from previous cell

$$\boldsymbol{f}_t = \sqrt{\sigma}\left(\boldsymbol{W}_f h_{t-1} + \boldsymbol{U}_f x_t + b_f\right)$$

Controls what part of new content is written to cell

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i h_{t-1} + \boldsymbol{U}_i x_t + b_i\right)$$

Controls what part of new cell is output to hidden state

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o h_{t-1} + \boldsymbol{U}_o x_t + b_o\right)$$

Gated recurrent unit (GRU)

- A simplification of LSTM proposed by Cho et al. 2014

Controls what part of the hidden state is updated/ preserved

Controls what part of the previous state is used to compute the new content

$$u_t = \sigma\left(\boldsymbol{W}_u h_{t-1} + u_u x_t + b_u\right)$$
$$r_t = \sigma\left(\boldsymbol{W}_r h_{t-1} + u_r x_t + b_r\right)$$
$$\tilde{h}_t = \tanh\left(\boldsymbol{W}_h \left(r_t \circ h_{t-1}\right) + u_h x_t + b_h\right)$$
$$h_t = \left(1 - u_t\right) \circ h_{t-1} + u_t \circ \tilde{h}_t$$

Discussions

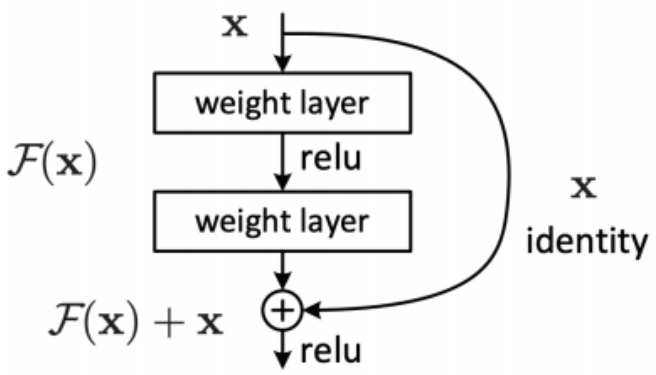
- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used

- With LSTM/GRU, gradient vanishing/explosion problem is alleviated but still exists

- Vanishing/exploding gradient is not just an RNN problem
  - Due to chain rule, gradient can be vanishingly small as it back propagate

Residual neural networks [He et al. 2016]

- Adding direct connections:



- Makes deep network easier to train

Summary

- Neural language models can help with the sparsity problem of n-gram language model
- RNN is better than NNLM because it can capture sequence of any lengths
- Word embedding can be used for improving the initialization of RNN
- RNN suffer from the vanishing gradient problem, the problem is alleviated using LSTM/GRU