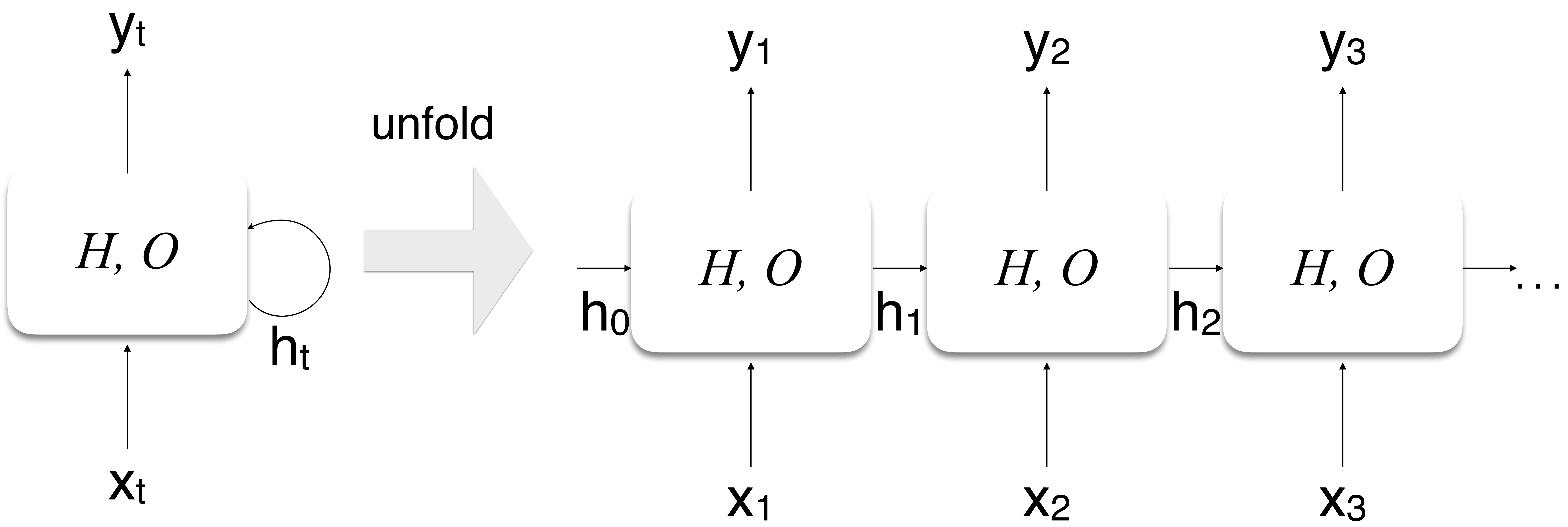
RNN-based AMs
+ Introduction to Language Modeling

Lecture 9

CS 753

Instructor: Preethi Jyothi

Recall RNN definition



Two main equations govern RNNs:

$$h_t = H(Wx_t + Vh_{t-1} + b^{(h)})$$

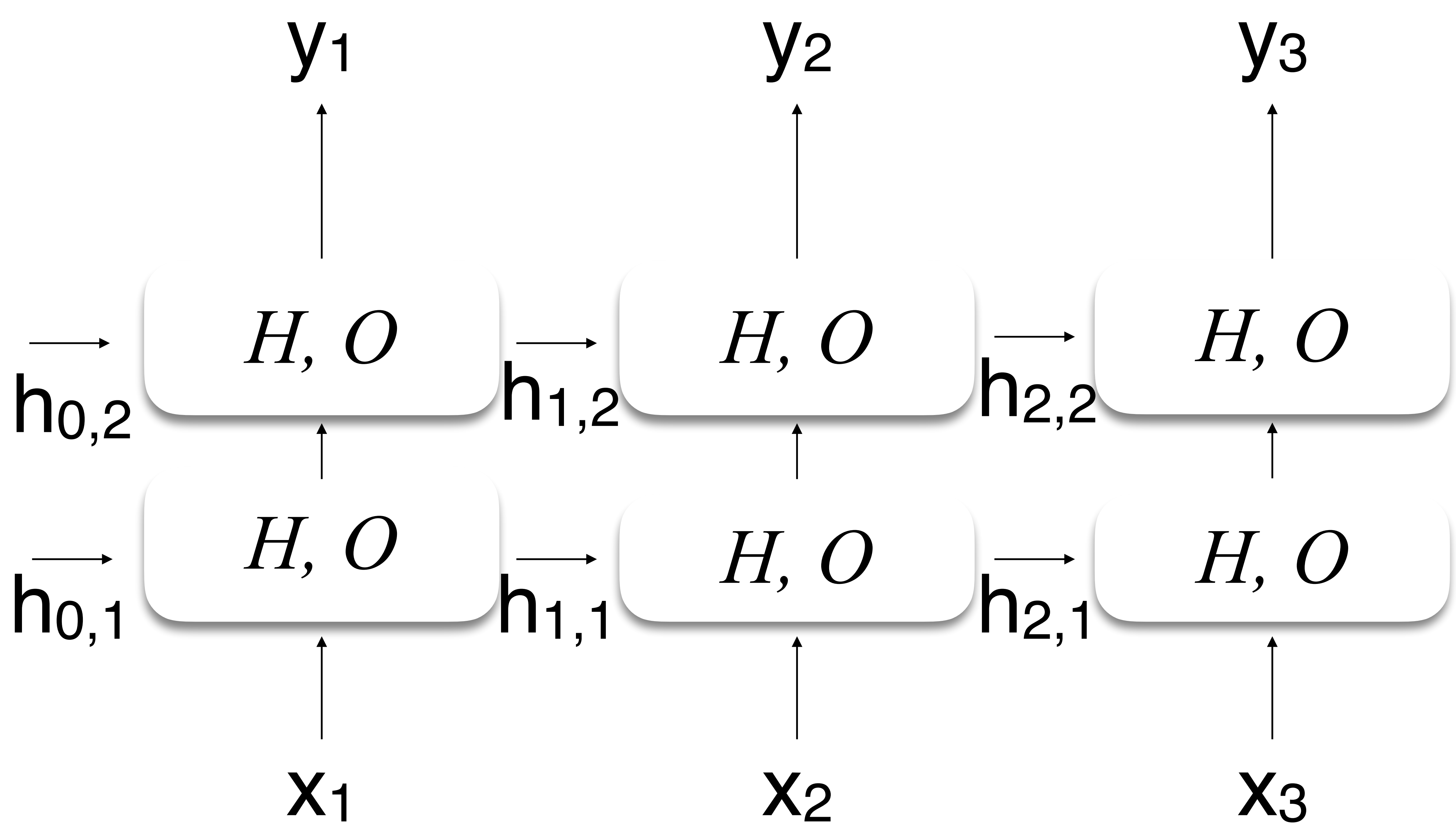
$$y_t = O(Uh_t + b^{(y)})$$

where W, V, U are matrices of input-hidden weights, hidden-hidden weights and hidden-output weights resp; $b^{(h)}$ and $b^{(y)}$ are bias vectors and H is the activation function applied to the hidden layer

Training RNNs

- An unrolled RNN is just a very deep feedforward network
- For a given input sequence:
  - create the unrolled network
  - add a loss function node to the network
  - then, use backpropagation to compute the gradients
- This algorithm is known as backpropagation through time (BPTT)

Deep RNNs



- RNNs can be stacked in layers to form deep RNNs
- Empirically shown to perform better than shallow RNNs on ASR [G13]

[G13] A. Graves, A . Mohamed, G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks”, ICASSP, 2013.

Vanilla RNN Model

$$h_t = H(Wx_t + Vh_{t-1} + b^{(h)})$$

$$y_t = O(Uh_t + b^{(y)})$$

H : element wise application of the sigmoid or tanh function

O : the softmax function

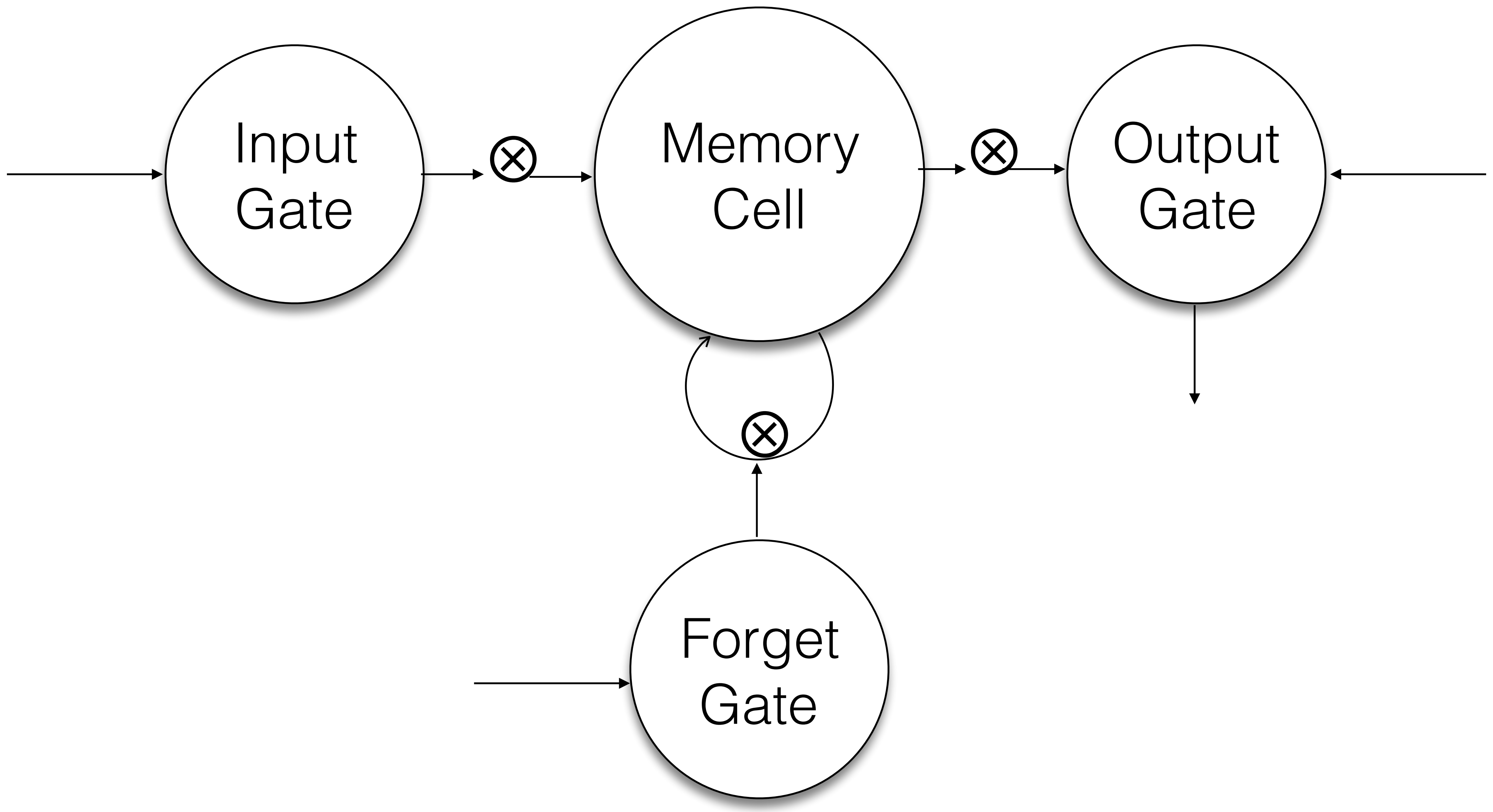
Run into problems of exploding and vanishing gradients.

Exploding/Vanishing Gradients

- In deep networks, gradients in early layers are computed as the product of terms from all the later layers
- This leads to unstable gradients:
  - If the terms in later layers are large enough, gradients in early layers (which is the product of these terms) can grow exponentially large: Exploding gradients
  - If the terms are in later layers are small, gradients in early layers will tend to exponentially decrease: Vanishing gradients
- To address this problem in RNNs, Long Short Term Memory (LSTM) units were proposed [HS97]

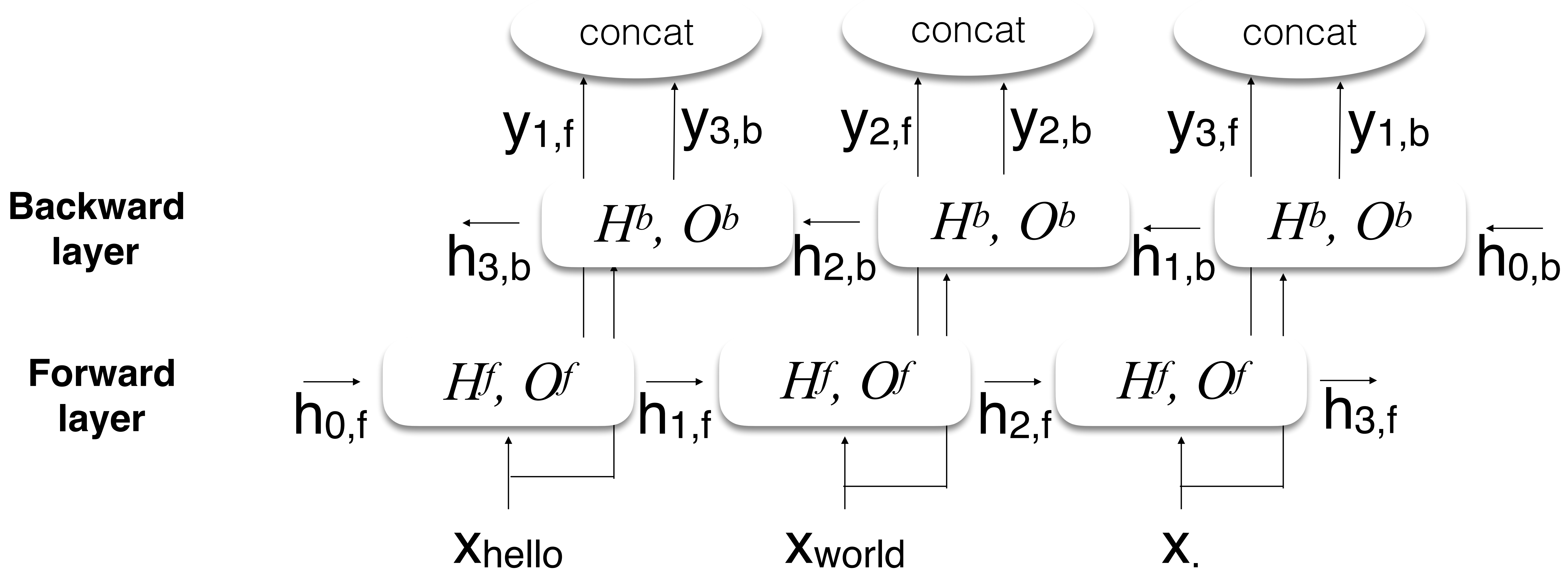
[HS97] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, 1997.

Long Short Term Memory Cells



- Memory cell: Neuron that stores information over long time periods
- Forget gate: When on, memory cell retains previous contents. Otherwise, memory cell forgets contents.
- When input gate is on, write into memory cell
- When output gate is on, read from the memory cell

Bidirectional RNNs



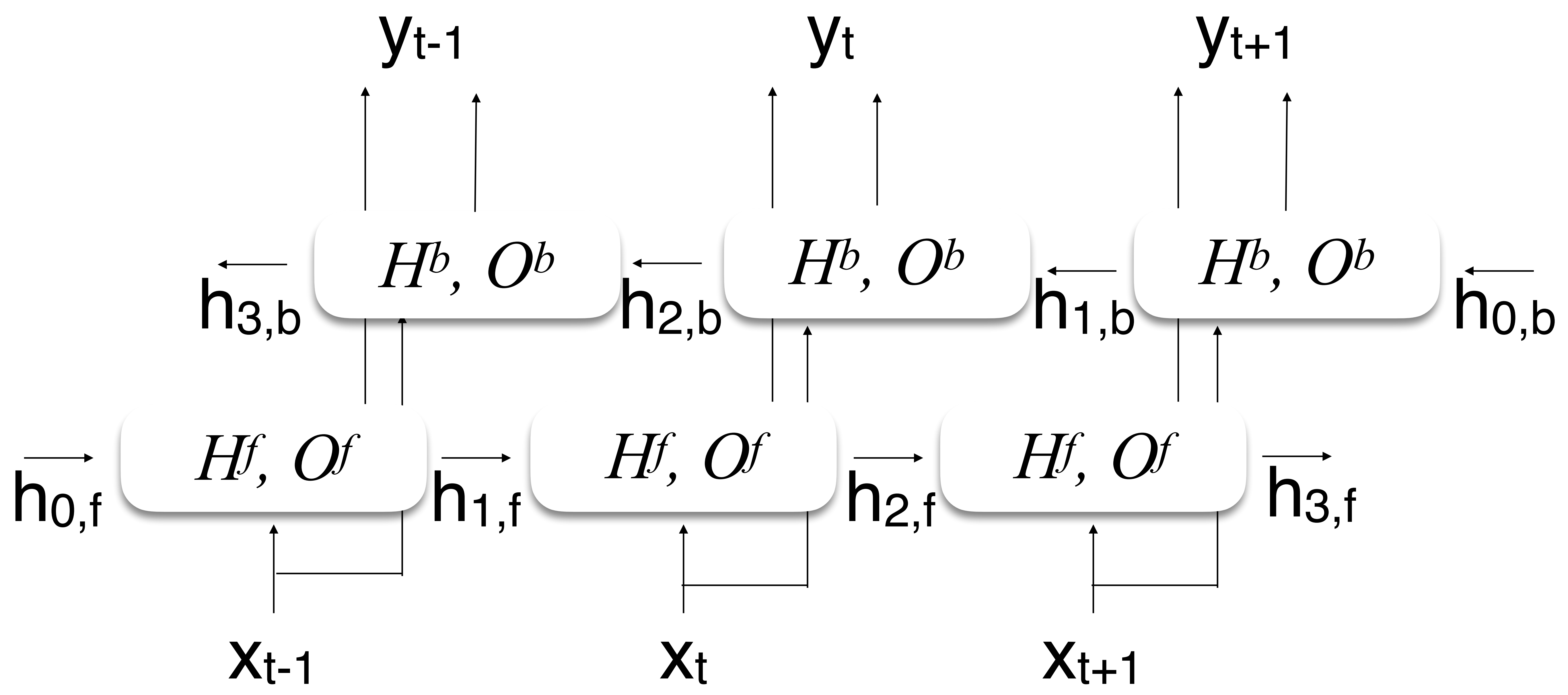
- BiRNNs process the data in both directions with two separate hidden layers
- Outputs from both hidden layers are concatenated at each position

ASR with RNNs

- We have seen how neural networks can be used for acoustic models in ASR systems
  - Main limitation: Frame-level training targets derived from HMM-based alignments
- Goal: Single RNN model that addresses this issues and does not rely on HMM-based alignments [G14]

[G14] A. Graves, N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks", ICML, 2014.

RNN-based Acoustic Model



- H was implemented using LSTMs in [G13]. Input: Acoustic feature vectors, one per frame; Output: Phones + space
- Deep bidirectional LSTM networks were used to do phone recognition on TIMIT
- Trained using the Connectionist Temporal Classification (CTC) loss [covered in later class]

[G13] A. Graves, et al., “Speech recognition with deep recurrent neural networks”, ICASSP, 2013.

RNN-based Acoustic Model

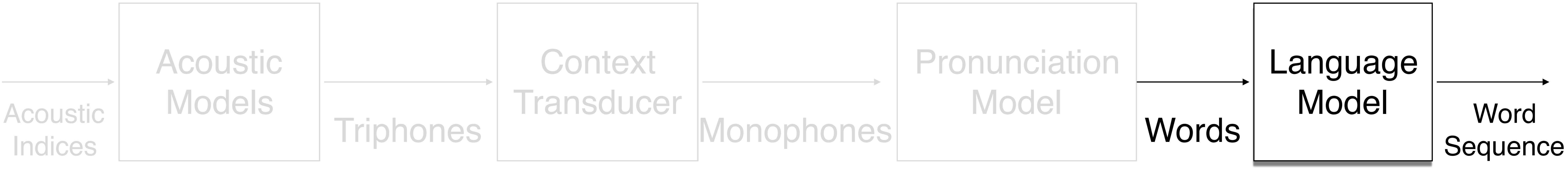
NETWORK	WEIGHTS	EPOCHS	PER
CTC-3L-500H-TANH	3.7M	107	37.6%
CTC-1L-250H	0.8M	82	23.9%
CTC-1L-622H	3.8M	87	23.0%
CTC-2L-250H	2.3M	55	21.0%
CTC-3L-421H-UNI	3.8M	115	19.6%
CTC-3L-250H	3.8M	124	18.6%
CTC-5L-250H	6.8M	150	18.4%

TIMIT phoneme recognition results

[G13] A. Graves, et al., "Speech recognition with deep recurrent neural networks", ICASSP, 2013.

So far, we've looked at acoustic models…

Next, language models



- Language models
  - provide information about word reordering

    Pr("she class taught a") < Pr("she taught a class")

  - provide information about the most likely next word

    Pr("she taught a class") > Pr("she taught a speech")

Application of language models

- Speech recognition
  - $\Pr$("she taught a class") > $\Pr$("sheet or tuck lass")
- Machine translation
- Handwriting recognition/Optical character recognition
- Spelling correction of sentences
- Summarization, dialog generation, information retrieval, etc.

Popular Language Modelling Toolkits

- SRILM Toolkit:

  http://www.speech.sri.com/projects/srilm/

- KenLM Toolkit:

  https://kheafield.com/code/kenlm/

- OpenGrm NGram Library:

  http://opengrm.org/

Introduction to probabilistic LMs

Probabilistic or Statistical Language Models

- Given a word sequence, $W = \{w_1, \ldots, w_n\}$, what is $\Pr(W)$?
- Decompose $\Pr(W)$ using the chain rule:

$$\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) = \Pr(w_1)\,\Pr(w_2|w_1)\,\Pr(w_3|w_1, w_2) \ldots \Pr(w_n|w_1, \ldots, w_{n-1})$$

- Sparse data with long word contexts: How do we estimate the probabilities $\Pr(w_n|w_1, \ldots, w_{n-1})$?

Estimating word probabilities

- Accumulate counts of words and word contexts
- Compute normalised counts to get next-word probabilities
- E.g. $\Pr(\text{“class | she taught a”})$

$$= \frac{\pi(\text{“she taught a class”})}{\pi(\text{“she taught a”})}$$

where π(“…”) refers to counts derived from a large English text corpus

- What is the obvious limitation here? We’ll never see enough data

Simplifying Markov Assumption

- Markov chain:
  - Limited memory of previous word history: Only last m words are included

- 1-order language model (or bigram model)

$$\Pr(w_1,w_2,\ldots,w_{n-1},w_n) \cong \Pr(w_1|<s>)\ \Pr(w_2|w_1)\ \Pr(w_3|w_2)\ldots\Pr(w_n|w_{n-1})$$

- 2-order language model (or trigram model)

$$\Pr(w_1,w_2,\ldots,w_{n-1},w_n) \cong \Pr(w_2|w_{1,}<s>)\ \Pr(w_3|w_1,w_2)\ldots\Pr(w_n|w_{n-2},w_{n-1})$$

- Ngram model is an N-1th order Markov model

Estimating Ngram Probabilities

- Maximum Likelihood Estimates
  - Unigram model

$$\Pr_{ML}(w_1) = \frac{\pi(w_1)}{\sum_i \pi(w_i)}$$

  - Bigram model

$$\Pr_{ML}(w_2|w_1) = \frac{\pi(w_1, w_2)}{\sum_i \pi(w_1, w_i)}$$

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The cat chased a mouse") using a bigram model?

Pr("<s> The cat chased a mouse </s>") =

Pr("The|<s>") · Pr("cat|The") · Pr("chased|cat") · Pr("a|chased") · Pr("mouse|a") · Pr("</s>|mouse") =

3/3 · 1/3 · 1/2 · 1/2 · 1/2 · 1/2 = 1/48

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The dog eats cheese") using a bigram model?

Pr("<s> The dog eats cheese </s>") =

Pr("The|<s>") · Pr("dog|The") · Pr("eats|dog") · Pr("cheese|eats") · Pr("</s>|cheese") =

3/3 · 1/3 · 0/1 · 1/1 · 1/1 = 0! Due to unseen bigrams

How do we deal with unseen bigrams? We'll come back to it.

Open vs. closed vocabulary task

- Closed vocabulary task: Use a fixed vocabulary, V. We know all the words in advance.
- More realistic setting, we don't know all the words in advance. Open vocabulary task. Encounter out-of-vocabulary (OOV) words during test time.
- Create an unknown word: <UNK>
  - Estimating <UNK> probabilities: Determine a vocabulary V. Change all words in the training set not in V to <UNK>
  - Now train its probabilities like a regular word
  - At test time, use <UNK> probabilities for words not in training

Evaluating Language Models

- Extrinsic evaluation:
  - To compare Ngram models A and B, use both within a specific speech recognition system (keeping all other components the same)
  - Compare word error rates (WERs) for A and B
  - Time-consuming process!

Intrinsic Evaluation

- Evaluate the language model in a standalone manner
- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
    - Same measure can be used to address both questions — perplexity!

Measures of LM quality

- How likely does the model consider the text in a test set?

- How closely does the model approximate the actual (test set) distribution?

  - Same measure can be used to address both questions — perplexity!

Perplexity (I)

- How likely does the model consider the text in a test set?
  - Perplexity(test) = $1/\Pr_{model}[\text{text}]$
  - Normalized by text length:
    - Perplexity(test) = $(1/\Pr_{model}[\text{text}])^{1/N}$ where N = number of tokens in test
    - e.g. If model predicts i.i.d. words from a dictionary of size L, per word perplexity = $(1/(1/L)^N)^{1/N} = L$

Intuition for Perplexity

- Shannon's guessing game builds intuition for perplexity
  - What is the surprisal factor in predicting the next word?
    - At the stall, I had tea and _________

biscuits	0.1
samosa	0.1
coffee	0.01
rice	0.001
⋮	
but	0.00000000001

- A better language model would assign a higher probability to the actual word that fills the blank (and hence lead to lesser surprisal/perplexity)

Measures of LM quality

- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
  - Same measure can be used to address both questions — perplexity!

Perplexity (II)

- How closely does the model approximate the actual (test set) distribution?
  - KL-divergence between two distributions X and Y
    $$D_{KL}(X||Y) = \Sigma_\sigma Pr_X[\sigma] \log (Pr_X[\sigma]/Pr_Y[\sigma])$$
  - Equals zero iff X = Y ; Otherwise, positive

- How to measure $D_{KL}(X||Y)$? We don't know X!

  Cross entropy between X and Y

  - $D_{KL}(X||Y) = \boxed{\Sigma_\sigma Pr_X[\sigma] \log(1/Pr_Y[\sigma])} - H(X)$
    where $H(X) = -\Sigma_\sigma Pr_X[\sigma] \log Pr_X[\sigma]$
  - Empirical cross entropy:

$$\frac{1}{|test|} \sum_{\sigma \in test} \log(\frac{1}{\Pr_y[\sigma]})$$

Perplexity vs. Empirical Cross Entropy

- Empirical Cross Entropy (ECE)

$$\frac{1}{|\#sents|} \sum_{\sigma \in test} \log(\frac{1}{\Pr_{model}[\sigma]})$$

- Normalized Empirical Cross Entropy = ECE/(avg. length) =

$$\frac{1}{|\#words/\#sents|} \frac{1}{|\#sents|} \sum_{\sigma \in test} \log(\frac{1}{\Pr_{model}[\sigma]})$$

$$= \frac{1}{N} \sum_{\sigma} \log(\frac{1}{\Pr_{\mathrm{model}}[\sigma]})$$

where N = *#words*

- How does $\frac{1}{N} \sum_{\sigma} \log(\frac{1}{\Pr_{model}[\sigma]})$ relate to perplexity?

Perplexity vs. Empirical Cross Entropy

$$\begin{aligned}\log(\text{perplexity}) &= \frac{1}{N}\log\frac{1}{\Pr[test]} \\ &= \frac{1}{N}\log\prod_{\sigma}(\frac{1}{\Pr_{model}[\sigma]}) \\ &= \frac{1}{N}\sum_{\sigma}\log(\frac{1}{\Pr_{model}[\sigma]})\end{aligned}$$

Thus, perplexity = exp$^{\text{(normalized cross entropy)}}$

Example perplexities for Ngram models trained on WSJ (80M words):

Unigram: 962, Bigram: 170, Trigram: 109

Introduction to smoothing of LMs

Recall example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is Pr("The dog eats cheese")?

Pr("<s> The dog eats cheese </s>") =

Pr("The|<s>") · Pr("dog|The") · Pr("eats|dog") · Pr("cheese|eats") · Pr("</s>|cheese") =

3/3 · 1/3 · 0/1 · 1/1 · 1/1 = 0! Due to unseen bigrams

Unseen Ngrams

- Even with MLE estimates based on counts from large text corpora, there will be many unseen bigrams/trigrams that never appear in the corpus
- If any unseen Ngram appears in a test sentence, the sentence will be assigned probability 0
- Problem with MLE estimates: maximises the likelihood of the observed data by assuming anything unseen cannot happen and overfits to the training data
  - Smoothing methods: Reserve some probability mass to Ngrams that don’t occur in the training corpus

Add-one (Laplace) smoothing

Simple idea: Add one to all bigram counts. That means,

$$\Pr_{ML}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

becomes

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V} \quad ✓$$

where V is the vocabulary size