

CS 1671/2071

# Human Language Technologies

Session 13: Neural networks, part 2

---

Michael Miller Yoder









February 24, 2025



School of Computing and Information

# Course logistics: homework

- Thanks for your Homework 2 submissions
- Kaggle results on private leaderboard

BridgetBrinkman		0.672
Abe		0.672
César Guerra-Solano		0.663
braynguyen		0.663
Stephen Gwon		0.655
Jeremy Luu		0.646
Jal355		0.620
Benjup		0.586

# Course logistics: project proposals

- [Project proposal](#) due **this Fri Feb 28**
- Set of 8 questions to answer with a PDF report
- Think about how you would apply approaches we have covered so far (n-gram feature extraction, logistic regression, n-gram language modeling) to your task
- Feel free to email or book office hours with Michael to discuss
- Peer review where you will rate your own performance and the performance of other group members (*will be posted*)
  - Not used for grading, just to identify and address any issues

# Course logistics: project proposal presentations

- [Proposal presentations](#) in class **Mon Mar 10** (Mon after spring break)
- Michael will soon post:
  - More instructions
  - A link to a shared PowerPoint document to add your slides to

# Midterm OMET feedback

- 20% participation
- What's helping learning
  - Lecture slides, going over problems in class, notebooks, quizzes (that make you read the textbook)
- What could change to help learning
  - Slowing the pace
  - Notebooks:
    - More explanation (especially for those unfamiliar with Python). Images/diagrams of what code is doing could help.
    - Releasing filled-out notebooks after class
    - Either completely filled-in or less filled-in
    - Is ML programming just calling libraries?
- Changes I will try to make
  - Release filled-in notebooks after class
  - More time on explanation of code in notebooks

# Course logistics

- This Wed Feb 26, Norah Almousa will be giving the lecture
  - Michael will be at a workshop downtown (feel free to still email me, etc)

# Structure of this course

## MODULE 1

### Prerequisite skills for NLP

text normalization, linear alg., prob., machine learning

### Approaches

### How text is represented

### NLP tasks

## MODULE 2

statistical machine learning

n-grams

language modeling  
text classification

## MODULE 3

neural networks

static word vectors

language modeling  
text classification

## MODULE 4

## MODULE 5

### NLP applications and ethics

machine translation, chatbots, information retrieval, bias

# A brief history of NLP



Sentences in Russian are punched into standard cards for feeding into the electronic data processing machine for translation into English

*The Georgetown-IBM Experiment.  
Credit: John Hutchins*

- 1950s: **foundations**
  - Turing Test ("Computing Machinery and Intelligence" paper)
  - Georgetown-IBM Experiment translating Russian to English
- 1960s-1980s: **symbolic reasoning**
  - Processing language with hand-built rules
- 1990s-2010s: **statistical NLP**
  - Learn patterns from large corpora (feature-based machine learning)
- 2000s-today: **neural NLP**
  - Powerful models of language from "deep" layers of neural networks trained on tons of text



# Lecture overview: neural networks, part 1

- Word embeddings as input to neural networks
- Training neural networks
- Recurrent neural networks (RNNs)
- Review activity
- (If time allows) project work time

# Feedforward neural networks with word embedding input

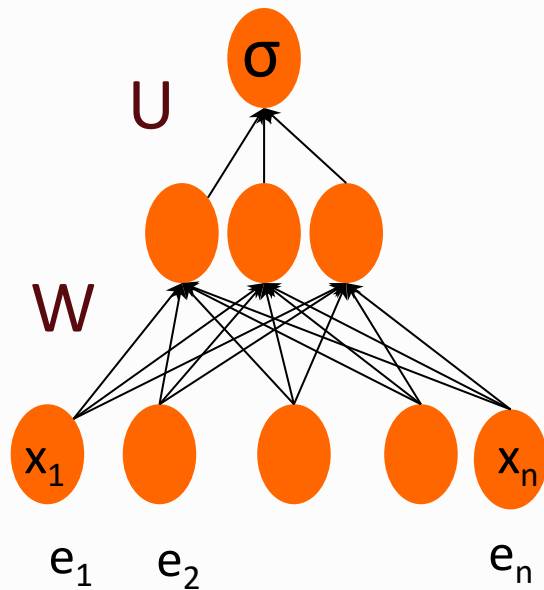
---

# Even better: representation learning

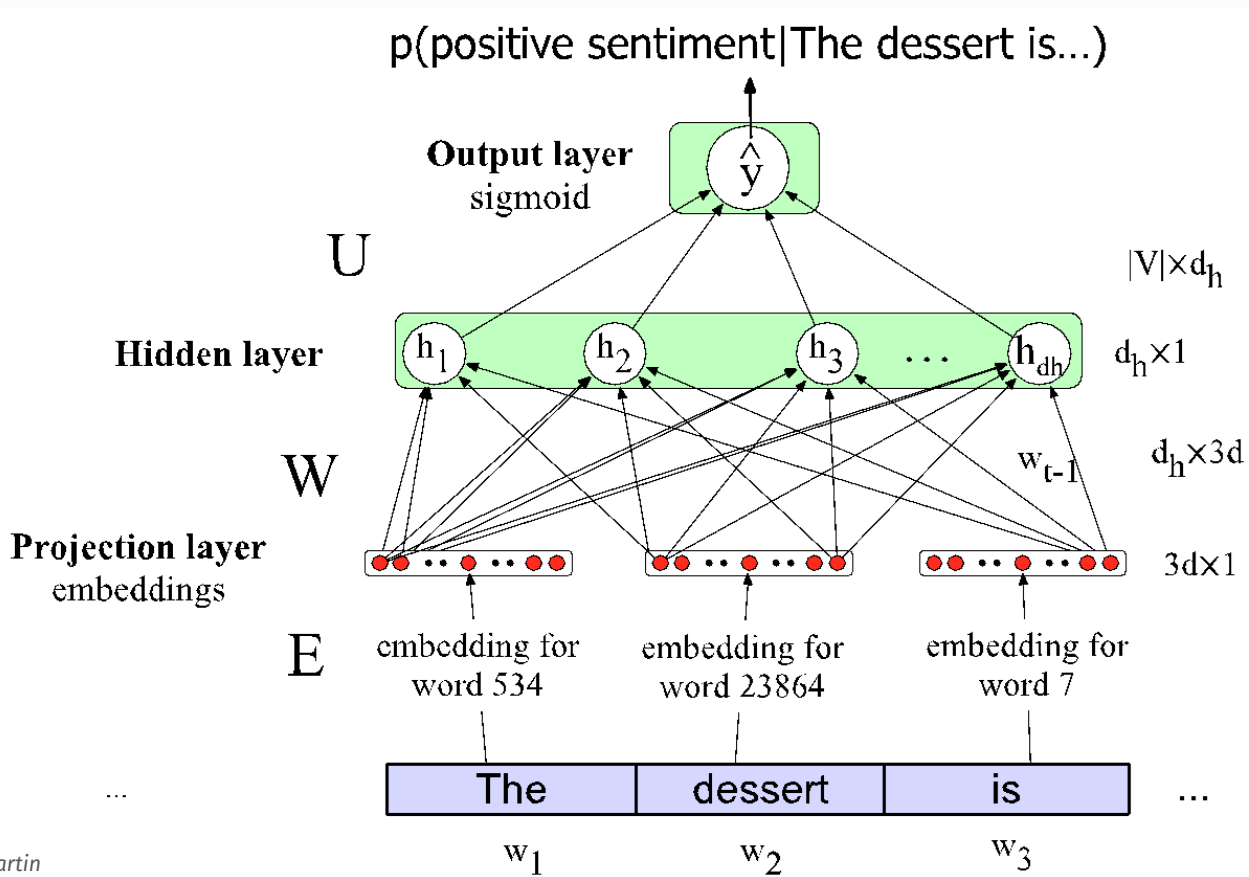
The real power of deep learning comes from the ability to **learn** features from the data

Instead of using hand-built human-engineered features for classification

Use learned representations like embeddings!



# Neural net classification with embeddings as input features!



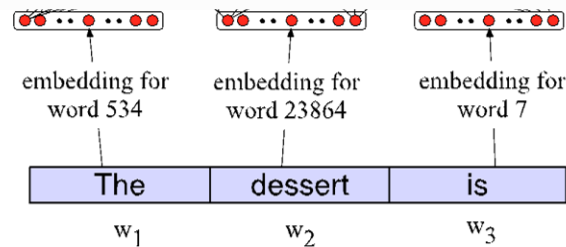
# Issue: texts come in different sizes

This assumes a fixed size length (3)!

Kind of unrealistic.

Some simple solutions:

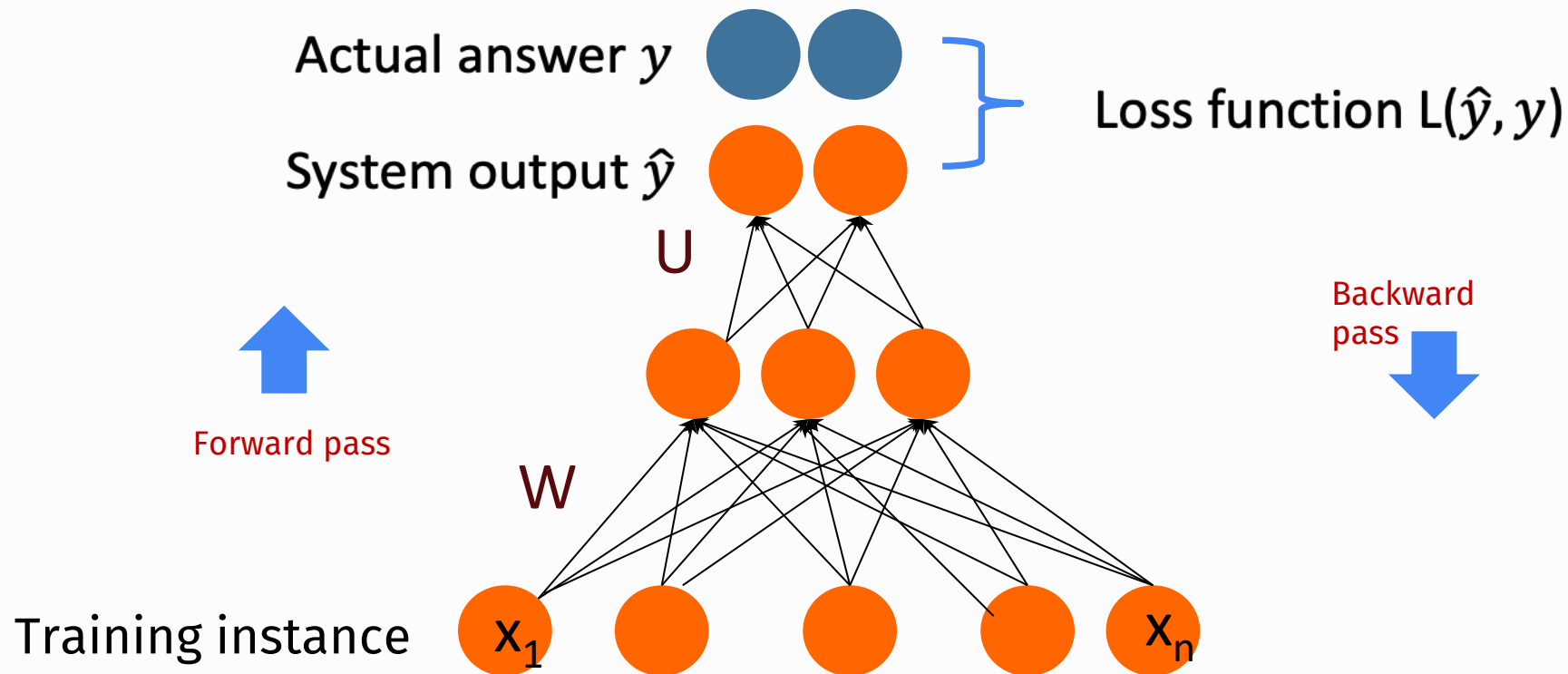
1. Make the input the length of the longest review
  - If shorter then pad with zero embeddings
  - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
  - Take the mean of all the word embeddings
  - Take the element-wise max of all the word embeddings
    - For each dimension, pick the max value from all words



# Training feedforward neural networks

---

# Intuition: training a 2-layer Network



Remember stochastic gradient descent  
from the logistic regression lecture—find  
gradient and optimize



# The Intuition Behind Training a 2-Layer Network

For every training tuple  $(x, y)$

1. Run **forward** computation to find the estimate  $\hat{y}$
2. Run **backward** computation to update weights
  - For every output node
    - Compute the loss  $L$  between true  $y$  and estimated  $\hat{y}$
    - For every weight  $w$  from the hidden layer to the output layer: update the weights
  - For every hidden node
    - Assess how much blame it deserves for the current answer
    - From every weight  $w$  from the input layer to the hidden layer
    - Update the weight

Computing the gradient requires finding the derivative of the loss with respect to each weight in every layer of the network.

**Error backpropagation through computation graphs.**

# Reminder: gradient descent for weight updates

Use the derivative of the loss function with respect to weights  $\frac{d}{dw} L(f(x; w), y)$

To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

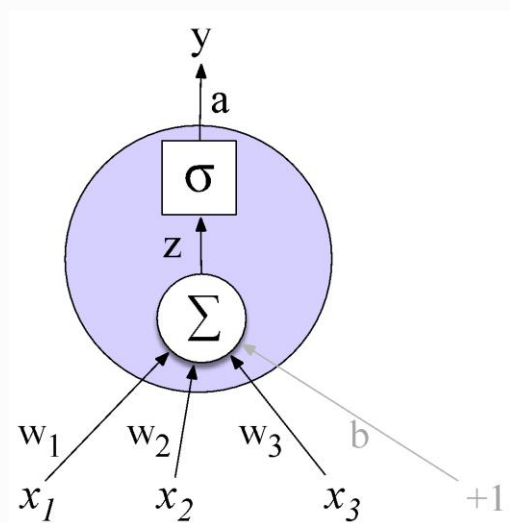
$$w_{t+1} = w_t - \eta \frac{d}{dw} L_{CE}(f(x; w), y)$$

- For logistic regression

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

# Where did that derivative come from?

Using the chain rule of derivatives!  $f(x) = u(v(x))$   $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

# How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

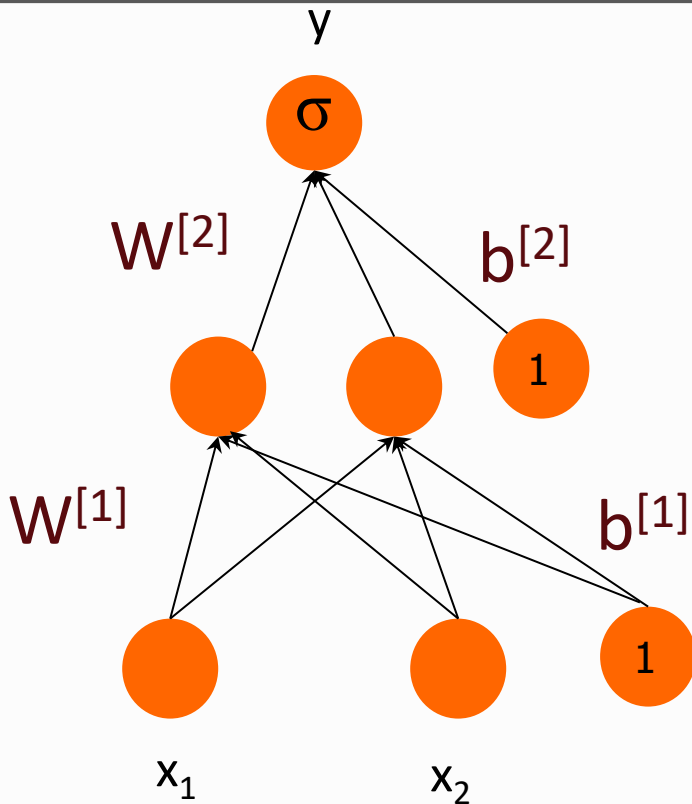
What about deeper networks? For training, we need the derivative of the loss with respect to each weight in every layer of the network

- Lots of layers, different activation functions?
- But the loss is computed only at the very end of the network!

Solution:

- Even more use of the chain rule!!
- This process is called **error backpropagation** (Rumelhart, Hinton, Williams, 1986)

# Backward differentiation on a two layer network



$$z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$

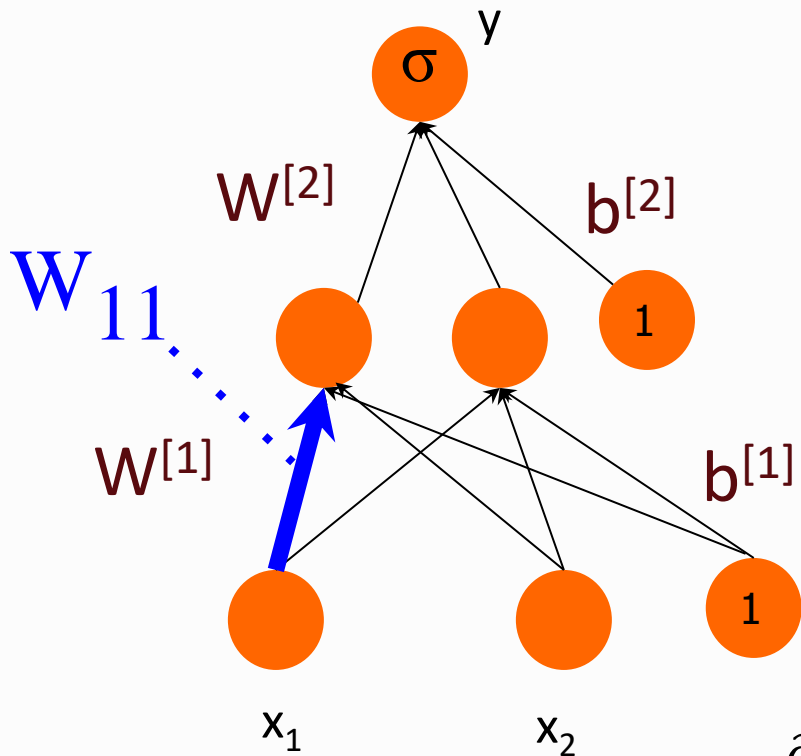
$$a^{[1]} = \text{ReLU}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

# Backward differentiation on a two layer network



$$z^{[1]} = W^{[1]} \mathbf{x} + b^{[1]}$$

$$a^{[1]} = \text{ReLU}(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

$$\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W_{11}^{[1]}}$$

# Summary

For training, we need the derivative of the loss with respect to weights in early layers of the network

- But loss is computed only at the very end of the network!

Solution: **backpropagation**

Given the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.



# Recurrent neural networks (RNNs)

---

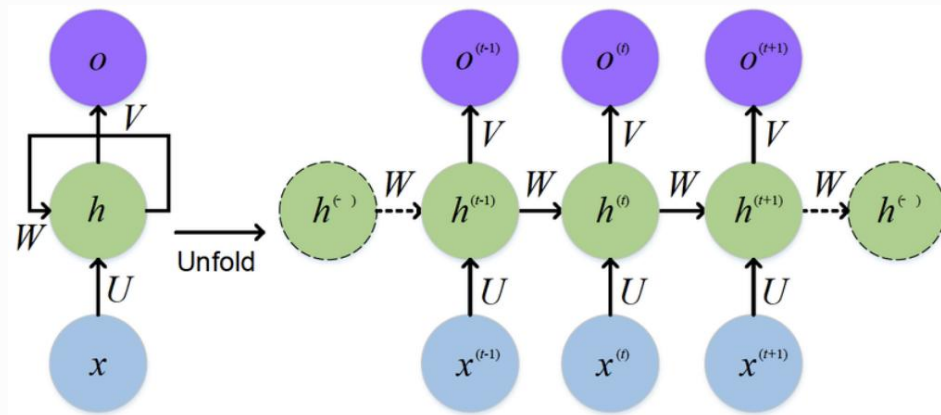
**FFNNs take an input of fixed dimensions**—a fixed number of features, a fixed number of tokens

The number tokens in a text—even a sentence—can be **arbitrarily large** (or short)

RNNs help us address this issue

# The architecture of an RNN

- Special kind of multilayer neural network for modeling sequences
- Hidden layers between the input and output receive input not just from the input layer, **but also from the hidden layer at a preceding timestep**
- RNNs can “remember” information from earlier on



# An RNN Language Model

## output distribution

$$\hat{y}^{(t)} = \text{softmax}(Uh^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

## hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

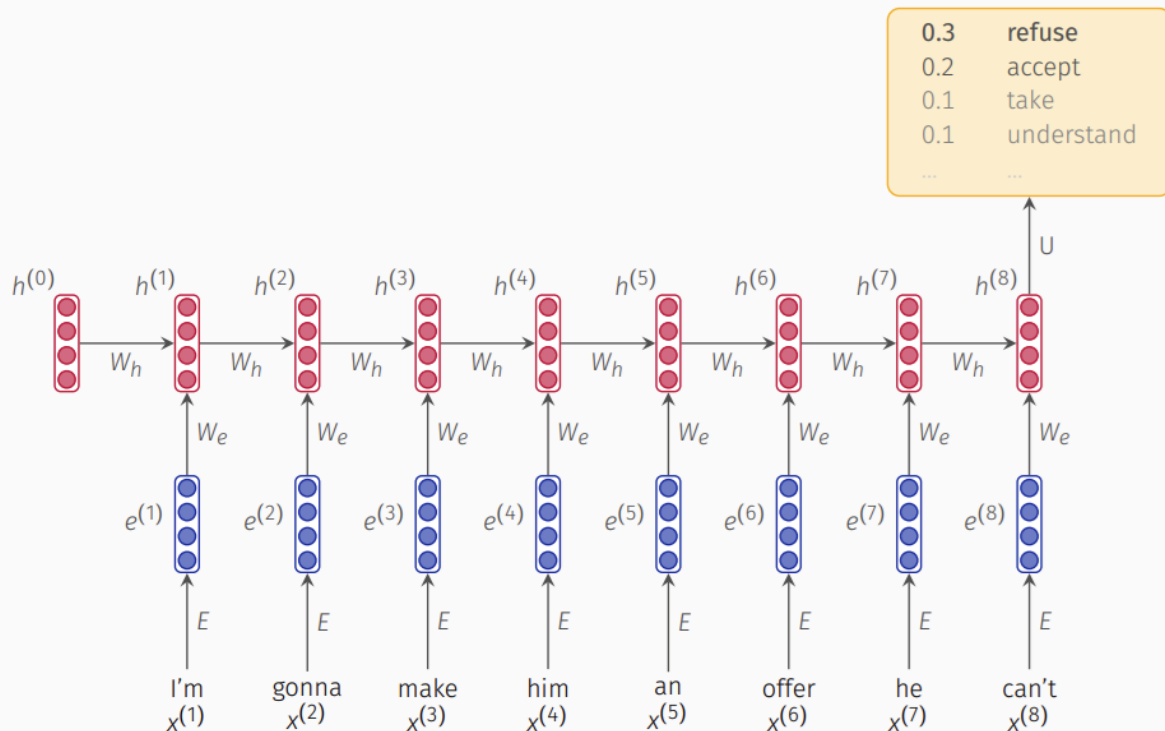
$h^{(0)}$  is the initial hidden state

## word embeddings

$$e^{(t)} = Ex^{(t)}$$

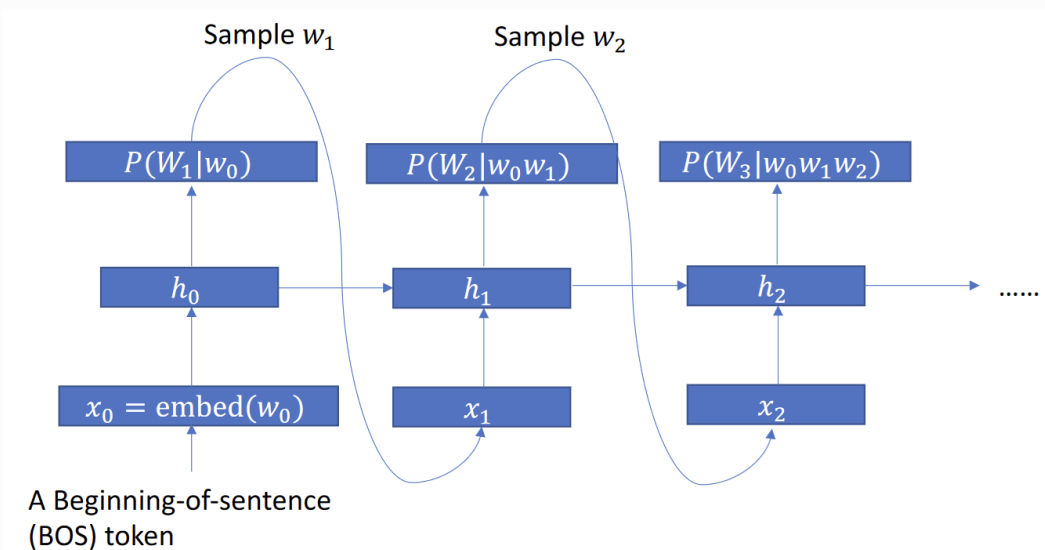
## one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



# Generation with RNN LMs

- At each time step  $t$ , we sample  $w_t$  from  $P(W_t | \dots)$ , and feed it to the next timestep!
- LM with this kind of generation process is called autoregressive LM



# Training an RNN Language Model

- Get a big corpus of text, which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$
- Feed it into the RNN-LM, computing output distribution  $y^{(t)}$  for every step  $t$ .
- Loss function on step  $t$  is **cross-entropy** between the predicted probability distribution  $\hat{y}^{(t)}$  and the true next word  $y^{(t)}$  (one-hot for  $x^{(t+1)}$ ):

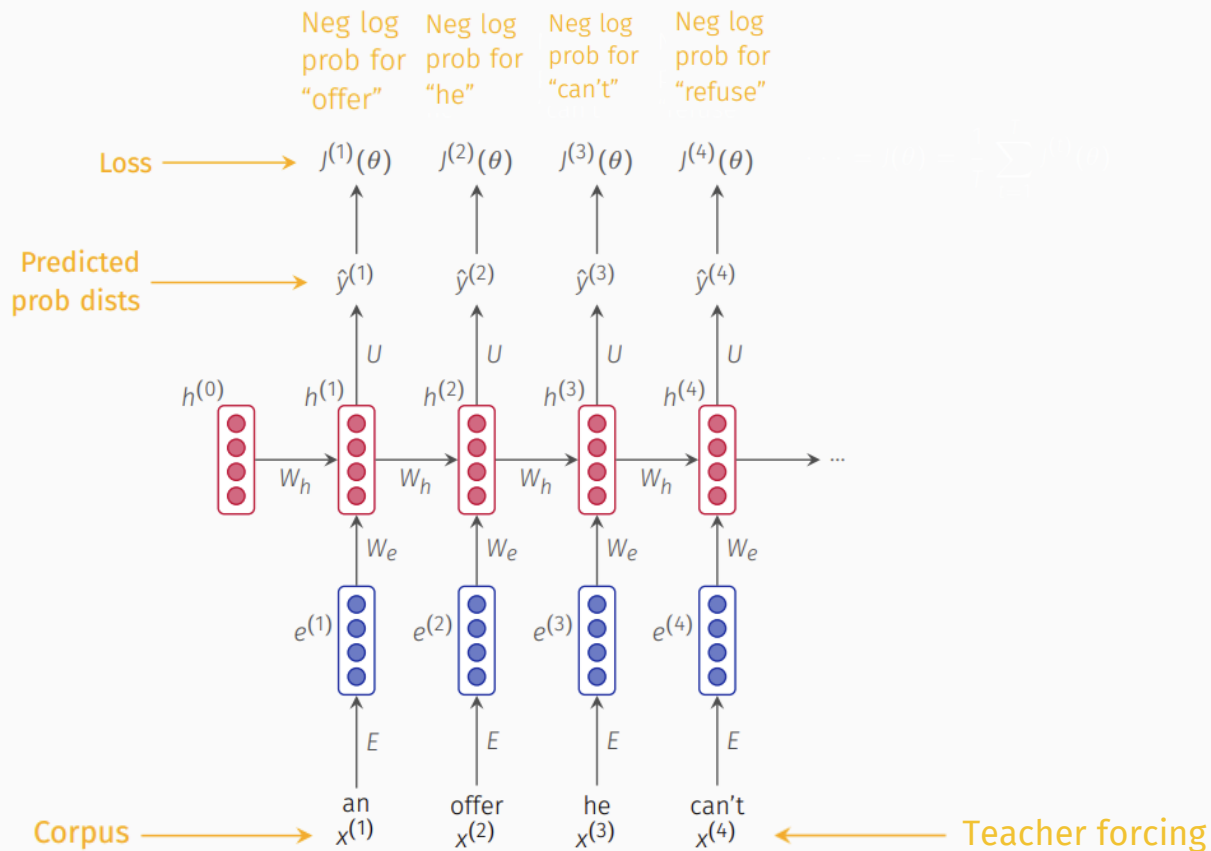
$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}$$

- Average this to get overall loss for the entire training set:

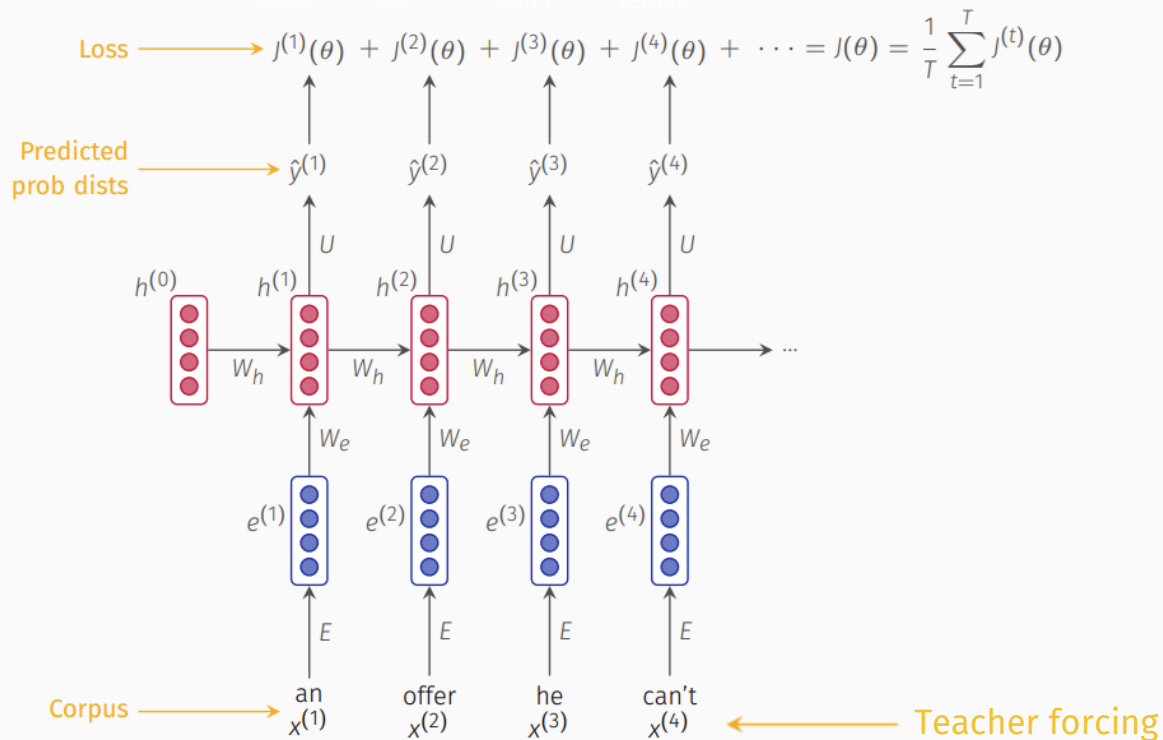
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \hat{y}_{x_{t+1}}^{(t)}$$



# Training an RNN Language Model



# Training an RNN Language Model



# Computing Loss and Gradients in Practice

- In principle, we could compute loss and gradients across the whole corpus  $(x^{(1)}, \dots, x^{(T)})$  but that would be incredibly expensive!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- Instead, we usually treat  $x^{(1)}, \dots, x^{(T)}$  as a document, or even a sentence
- This works much better with **Stochastic Gradient Descent**, which lets us compute loss and gradients for little chunks and update as we go.
- Actually, we do this in batches: compute  $J(\theta)$  for a batch of sentences; update weights; repeat.

## We Will Skip the Details of Backpropagation in RNNs for Now

- The fact that training RNNs involves backpropagation over timesteps, summing as you go, means that it (the **backpropagation through time** algorithm) is a bit more complicated than backpropagation in feedforward neural networks.
- We will skip these details for now, but you will want to learn them if you are doing serious work with RNNs.

# Review activity

---

# These concepts are confusing!

I don't expect you to understand the intuition of backpropagation or RNNs the first time around. With a group, decide which concept you are most comfortable with and which you find most confusing:

1. Neural network training and backpropagation
2. RNNs

Make a list of questions you have about the concept you are most confused about. Then find a group that is comfortable with the concept you find confusing. Maybe they can answer your questions! I am also available to answer questions.

# Project work time

---