CS 2731
Introduction to Natural Language Processing

Session 12: Neural networks part 2

Michael Miller Yoder

October 6, 2025

University of Pittsburgh | School of Computing and Information

Course logistics: quiz

- Quiz in class **this Wed Oct 8.** Readings to review:
  - Session 11: J+M 6-6.1, 6.3-6.5
  - Session 12: J+M 6.6, 13-13.2
  - You will have 10 minutes to complete the quiz (until 2:40pm)

Course logistics: homework and project

- Homework 2 is **due this Thu Oct 9**
- Next project deliverable: project proposal due **Oct 16**
    - Will include plans for **task, data, methods, evaluation**
    - Include example input and output
    - Literature review of at least 3 related papers
    - Feel free to email or book office hours with Michael to discuss
- We have $150 total as a class to use on OpenAI LLM credits
- Access to open-source LLM set up on School of Computing and Information servers for API access is coming soon
    - Gemma, LLaMa, Deepseek

Midterm course evaluation (OMETs)

- https://go.blueja.io/Iq36newH2UeDZRnTEA4pDg
- All types of feedback are welcome (critical and positive)
- Completely anonymous, will not affect grades
- Let me know what's working and what to improve on while the course is still running!
- Please be as specific as possible
- Available until **11:59pm today, Mon Oct 6**

Review: neural networks

Discuss with a neighbor:

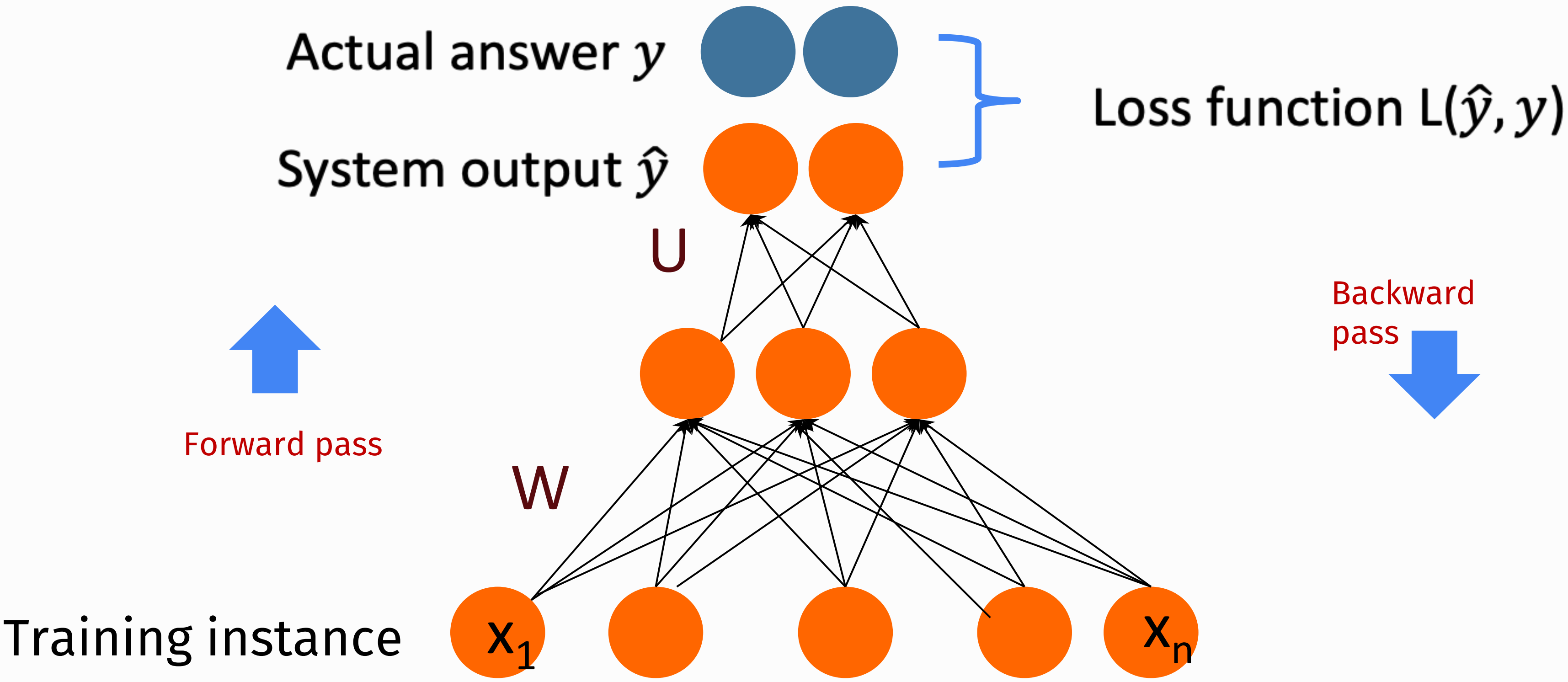
1. Describe the steps of computation that occurs at every level of a neural network
2. What is used as typical input to neural networks in NLP?

Lecture overview: neural networks part 2

- Training neural networks
- Recurrent neural networks (RNNs)
- Review activity

Training feedforward neural networks

Intuition: training a 2-layer Network



Slide adapted from Jurafsky & Martin

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

Slide credit: David Mortensen

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.**

Slide credit: David Mortensen

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

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

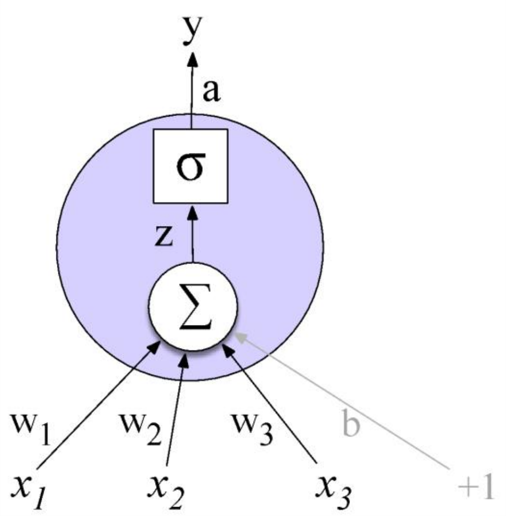
- For logistic regression

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

Slide adapted from Jurafsky & Martin

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

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

Slide adapted from Jurafsky & Martin

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

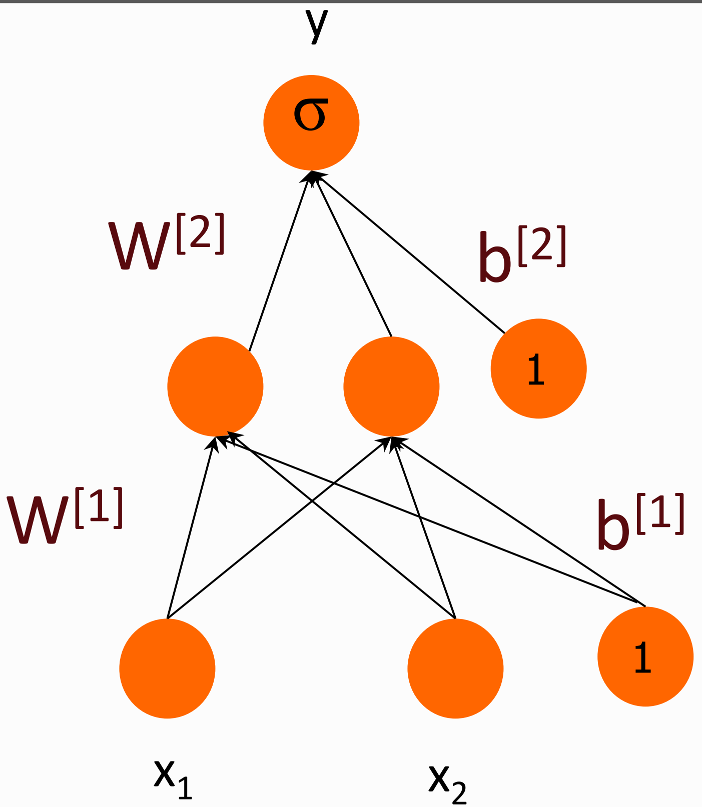
- Lots of layers, different activation functions?
- But the loss is computed only at the very end of the network!

Solution:

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

Slide adapted from Jurafsky & Martin

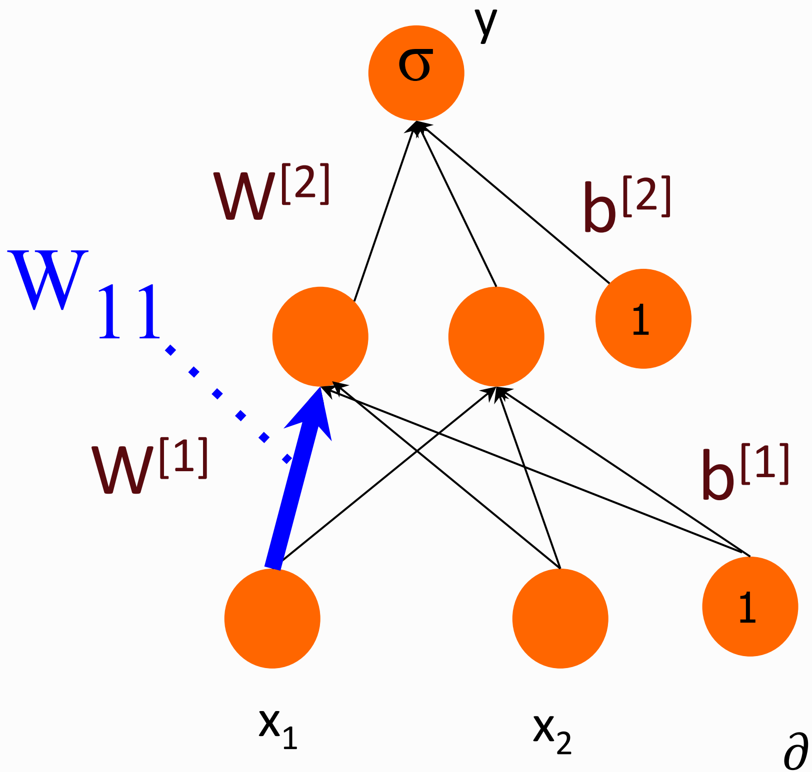
Backward differentiation on a two layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

Slide adapted from Jurafsky & Martin

Backward differentiation on a two layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

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

Slide adapted from Jurafsky & Martin

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

Slide credit: David Mortensen

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

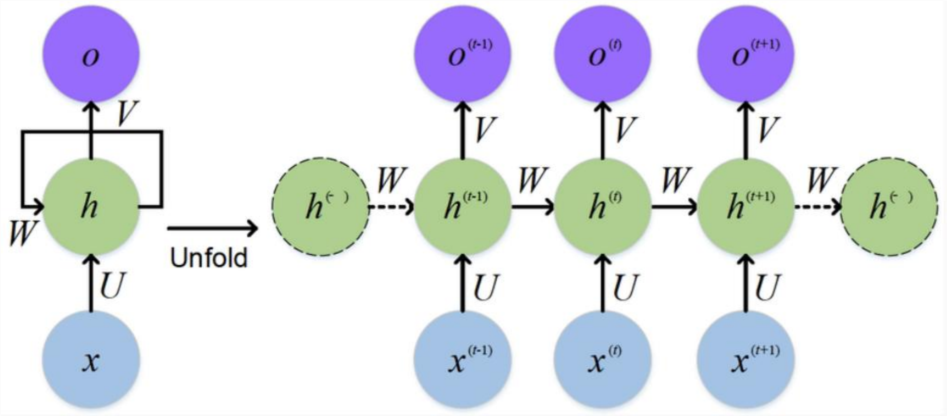
Slide credit: David Mortensen

RNNs help us address this issue

Slide credit: David Mortensen

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 form the input layer, **but also from the hidden layer at a preceding timestep**
- RNNs can “remember” information from earlier on



Slide credit: David Mortensen

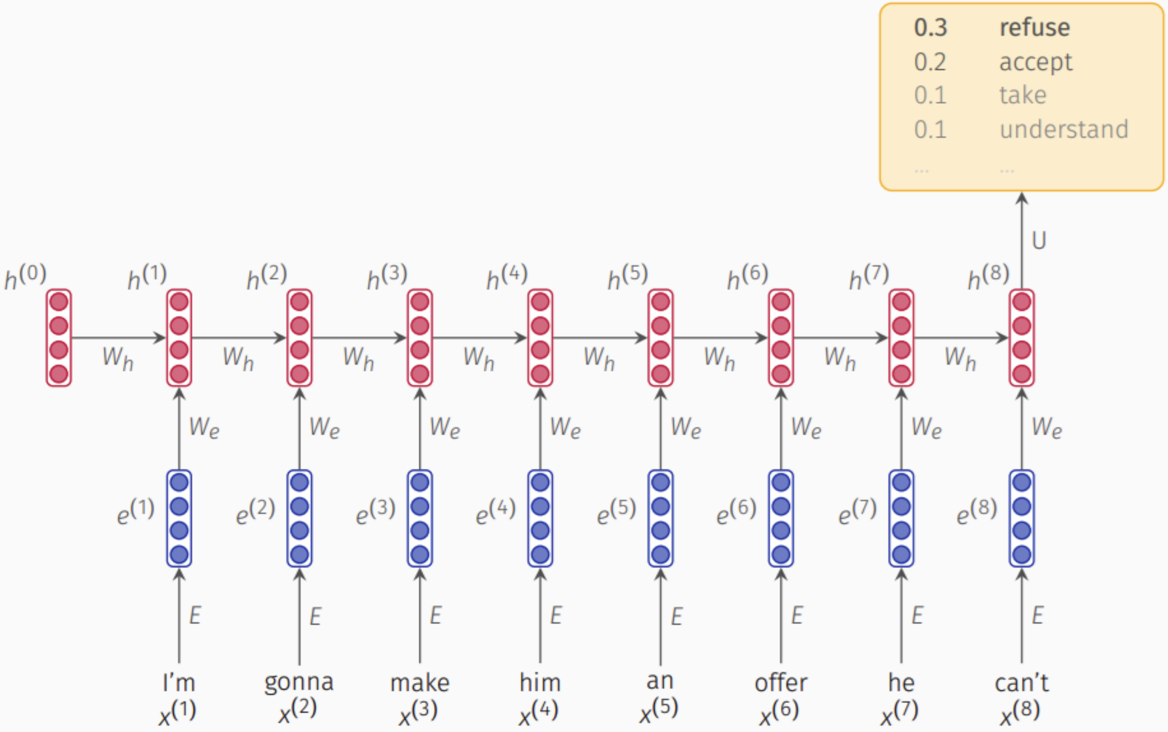
An RNN Language Model

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

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

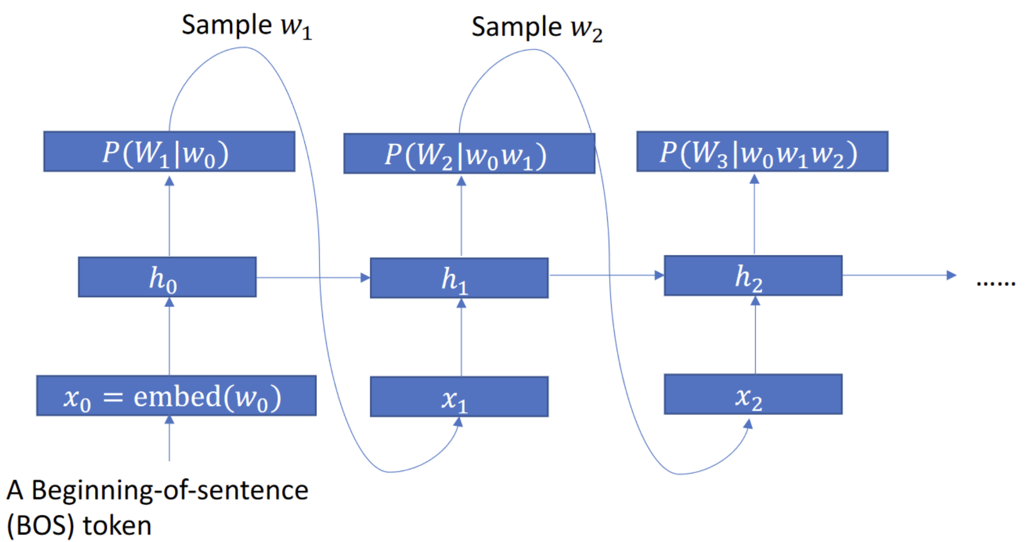
one-hot vectors
$x^{(t)} \in \mathbb{R}^{|V|}$



Slide credit: David Mortensen

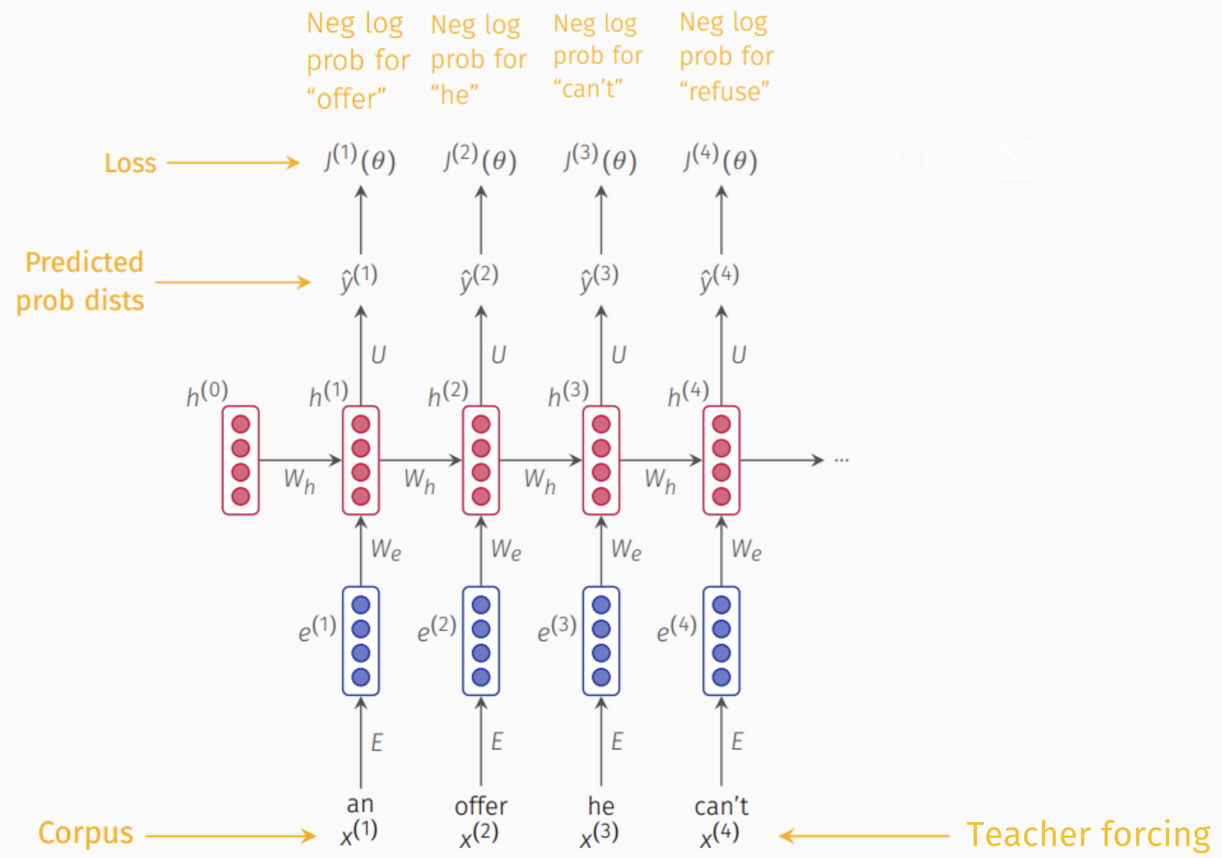
Generation with RNN LMs

- 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



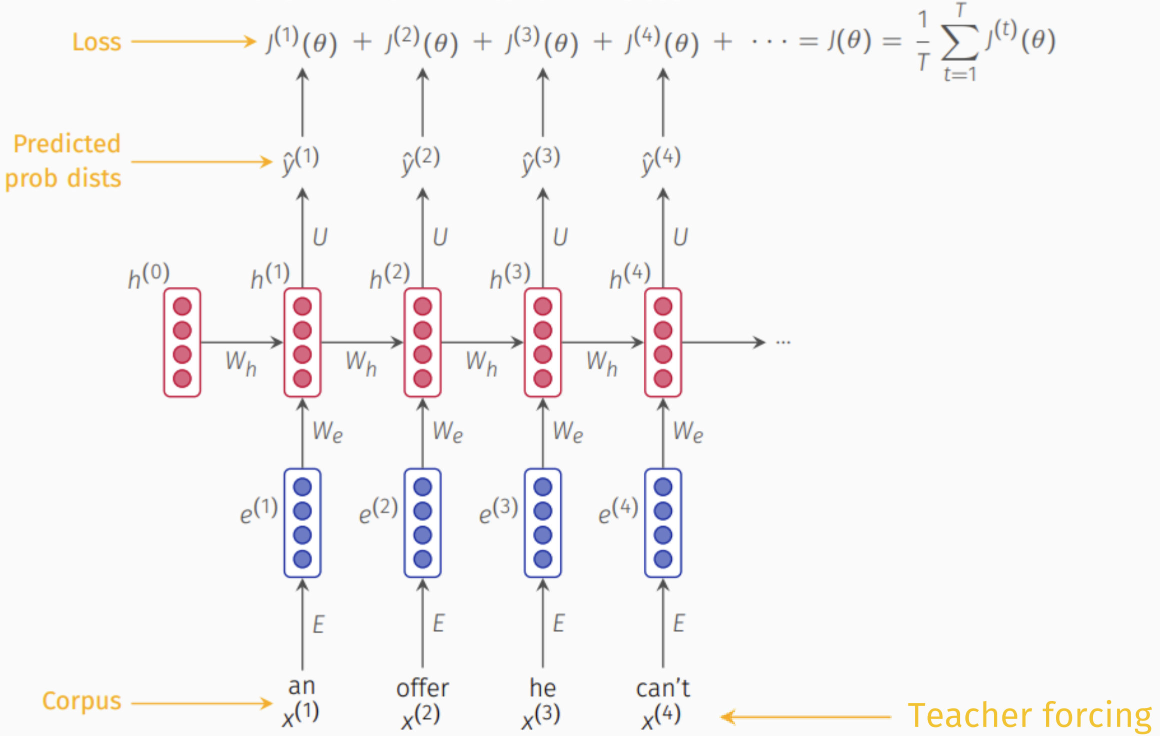
Slide adapted from Tianxing He

Training an RNN Language Model



Slide credit: David Mortensen

Training an RNN Language Model



Slide credit: David Mortensen

Training an RNN Language Model

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

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

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

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

Slide credit: David Mortensen

Computing Loss and Gradients in Practice

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

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

- Instead, we usually treat $x^{(1)}, \ldots, x^{(T)}$ 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.

Slide credit: David Mortensen

We Will Skip the Details of Backpropogation 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.

Slide credit: David Mortensen

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 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 might be comfortable with the concept you find confusing. Maybe they can answer your questions! I am also available to answer questions. The textbook is also a great resource!