Neural network language models

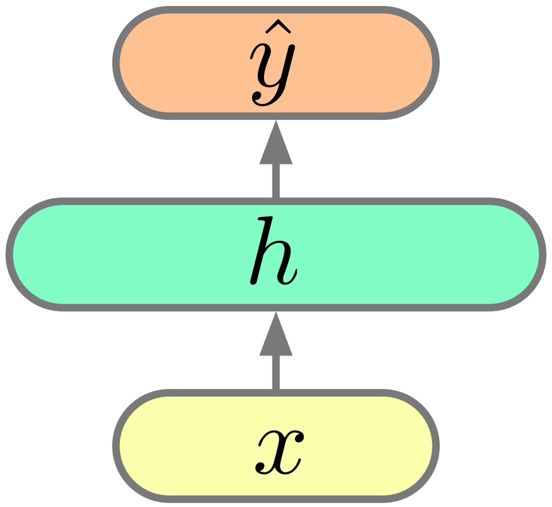
Lecture, Feb 16
CS 690N, Spring 2017
Advanced Natural Language Processing
http://people.cs.umass.edu/~brenocon/anlp2017/

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

Neural Language Models

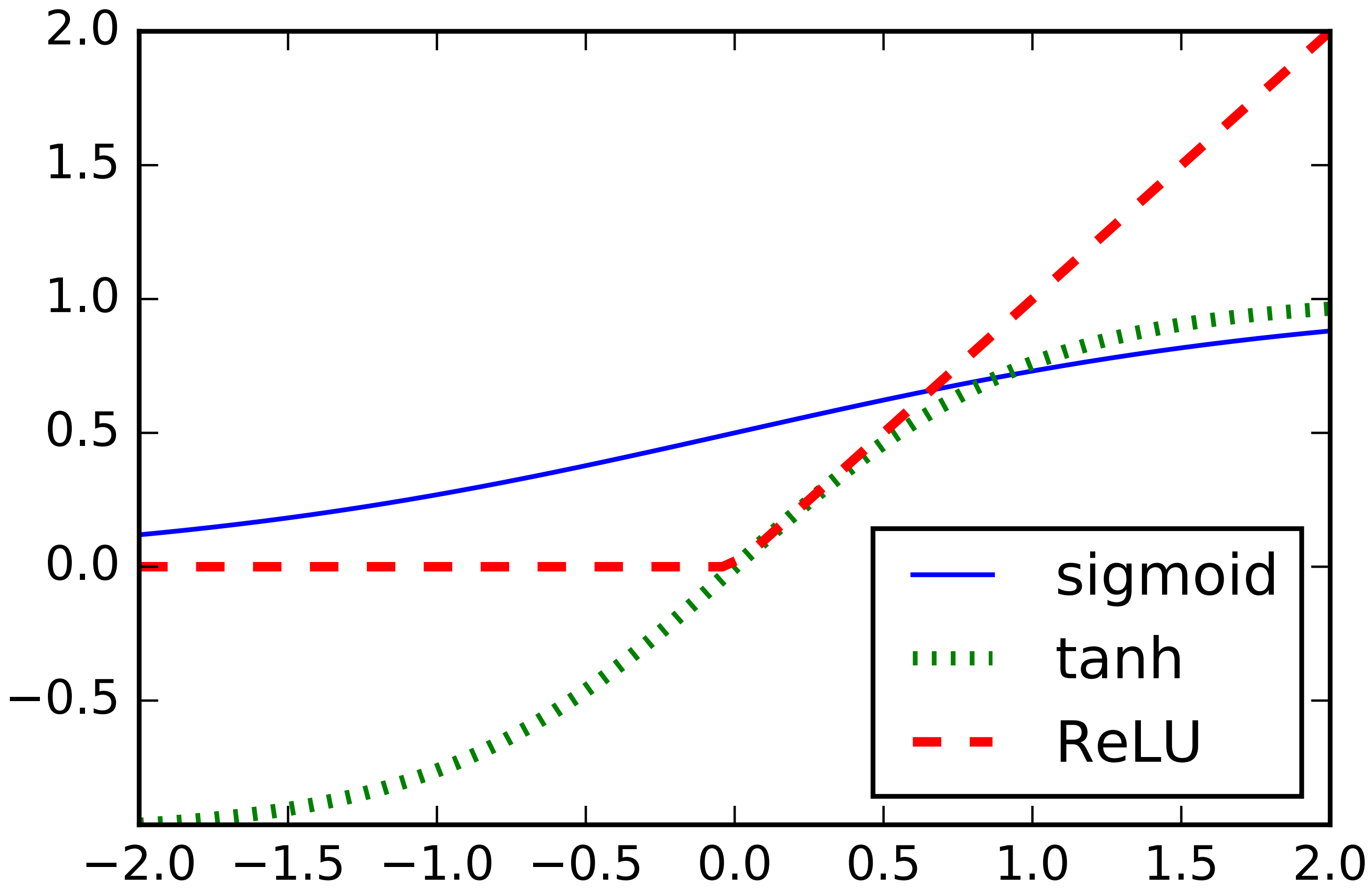
Feed forward network

$$
\begin{aligned}
h &= g(Vx + c) \\
\hat{y} &= Wh + b
\end{aligned}
$$



[Slide: Phil Blunsom]

Nonlinear activation functions



$$\text{sigmoid}(x) = \frac{e^x}{1 + e^x}$$

$$\tanh(x) = 2 \times \text{sgm}(x) - 1$$

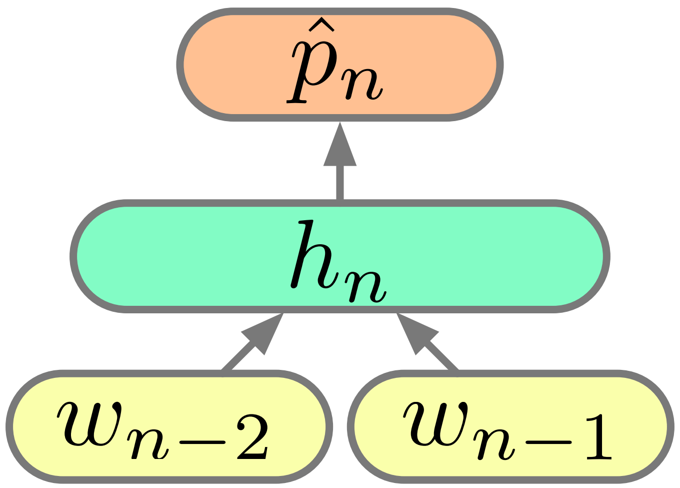
$$(x)_+ = \max(0, x)$$

a.k.a. "ReLU"

Trigram NN language model

Word embeddings

$$\begin{aligned} h_n &= g(V[w_{n-1}; w_{n-2}] + c) \\ \hat{p}_n &= \mathrm{softmax}(W h_n + b) \\ \mathrm{softmax}(u)_i &= \frac{\exp u_i}{\sum_j \exp u_j} \end{aligned}$$

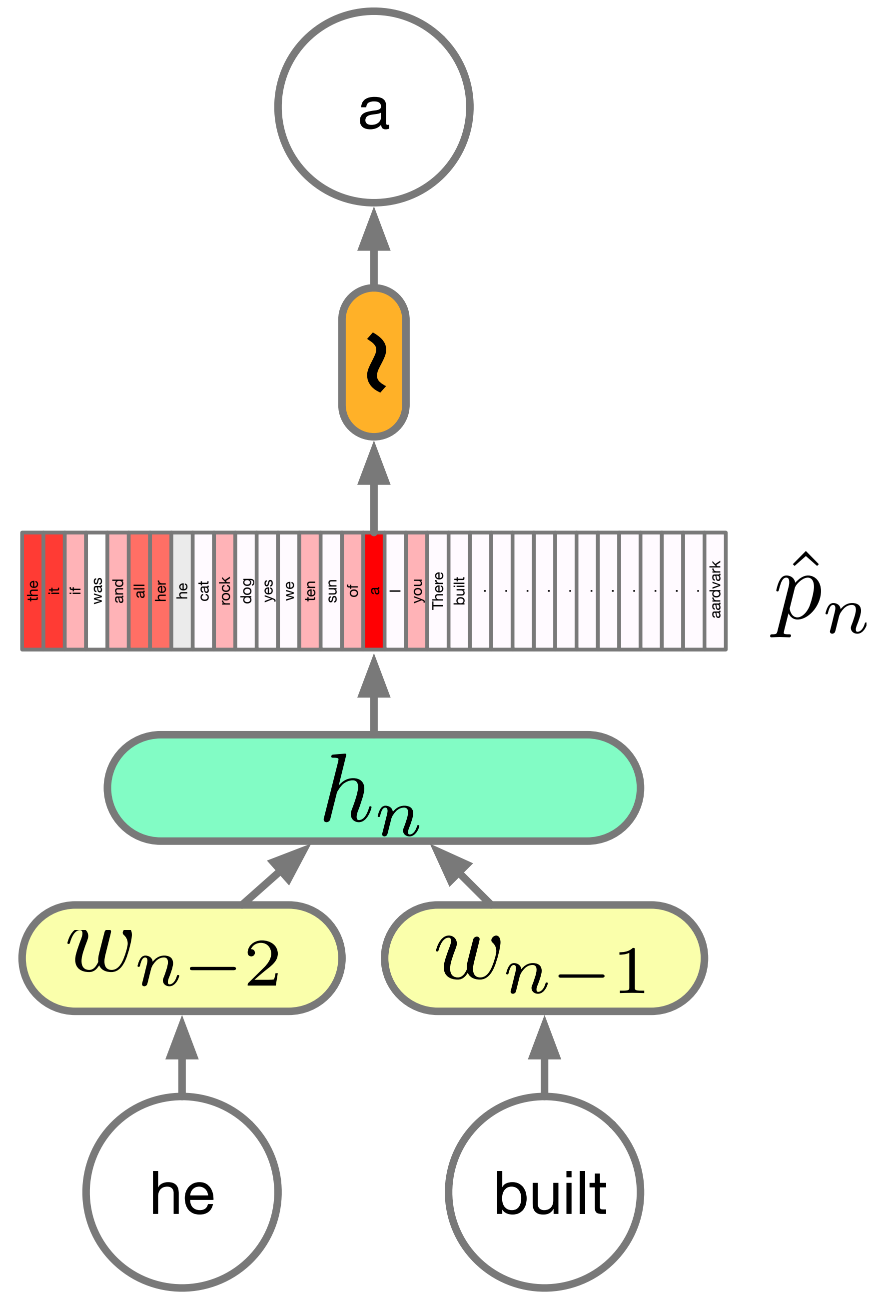


- w_i are one hot vetors and $\hat{p}_i$ are distributions,
- $|w_i| = |\hat{p}_i| = V$ (words in the vocabulary),
- V is usually very large $> 1e5$.

[Slide: Phil Blunsom]

Neural Language Models: Sampling

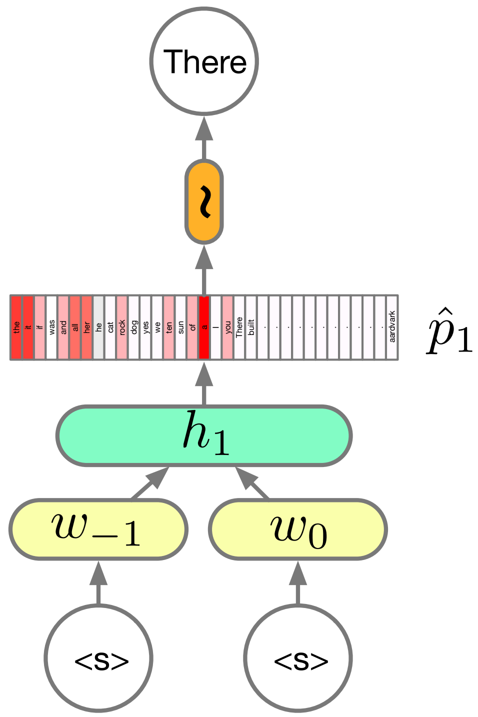
$$w_n | w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

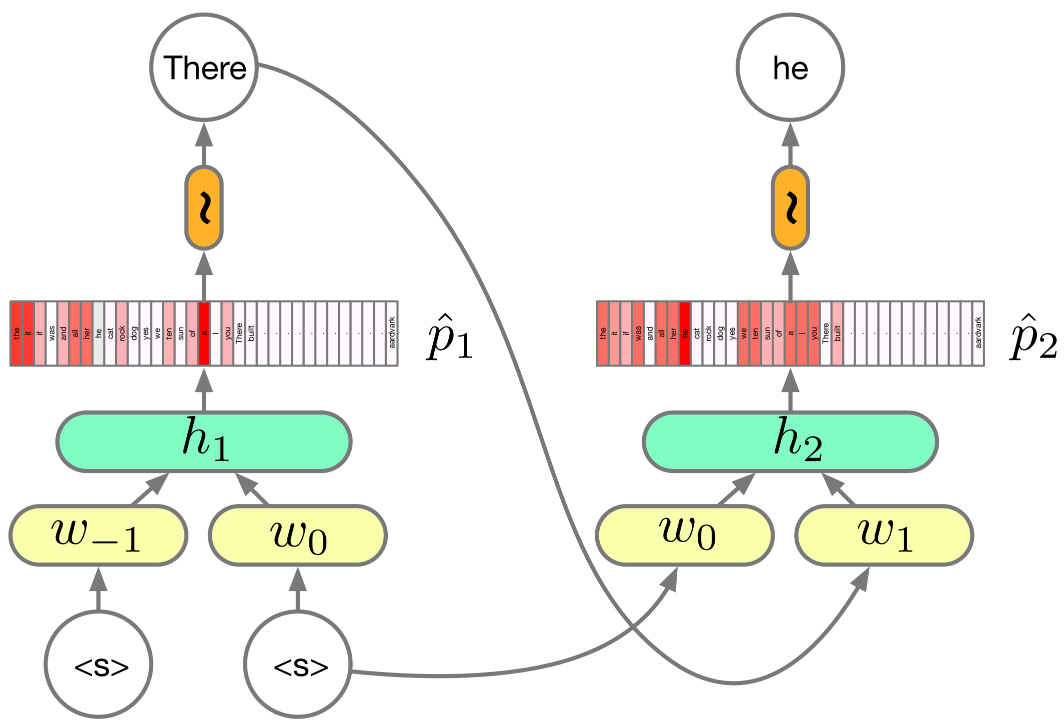
$$w_n | w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

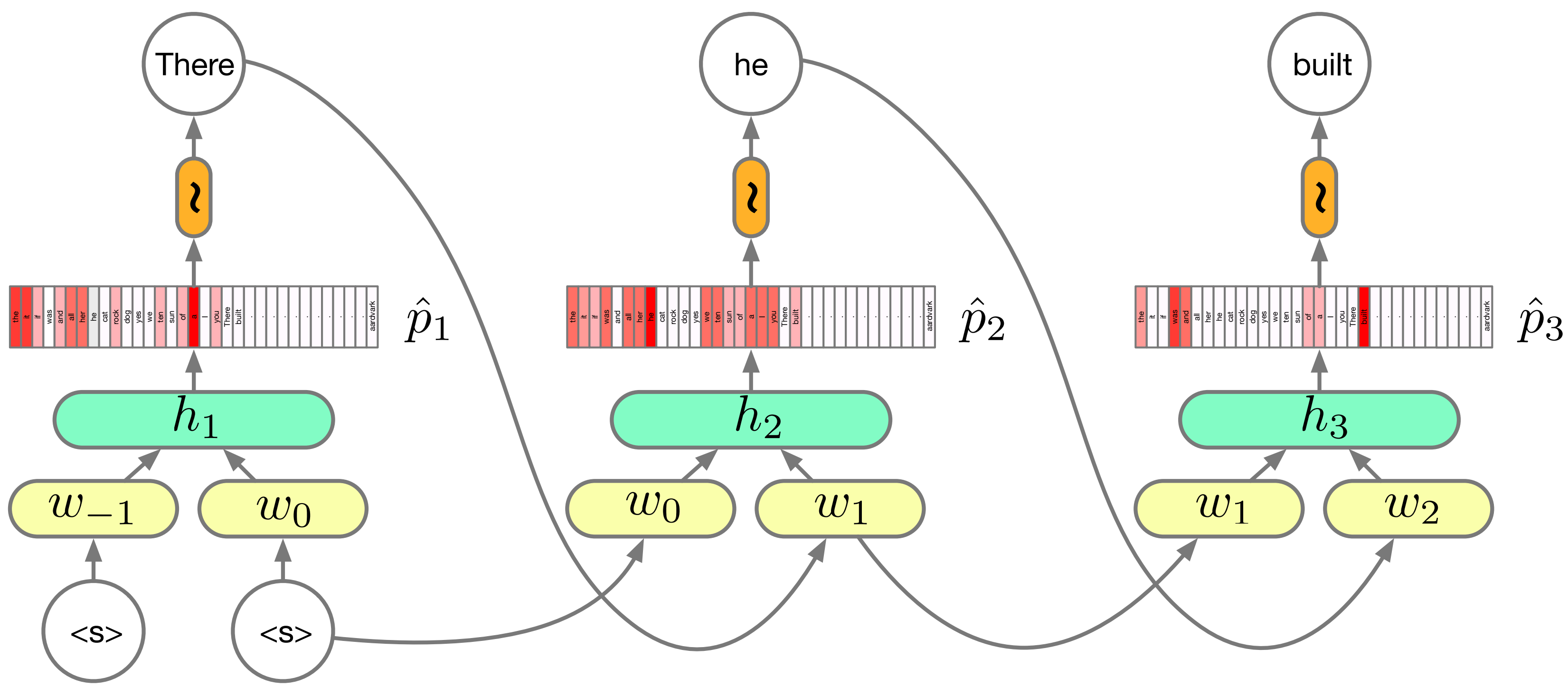
$$w_n \big| w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

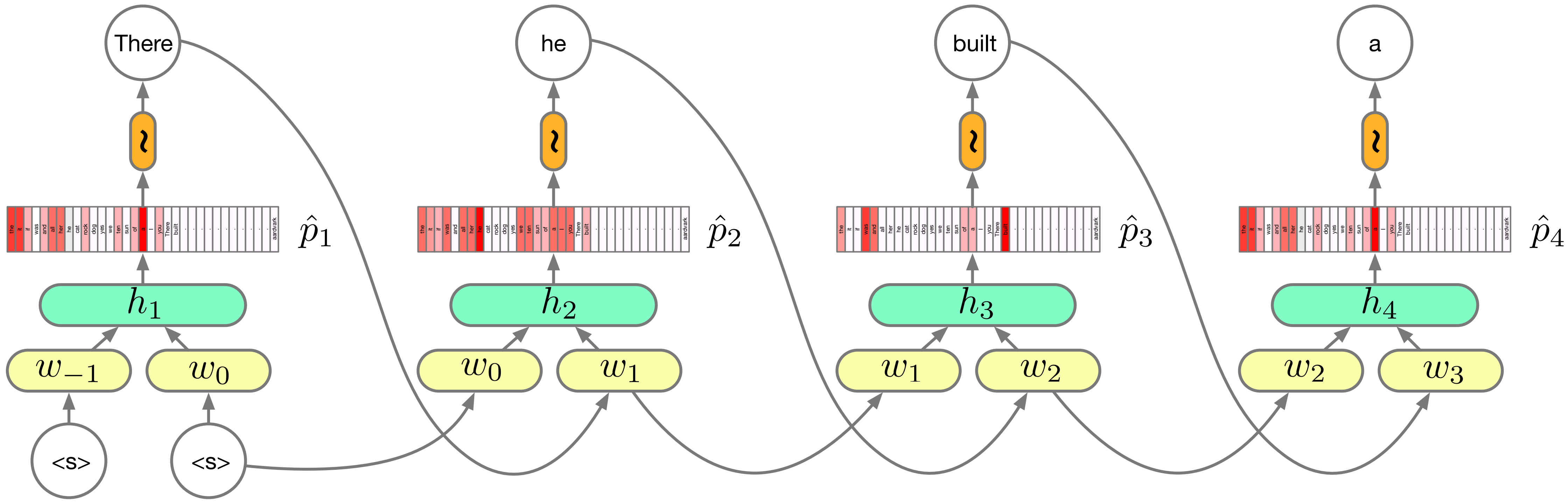
$$w_n \big| w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

$$w_n \mid w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Training

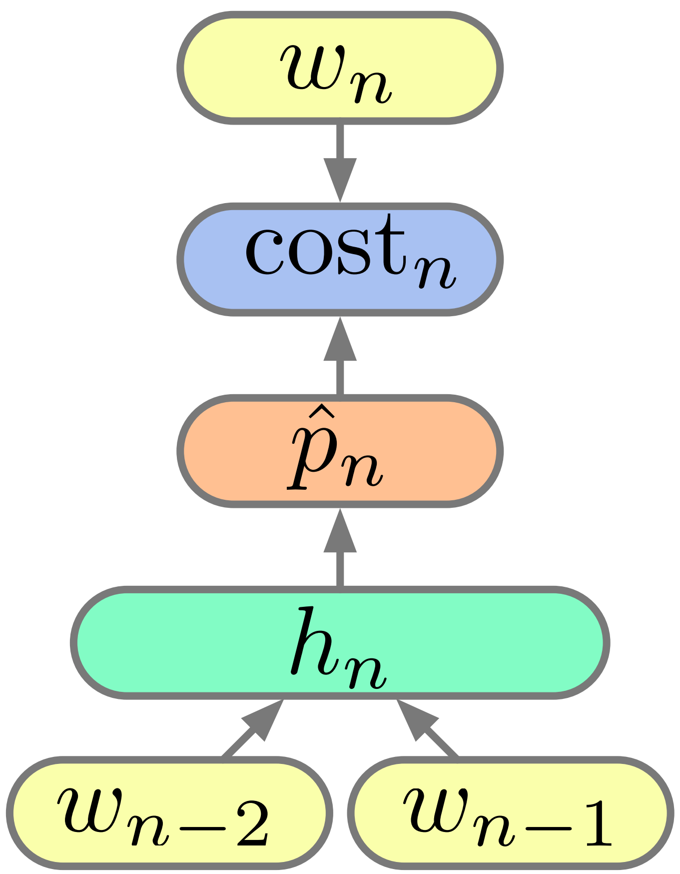
The usual training objective is the cross entropy of the data given the model (MLE):

$$\mathcal{F} = -\frac{1}{N}\sum_{n} \text{cost}_n(w_n, \hat{p}_n)$$

The cost function is simply the model's estimated log-probability of w_n:

$$\text{cost}(a, b) = a^T \log b$$

(assuming w_i is a one hot encoding of the word)



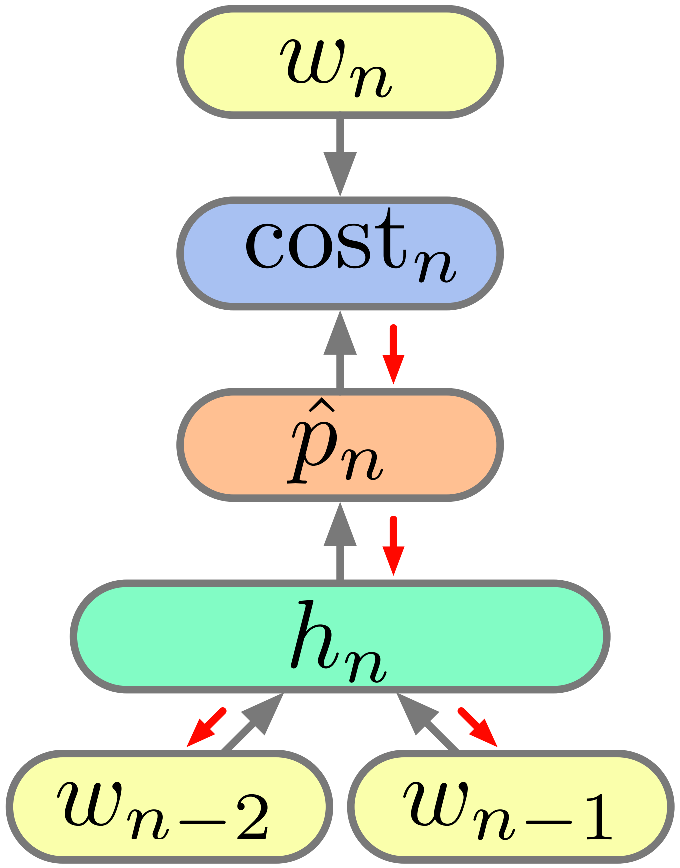
[Slide: Phil Blunsom]

Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

$$\frac{\partial \mathcal{F}}{\partial W} = -\frac{1}{N}\sum_n \frac{\partial \mathrm{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial W}$$

$$\frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{N}\sum_n \frac{\partial \mathrm{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$

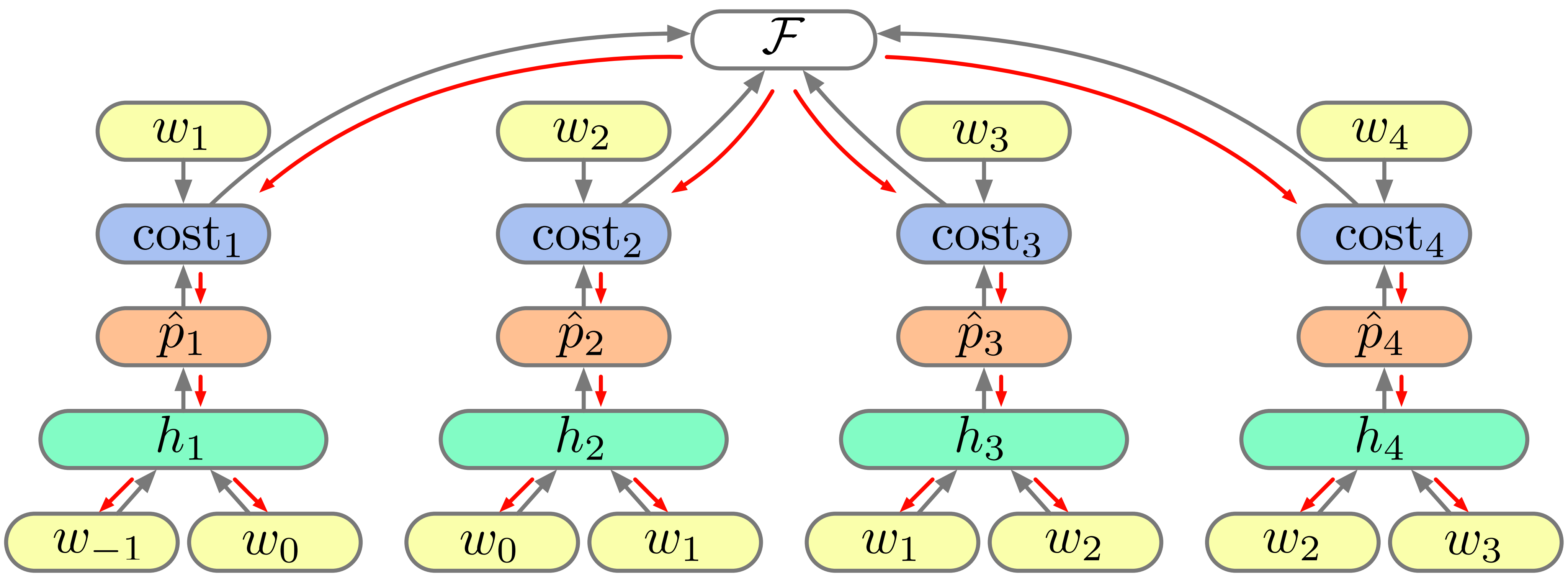


[Slide: Phil Blunsom]

Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

$$\frac{\partial \mathcal{F}}{\partial W} = -\frac{1}{4}\sum_{n=1}^{4}\frac{\partial \mathrm{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial W} \quad , \quad \frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{4}\sum_{n=1}^{4}\frac{\partial \mathrm{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$



Note that calculating the gradients for each time step n is independent of all other timesteps, as such they are calculated in parallel and summed.

[Slide: Phil Blunsom]

Comparison with Count Based N-Gram LMs

Good

- Better generalisation on unseen n-grams, poorer on seen n-grams. Solution: direct (linear) ngram features.
- Simple NLMs are often an order magnitude smaller in memory footprint than their vanilla n-gram cousins (though not if you use the linear features suggested above!).

Bad

- The number of parameters in the model scales with the n-gram size and thus the length of the history captured.
- The n-gram history is finite and thus there is a limit on the longest dependencies that an be captured.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.

[Slide: Phil Blunsom]

Training NNs

- Dropout (preferred regularization method)
- Minibatching
- Parallelization (GPUs)

- Local optima?

Word/feature embeddings

- "Lookup layer": from discrete input features (words, ngrams, etc.) to continuous vectors
  - Anything that was directly used in log-linear models, move to using vectors
- Learn or not?
  - Learn: they're just model parameters
  - Fixed: use pretrained embeddings
    - Use a faster-to-train model on very large, perhaps different, dataset [e.g. *word2vec*, *glove* pretrained word vectors]
  - Both: initialize with pretrained, then learn
    - Word at test but not training time?
- Shared representations for domain adaptation and multitask learning

Local models

$$w_t \mid w_{t-2}, w_{t-1}$$

Long-history models

$$w_t \mid w_1, \ldots w_{t-1}$$

Fully observed
direct word models

Latent-class
direct word models

$\ldots\ldots$ Log-linear models $\ldots\ldots$

Markovian neural LM

Recurrent neural LM