Neural network LM

CS 690N, Spring 2018

Advanced Natural Language Processing

http://people.cs.umass.edu/~brenocon/anlp2018/

Brendan O'Connor

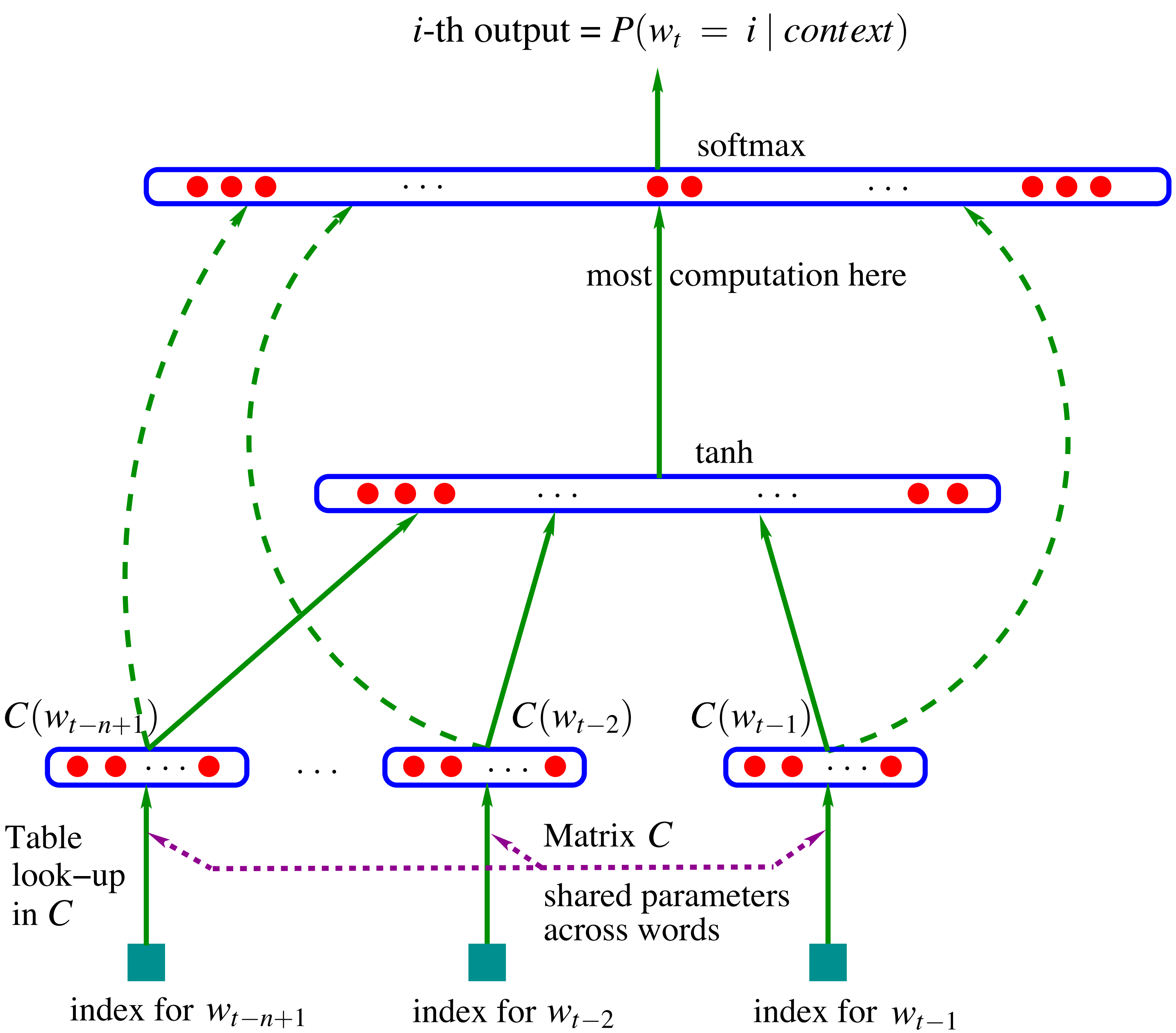
College of Information and Computer Sciences

University of Massachusetts Amherst

- Paper presentations
  - Groups 2-3
  - Aim for 20 minutes
- Choose a full-length research paper in NLP, or computational linguistics
  - Choose yourself (and get our approval >1 week out), or choose from a list
- Similar to the reading feedback writing: Summarization (what did they do? what methods? what data), explanation (what are the contributions?), synthesis and critique (what are the strengths/ weaknesses? relationships to other work or future work?)

Bengio et al. 2003: N-gram multilayer perceptron

$$f(w_t, \cdots, w_{t-n+1}) = \hat{P}(w_t | w_1^{t-1})$$



Learn: C, W, U, H, d (chain rule)

$$C(i) \in \mathbb{R}^m$$ Word embedding parameters

$$x = (C(w_{t-1}), C(w_{t-2}), \cdots, C(w_{t-n+1}))$$

Lookup layer with concatenation: (kinda) hidden layer size *(n-1)m*

shortcut linear layer

another hidden layer, size *h*

$$y = \underline{b + Wx} + U \tanh(\underline{d + Hx})$$

Vocab output: log-probs size *V*

$$\hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}.$$

Output layer (softmax / log-linear)

- Embedding lookup (C: dim (V,m)) equivalent to one-hot encoding (len V) + hidden layer (C)

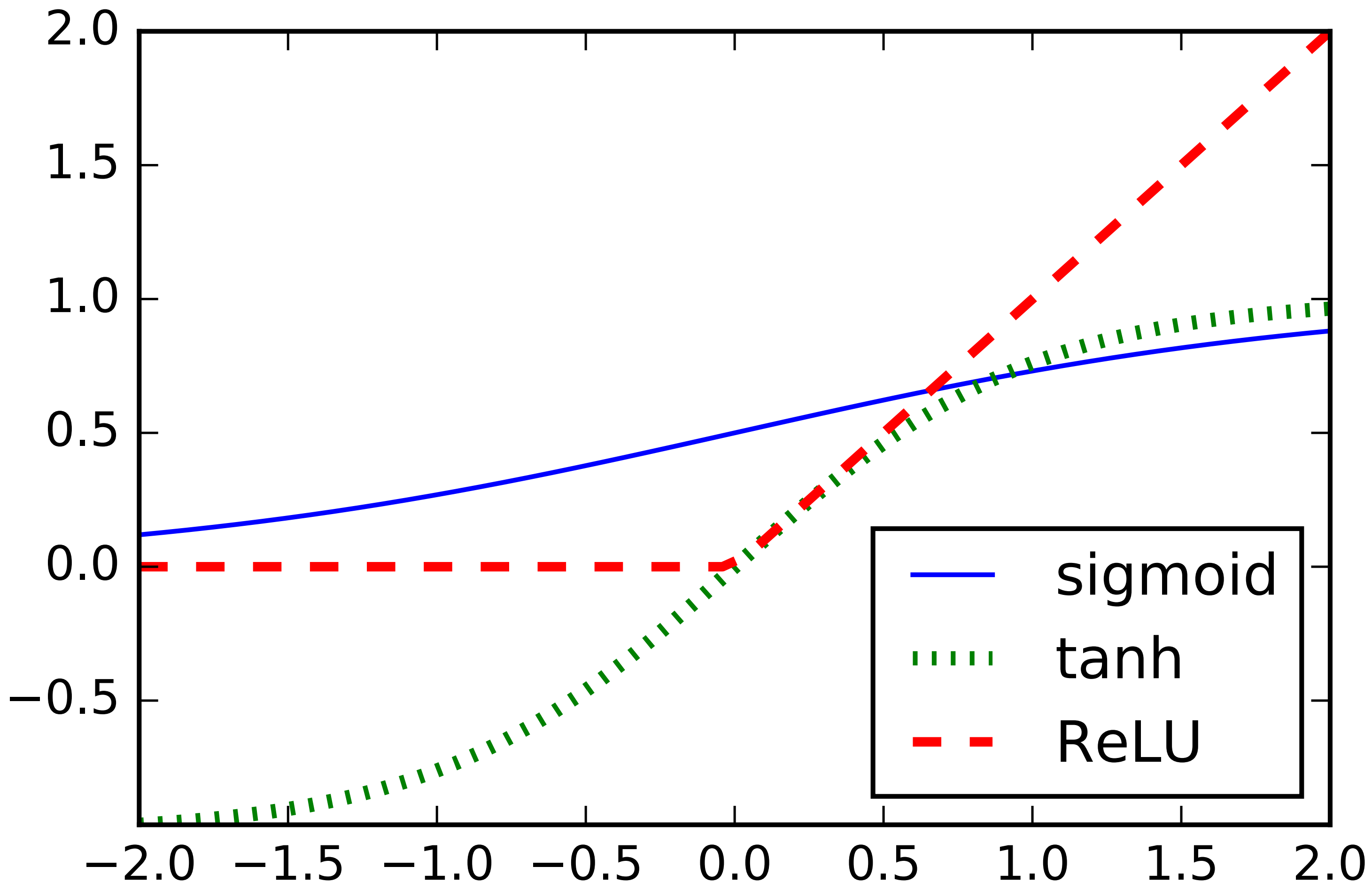
Why?

- Curse of dimensionality: bottleneck information into K~30 hidden dimensions (K<<V)
- NNs can learn complicated functions
  - ... we don't really have a good grip on what's learnable beyond universal function approximation
  - ... but seems better than linear dim reduction (e.g. S+P). Non-planar regions in embedding space?
- Multilayer structures
  - Maybe: "deep" models learn more abstract concepts (clearly in vision; less clear for NLP, though can help)
  - Definitely: hierarchical and sequential NNs to match hierarchical/memory-ful structure in language (recursive/ recurrent NNs)

Word/feature embeddings

- "Lookup layer": from discrete input features (words, ngrams, etc.) to continuous vectors
  - Any binary feature that was directly used in log-linear models, give it a vector
  - Character n-grams, part-of-speech tags, etc.
- As model parameters: learn them like everything else
- Or, as external information: use pretrained embeddings
  - Common in practice: use a faster-to-train model on very large, perhaps different, dataset
    [e.g. *word2vec*, *glove* pretrained word vectors]
- Shared representations for
  domain adaptation and multitask learning

Nonlinear activation functions



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

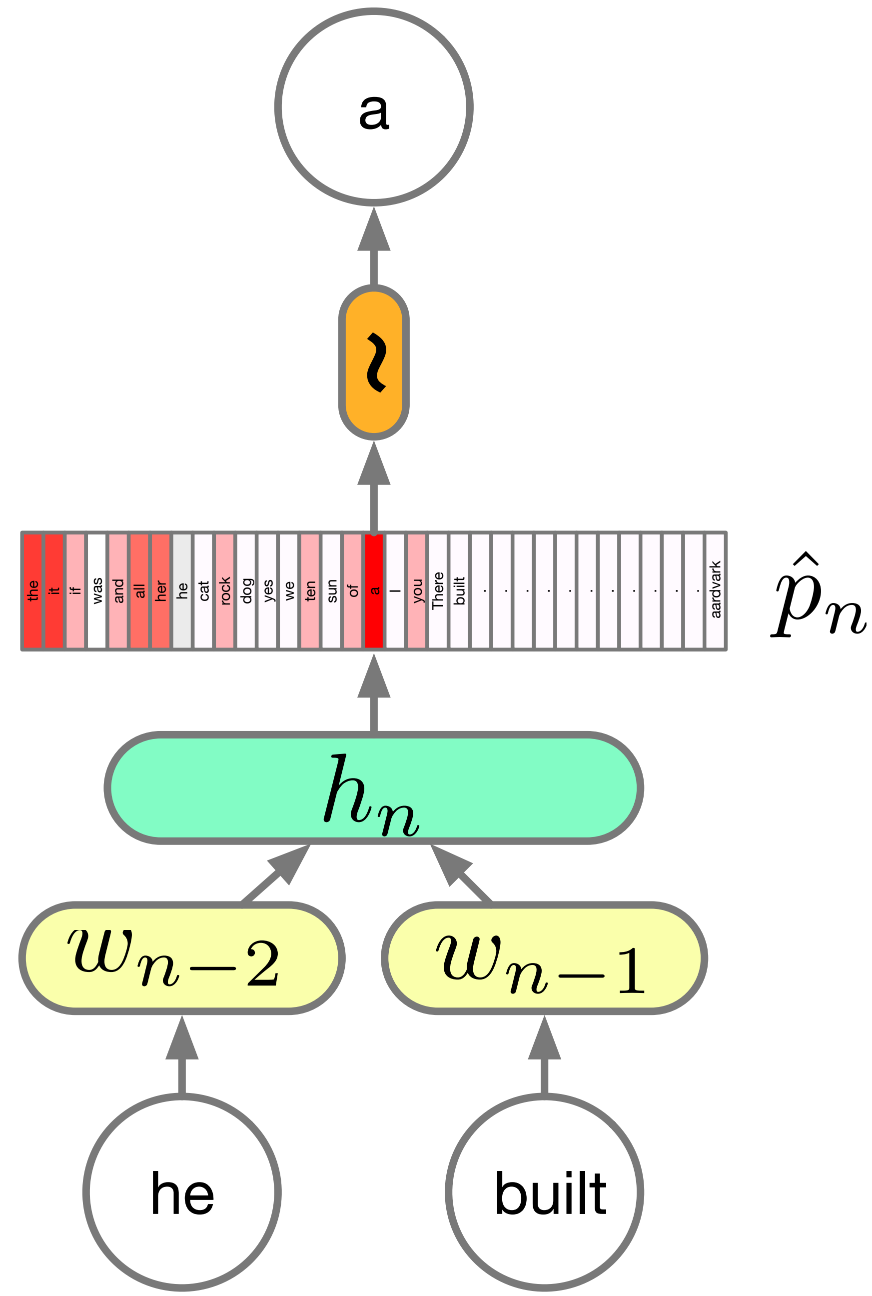
$$\tanh(x) = 2 \times \mathrm{sgm}(x) - 1$$

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

positive part a.k.a. ReLU

Neural Language Models: Sampling

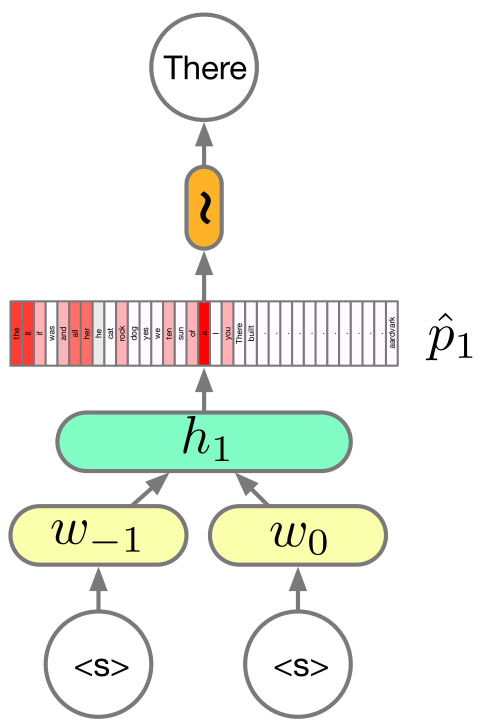
$$w_n | w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

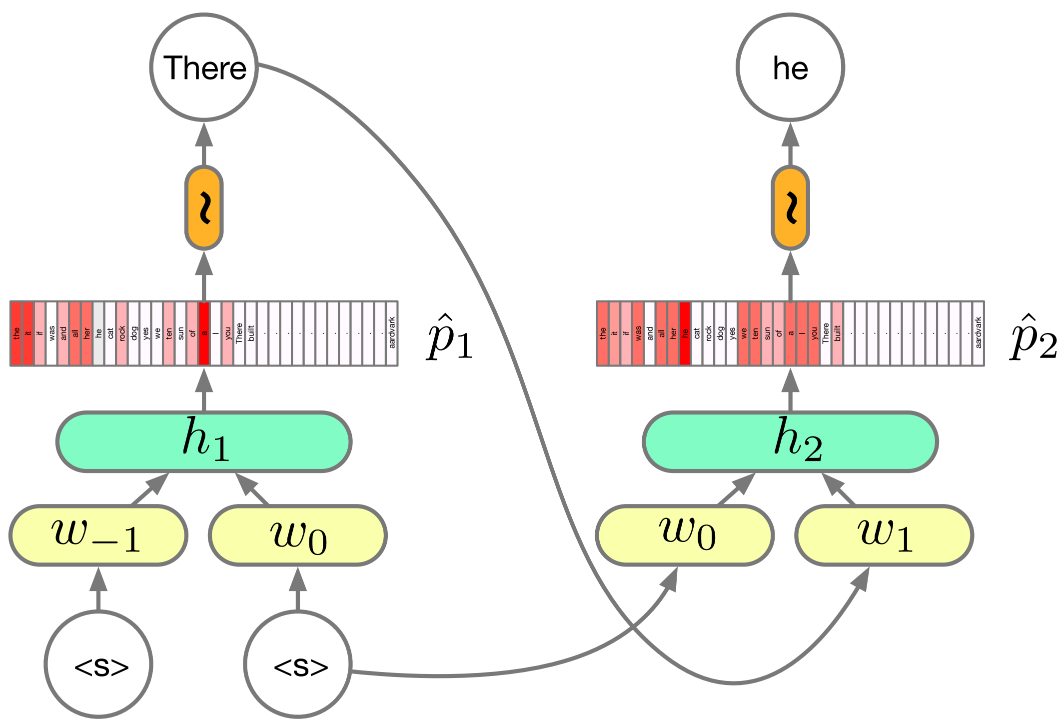
$$w_n | w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

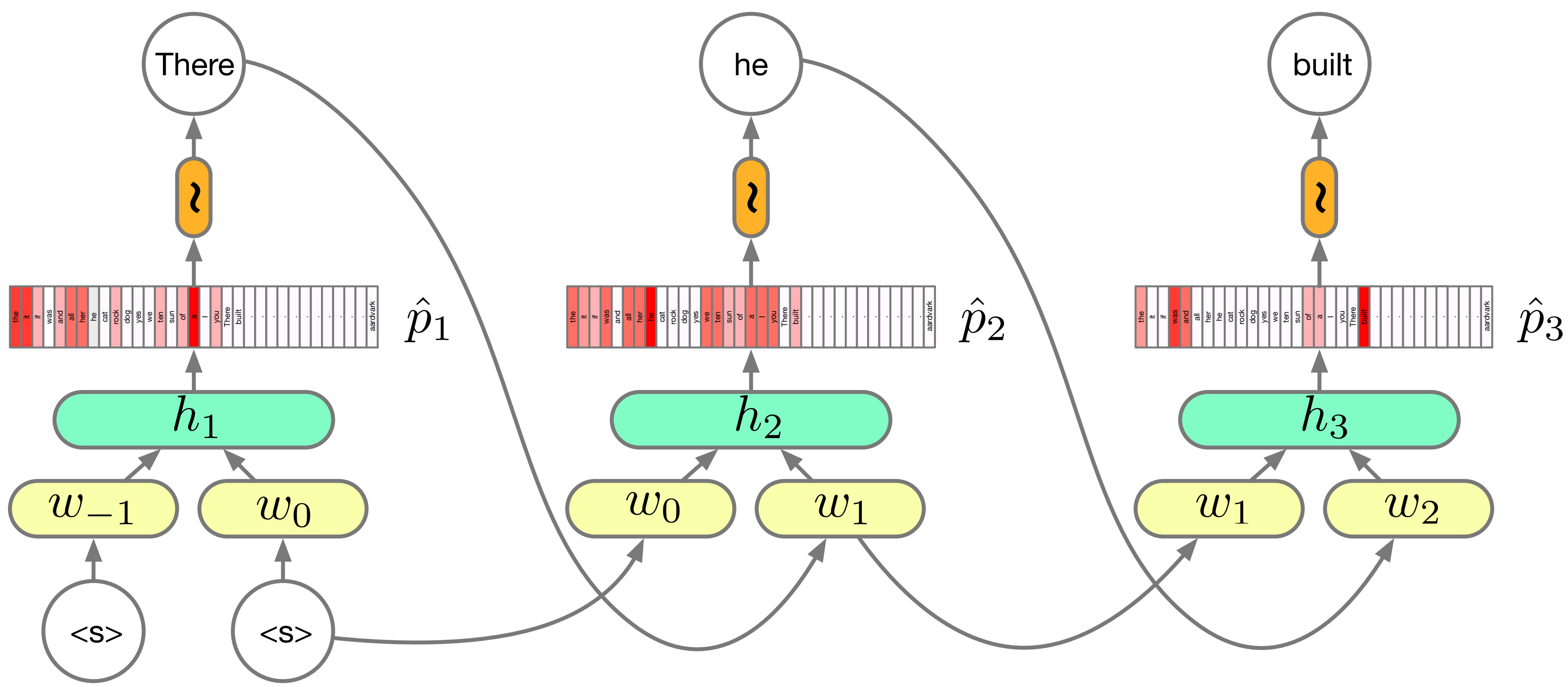
$$w_n \big| w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[Slide: Phil Blunsom]

Neural Language Models: Sampling

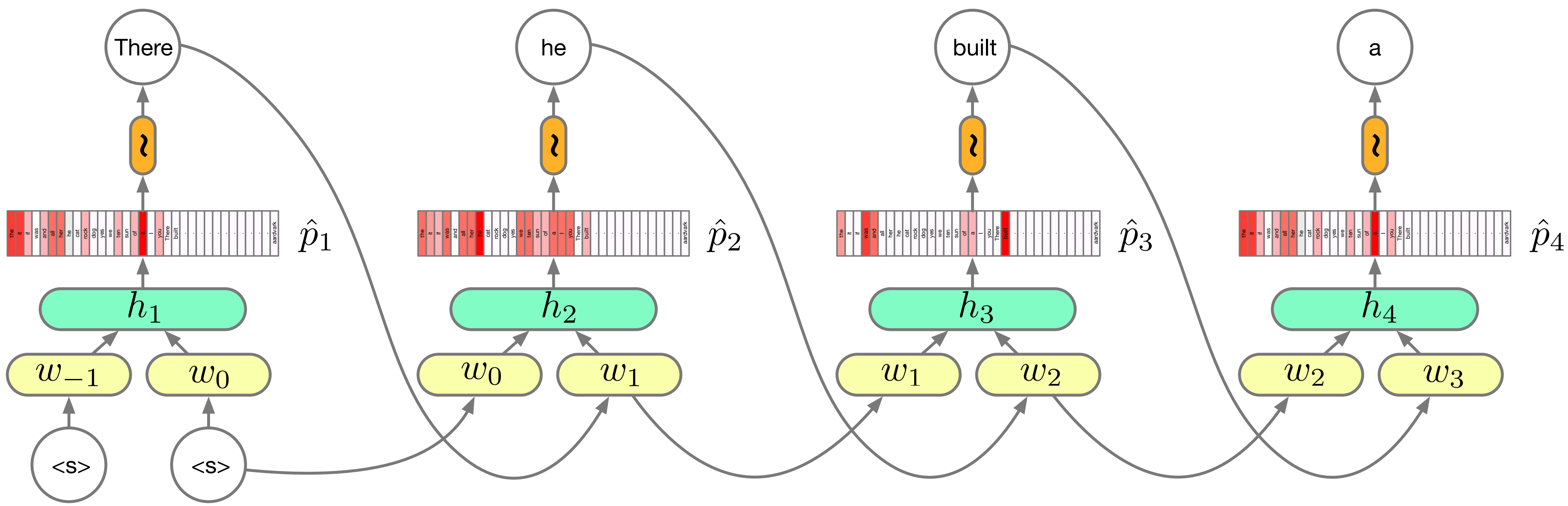
$$w_n | w_{n-1}, w_{n-2} \quad \sim \quad \hat{p}_n$$



[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: Training

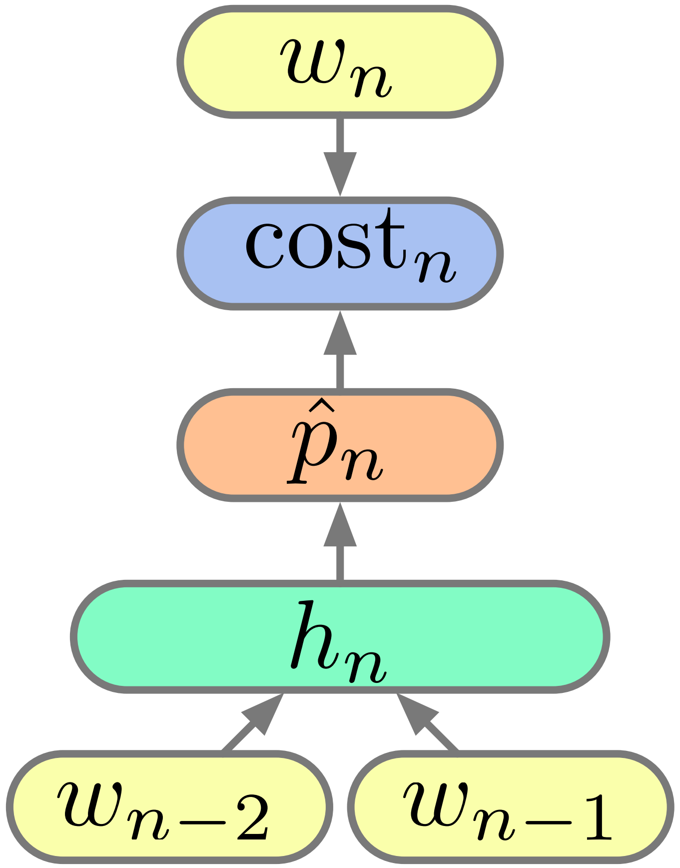
The usual training objective is the cross entropy of the data given the model (MLE):

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

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)



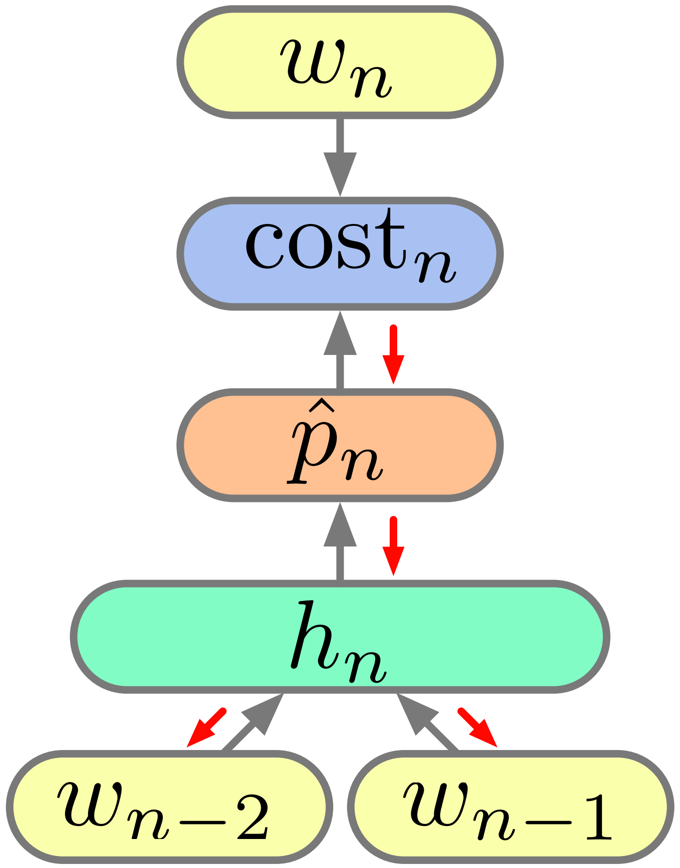
[Slide: Phil Blunsom]

Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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

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

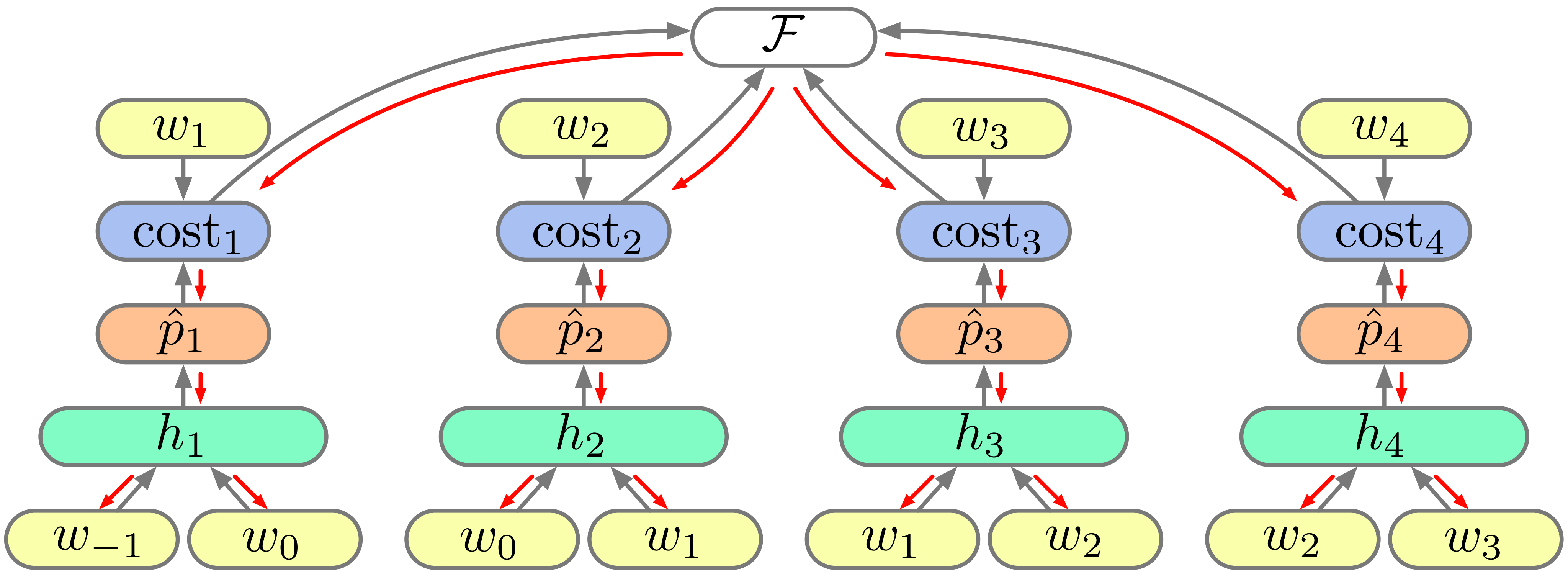


[Slide: Phil Blunsom]

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)
- Minibatch (adaptive) SGD
  - Parallelization (CPUs, GPUs) within a minibatch

- Local optima (?)

Boring old SGD

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \boldsymbol{g}_t$$

params $\boldsymbol{x}$, learning rate $\boldsymbol{\eta}$, minibatch timestep $\boldsymbol{t}$, gradient $\boldsymbol{g_t}$
(typically: learning rate decay on fixed schedule. or constant learning rate?)

Boring old SGD

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \boldsymbol{g}_t$$

params $\boldsymbol{x}$, learning rate $\boldsymbol{\eta}$, minibatch timestep $\boldsymbol{t}$, gradient $\mathbf{g_t}$ (typically: learning rate decay on fixed schedule. or constant learning rate?)

Adaptive SGD

- AdaGrad: simplest of adaptive SGD methods.
  - Has *per-parameter*, *adaptive* learning rates

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} g_{t',i}^2}} g_{t,i}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \mathbf{G}_t^{-1/2} \odot \boldsymbol{g}_t$$

- If G was the Hessian, and we calculated g and G on the whole batch, this would be a Newton-Raphson step
  - Related: (Nesterov) momentum
- Variants with tricks about history decay, etc. (e.g. Adam, RMSprop, Adadelta...)

Local vs. global models

Local models	Long-history models
$w_t \mid w_{t-2}, w_{t-1}$	$w_t \mid w_1, \ldots w_{t-1}$
Fully observed direct word models	
Latent-class direct word models	

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

Markovian neural LM	Recurrent neural LM