Recurrent Neural Networks and Language Modelling: Part 1

Phil Blunsom

phil.blunsom@cs.ox.ac.uk

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?

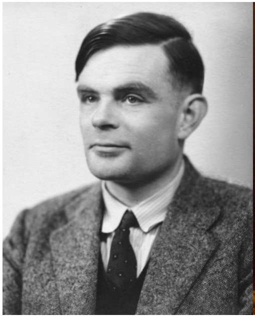
Thus we can compare different orderings of words (e.g. Translation):

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

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

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

History: cryptography



Language models

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

Translation

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

Question Answering

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

Dialogue

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

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 ($w_1, \ldots, w_{n-1}$).

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.

Language Modelling Data

Language modelling is a time series prediction problem in which we must be careful to train on the past and test on the future.

If the corpus is composed of articles, it is best to ensure the test data is drawn from a disjoint set of articles to the training data.

Language Modelling Data

Two popular data sets for language modelling evaluation are a preprocessed version of the Penn Treebank,[1] and the Billion Word Corpus.[2] Both are flawed:

- the PTB is very small and has been heavily processed. As such it is not representative of natural language.
- The Billion Word corpus was extracted by first randomly permuting sentences in news articles and then splitting into training and test sets. As such train and test sentences come from the same articles and overlap in time.

The recently introduced WikiText datasets[3] are a better option.

[1] `www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz`

[2] `code.google.com/p/1-billion-word-language-modeling-benchmark/`

[3] Pointer Sentinel Mixture Models. Merity et al., arXiv 2016

Lecture Overview

The rest of this lecture will survey three approaches to parametrising language models:

- With count based n-gram models we approximate the history of observed words with just the previous *n* words.
- Neural n-gram models embed the same fixed n-gram history in a continues space and thus better capture correlations between histories.
- With Recurrent Neural Networks we drop the fixed n-gram history and compress the entire history in a fixed length vector, enabling long range correlations to be captured.

Outline

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:

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

The conditioning context, w_{i-1}, is called the **history**.

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.

N-Gram Models: Back-Off

In our training corpus we may never observe the trigrams:

- Oxford Pimm's eater
- Oxford Pimm's drinker

If both have count 0 our smoothing methods will assign the same probability to them.

A better solution is to interpolate with the bigram probability:

- Pimm's eater
- Pimm's drinker

N-Gram Models: Interpolated Back-Off

By recursively interpolating the n-gram probabilities with the $(n-1)$-gram probabilities we can smooth our language model and ensure all words have non-zero probability in a given context.

A simple approach is linear interpolation:

$$\begin{aligned} p_I(w_n|w_{n-2}, w_{n-1}) &= \lambda_3 p(w_n|w_{n-2}, w_{n-1}) + \\ &\quad \lambda_2 p(w_n|w_{n-1}) + \\ &\quad \lambda_1 p(w_n). \end{aligned}$$

where $\lambda_3 + \lambda_2 + \lambda_1 = 1$.

A number of more advanced smoothing and interpolation schemes have been proposed, with Kneser-Ney being the most common.[4]

[4] *An empirical study of smoothing techniques for language modeling. Stanley Chen and Joshua Goodman. Harvard University, 1998.* `research.microsoft.com/en-us/um/people/joshuago/tr-10-98.pdf`

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.

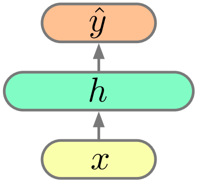
[5] Heaps' Law: `en.wikipedia.org/wiki/Heaps'_law`

Outline

Neural Language Models

Feed forward network

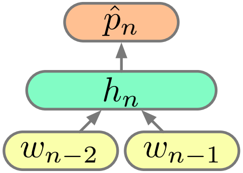
$$\begin{aligned} h &= g(Vx + c) \\ \hat{y} &= Wh + b \end{aligned}$$



Neural Language Models

Trigram NN language model

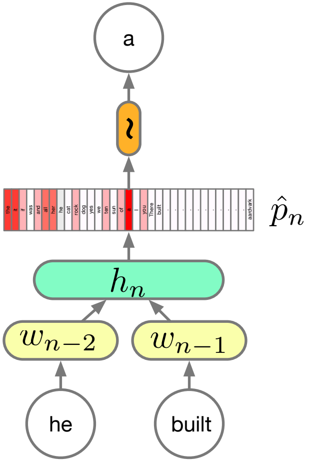
$$
\begin{aligned}
h_n &= g(V[w_{n-1}; w_{n-2}] + c) \\
\hat{p}_n &= \text{softmax}(Wh_n + b) \\
\text{softmax}(u)_i &= \frac{\exp u_i}{\sum_j \exp u_j}
\end{aligned}
$$



- 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 $> 1e5$.

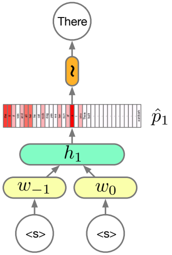
Neural Language Models: Sampling

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



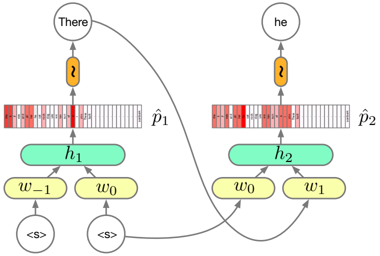
Neural Language Models: Sampling

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



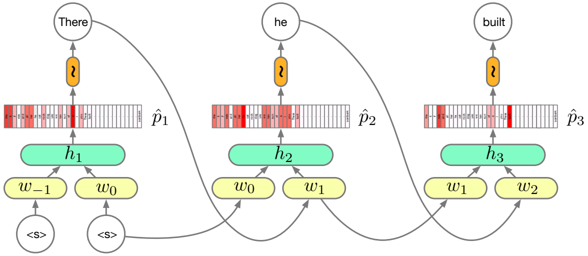
Neural Language Models: Sampling

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



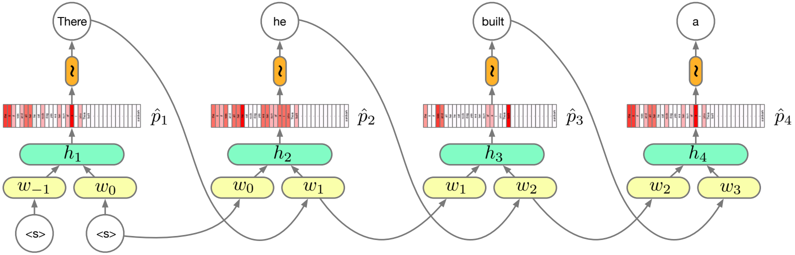
Neural Language Models: Sampling

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



Neural Language Models: Sampling

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



Neural Language Models: Training

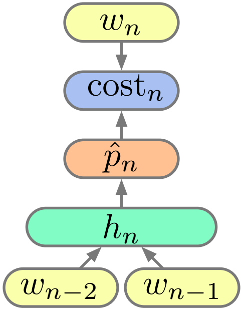
The usual training objective is the cross entropy of the data given the model (MLE):

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

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)

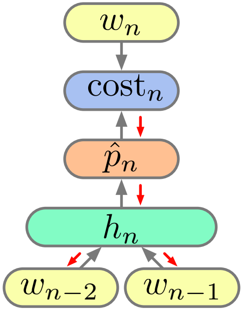


Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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

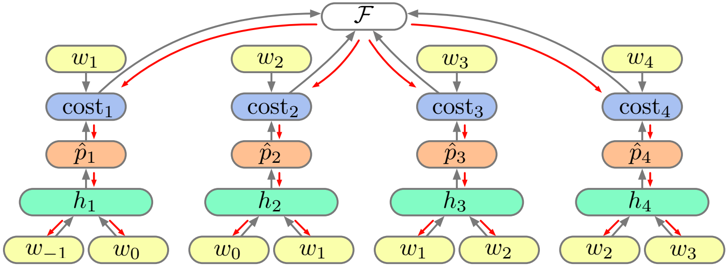
$$\frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{N}\sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$



Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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



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.

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

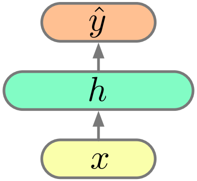
- The number of parameters in the model scales with the n-gram size and thus the length of the history captured.
- The n-gram history is finite and thus there is a limit on the longest dependencies that an be captured.
- Mostly trained with Maximum Likelihood based objectives which do not encode the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

Outline

Recurrent Neural Network Language Models

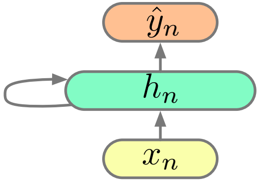
Feed Forward

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



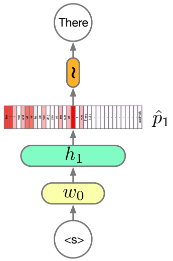
Recurrent Network

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



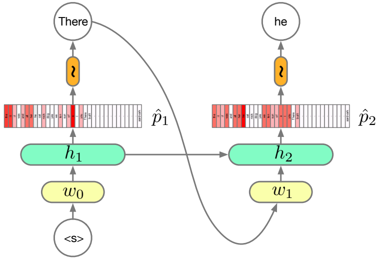
Recurrent Neural Network Language Models

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



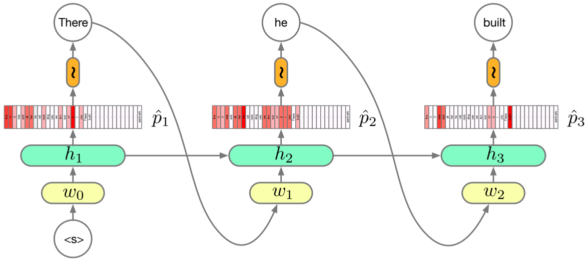
Recurrent Neural Network Language Models

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



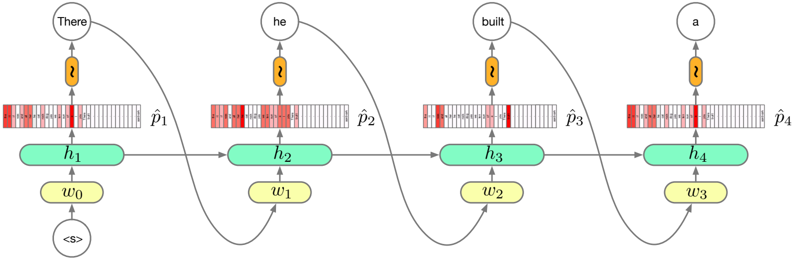
Recurrent Neural Network Language Models

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



Recurrent Neural Network Language Models

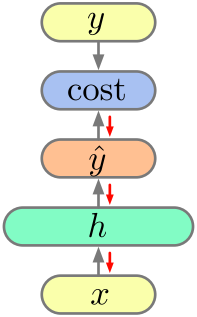
$$h_n = g(V[x_n; h_{n-1}] + c)$$



Recurrent Neural Network Language Models

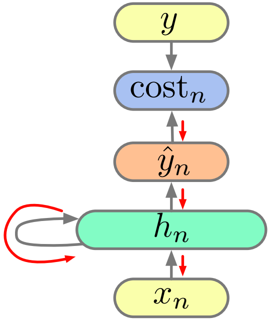
Feed Forward

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



Recurrent Network

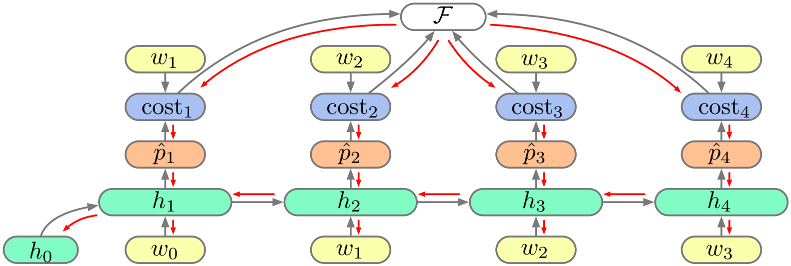
$$\begin{aligned} h_n &= g(V[x_n; h_{n-1}] + c) \\ \hat{y}_n &= Wh_n + b \end{aligned}$$



Recurrent Neural Network Language Models

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

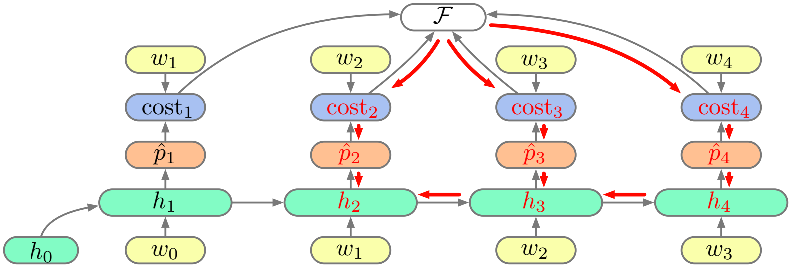
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

This algorithm is called Back Propagation Through Time (BPTT). Note the dependence of derivatives at time n with those at time $n + \alpha$:

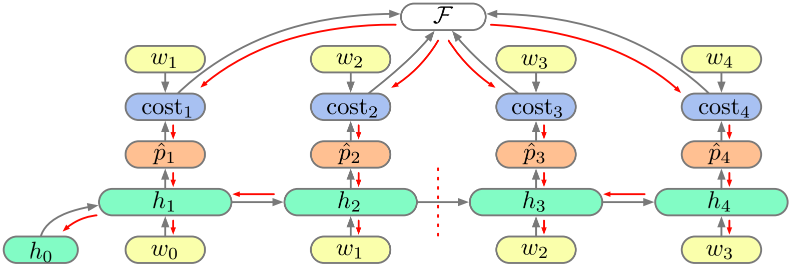
$$\frac{\partial \mathcal{F}}{\partial h_2} = \frac{\partial \mathcal{F}}{\partial \text{cost}_2}\frac{\partial \text{cost}_2}{\partial \hat{p}_2}\frac{\partial \hat{p}_2}{\partial h_2} + \frac{\partial \mathcal{F}}{\partial \text{cost}_3}\frac{\partial \text{cost}_3}{\partial \hat{p}_3}\frac{\partial \hat{p}_3}{\partial h_3}\frac{\partial h_3}{\partial h_2} + \frac{\partial \mathcal{F}}{\partial \text{cost}_4}\frac{\partial \text{cost}_4}{\partial \hat{p}_4}\frac{\partial \hat{p}_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}$$



Recurrent Neural Network Language Models

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

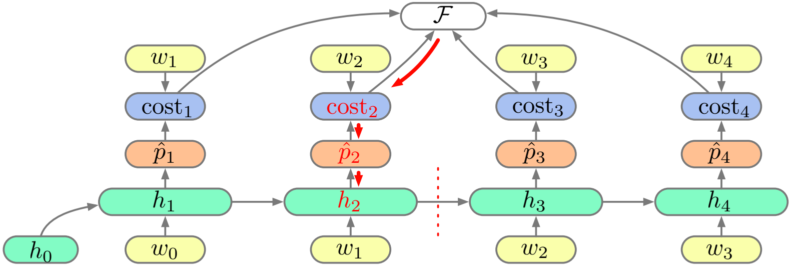
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

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

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

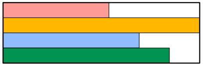


RNNs: Minibatching and Complexity

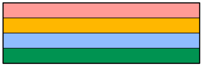
Mini-batching on a GPU is an effective way of speeding up big matrix vector products. RNNLMs have two such products that dominate their computation: the recurrent matrix V and the softmax matrix W.

Complexity:

BPTT Linear in the length of the longest sequence. Minibatching can be inefficient as the sequences in a batch may have different lengths.



TBPTT Constant in the truncation length T. Efficient for mini-batching as all sequences have length T.



More on this in the GPU lecture next week.

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 (more on this next lecture).
- Increasing the size of the hidden layer, and thus memory, increases the computation and memory quadratically.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.

Bias vs Variance in LM Approximations

The main issue in language modelling is compressing the history (a string). This is useful beyond language modelling in classification and representation tasks (more next week).

- With n-gram models we approximate the history with only the last n words.
- With recurrent models (RNNs, next) we compress the unbounded history into a fixed sized vector.

We can view this progression as the classic Bias vs. Variance tradeoff in ML. N-gram models are biased but low variance. RNNs decrease the bias considerably, hopefully at a small cost to variance.

Consider predicting the probability of a sentence by how many times you have seen it before. This is an unbiased estimator with (extremely) high variance.

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`

In the next lecture I will discuss the architechtural and algorithmic solutions to issues encountered when training RNNs.