CA4012

Statistical Machine Translation

Week 10: Neural Language Models

Lecturer: Mohammed Hasanuzzaman

Lab Tutors: Longyue Wang and Meghan Dowling

Date: 12.04.2018

* slides are adopted from Phil Blunsom lecture

Language models

A language model assigns a probability to a sequence of words, such that $\sum_{w \in \Sigma^*} p(w) = 1$:

Given the observed training text, how probable is this new utterance?

Thus we can compare different orderings of words (e.g. Translation):

$$p(\text{he likes apples}) > p(\text{apples likes he})$$

or choice of words (e.g. Speech Recognition):

$$p(\text{he likes apples}) > p(\text{he licks apples})$$

Language models

Much of Natural Language Processing can be structured as (conditional) language modelling:

Translation

$$p_{\mathrm{LM}}(\text{Les chiens aiment les os } ||| \text{ Dogs love bones})$$

Question Answering

$$p_{\mathrm{LM}}(\text{What do dogs love? } ||| \text{ bones . } | \beta)$$

Dialogue

$$p_{\mathrm{LM}}(\text{How are you? } ||| \text{ Fine thanks. And you? } | \beta)$$

Language models

Most language models employ the chain rule to decompose the joint probability into a sequence of conditional probabilities:

$$p(w_1, w_2, w_3, \ldots, w_N) =$$
$$p(w_1)\, p(w_2|w_1)\, p(w_3|w_1, w_2) \times \ldots \times p(w_N|w_1, w_2, \ldots w_{N-1})$$

Note that this decomposition is exact and allows us to model complex joint distributions by learning conditional distributions over the next word (w_n) given the history of words observed ($w_1, \ldots, w_{n-1}$).

DCU

Language models

The simple objective of modelling the next word given the observed history contains much of the complexity of natural language understanding.

Consider predicting the extension of the utterance:

$$p(\cdot|\text{ There she built a})$$

With more context we are able to use our knowledge of both language and the world to heavily constrain the distribution over the next word:

$$p(\cdot|\text{ Alice went to the beach. There she built a})$$

There is evidence that human language acquisition partly relies on future prediction.

Evaluating a Language Model

A good model assigns real utterances w_1^N from a language a high probability. This can be measured with cross entropy:

$$H(w_1^N) = -\frac{1}{N} \log_2 p(w_1^N)$$

Intuition 1: Cross entropy is a measure of how many bits are needed to encode text with our model.

Alternatively we can use **perplexity**:

$$\text{perplexity}(w_1^N) = 2^{H(w_1^N)}$$

Intuition 2: Perplexity is a measure of how surprised our model is on seeing each word.

N-Gram Models: The Markov Chain Assumption

Markov assumption:

- only previous history matters
- limited memory: only last $k-1$ words are included in history (older words less relevant)
- ***k*th order Markov model**

For instance 2-gram language model:

$$
\begin{aligned}
p(w_1, w_2, w_3, \ldots, w_n) & \\
&= p(w_1)\, p(w_2|w_1)\, p(w_3|w_1, w_2) \times \ldots \\
&\quad \times p(w_n|w_1, w_2, \ldots w_{n-1}) \\
&\approx p(w_1)\, p(w_2|w_1)\, p(w_3|w_2) \times \ldots \times p(w_n|w_{n-1})
\end{aligned}
$$

N-Gram Models: Estimating Probabilities

Maximum likelihood estimation for 3-grams:

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

Collect counts over a large text corpus. Billions to trillions of words are easily available by scraping the web.

Provisional Summary

Good

- Count based n-gram models are exceptionally scalable and are able to be trained on trillions of words of data,
- fast constant time evaluation of probabilities at test time,
- sophisticated smoothing techniques match the empirical distribution of language.[5]

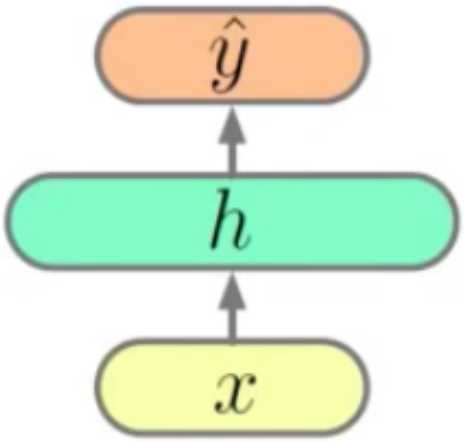
Bad

- Large ngrams are sparse, so hard to capture long dependencies,
- symbolic nature does not capture correlations between semantically similary word distributions, e.g. cat $\leftrightarrow$ dog,
- similarly morphological regularities, running $\leftrightarrow$ jumping, or gender.

Neural Language Models

Feed forward network

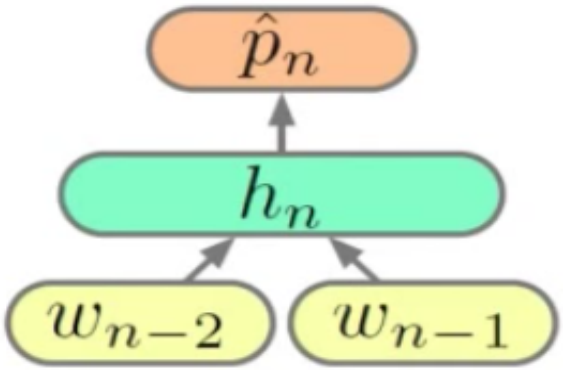
$$
\begin{aligned}
h &= g(Vx + c) \\
\hat{y} &= Wh + b
\end{aligned}
$$



Neural Language Models

Trigram NN language model

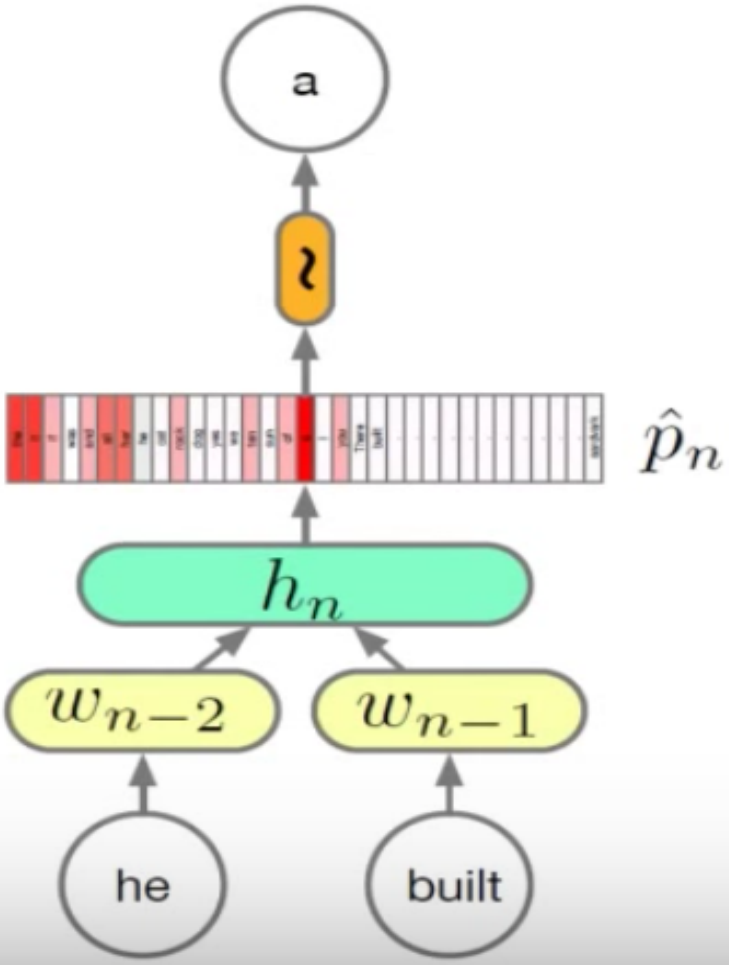
$$
\begin{aligned}
h_n &= g(V[w_{n-1}; w_{n-2}] + c) \\
\hat{p}_n &= \text{softmax}(Wh_n + b) \\
\text{softmax}(u)_i &= \frac{\exp u_i}{\sum_j \exp u_j}
\end{aligned}
$$



- 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 $> 1e5$.

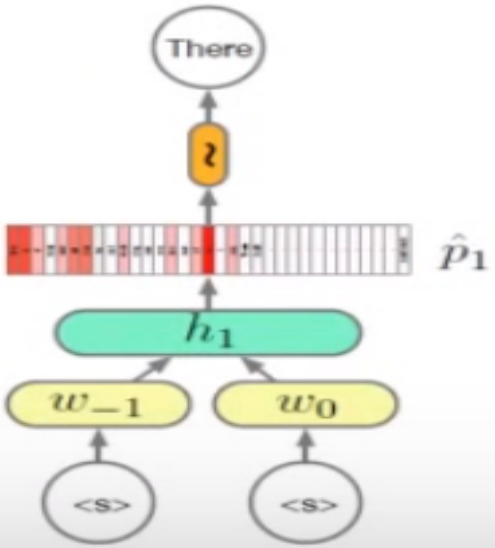
Neural Language Models: Sampling

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



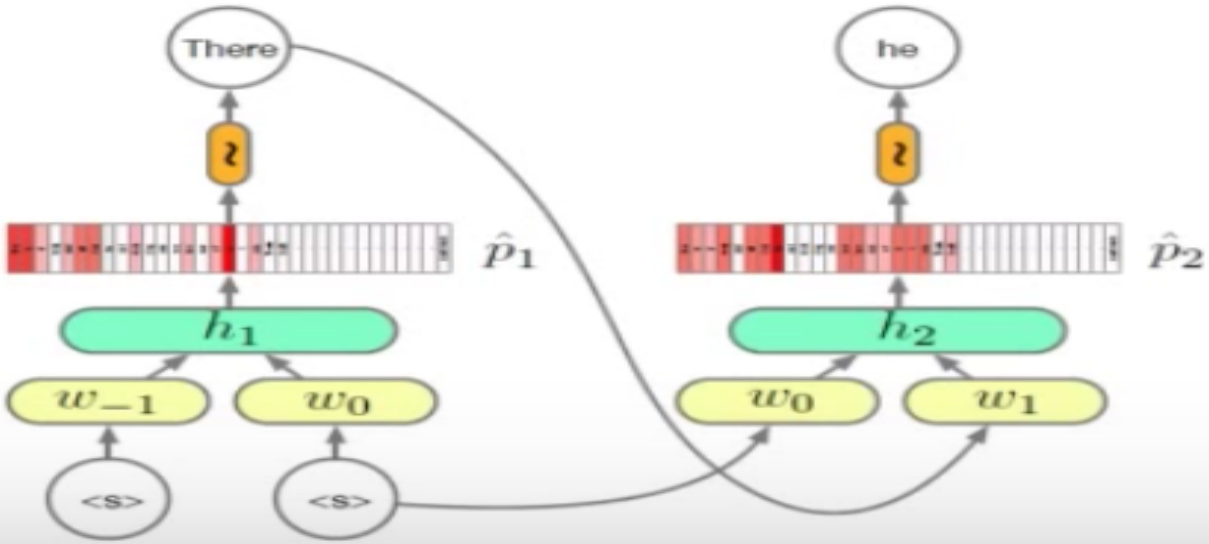
Neural Language Models: Sampling

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



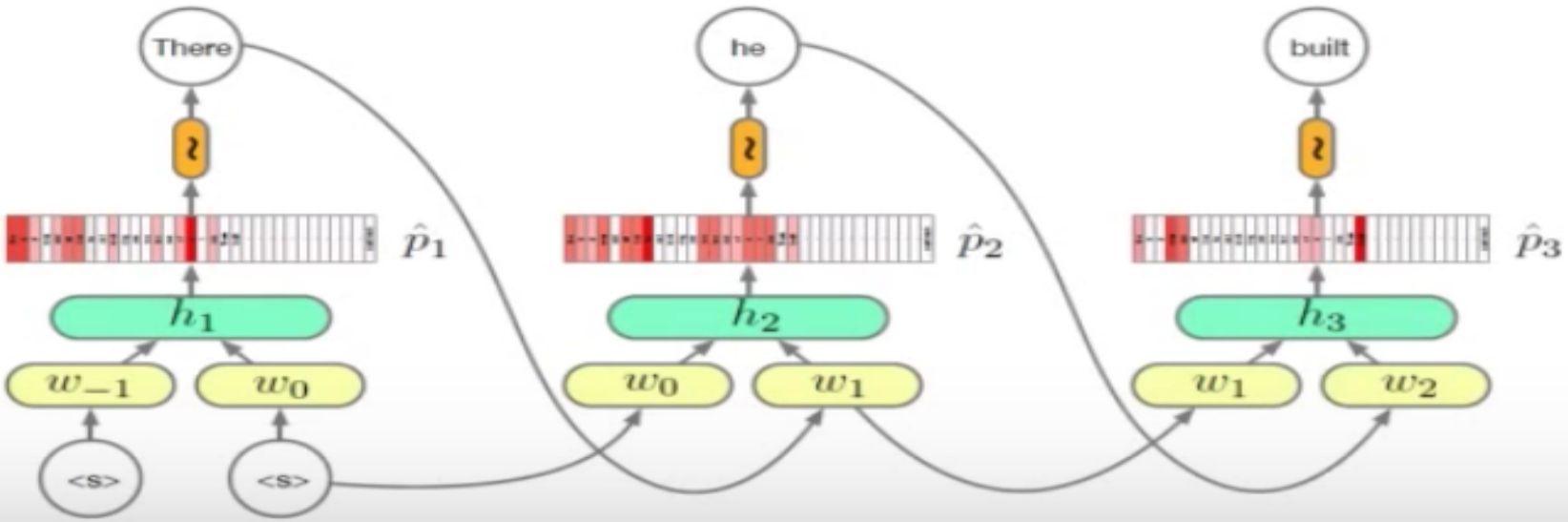
Neural Language Models: Sampling

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



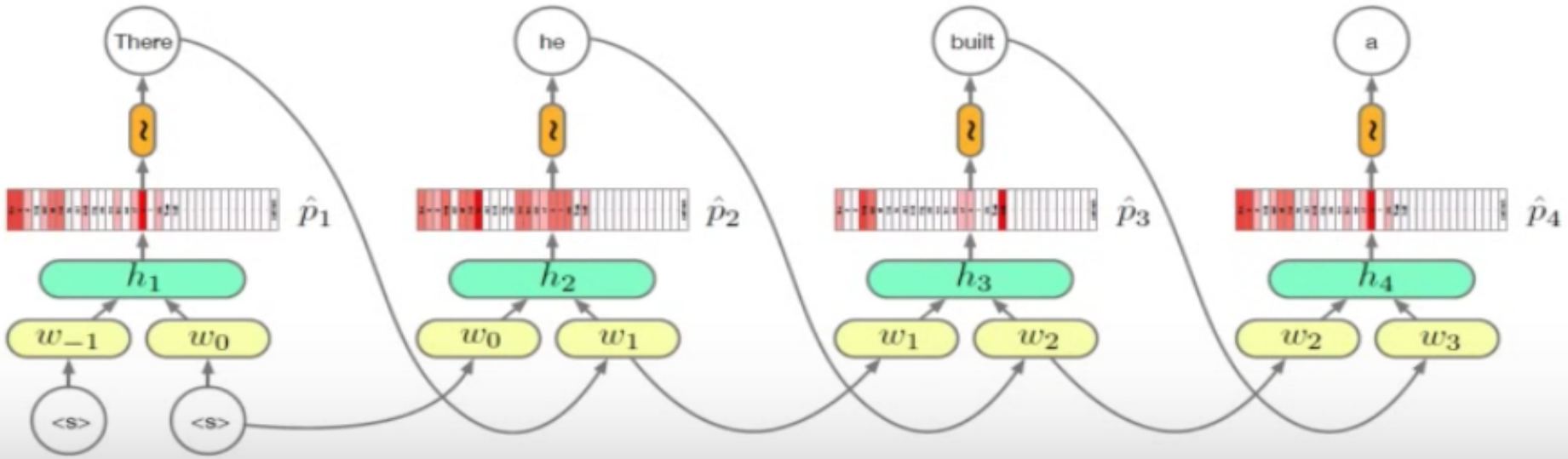
Neural Language Models: Sampling

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



Neural Language Models: Sampling

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



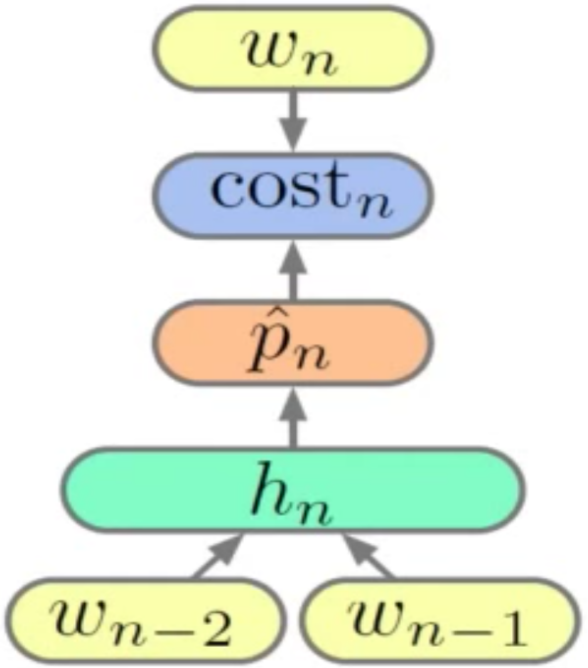
The usual training objective is the cross entropy of the data given the model (MLE):

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

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)

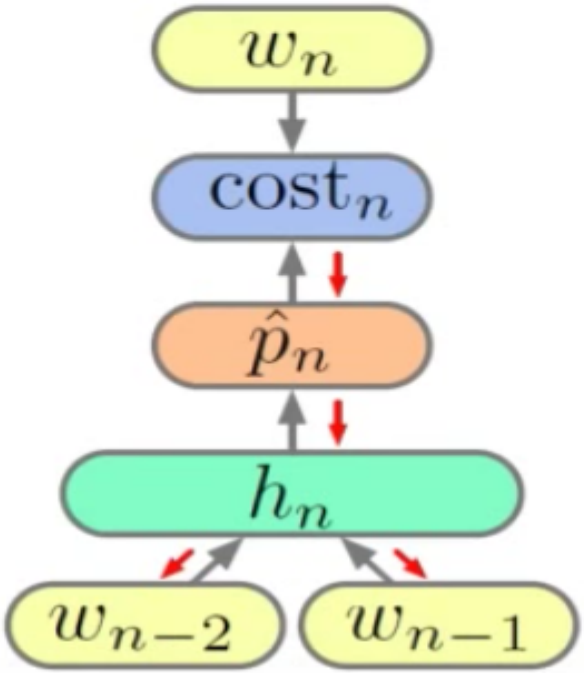


Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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

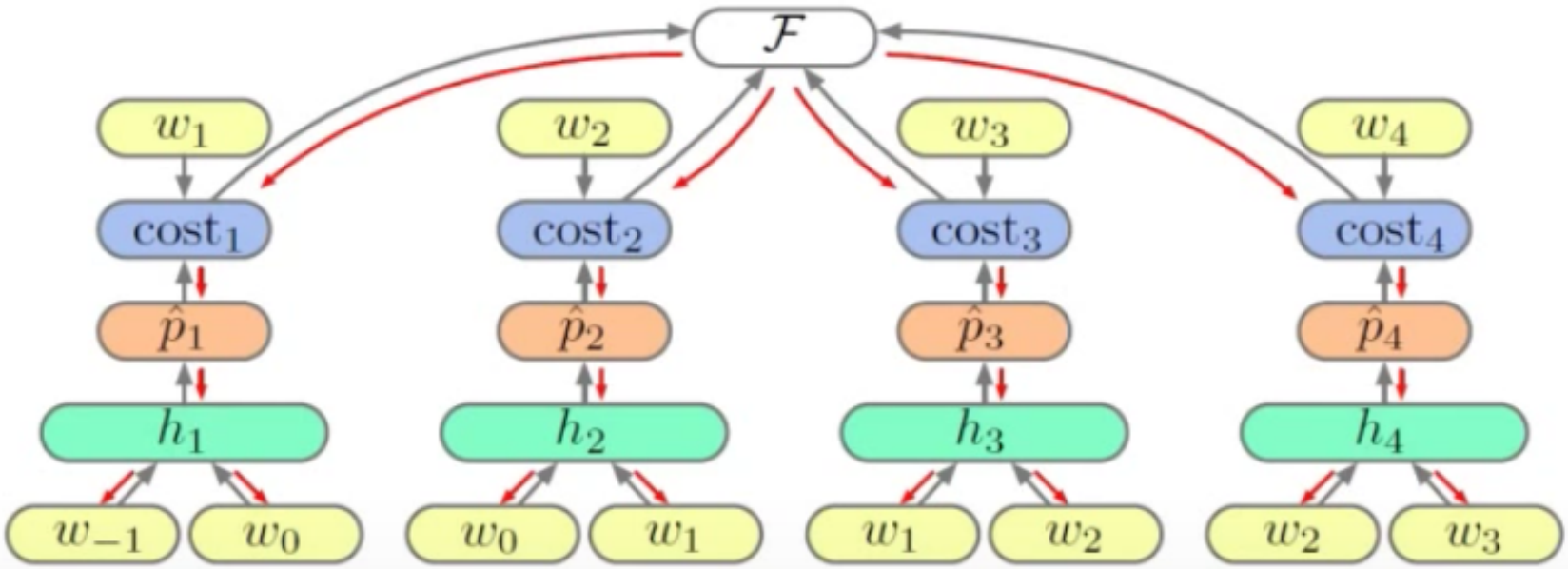
$$\frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{N} \sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial h_n} \frac{\partial h_n}{\partial V}$$



Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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



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!).

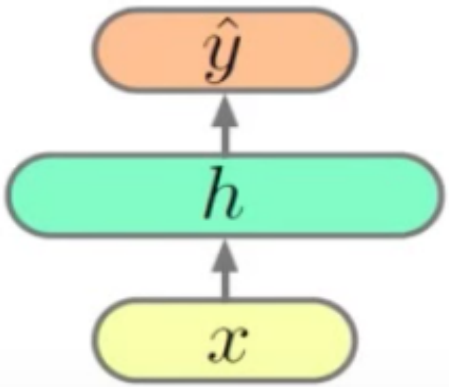
Bad

- 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.

Recurrent Neural Network Language Models

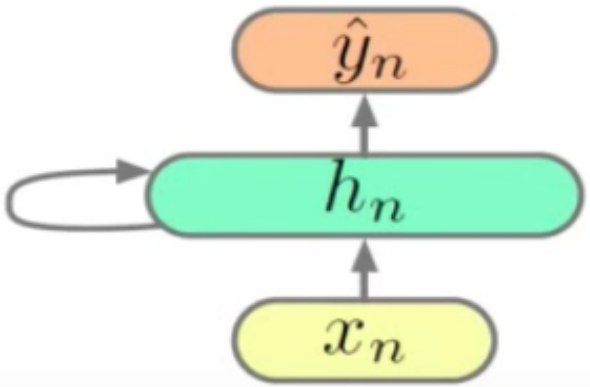
Feed Forward

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



Recurrent Network

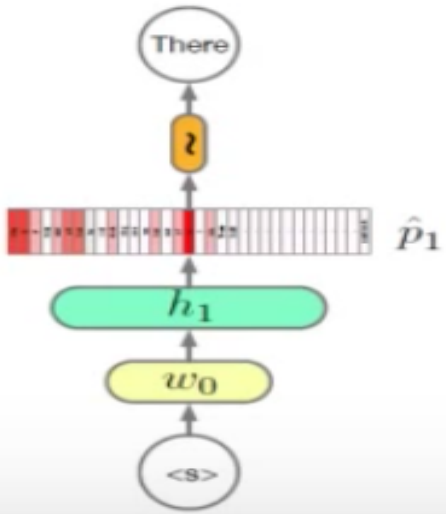
$$h_n = g(V[x_n; h_{n-1}] + c)$$
$$\hat{y}_n = Wh_n + b$$



DCU

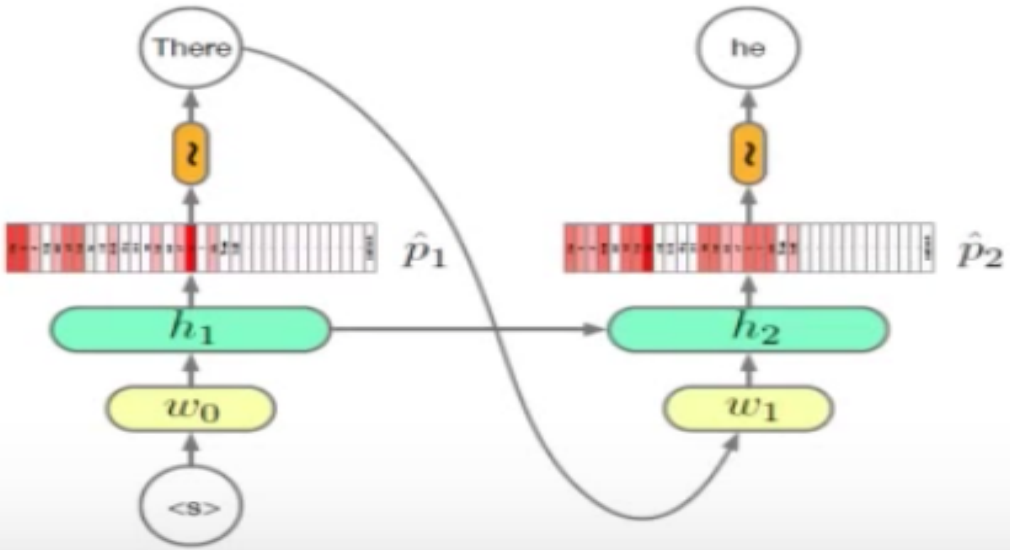
Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



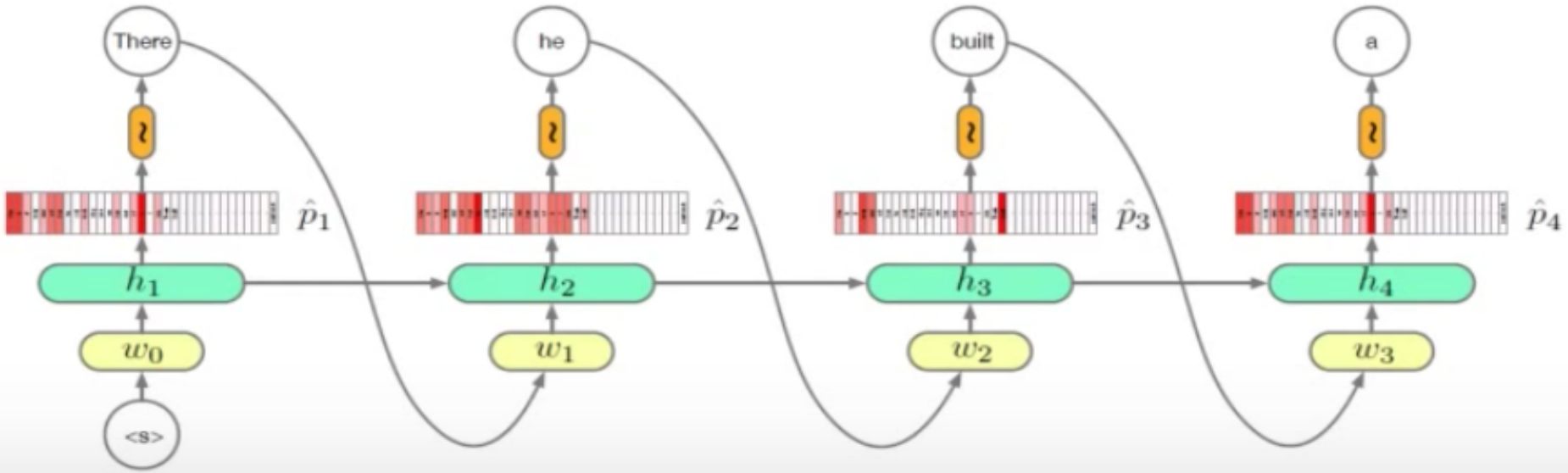
Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$

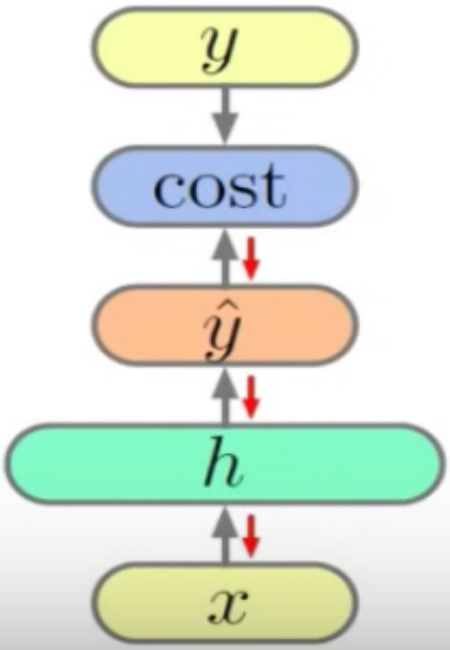


DCU

Recurrent Neural Network Language Models

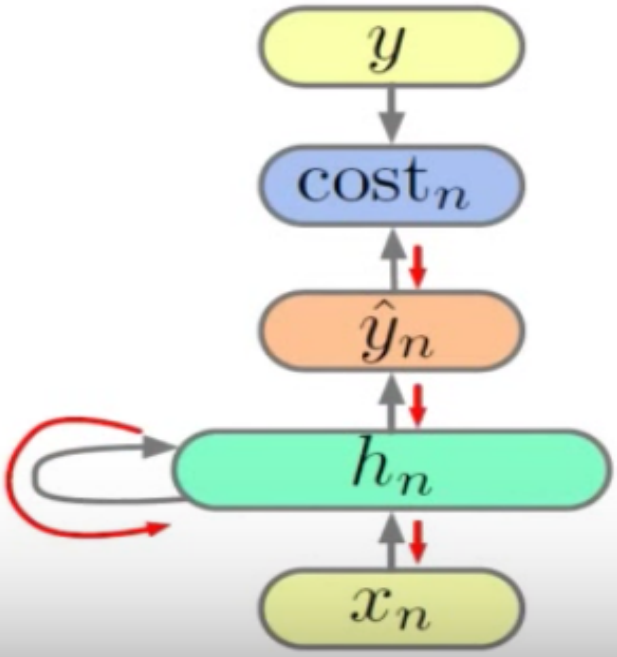
Feed Forward

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



Recurrent Network

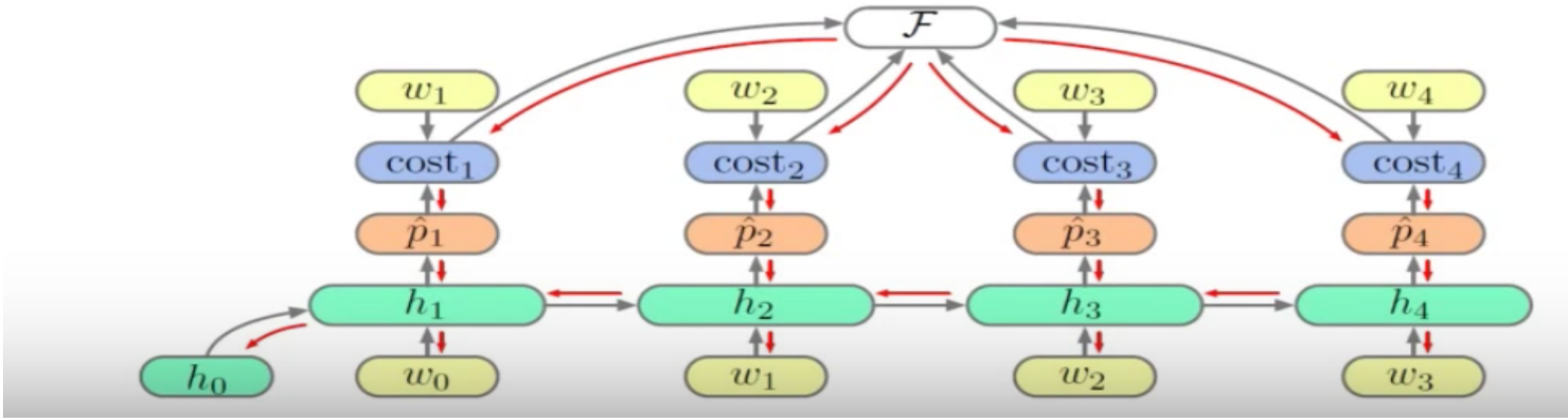
$$h_n = g(V[x_n; h_{n-1}] + c)$$
$$\hat{y}_n = Wh_n + b$$



Recurrent Neural Network Language Models

The unrolled recurrent network is a directed acyclic computation graph. We can run backpropagation as usual:

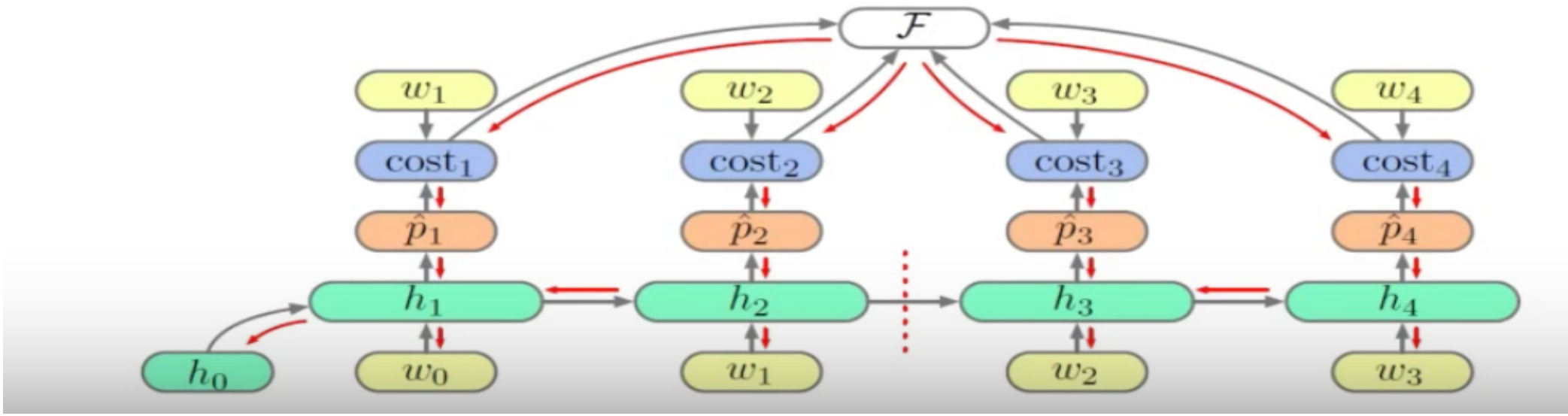
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

If we break these depdencies after a fixed number of timesteps we get Truncated Back Propagation Through Time (TBPTT):

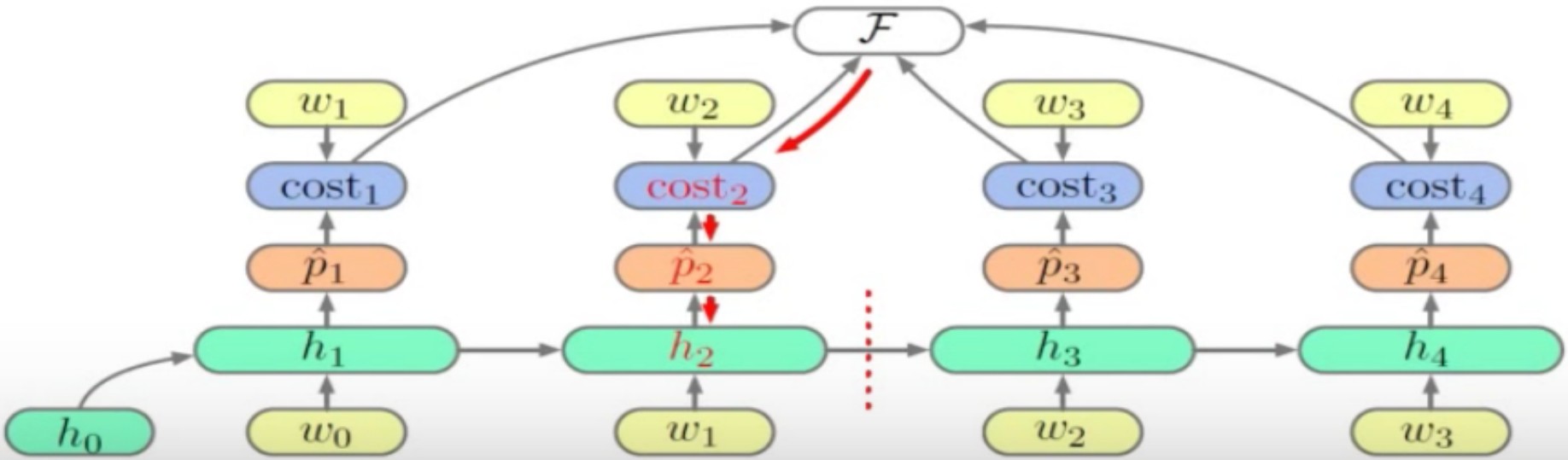
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

If we break these depdencies after a fixed number of timesteps we get Truncated Back Propagation Through Time (TBPTT):

$$\frac{\partial \mathcal{F}}{\partial h_2} \approx \frac{\partial \mathcal{F}}{\partial \text{cost}_2} \frac{\partial \text{cost}_2}{\partial \hat{p}_2} \frac{\partial \hat{p}_2}{\partial h_2}$$



Comparison with N-Gram LMs

Good

- RNNs can represent unbounded dependencies, unlike models with a fixed n-gram order.
- RNNs compress histories of words into a fixed size hidden vector.
- The number of parameters does not grow with the length of dependencies captured, but they do grow with the amount of information stored in the hidden layer.

Bad

- RNNs are hard to learn and often will not discover long range dependencies present in the data
- Increasing the size of the hidden layer, and thus memory, increases the computation and memory quadratically.

References

Textbook

Deep Learning, Chapter 10.
`www.deeplearningbook.org/contents/rnn.html`

Blog Posts

Andrej Karpathy: The Unreasonable Effectiveness of Recurrent Neural Networks
`karpathy.github.io/2015/05/21/rnn-effectiveness/`

Yoav Goldberg: The unreasonable effectiveness of Character-level Language Models
`nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139`

Stephen Merity: Explaining and illustrating orthogonal initialization for recurrent neural networks.
`smerity.com/articles/2016/orthogonal_init.html`