Parameter Initialization

- You normally must initialize weights to small random values (i.e., not zero matrices!)
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights** ~ Uniform($-r$, r), with r chosen so numbers get neither too big or too small [later the need for this is removed with use of layer normalization]
- Xavier initialization has variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size):

$$\mathrm{Var}(W_i) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

Optimizers

- Usually, plain SGD will work just fine!
  - However, getting good results will often require hand-tuning the learning rate
    - See next slide
- For more complex nets and situations, or just to avoid worry, you often do better with one of a family of more sophisticated "adaptive" optimizers that scale the parameter adjustment by an accumulated gradient.
  - These models give differential per-parameter learning rates
    - Adagrad
    - RMSprop
    - Adam ← A fairly good, safe place to begin in many cases
    - SparseAdam
    - …

Optimizers: SGD with momentum

On iteration t:

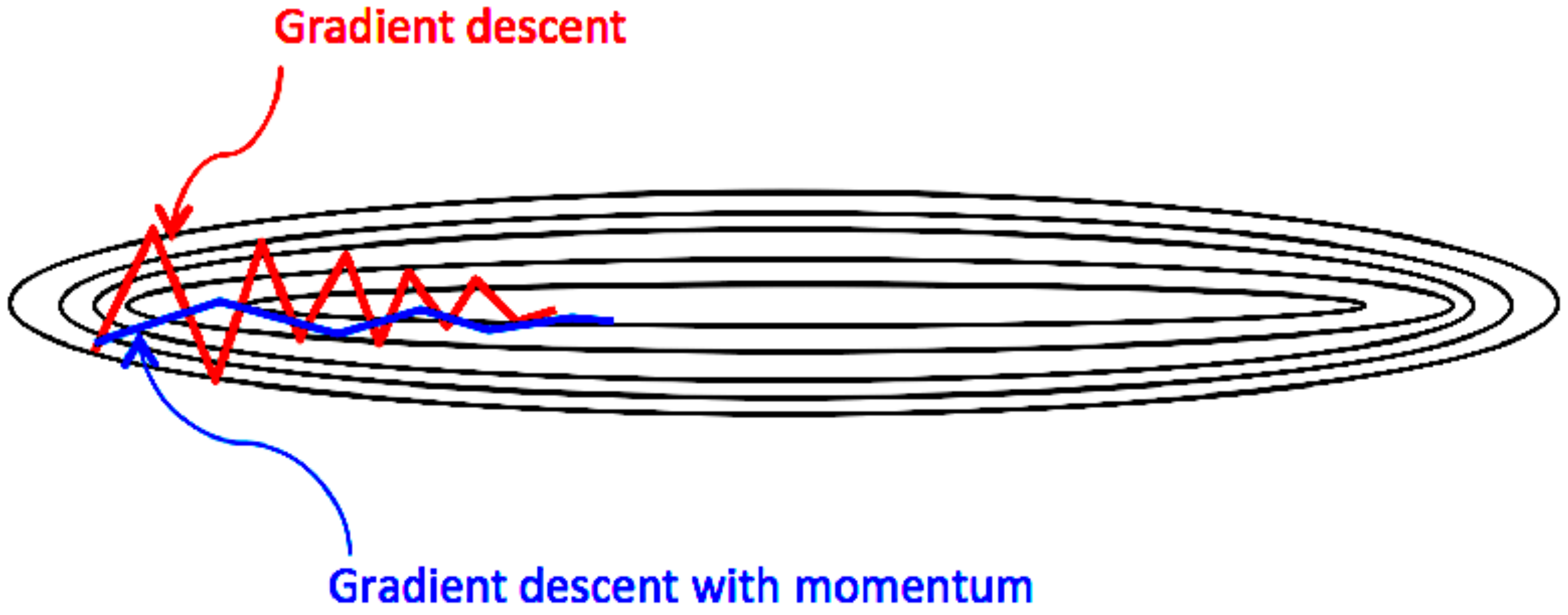
Compute dW, db

$$v_{dW} = \beta v_{dW} + (1 - \beta) dW$$

$$v_{db} = \beta v_{db} + (1 - \beta) db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\qquad \beta = 0.9$



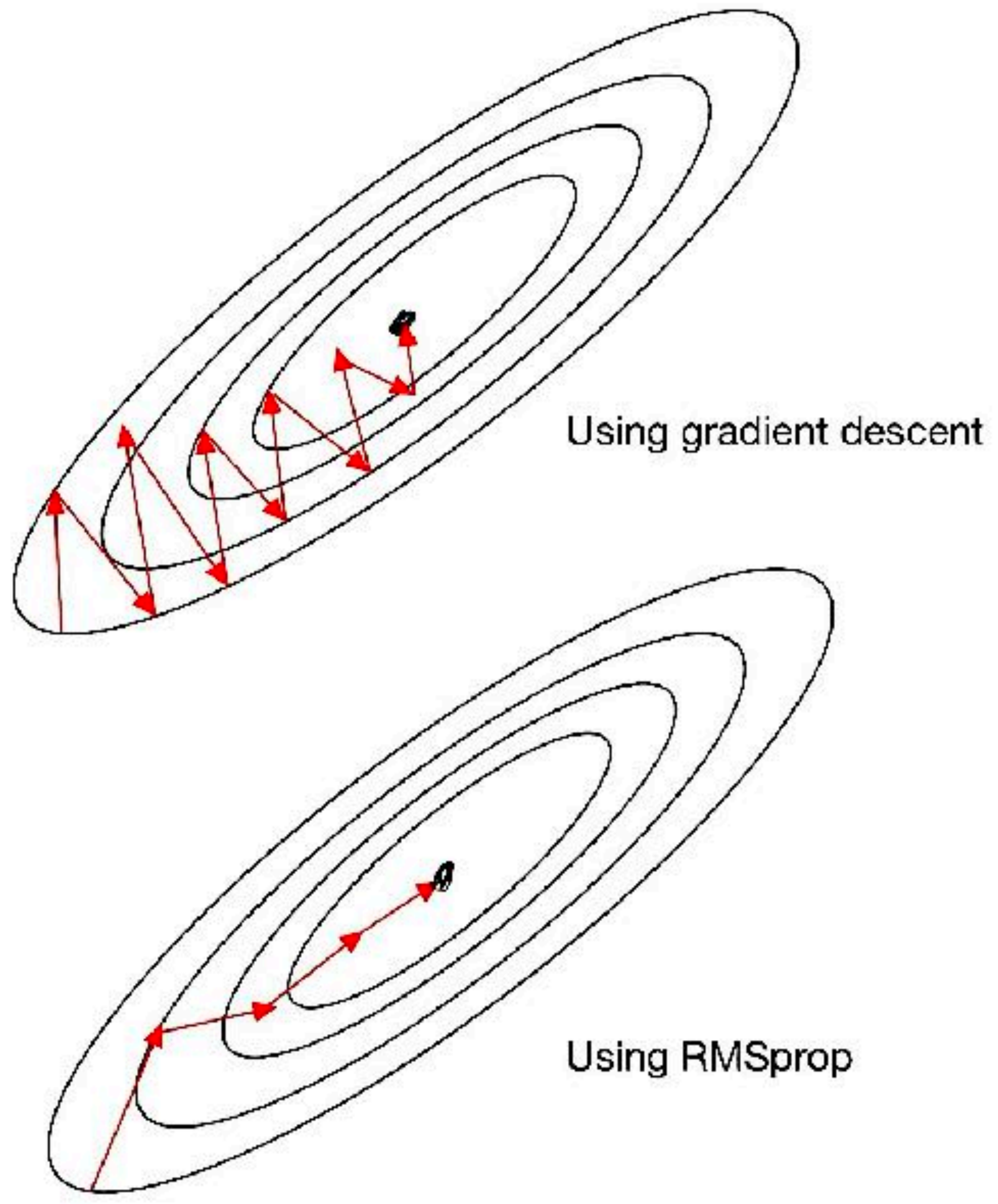
Optimizers: RMSprop

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$



Optimizers: Adam

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db \longrightarrow \text{MOMENTUM}$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dW^2, \quad S_{db} = \beta_1 S_{db} + (1-\beta_1)\, db^2 \longrightarrow \text{RMS PROP}$$

$$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t), \quad S_{db} = S_{db} / (1-\beta_2^t)$$

$$\left. W = W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}, \quad b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon} \right\} \text{UPDATE}$$

$$v_{dW} = \beta v_{dW} + (1-\beta) dW$$

$$v_{db} = \beta v_{db} + (1-\beta) db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

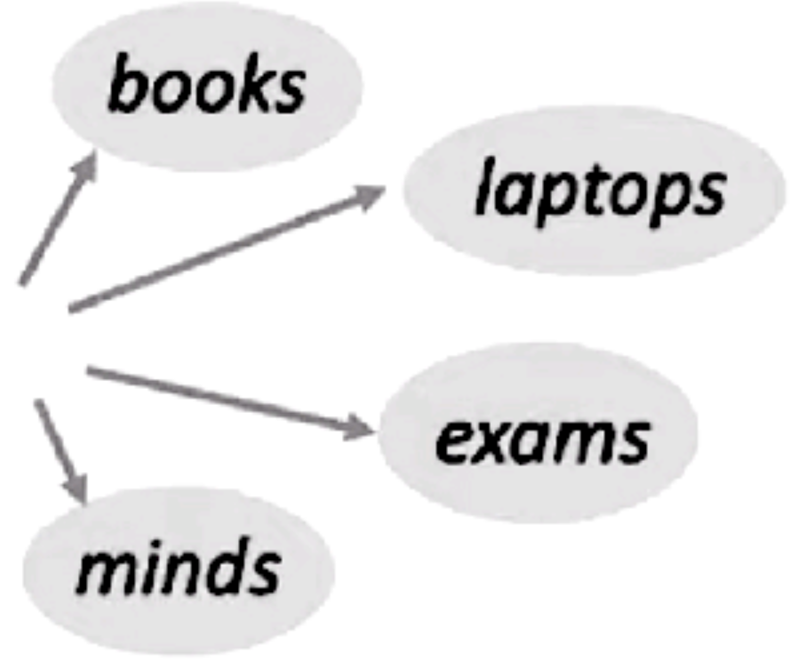
Choosing learning rate

- You can just use a constant learning rate. Start around *lr* = 0.001?
  - It must be order of magnitude right – try powers of 10
    - Too big: model may diverge or not converge
    - Too small: your model may not have trained by the assignment deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every *k* epochs
    - An epoch = a pass through the data (shuffled or sampled – not in same order each time)
  - By a formula: $lr = lr_0 e^{-kt}$, for epoch *t*
  - There are fancier methods like cyclic learning rates (q.v.)
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so you may want to start with a higher learning rate

Language Modeling

- Language Modeling is the task of predicting what word comes next

the students opened their ______



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

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a Language Model

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text

- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\ &= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}} \end{aligned}$$

N-gram Language Model

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- **Definition:** A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “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 Model

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}$$

(definition of conditional prob)

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

N-gram Language Model

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* ______

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the 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

Should we have discarded the "proctor" context?

N-gram Language Model

Sparsity Problem 1

<u>Problem:</u> What if *"students opened their w"* never occurred in data? Then w has probability 0!

<u>(Partial) Solution:</u> Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\boxed{\text{count}(\text{students opened their})}}$$

Sparsity Problem 2

<u>Problem:</u> What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

<u>(Partial) Solution:</u> Just condition on *"opened their"* instead. This is called *backoff*.

N-gram Language Model: Text generation

today the _______

today the _______

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

get probability distribution

company	$\boxed{0.153}$
bank	$\boxed{0.153}$
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

N-gram Language Model: Text generation

$\underbrace{\textit{today the}}_{\text{condition on this}}$ ______

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

sampl

today $\underbrace{\textit{the price}}_{\text{condition on this}}$ ______

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

today the $\underbrace{\textit{price of}}_{\text{condition on this}}$ ______

get probability distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

N-gram Language Model: Text generation

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 .

Neural Language Model

output distribution

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

hidden layer

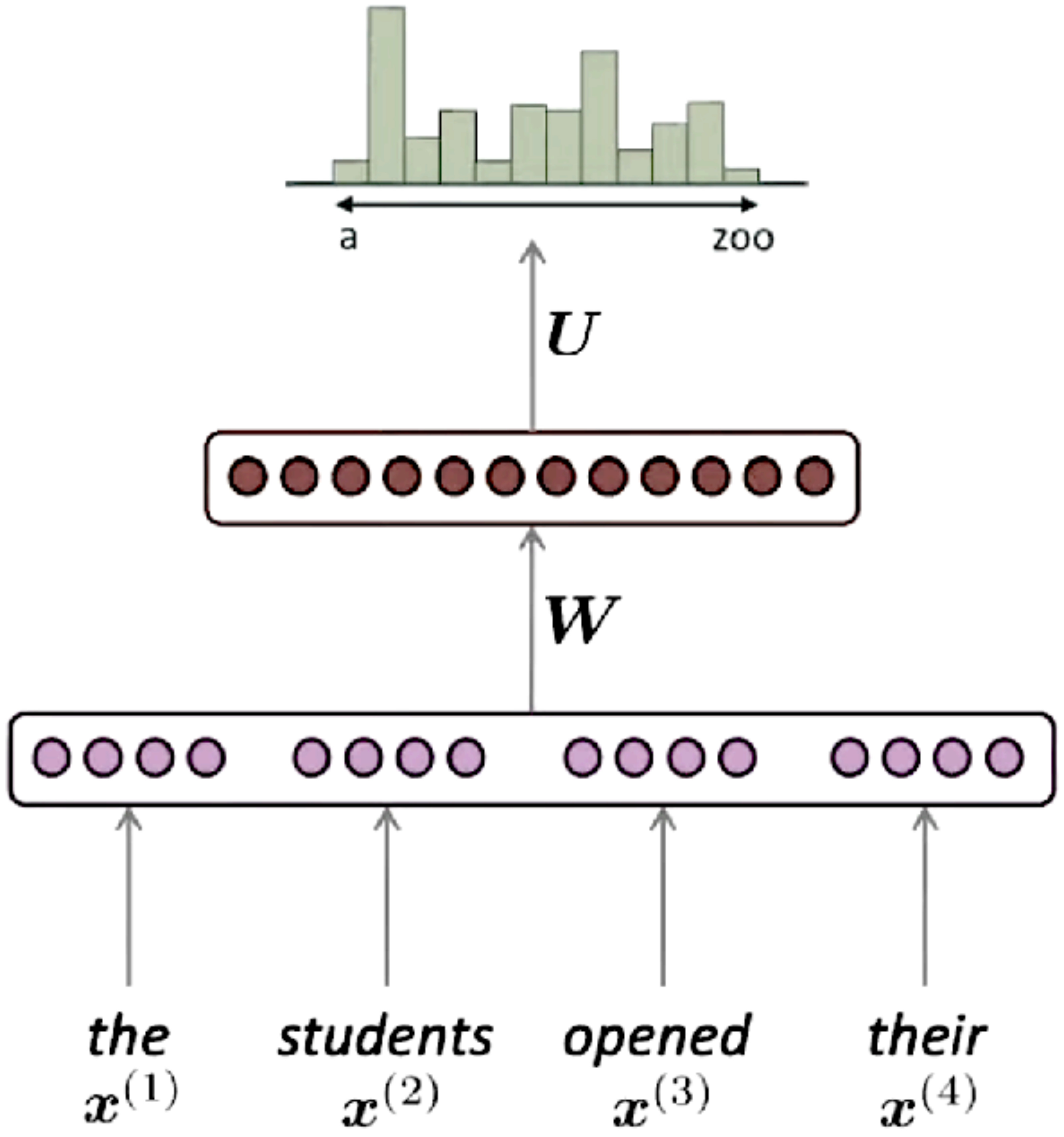
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

Y. Bengio, et al. (2000/2003)

Neural Language Model

We need a neural architecture
that can process *any length input*

Recurrent Neural Networks (RNNs)