CS 589 Fall 2021 Lecture 7

word embedding

Monday 6:30-9:00
Babbio 122

photo: https://www.scubedstudios.com/information-retrieval/

All zoom links in Canvas
Most slides adapted from Stanford CS224 (Instructor: Abigail See)

Review of Lecture 6

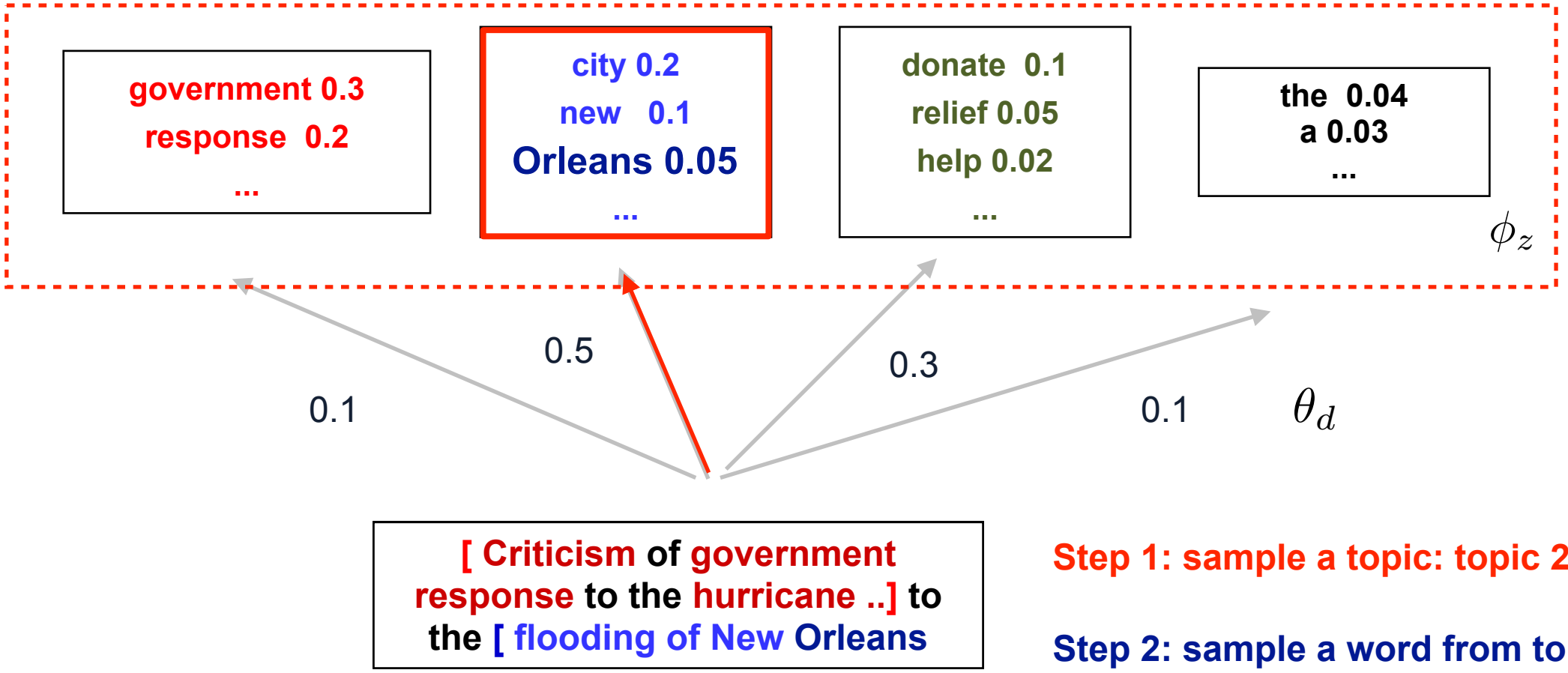
- Expectation maximization
  - Maximizing the likelihood estimation under the latent space assumption
  - Complete likelihood vs. incomplete likelihood
  - E step vs M step
  - Theorem: The incomplete log likelihood p(x|theta) is guaranteed to not decrease during EM iterations

- The coin topic problem
  - Derive the EM algorithm when p(w|T) and p(w|H) are known, lambda is unknown, etc.

Review of PLSA

Phi T x V, V: vocabulary size ~50,000, T: #topics, T=20

theta D x T, D: #documents: ~10,000, T: #topics, T=20



Step 1: sample a topic: topic 2

Step 2: sample a word from topic 2

Probabilistic Latent Semantic Analysis. Thomas Hoffman. 2001.

LSI vs PLSA vs LDA

- Indexing by latent semantic analysis. Deerwester et al. 1998
- Probabilistic latent semantic analysis. Thomas Hoffman. 2001
- Latent Dirichlet allocation. Blei et al. 2003

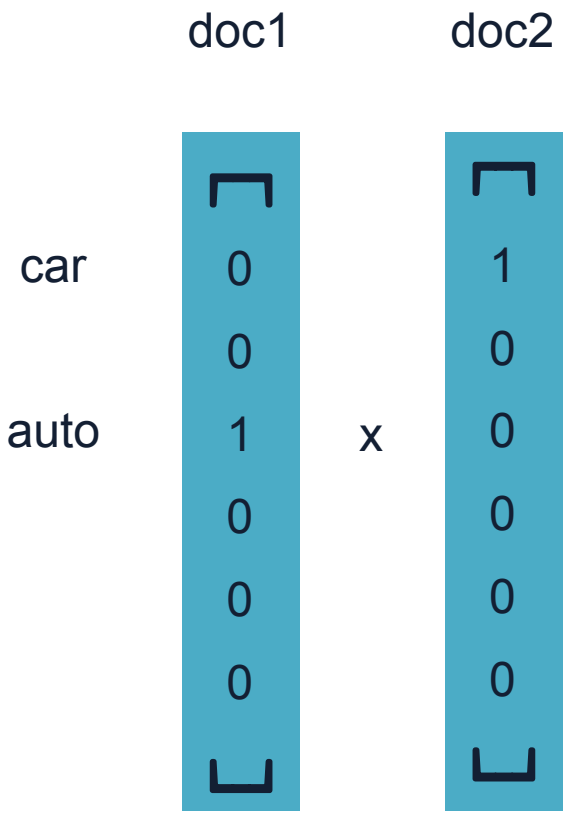
Vocabulary gap problem with the vector space model

- Vector space model:

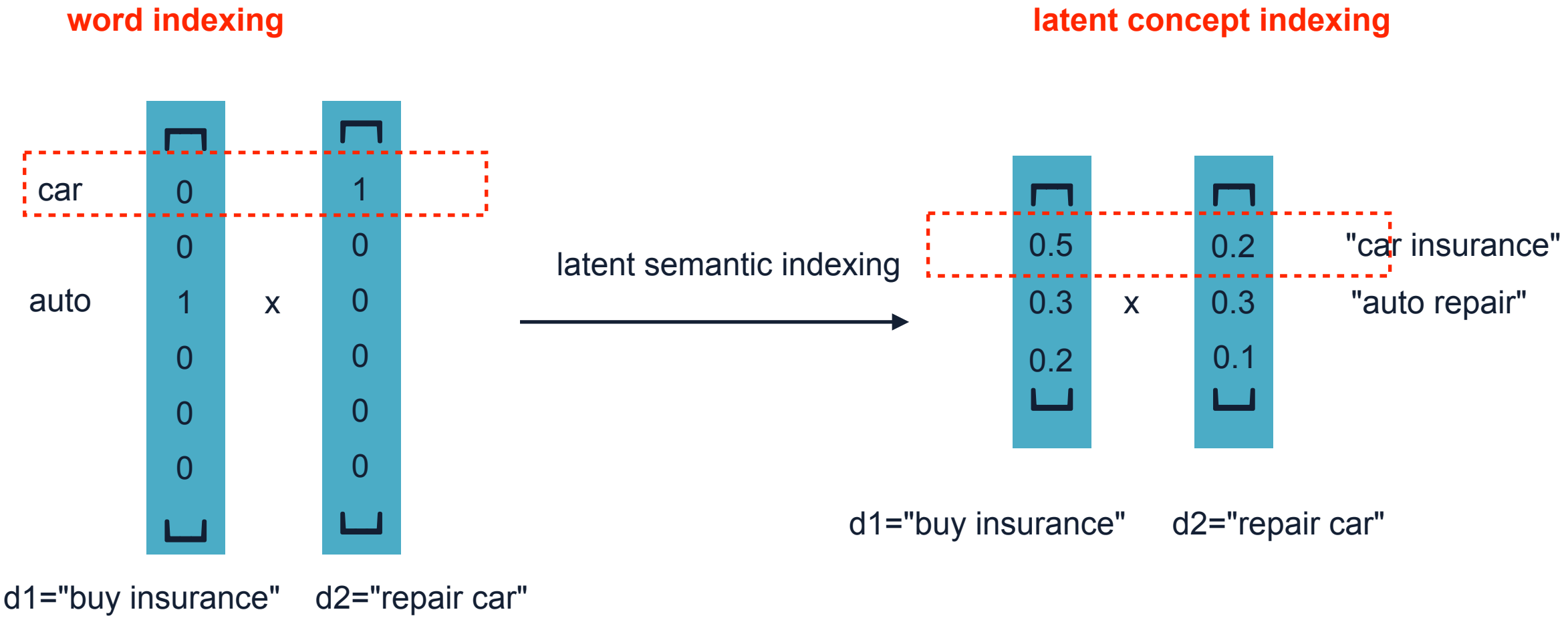
$$score(q, d) = \frac{q \cdot d}{\|q\| \cdot \|d\|}$$

- Challenges matching synonyms
  - e.g., auto vs. car

- Challenges matching polysemy
  - e.g., apple (fruit vs. company)



Latent semantic indexing



Indexing by latent semantic analysis. Deerwester et al. 1998

Singular value decomposition (matrix factorization)

- For a matrix A $\in \mathbb{R}^{m \times n}$, approximate A with a singular value decomposition (SVD):

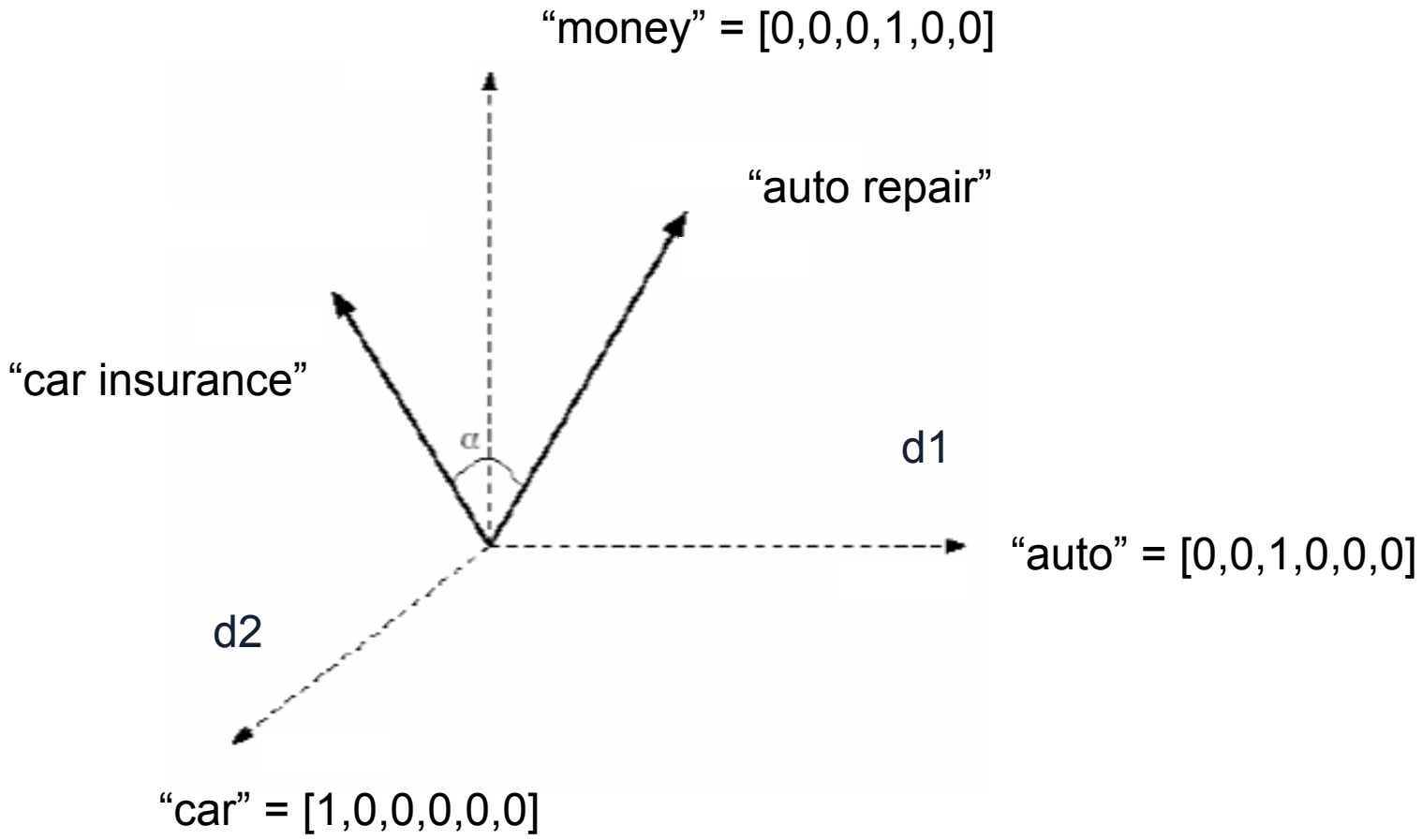
$$\min_{U,\Sigma,V} \|A - U\Sigma V^T\|^2$$

- U: V x r
- V: r x D
- Sigma: r x r

$$\Sigma = \mathrm{diag}\,(\sigma_1 \ldots \sigma_r)$$

Low dimensional vector representation of words

- Axes are concepts, also called principal components (PCA)



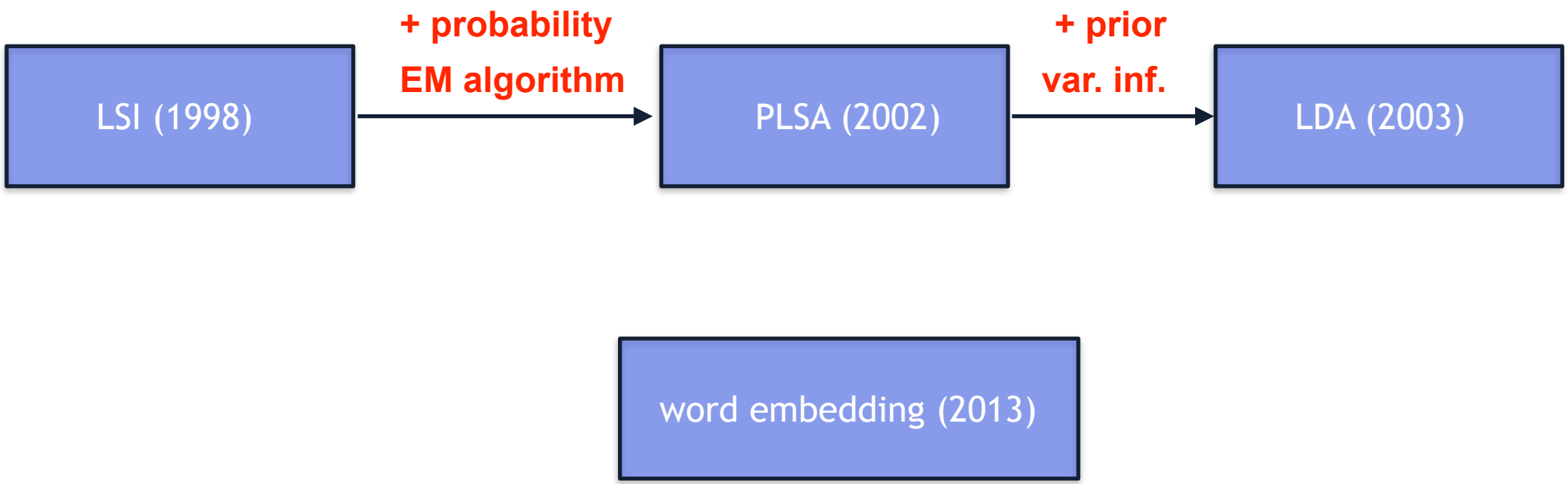
Latent semantic indexing

- Uses statistically derived **conceptual indices** instead of individual word indexing for retrieval
- Assumes that there is some underlying or latent structure in word usage that is obscured by variability in word choice
- Key idea: instead of representing documents and queries as vectors in a **low dimensional space** of terms

What LSI can do

- LSI effectively does
  - Dimensionality reduction
  - Noise reduction
  - Exploitation of redundant data (linearity)
  - Correlation analysis and query expansion (with related words)
- Some of the individual effects can be achieved with simpler techniques (e.g. thesaurus construction). LSI does them together
- LSI handles **synonymy** well, not so much **polysemy** (vs word embedding)
- Challenge: SVD is complex to compute (O(n3)) – Needs to be updated as new documents are found/updated

From LSI to word embedding



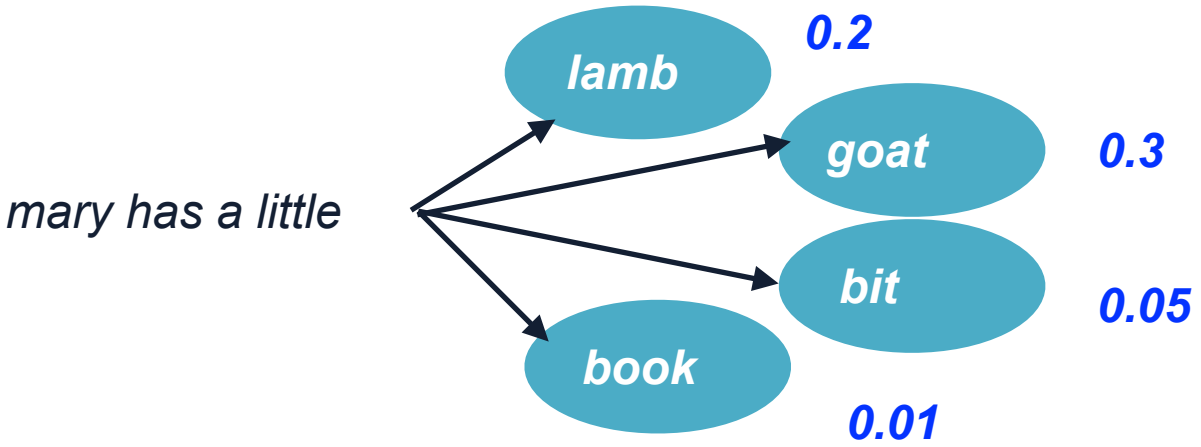
Today's lecture

- Language models
  - Word2vec [Mikolov et al. 2013]

- 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

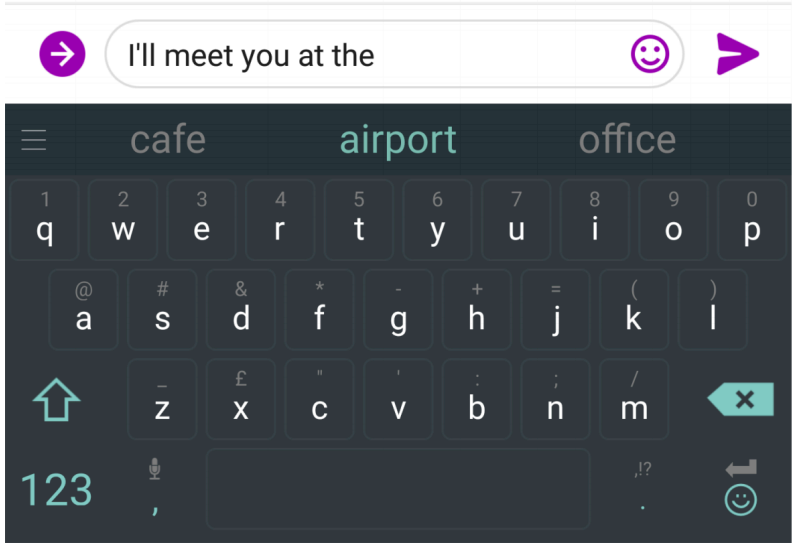
- Training (chain rule):

$$\max_{\theta} \; P(x_1, \ldots, x_T) = P(x_1) \times P(x_2 \mid x_1) \times \cdots \times P(x_T \mid x_{T-1}, \ldots, x_1)$$
$$= \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots, x_1)$$

- Prediction:

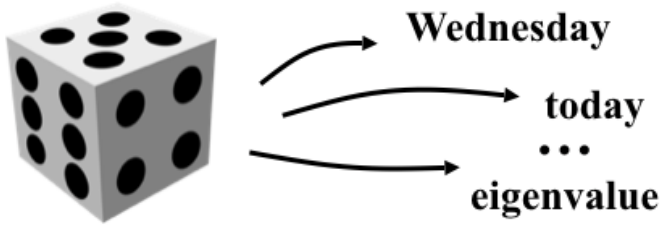
$$x_t = \max_{x'} p(x' | x_1, \cdots, x_{t-1})$$

How to model the probability in language model?



Statistical 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:

$$p(\text{" Mary little lamb"}) = p(\text{"Mary"}) \times p(\text{"little"} \mid \text{" Mary"}) \times p(\text{"lamb"} \mid \text{"little"}) = \frac{30}{160} \times \frac{5}{30} \times \frac{25}{100}$$

Language models

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

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

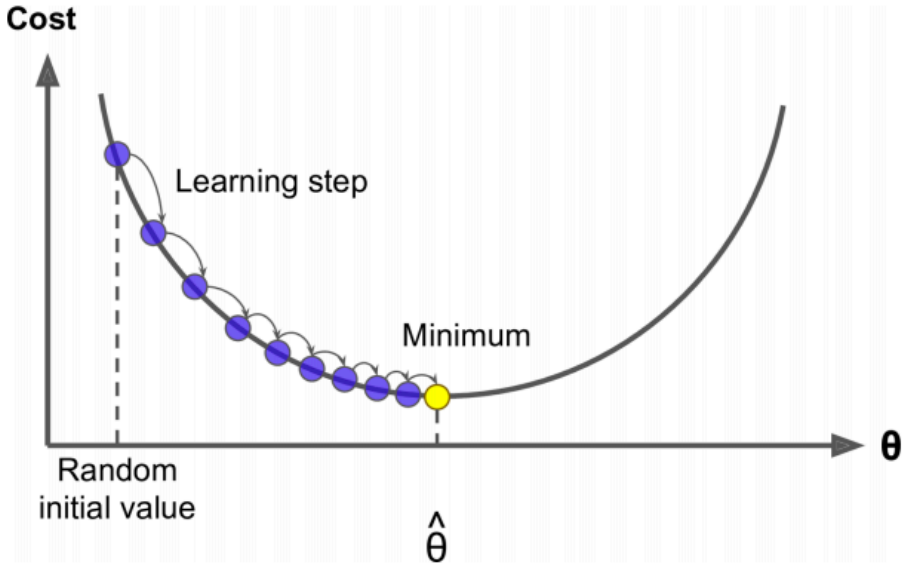
$$= \frac{P\left(x_{t+1}, x_t, \ldots, x_{t-n+2}\right)}{P\left(x_t, \ldots, x_{t-n+2}\right)}$$

- 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

Gradient descent (optimization method)

- 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



$$\min_{\theta} J(\theta)$$

Training loss: CE, margin loss

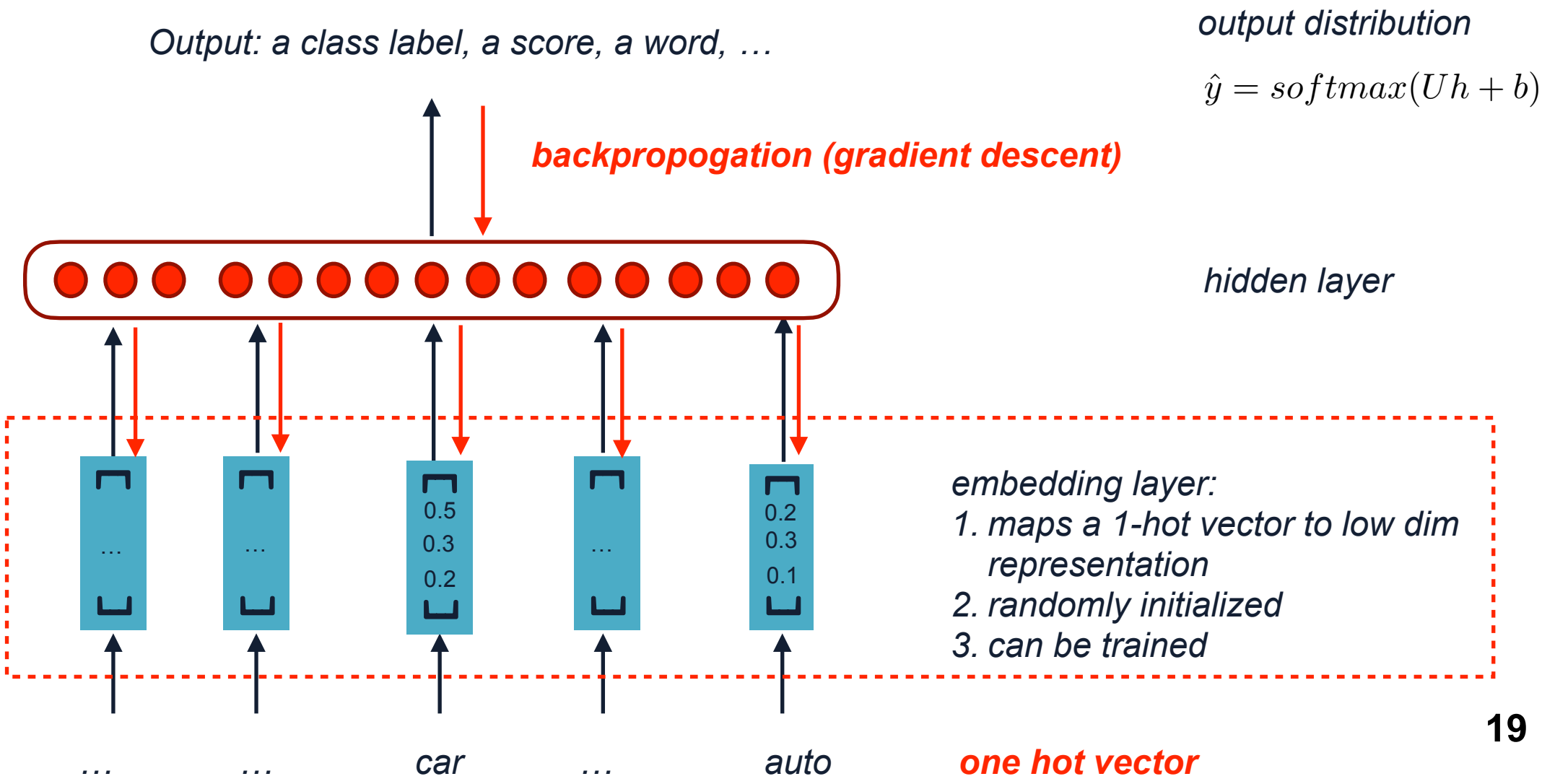
Gradient descent algorithm

- Repeat

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

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

Neural network for NLP



Neural language models

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

Improvements over n-gram LM:

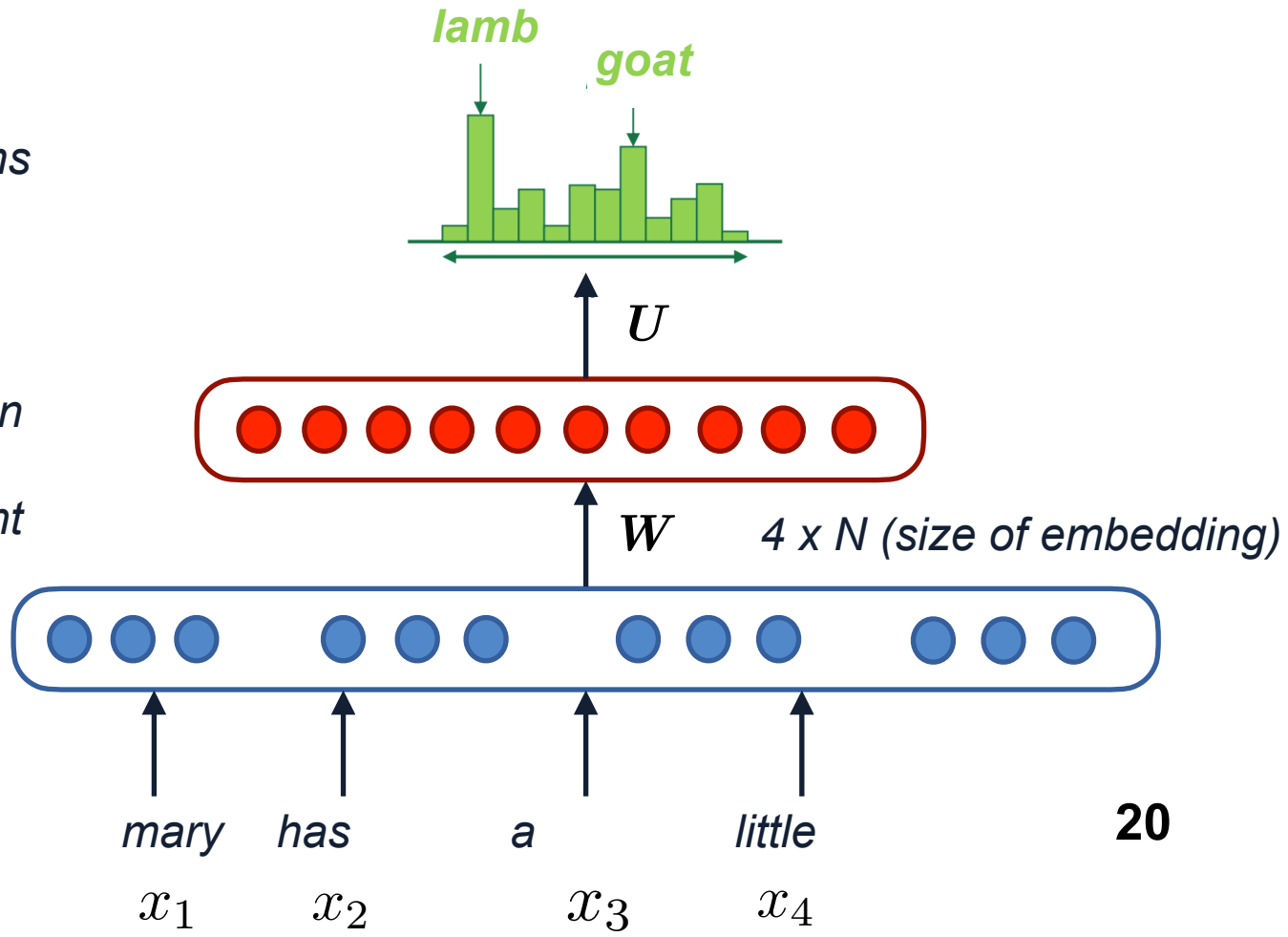
1. No sparsity problem

2. Don't need to store all observed n-grams

Remaining problems:

1. Fixed window size too small, enlarging window size enlarges W but window can never be large enough

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

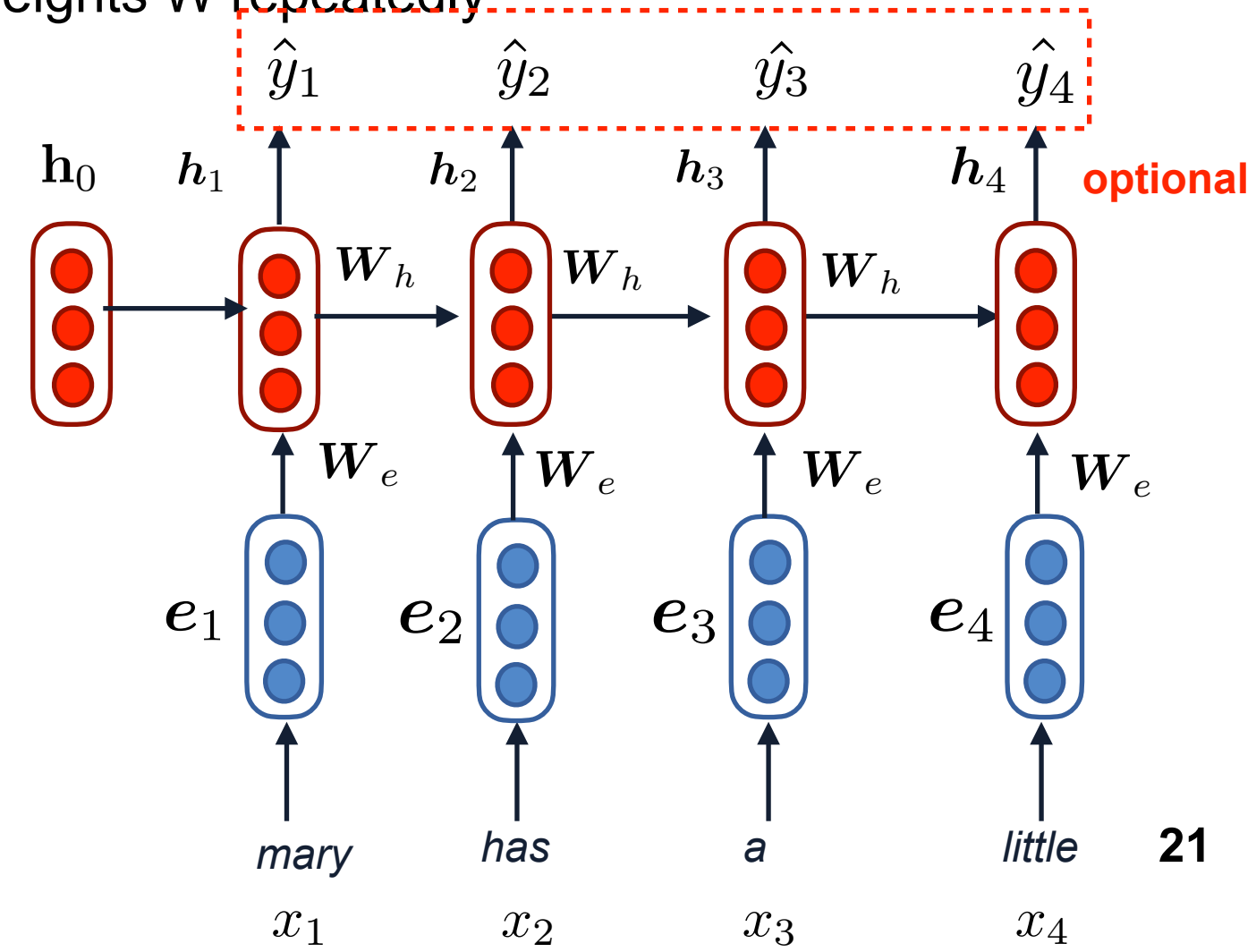
$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

word embeddings

$e_t = \boldsymbol{E}x_t$ N x V

word/one-hot vectors

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

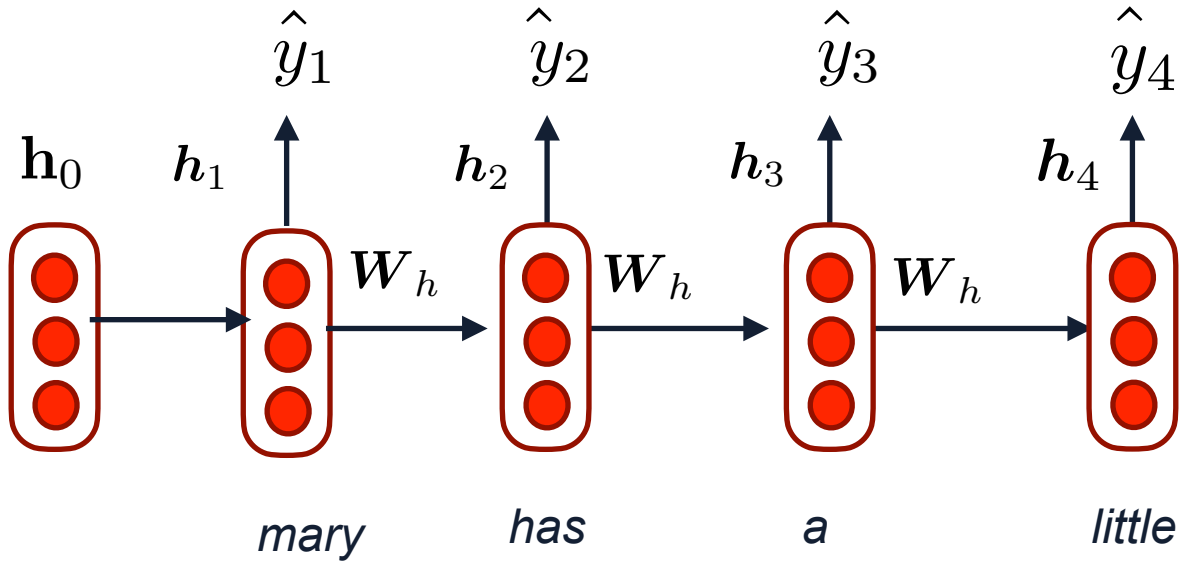


Training RNN method 1: Language modeling

- Minimizes the cross entropy loss:

$$J_t(\theta) = CE(y_t, \hat{y}_t)$$
$$= -\sum_{w \in V} y_{t,w} \log \hat{y}_{t,w}$$
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$
$$= \frac{1}{T}(-\log 0.85$$
$$-\log 0.62$$
$$-\log 0.79$$
$$-\log 0.82$$
$$+ \cdots)$$

a	[0.01,	[0.09,	**[0.79,**	[0.09,
...	...	**0.62,**	...	...
little	0.01,	...	0.01,	**0.82,**
...				
mary	**0.85]**	0.10]	0.10]	0.10]



Training RNN method 2: Text classification

output distribution

$$\hat{y}_t = \text{softmax}\left(Uh_t + b_2\right) \in \mathbb{R}^3$$

- Cross entropy loss (min):

$$J_s(\theta) = CE(y_s, \hat{y}_s)$$
$$= -\sum_{label \in \{pos, neutral, neg\}} \mathbb{1}[y_s == label] \log \hat{y}_{s,label}$$

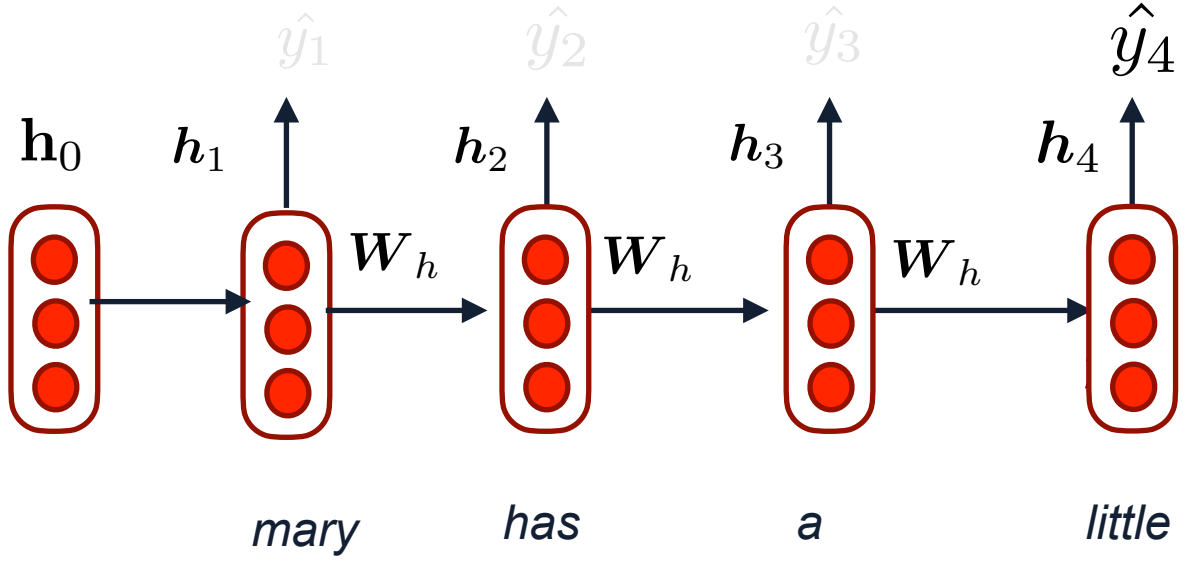
- Accuracy (max):

$$J_s(\theta) = Acc(y_s, \hat{y}_s)$$
$$= \sum_{label \in \{pos, neutral, neg\}} \mathbb{1}[y_s == label] \mathbb{1}[\hat{y}_{s,label} > 0.333]$$

pos [0.2,

neutral **0.5,**

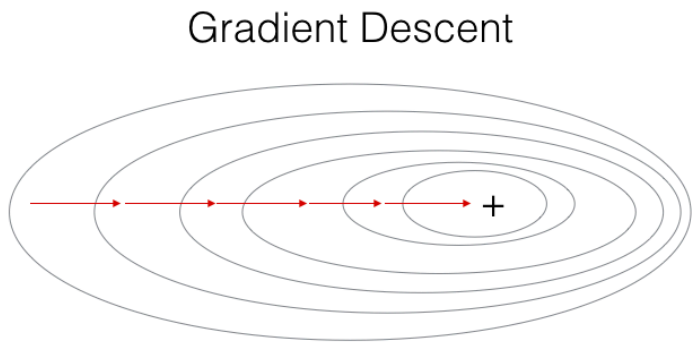
neg 0.3]



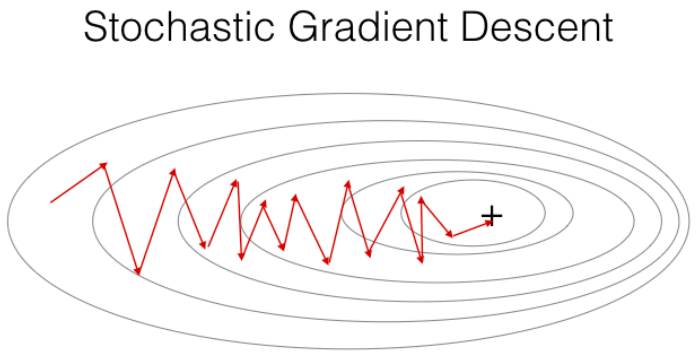
Training RNN with SGD

- Gradient descent (computing loss and gradients across entire corpus) is too expensive

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



- Improving the training efficiency using SGD:
  - Smaller batch size: takes less memory
  - Large batch size: converges faster
  - Typical batch size: 1, 2, 4, 8, 16, 32, ...



Advantages and disadvantages of RNN

- Advantages (over the neural probabilistic language model):
  - Can process input of any length
  - Theoretically, at step t, we can look back for a long distance
  - Model size does not increase for longer sentences
  - Words at different positions share the same weight, so more efficient at training these weights
- Disadvantages:
  - Backprop is slow if the sequence is long
  - In practice, it is still difficult to access information many steps back

Chain rule

- Chain rule of partial derivative:

$$h(x) = f(g(x))$$

$$h'(x) = f'(g(x))g'(x)$$

- Multivariate chain rule:

$$\frac{\partial f(x(t), y(t))}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$

Back propagation through time (BPTT)

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

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

depends on Wh, needs multivariate chain rule

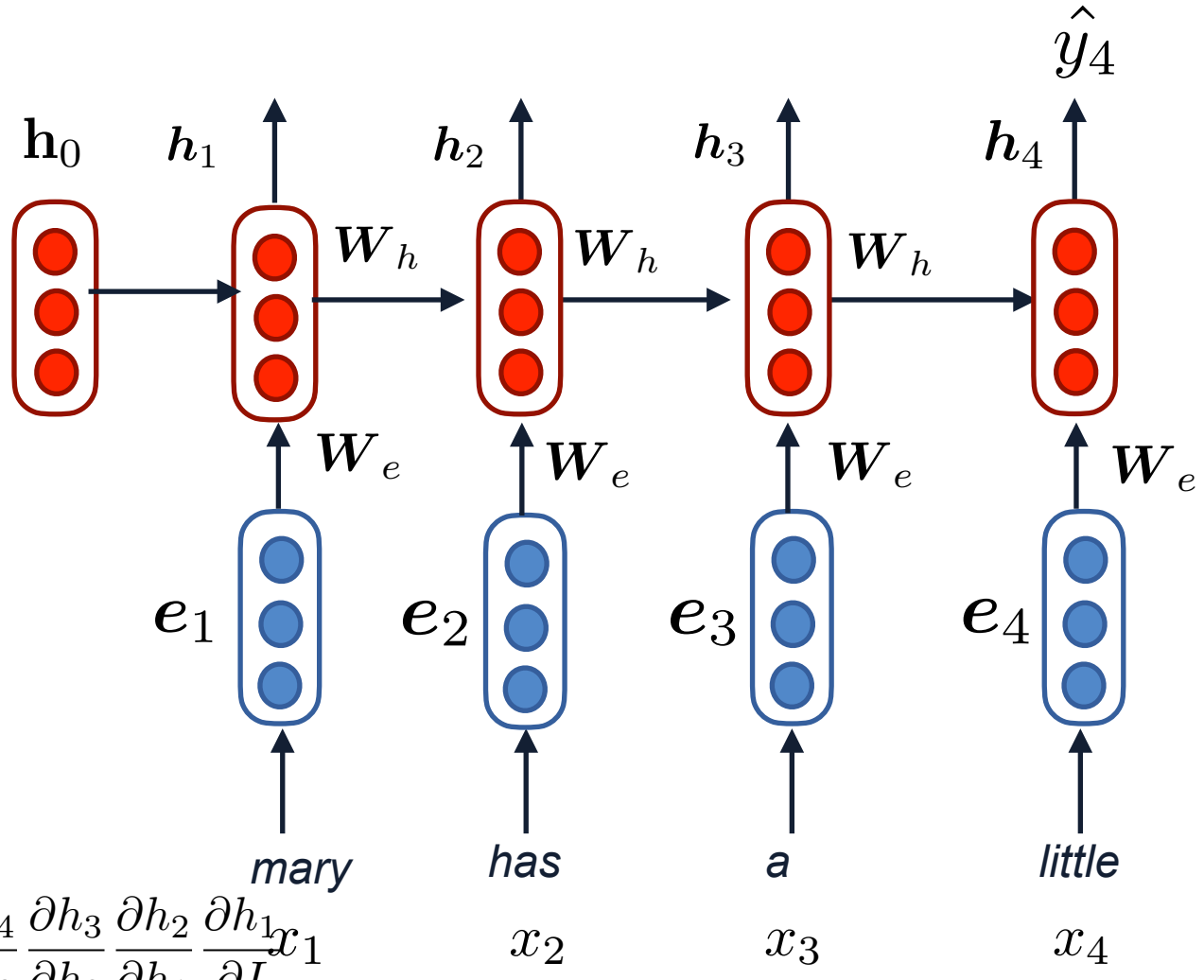
$$\frac{\partial J_4(\theta)}{W_h} = \frac{\partial J}{\partial h_4} \times \frac{\partial h_4}{\partial W_h}$$

$$\frac{\partial h_4}{\partial W_h} = \sigma(I(W_h)h_3(W_h) + W_e e_4 + b_1)$$

$$= \frac{\partial h_4}{\partial I(W_h)} + \frac{\partial h_4}{\partial h_3(W_h)} \frac{\partial h_3(W_h)}{\partial W_h}$$

$$= \frac{\partial h_4}{\partial I(W_h)} + \frac{\partial h_4}{\partial h_3}\left(\frac{\partial h_3}{\partial I} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_h}\right)$$

$$= \frac{\partial h_4}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial I}$$



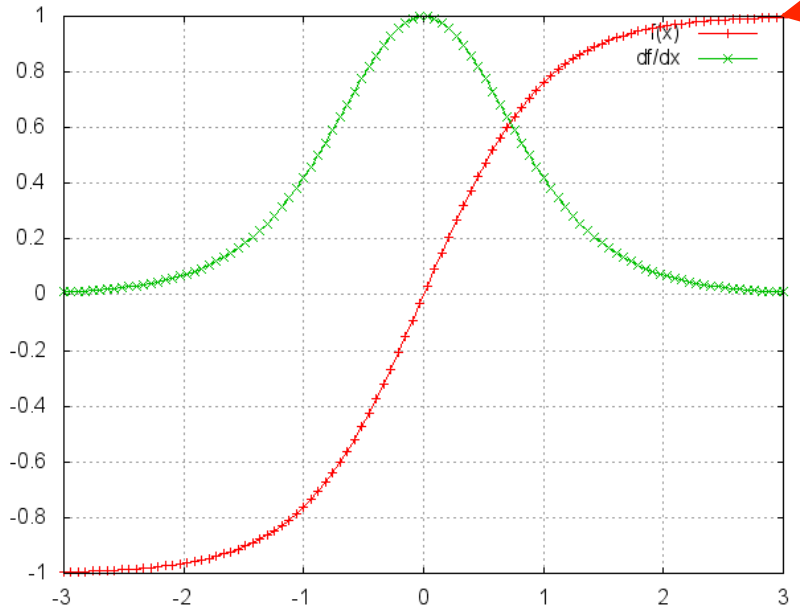
Vanishing gradient problem in RNN

$$\frac{\partial h_4}{\partial W_h} = \frac{\partial h_4}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial I} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial I}$$

h1's effect does not matter

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

partial derivative ~ zero at both ends



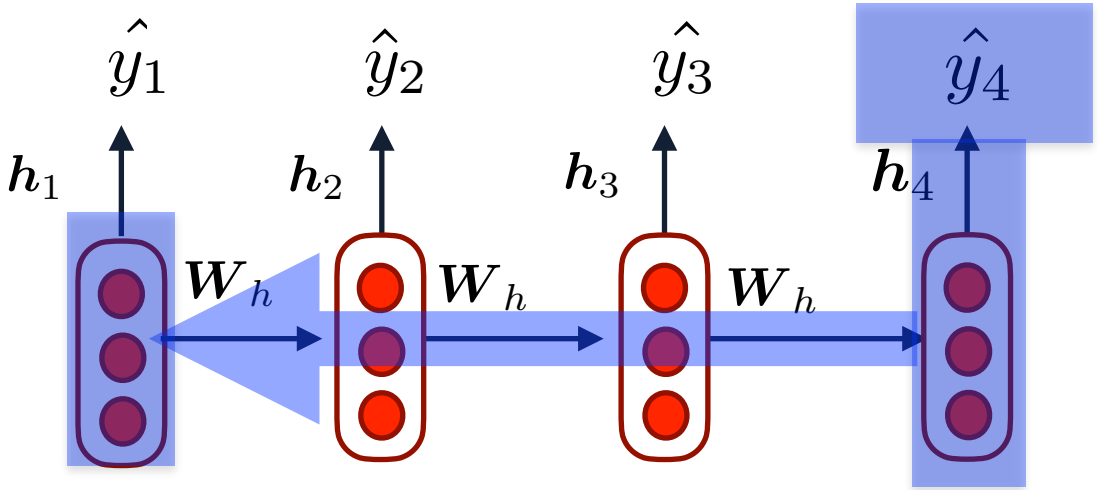


Image source: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

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

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

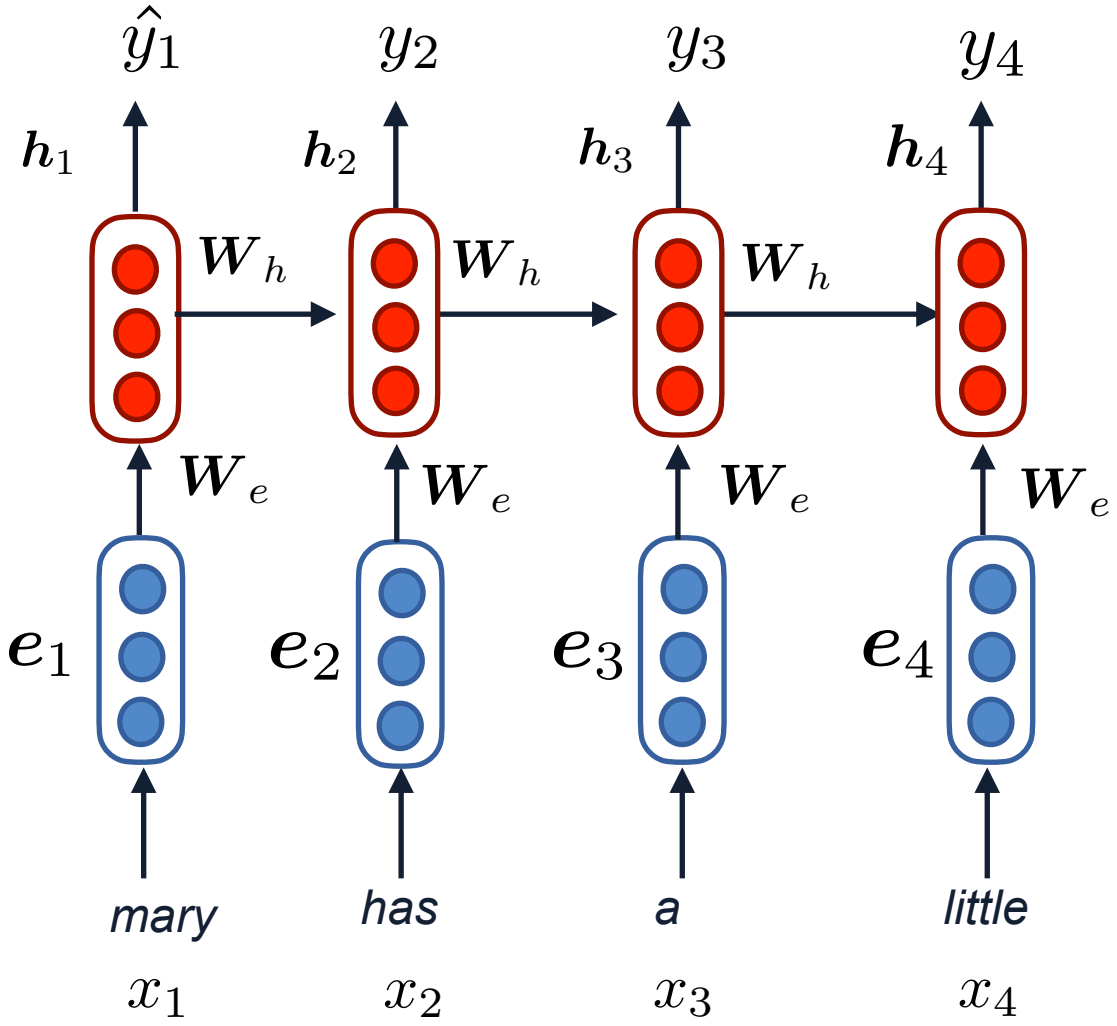
- Problem 1: we take a large step and reach a bad parameter position

- 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

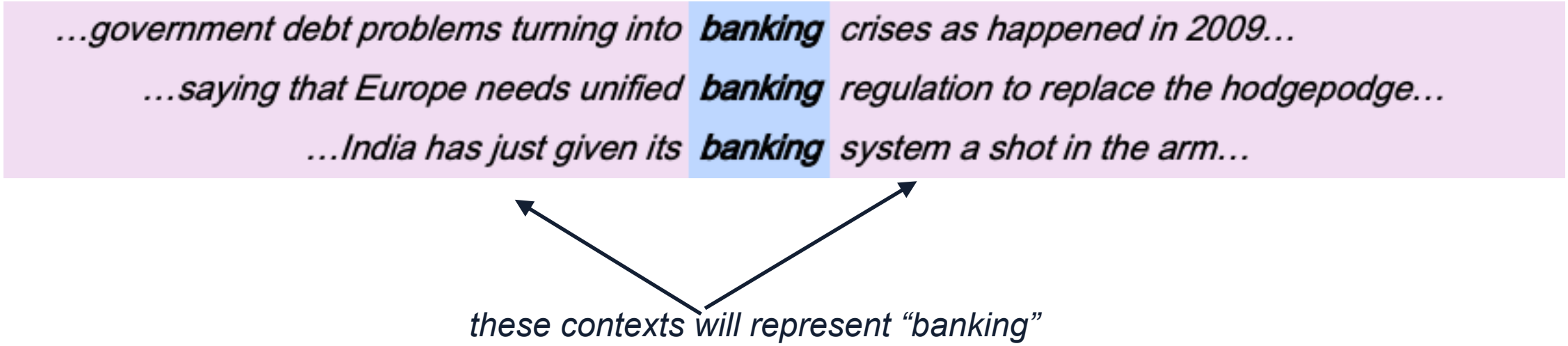
Why word embedding

- 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 starting point?

- 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)



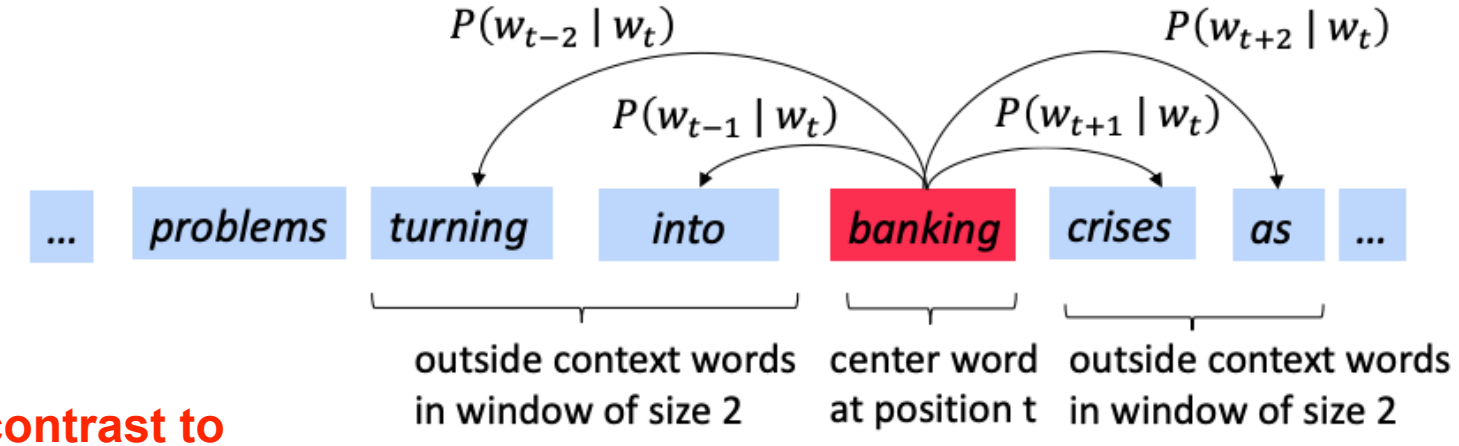
Word2vec Overview [Mikolov et al. 13]

- Skipgram: maximizing the probability of the context words given w_t

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

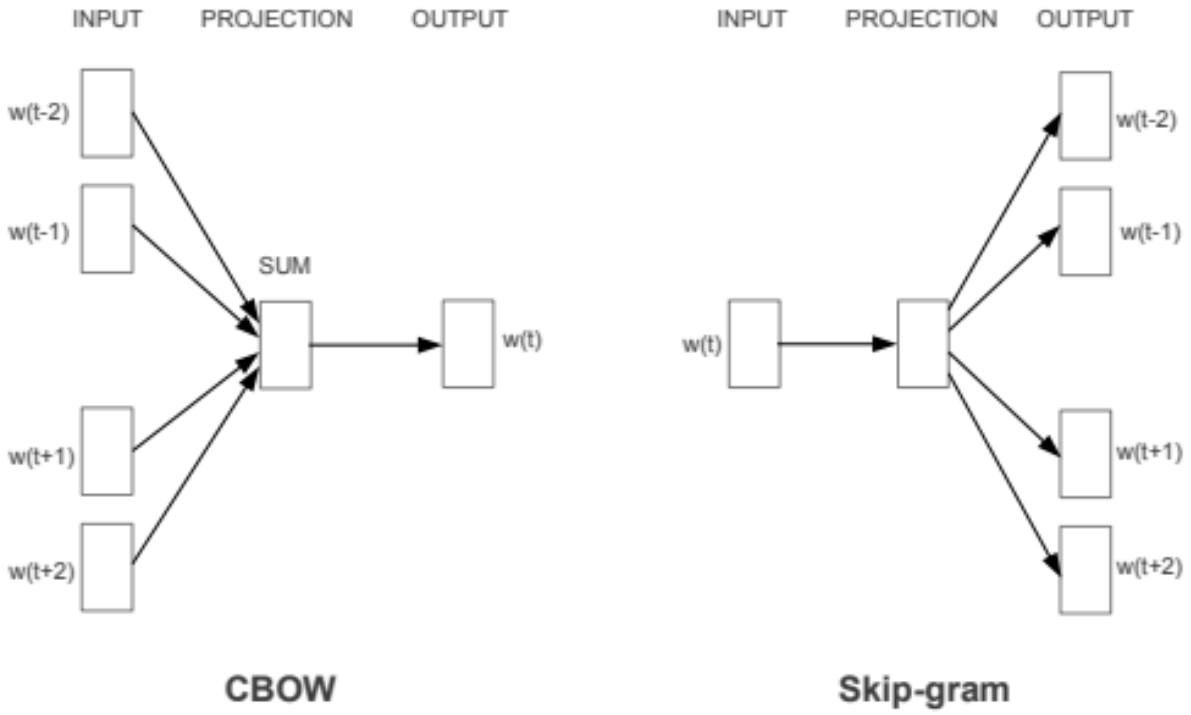
$$P(w_{t+j} \mid w_t) = \frac{\exp\left(u_{w_{t+j}}^T v_{w_t}\right)}{\sum_{w\in V}\exp\left(u_w^T v_{w_t}\right)}$$

probability of predicting w_t+j in contrast to predicting any other words



Word2vec Overview [Mikolov et al. 13]

- CBOW: maximizing the probability of w_t given context words

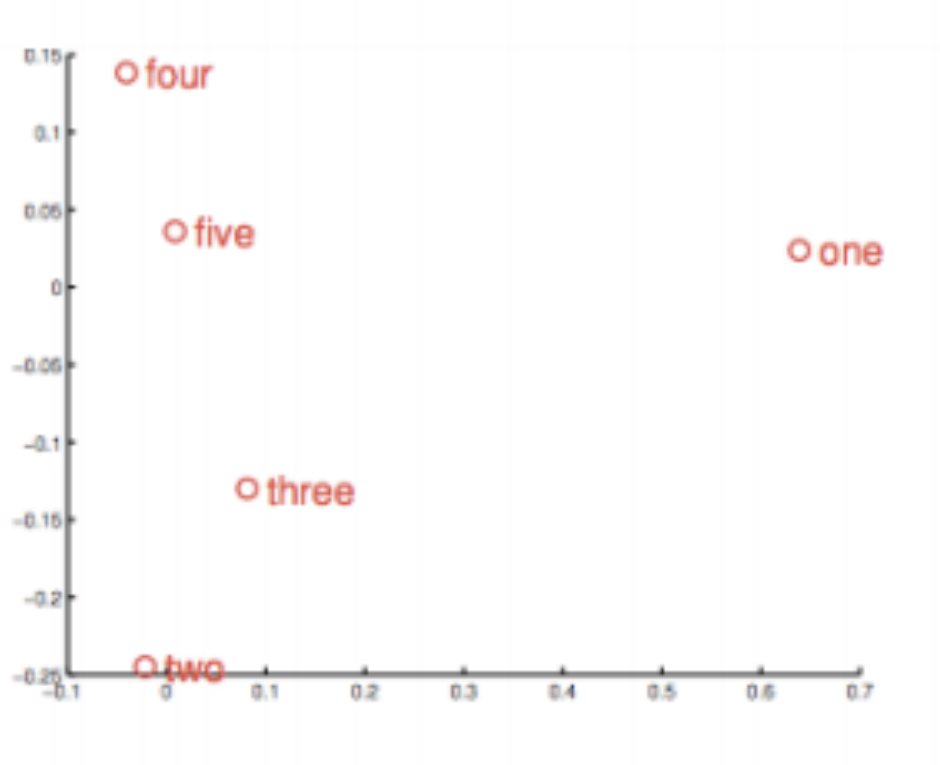


Word embedding results

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

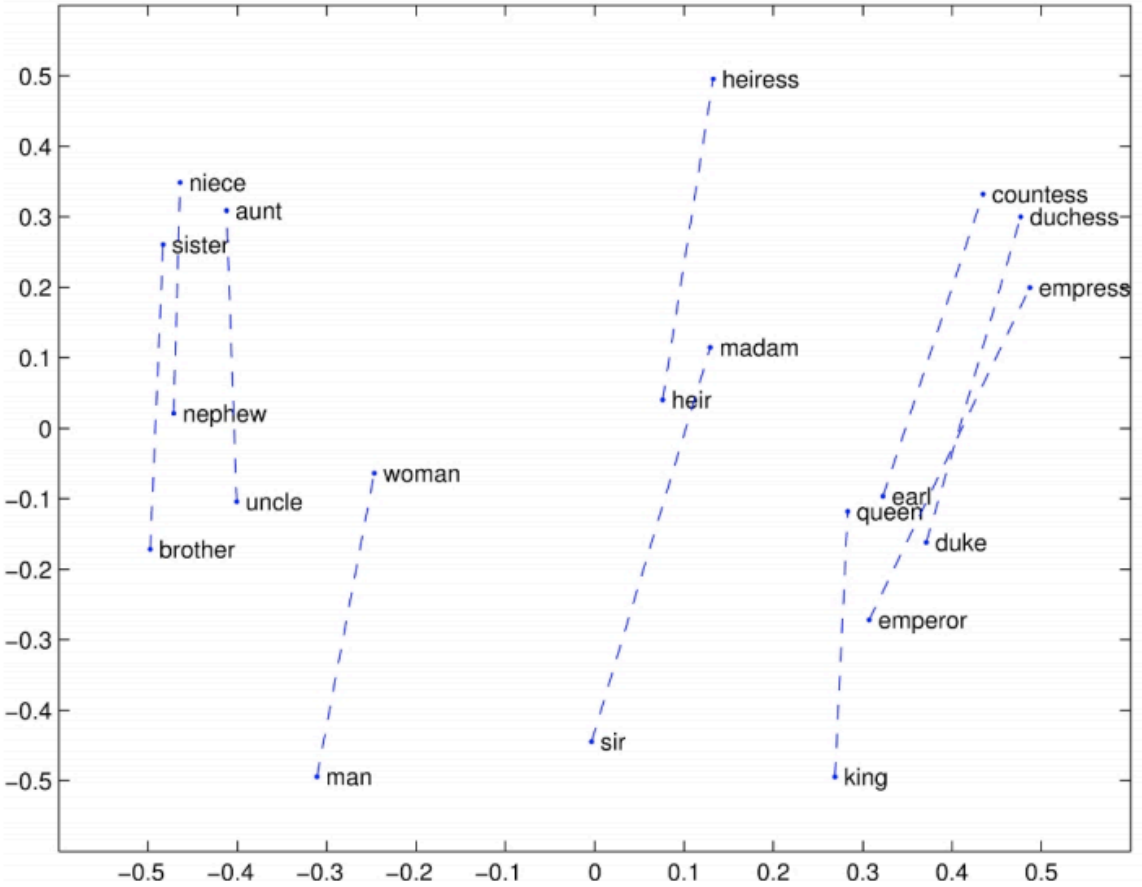
words closest to “sweden”

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

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

expensive to compute

- 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

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

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:

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

Summary of word embedding

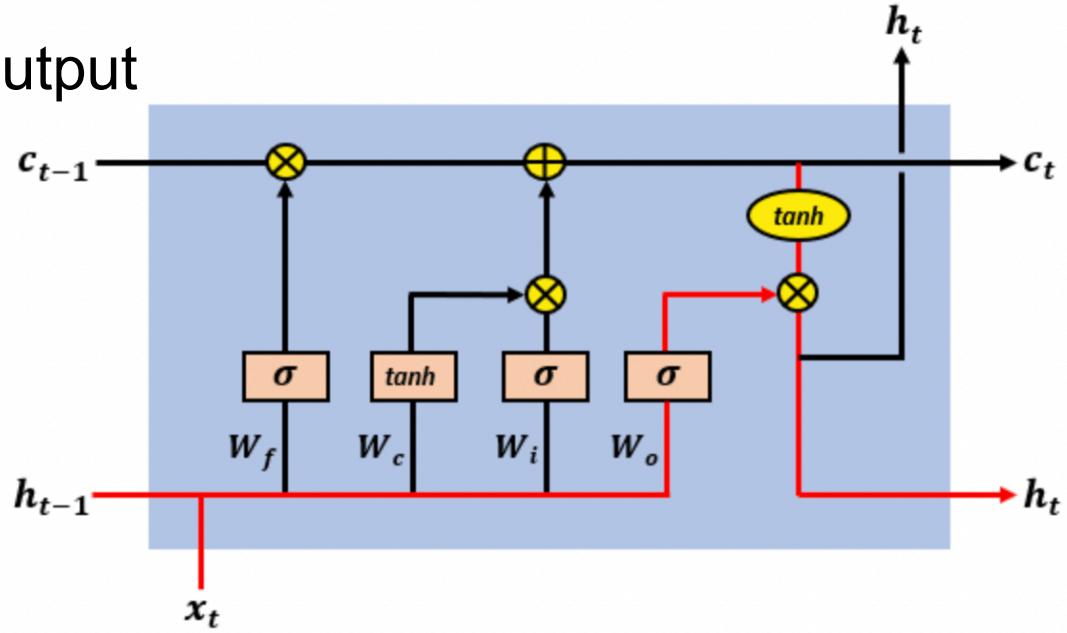
- Word embedding is designed for initializing the embedding layer of a neural network
- Skipgram and CBOW train word embedding vectors by predicting the probability of word generation from context words
- Training of word embedding vectors can be improved using negative sampling, subsampling of frequent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

Long short term memory [Hochreiter & Schmidhuber 97]

- Solution to the vanishing gradient problem

- At time t, there is a hidden state h_t and a cell state c_t
  - Both vectors are of the same length
  - The cell state stores long-term information
  - Four gates: forget, input modulation, input, output

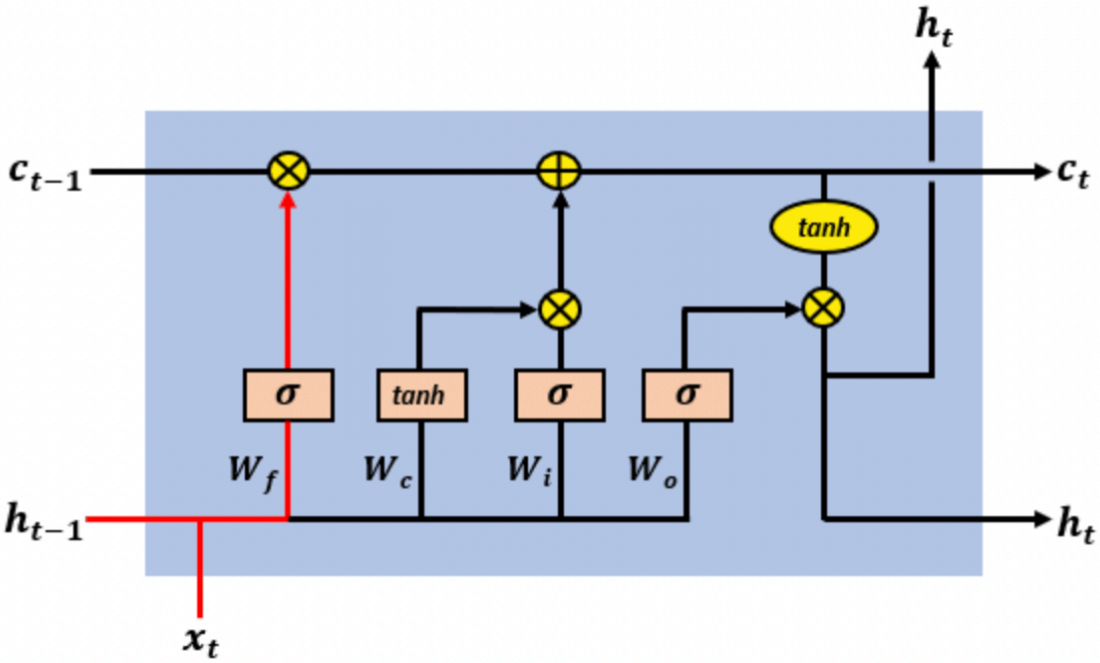
- The forget gate helps solve vanishing gradient



Long short term memory [Hochreiter & Schmidhuber 97]

- Forget gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t])$$

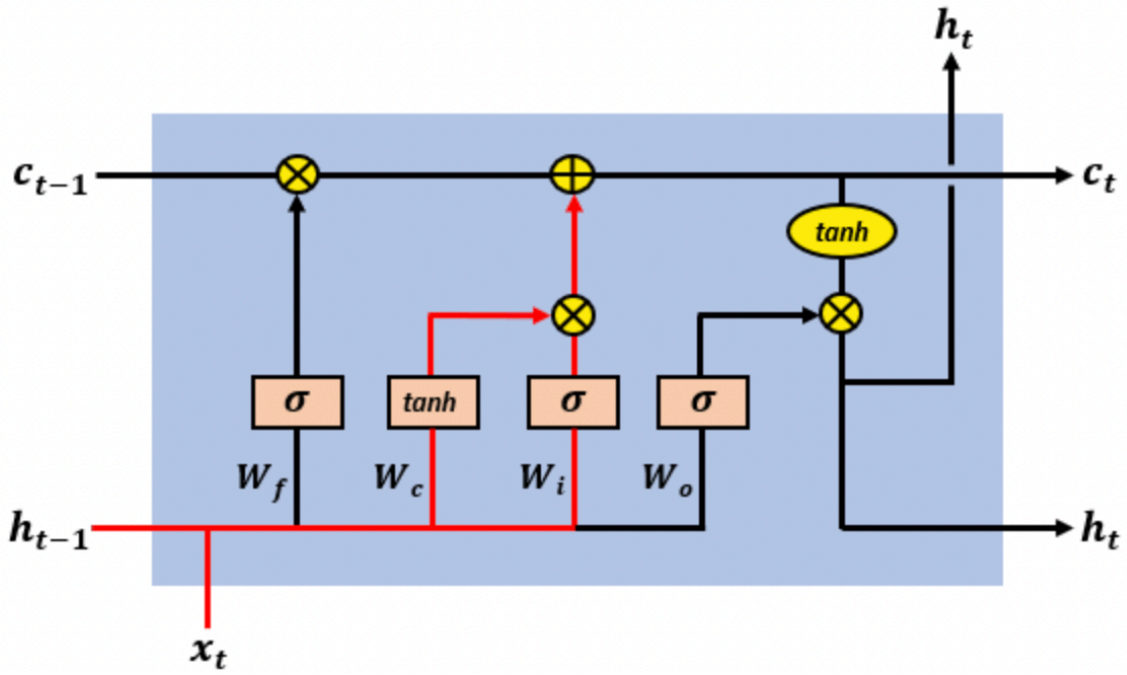


https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

Long short term memory [Hochreiter & Schmidhuber 97]

- Input and input modulation gate:

$$\tilde{c}_t \times i_t = tanh(W_c \cdot [h_{t-1}, x_t]) \times \sigma(W_i \cdot [h_{t-1}, x_t])$$

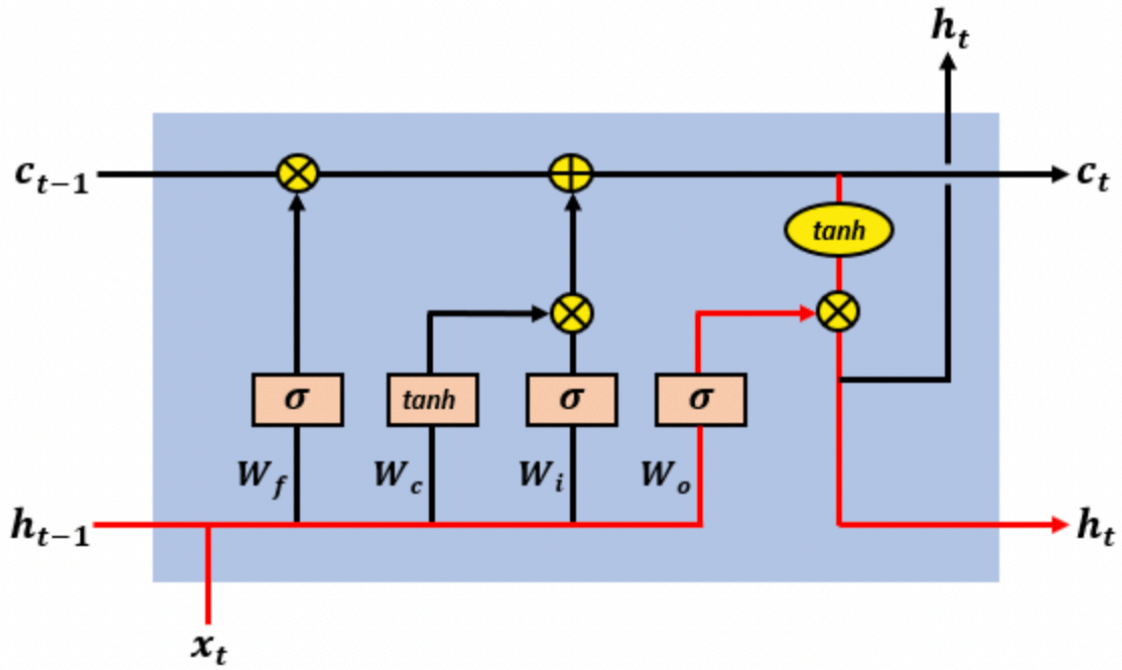


https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

Long short term memory [Hochreiter & Schmidhuber 97]

- Output gate (controls the output hidden state):

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$
$$h_t = o_t \times tanh(c_t)$$
$$= \sigma(W_o \cdot [h_{t-1}, x_t]) \times tanh(c_t)$$



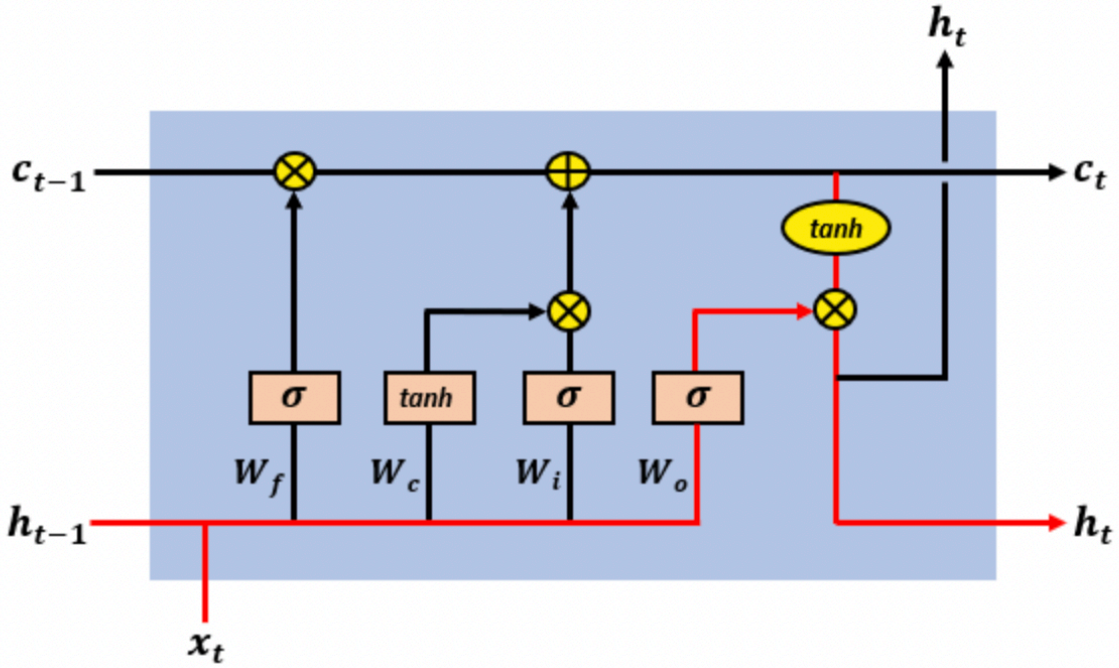
https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

BPTT for LSTM

$$c_t = (c_{t-1} \times f_t) + (\tilde{c}_t \times i_t)$$

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t]) \times tanh(c_t)$$

- From BPTT for RNN, we saw that the major reason for vanishing gradient to happen is the product of $\frac{\partial h_t}{\partial h_{t-1}}$'s.
- This problem, however, does not occur for LSTM



https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

BPTT for LSTM

- Theorem 1: The gradient of the objective function $J(\theta)$ w.r.t. parameter θ is the sum of derivative of $J(\theta)$ w.r.t. θ along all possible paths.

- As a result, the gradient vanishes when the gradient flows through EVERY PATH from θ to $J(\theta)$ vanish.

- Let's look at one specific path and prove the gradient does not vanish there

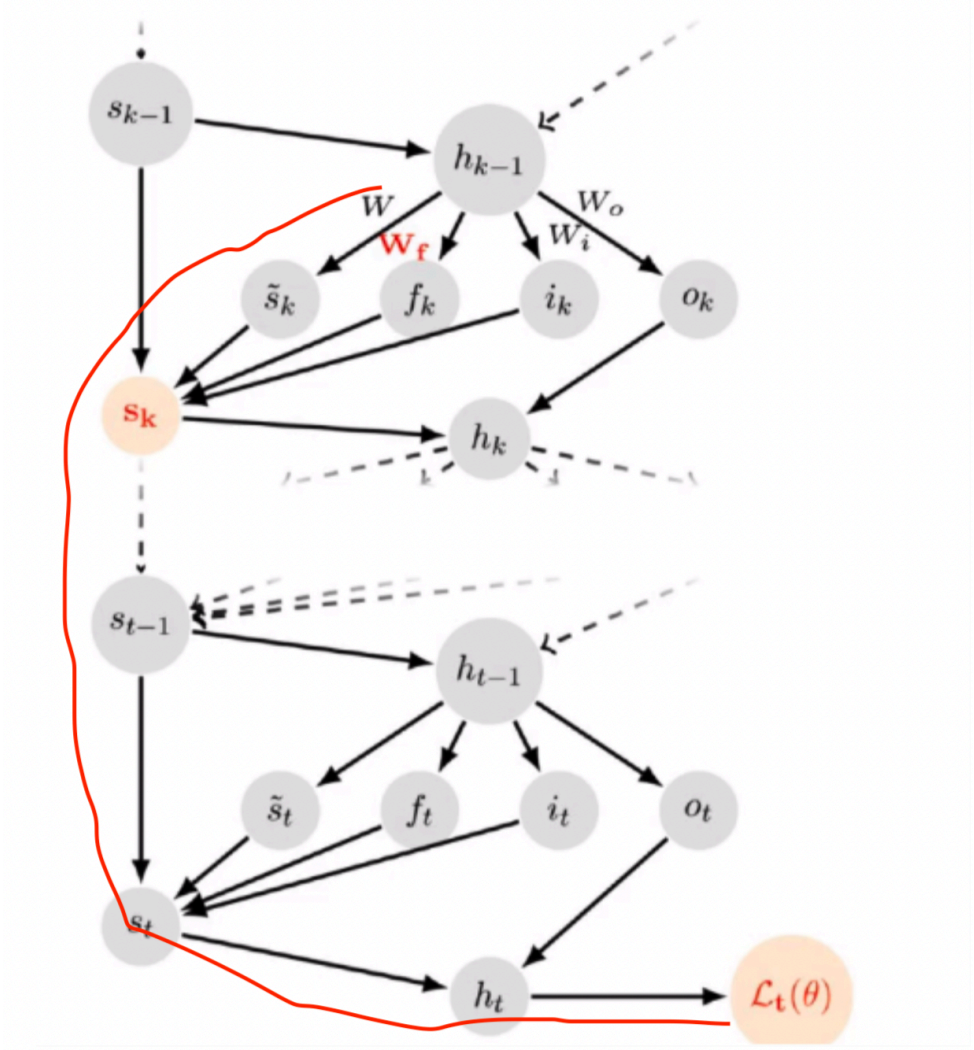
https://prvnk10.medium.com/how-lstms-solve-the-problem-of-vanishing-gradients-ea88f08c78ca

BPTT for LSTM

$$\frac{\partial J_t(\theta)}{\partial c_k} = \frac{\partial J_t(\theta)}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\cdots\frac{\partial c_{k+1}}{\partial c_k}$$

$$\begin{aligned}\frac{\partial c_t}{\partial c_{t-1}} &= \frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial c_{t-1}} + f_t \\ &+ \frac{\partial \tilde{c}_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial c_{t-1}}\cdot i_t + \frac{\partial i_t}{\partial h_{t-1}}\cdot\frac{\partial h_{t-1}}{\partial c_{t-1}}\tilde{c}_t \\ &\equiv c_{t-1}\cdot\sigma'(W_t[h_{t-1},x_t])\cdot W_f\cdot o_t\cdot tanh'(c_{t-1}) \\ &+ f_t \\ &+ i_t\cdot tanh(W_t[h_{t-1},x_t])\cdot W_c\cdot o_t\cdot tanh'(c_{t-1}) \\ &+ \tilde{c}_t\cdot\sigma'(W_i[h_{t-1},x_t])\cdot W_i\cdot o_t\cdot tanh'(c_{t-1})\end{aligned}$$

avoid gradient vanishing problem by learning f_t



https://prvnk10.medium.com/how-lstms-solve-the-problem-of-vanishing-gradients-ea88f08c78ca

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

Final Project

- Students will form into groups of 2
- Propose a project by Nov 15
- Discuss with me if you are not sure

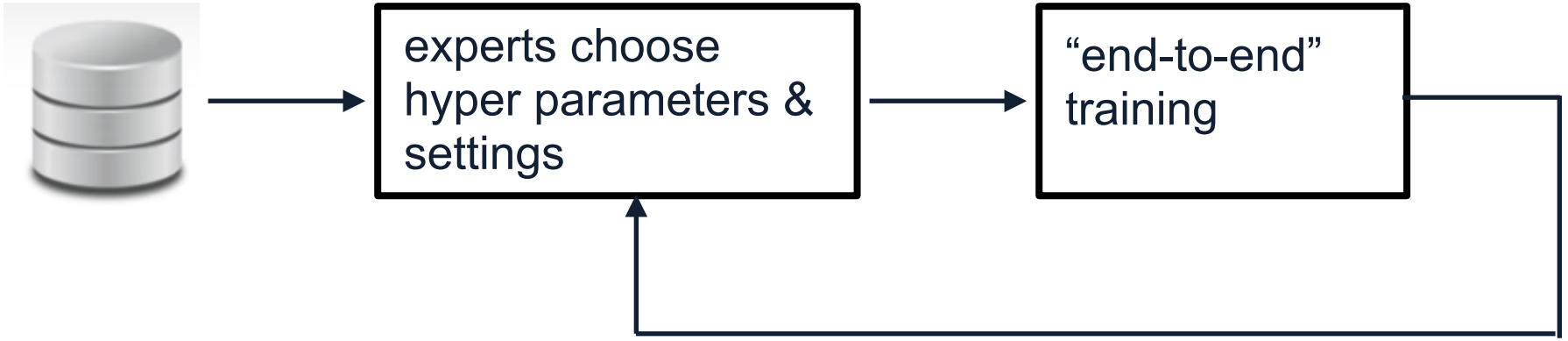
Hyperparameter optimization

- Deep neural network is sensitive to hyperparameters
- Tuning hyperparameter is a black magic
- Training requires significant time and efforts!
  - P100 ~$500 per month
  - V100 ~$1,000- $1,500 per month

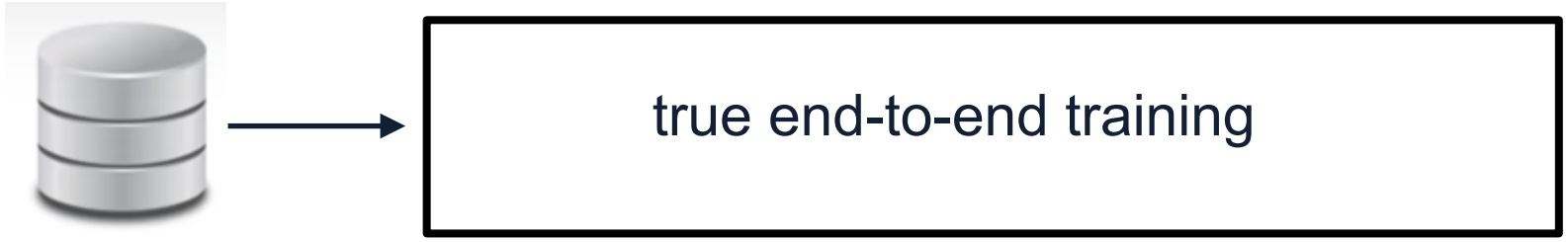
NVIDIA® Tesla® V100	1 GPU	16 GB HBM2	$2.48 USD per GPU
	2 GPUs	32 GB HBM2	
	4 GPUs	64 GB HBM2	
	8 GPUs	128 GB HBM2	

Automated machine learning

Current deep learning practice



AutoML: true end-to-end training



Auto-sklearn [Feurer et al. 2018]

- Tuning happens behind the scene

sklearn:
NB

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.66666667 0.33333333]]
```

auto-
sklearn

```
>>> import autosklearn.classification
>>> cls = autosklearn.classification.AutoSklearnClassifier()
>>> cls.fit(X_train, y_train)
>>> predictions = cls.predict(X_test)
```

Hyperparameter tuning

- Deep neural network is sensitive to hyper parameters

Config 1:
lr = 2e-5
batch size = 32
epoch = 3.0
max_seq_length = 128

Config 2:
lr = 5e-5
batch size = 128
epoch = 10.0
max_seq_length = 128

- Which config to choose?

BERT: epoch = 3.0, RoBERTa: epoch = 10.0

Hyperparameter optimization

- Definition of hyperparameter tuning:

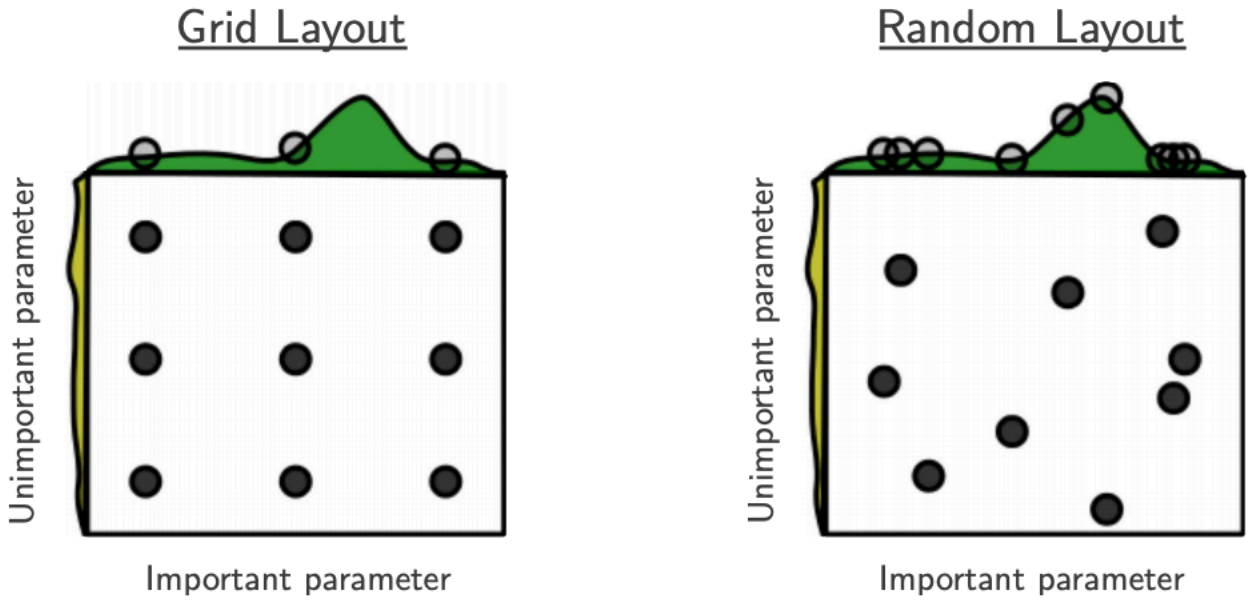
$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} \, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid}),$$

- $\boldsymbol{\lambda}$: hyperparameter
- $\boldsymbol{\Lambda}$: search space for hyperparameter

- D_{train}: training dataset
- D_{valid}: validation dataset
- L: loss function (objective)

Hyperparameter optimization [Bergstra et al. 2012]

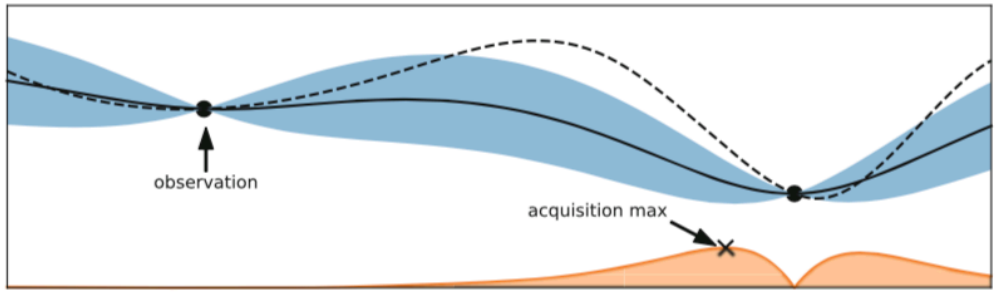
- Grid search vs. random search

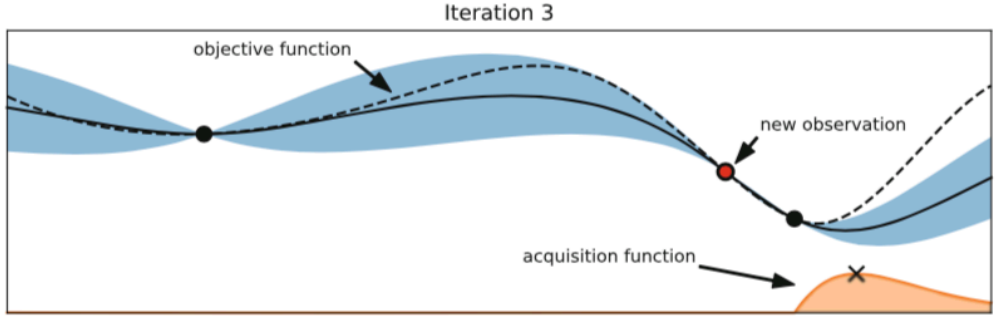


- Grid search: explored 3 important hyperparameters in 9 **trials**
- Random search: explored 9 hyperparameters in 9 trials

Bayesian optimization [Bergstra et al. 2012]

- Approach: fix a probabilistic model to the function evaluation
- Use that model for exploration vs. exploitation
- During the development of AlphaGO, many hyper parameters were tuned with BO many times, e.g., before the match with Sedol Lee, HPO improved the win-rate from 50% -> 65%





SMBO(f, M_0, T, S)

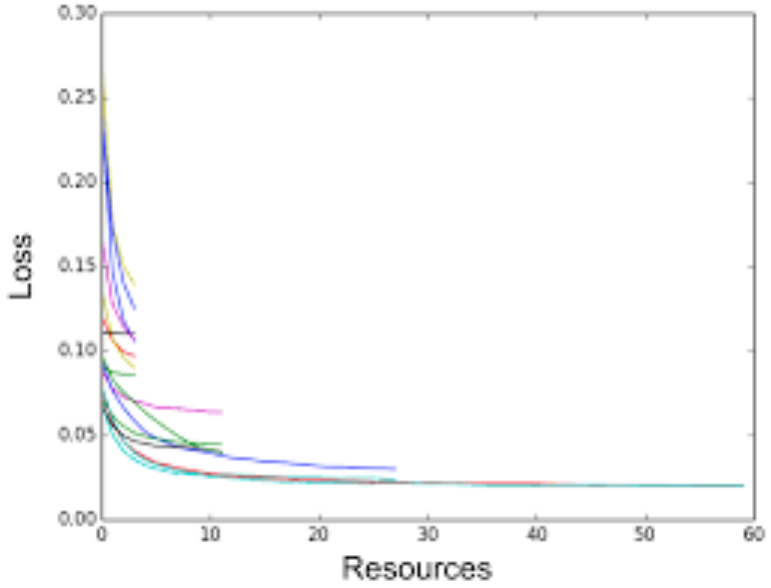
1 $\mathcal{H} \leftarrow \emptyset$,
2 For $t \leftarrow 1$ **to** T,
3 $x^* \leftarrow \text{argmin}_x S(x, M_{t-1})$,
4 Evaluate $f(x^*)$, ▷ *Expensive step*
5 $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6 Fit a new model M_t to $\mathcal{H}$.
7 **return** $\mathcal{H}$

acquisition function

Early stopping and pruning

- If a trial shows bad performance, no need to keep running it

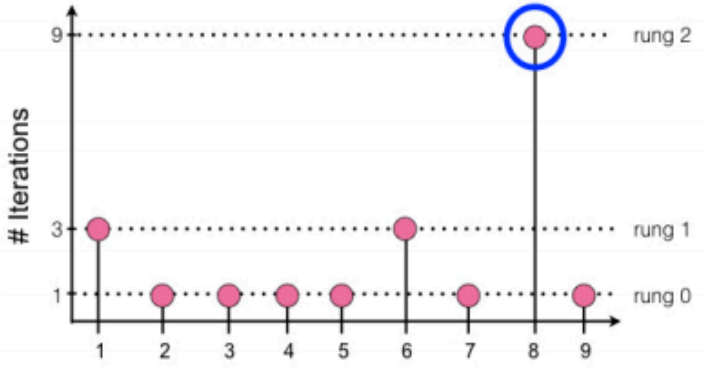
- Successive halving [Jamieson et al. 2015]
  - A fixed amount of resources
  - Allocate the resources by first aggressively exploring, then exploiting



	$s = 4$	
i	n_i	r_i
0	81	1
1	27	3
2	9	9
3	3	27
4	1	81

Asynchronous Successive Halving [Li et al. 2020]

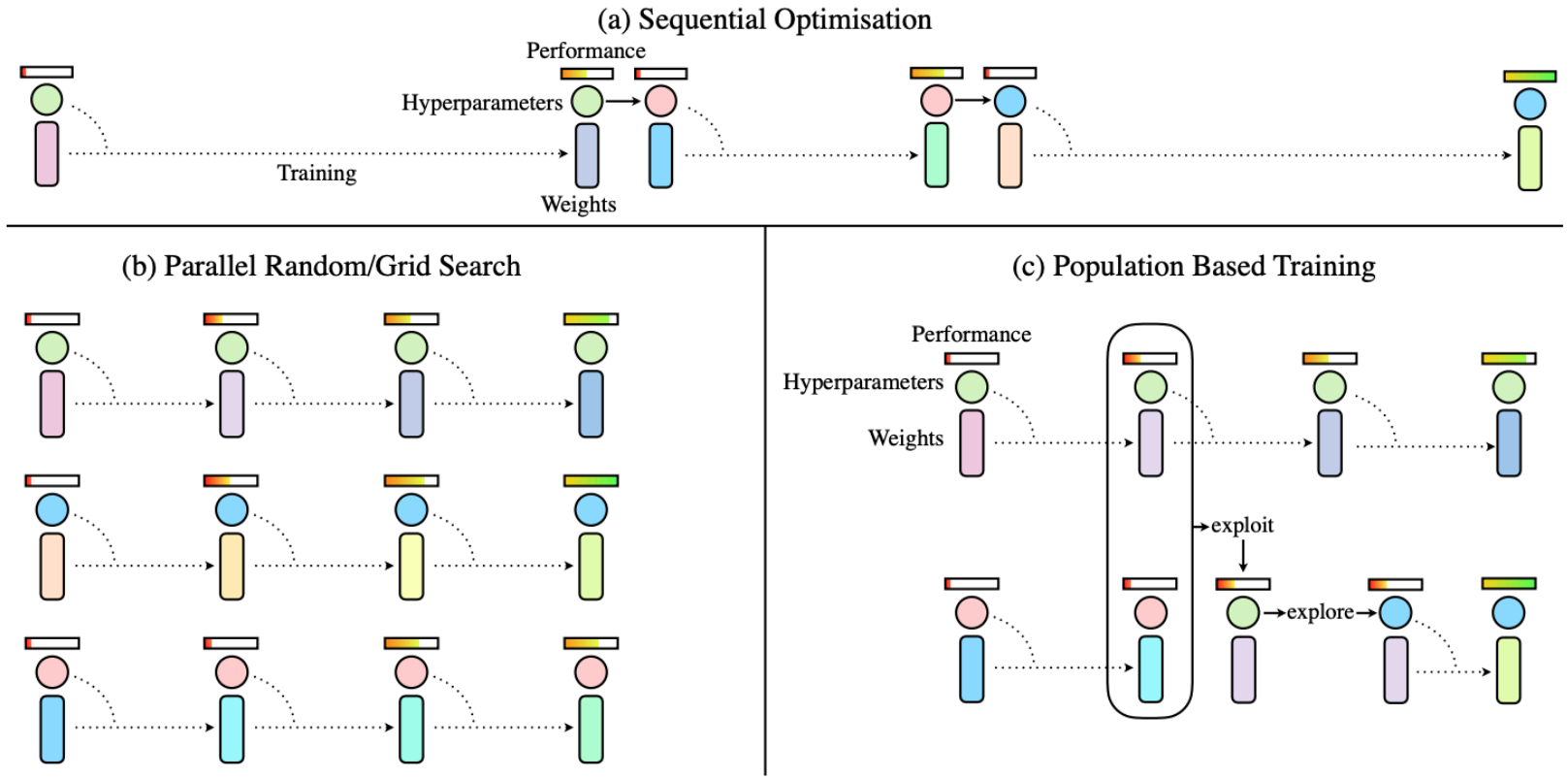
- Synchronous SHA are not efficient because the higher rungs are waiting for the lower rungs to finish
- Instead of waiting, we can promote lower rungs to higher rungs
- Time cost = 13/9 time (R) vs. 3 time (R)



bracket s	rung i	n_i	r_i	total budget
0	0	9	1	9
	1	3	3	9
	2	1	9	9
1	0	9	3	27
	1	3	9	27
2	0	9	9	81

Population-based training [Jaderberg et al. 2017]

- Evolutionary algorithm for hyperparameter optimization



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