Natural Language Processing, Language Modelling and Machine Translation

Phil Blunsom

in collaboration with the

DeepMind Natural Language Group

pblunsom@google.com

Natural Language Processing

Linguistics

Why are human languages the way that they are? How does the brain map from raw linguistic input to meaning and back again? And how do children learn language so quickly?

Computational Linguistics

Computational models of language and computational tools for studying language.

Natural Language Processing

Building tools for processing language and applications that use language:

- Intrinsic: Parsing, **Language Modelling**, etc.
- Extrinsic: ASR, **MT**, QA/Dialogue, etc.

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?

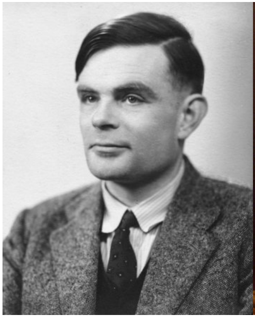
Thus we can compare different orderings of words (e.g. Translation):

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

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

Language Modelling Overview

In this lecture I 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:

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

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:

- Montreal beer eater
- Montreal beer 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:

- beer eater
- beer 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.

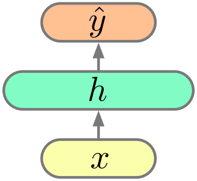
[5]Heaps' Law: `en.wikipedia.org/wiki/Heaps'_law`

Outline

Neural Language Models

Feed forward network

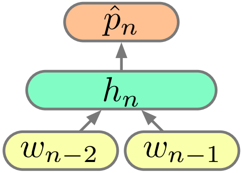
$$
\begin{aligned}
h &= g(Vx + c) \\
\hat{y} &= Wh + b
\end{aligned}
$$



Neural Language Models

Trigram NN language model

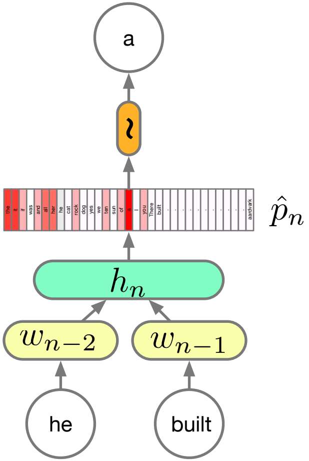
$$\begin{aligned} h_n &= g(V[w_{n-1}; w_{n-2}] + c) \\ \hat{p}_n &= \text{softmax}(W h_n + b) \\ \text{softmax}(u)_i &= \frac{\exp u_i}{\sum_j \exp u_j} \end{aligned}$$



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

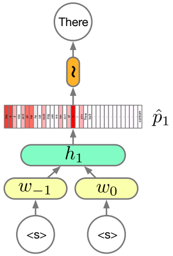
Neural Language Models: Sampling

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



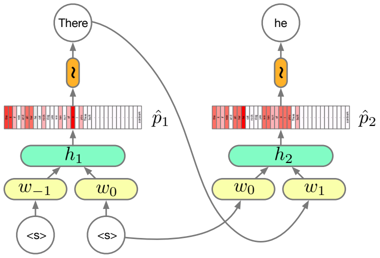
Neural Language Models: Sampling

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



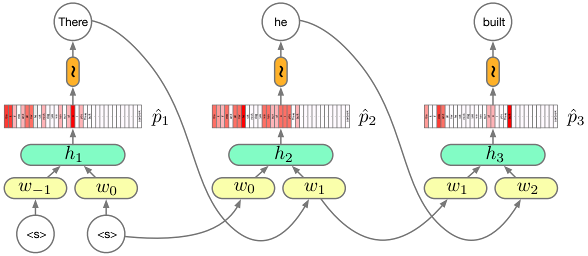
Neural Language Models: Sampling

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



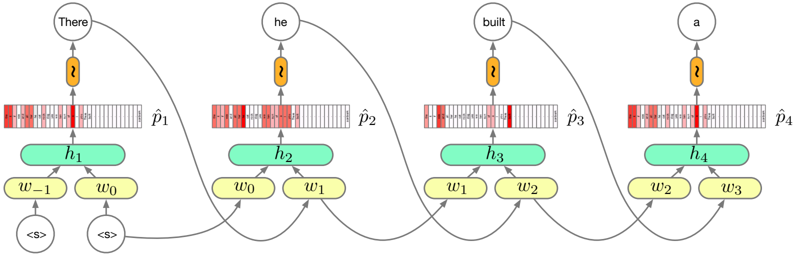
Neural Language Models: Sampling

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



Neural Language Models: Sampling

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



Neural Language Models: Training

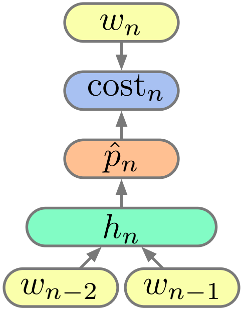
The usual training objective is the cross entropy of the data given the model (MLE):

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

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)

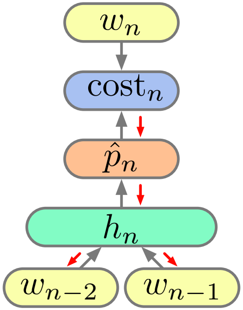


Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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

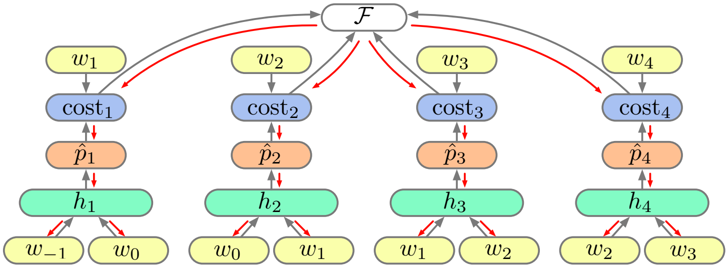
$$\frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{N}\sum_n \frac{\partial \mathrm{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$



Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

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



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

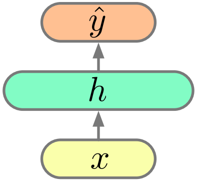
- The number of parameters in the model scales with the n-gram size and thus the length of the history captured.
- The n-gram history is finite and thus there is a limit on the longest dependencies that an be captured.
- Mostly trained with Maximum Likelihood based objectives which do not encode the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

Outline

Recurrent Neural Network Language Models

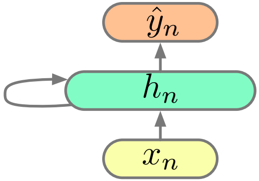
Feed Forward

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



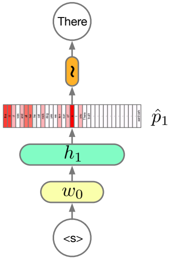
Recurrent Network

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



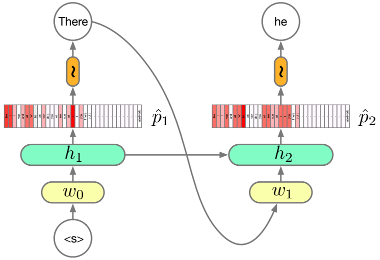
Recurrent Neural Network Language Models

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



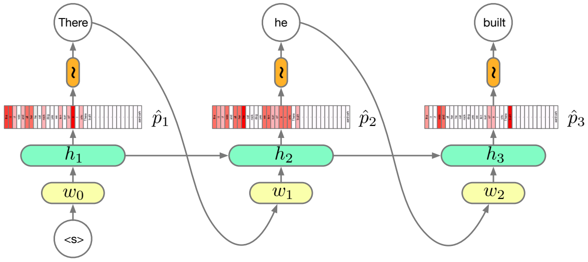
Recurrent Neural Network Language Models

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



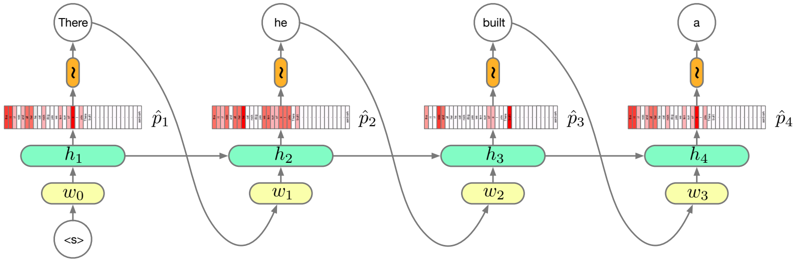
Recurrent Neural Network Language Models

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



Recurrent Neural Network Language Models

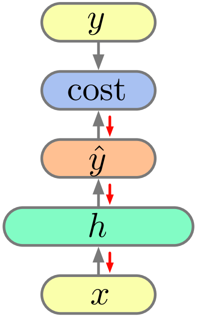
$$h_n = g(V[x_n; h_{n-1}] + c)$$



Recurrent Neural Network Language Models

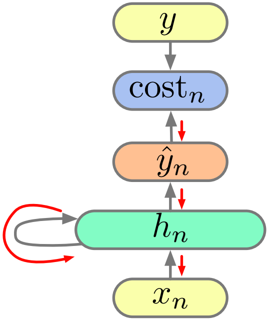
Feed Forward

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



Recurrent Network

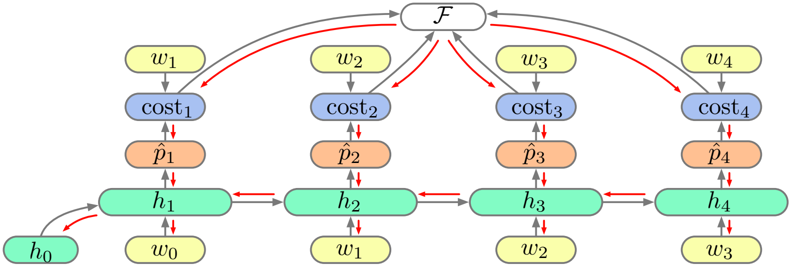
$$\begin{aligned} h_n &= g(V[x_n; h_{n-1}] + c) \\ \hat{y}_n &= Wh_n + b \end{aligned}$$



Recurrent Neural Network Language Models

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

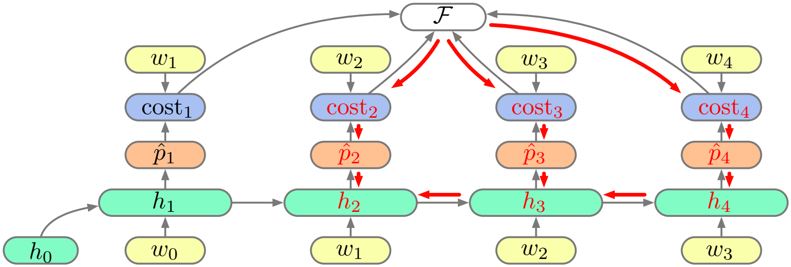
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \mathrm{cost}_n(w_n, \hat{p}_n)$$



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 $n + \alpha$:

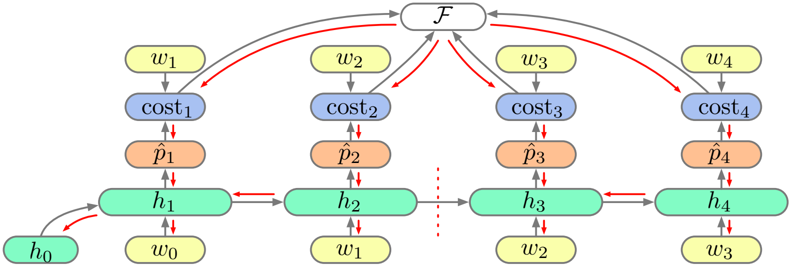
$$\frac{\partial \mathcal{F}}{\partial h_2} = \frac{\partial \mathcal{F}}{\partial \mathrm{cost}_2}\frac{\partial \mathrm{cost}_2}{\partial \hat{p}_2}\frac{\partial \hat{p}_2}{\partial h_2} + \frac{\partial \mathcal{F}}{\partial \mathrm{cost}_3}\frac{\partial \mathrm{cost}_3}{\partial \hat{p}_3}\frac{\partial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



Recurrent Neural Network Language Models

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

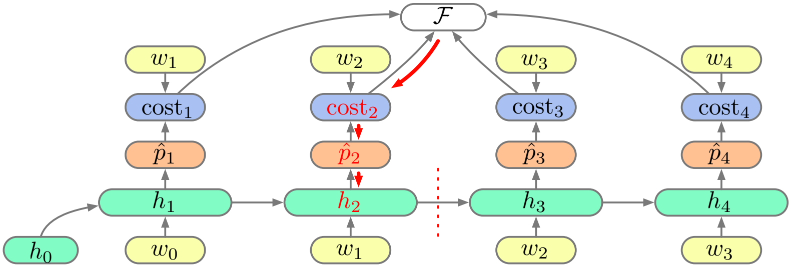
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

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

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



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.

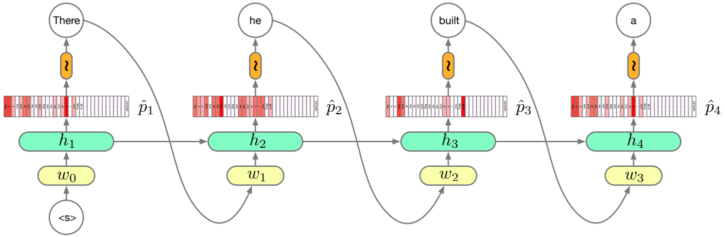
Bad

- 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.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.

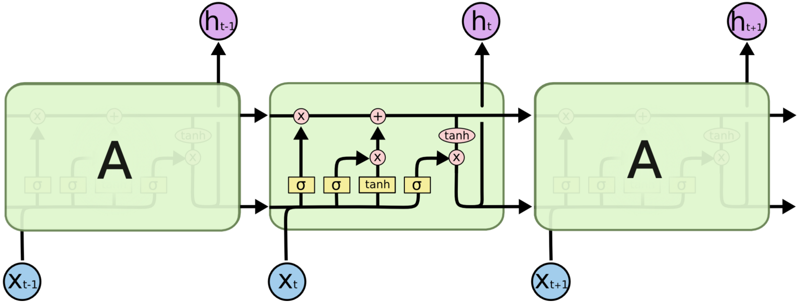
Language Modelling: Review

Language models aim to represent the history of observed text $(w_1, \ldots, w_{t-1})$ succinctly in order to predict the next word (w_t):

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



Gated Units: LSTMs and GRUs

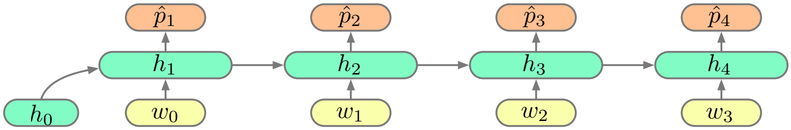


Christopher Olah: Understanding LSTM Networks
`colah.github.io/posts/2015-08-Understanding-LSTMs/`

Deep RNN LMs

The memory capacity of an RNN can be increased by employing a larger hidden layer h_n, but a linear increase in h_n results in a quadratic increase in model size and computation.

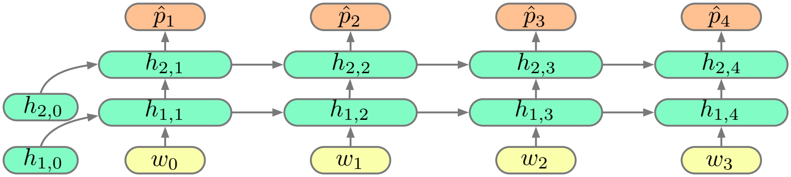
A Deep RNN increases the memory and representational ability with linear scaling.



Deep RNN LMs

The memory capacity of an RNN can be increased by employing a larger hidden layer h_n, but a linear increase in h_n results in a quadratic increase in model size and computation.

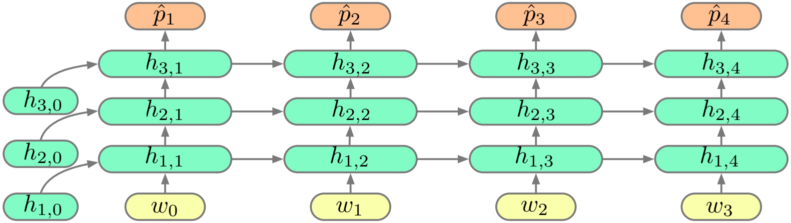
A Deep RNN increases the memory and representational ability with linear scaling.



Deep RNN LMs

The memory capacity of an RNN can be increased by employing a larger hidden layer h_n, but a linear increase in h_n results in a quadratic increase in model size and computation.

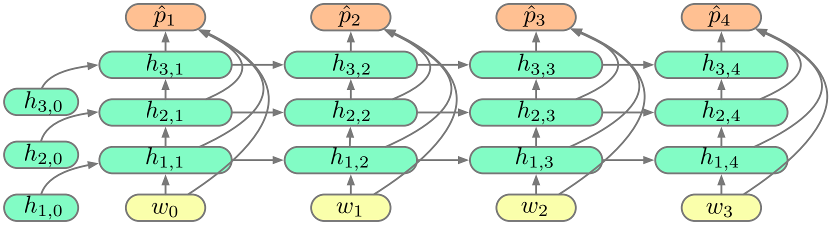
A Deep RNN increases the memory and representational ability with linear scaling.



Deep RNN LMs

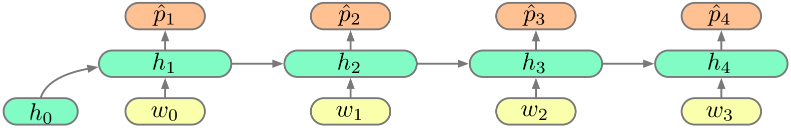
The memory capacity of an RNN can be increased by employing a larger hidden layer h_n, but a linear increase in h_n results in a quadratic increase in model size and computation.

A Deep RNN increases the memory and representational ability with linear scaling.



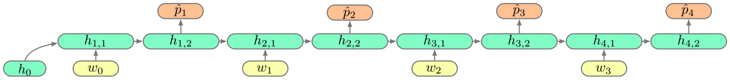
Deep RNN LM

Alternatively we can increase depth in the time dimension. This improves the representational ability, but not the memory capacity.



Deep RNN LM

Alternatively we can increase depth in the time dimension. This improves the representational ability, but not the memory capacity.



The recently proposed Recurrent Highway Network[6] employs a deep-in-time GRU-like cell with untied weights, and reports strong results on language modelling.

[6]Recurrent Highway Networks. Zilly et al., arXiv 2016.

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \mathrm{softmax}\left(W h_n + b\right)$$

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \text{softmax}\left(Wh_n + b\right)$$

Solutions

Short-lists: use the neural LM for the most frequent words, and a traditional ngram LM for the rest. While easy to implement, this nullifies the neural LM's main advantage, i.e. generalisation to rare events.

Batch local short-lists: approximate the full partition function for data instances from a segment of the data with a subset of the vocabulary chosen for that segment.[7]

[7] On Using Very Large Target Vocabulary for Neural Machine Translation. Jean et al., ACL 2015

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \text{softmax}\left(Wh_n + b\right)$$

Solutions

Approximate the gradient/change the objective: if we did not have to sum over the vocabulary to normalise during training it would be much faster. It is tempting to consider maximising likelihood by making the log partition function an independent parameter c, but this leads to an ill defined objective.

$$\hat{p}_n \equiv \exp\left(Wh_n + b\right) \times \exp(c)$$

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \text{softmax}\left(Wh_n + b\right)$$

Solutions

Approximate the gradient/change the objective: Mnih and Teh use Noise Contrastive Estimation (NCE). This amounts to learning a binary classifier to distinguish data samples from (k) samples from a noise distribution (a unigram is a good choice):

$$p(\text{Data} = 1|\hat{p}_n) = \frac{\hat{p}_n}{\hat{p}_n + kp_{\text{noise}}(w_n)}$$

Now parametrising the log partition function as c does not degenerate. This is very effective for speeding up training, but has no impact on testing time.[7]

[7] In practice fixing $c = 0$ is effective. It is tempting to believe that this noise contrastive objective justifies using unnormalised scores at test time. This is not the case and leads to high variance results.

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \text{softmax}\left(Wh_n + b\right)$$

Solutions

Approximate the gradient/change the objective: NCE defines a binary classification task between true or noise words with a logistic loss. An alternative, called Importance Sampling (IS)[7][8], defines a multiclass classification problem between the true word and noise samples, with a Softmax and cross entropy loss.

[7] Quick Training of Probabilistic Neural Nets by Importance Sampling. Bengio and Senecal. AISTATS 2003

[8] Exploring the Limits of Language Modeling. Jozefowicz et al., arXiv 2016.

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \mathrm{softmax}\left(Wh_n + b\right)$$

Solutions

Factorise the output vocabulary: One level factorisation works well (Brown clustering is a good choice, frequency binning is not):

$$p(w_n|\hat{p}_n^{\text{class}}, \hat{p}_n^{\text{word}}) = p(\text{class}(w_n)|\hat{p}_n^{\text{class}}) \times p(w_n|\text{class}(w_n), \hat{p}_n^{\text{word}}),$$

where the function class($\cdot$) maps each word to one class. Assuming balanced classes, this gives a $\sqrt{V}$ speedup.

Scaling: Large Vocabularies

Much of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating:

$$\hat{p}_n = \mathrm{softmax}\left(Wh_n + b\right)$$

Solutions

Factorise the output vocabulary: By extending the factorisation to a binary tree (or code) we can get a $\log V$ speedup,[7][8] but choosing a tree is hard (frequency based Huffman coding is a poor choice):

$$p(w_n|h_n) = \prod_i p(d_i|r_i, h_n),$$

where d_i is i^{th} digit in the code for word w_n, and r_i is the parameter vector for the i^{th} node in the path corresponding to that code.

Recently Grave et al. proposed optimising an n-ary factorisation tree for both perplexity and GPU throughput.[9]

[7] Hierarchical Probabilistic Neural Network Language Model. Morin and Bengio. AISTATS 2005.

[8] A scalable hierarchical distributed language model. Mnih and Hinton, NIPS'09.

[9] Efficient softmax approximation for GPUs. Grave et al., arXiv 2016

Scaling: Large Vocabularies

Full Softmax

Training: Computation and memory $O(V)$,
Evaluation: Computation and memory $O(V)$,
Sampling: Computation and memory $O(V)$.

Balanced Class Factorisation

Training: Computation $O(\sqrt{V})$ and memory $O(V)$,
Evaluation: Computation $O(\sqrt{V})$ and memory $O(V)$,
Sampling: Computation and memory $O(V)$ (but average case is better).

Balanced Tree Factorisation

Training: Computation $O(\log V)$ and Memory $O(V)$,
Evaluation: Computation $O(\log V)$ and Memory $O(V)$,
Sampling: Computation and Memory $O(V)$ (but average case is better).

NCE / IS

Training: Computation $O(k)$ and Memory $O(V)$,
Evaluation: Computation and Memory $O(V)$,
Sampling: Computation and Memory $O(V)$.

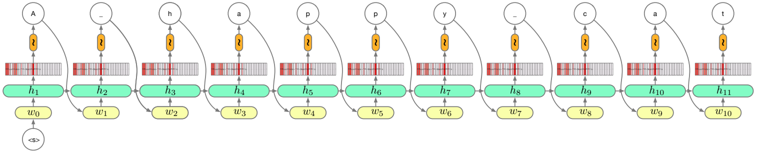
Sub-Word Level Language Modelling

An alternative to changing the softmax is to change the input granularity and model text at the morpheme or character level.

This results in a much smaller softmax and no unknown words, but the downsides are longer sequences and longer dependencies.

This also allows the model to capture subword structure and morphology: *disunited* $\leftrightarrow$ *disinherited* $\leftrightarrow$ *disinterested*.

Charater LMs lag word based models in perplexity, but are clearly the future of language modelling.



A
_
h
a
p
p
y
_
c
a
t
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11}
w_0 w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10}
<s>

Regularisation: Dropout

Large recurrent networks often overfit their training data by memorising the sequences observed. Such models generalise poorly to novel sequences.

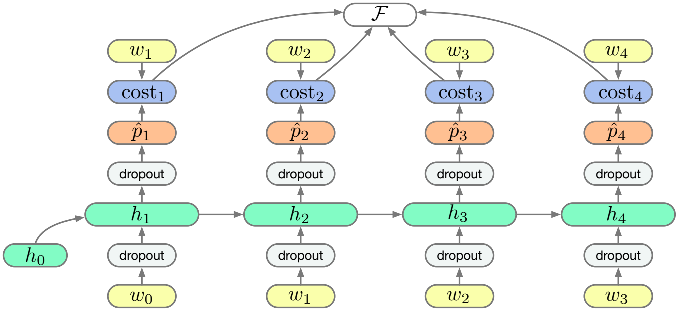
A common approach in Deep Learning is to overparametrise a model, such that it could easily memorise the training data, and then heavily regularise it to facilitate generalisation.

The regularisation method of choice is often Dropout.[10]

[10] Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Srivastava et al. JMLR 2014.

Regularisation: Dropout

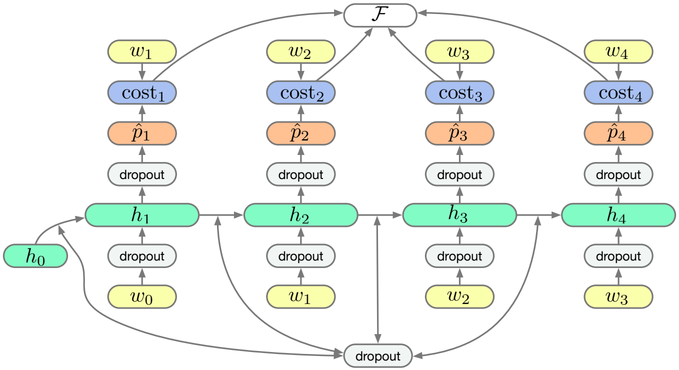
Dropout is ineffective when applied to recurrent connections, as repeated random masks zero all hidden units in the limit. The most common solution is to only apply dropout to non-recurrent connections.[11]



[11] Recurrent neural network regularization. Zaremba et al., arXiv 2014.

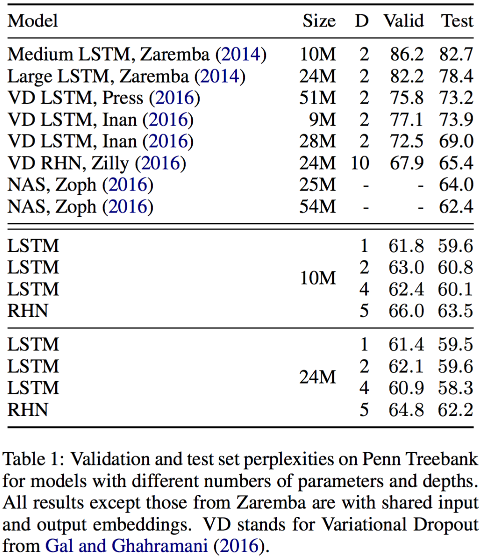
Regularisation: Bayesian Dropout (Gal)

Gal and Ghahramani[12] advocate tying the recurrent dropout mask and sampling at evaluation time:



[12]A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. Gal and Ghahramani, NIPS 2016.

Evaluation: hyperparamters are a confounding factor



Model	Size	D	Valid	Test
Medium LSTM, Zaremba (2014)	10M	2	86.2	82.7
Large LSTM, Zaremba (2014)	24M	2	82.2	78.4
VD LSTM, Press (2016)	51M	2	75.8	73.2
VD LSTM, Inan (2016)	9M	2	77.1	73.9
VD LSTM, Inan (2016)	28M	2	72.5	69.0
VD RHN, Zilly (2016)	24M	10	67.9	65.4
NAS, Zoph (2016)	25M	-	-	64.0
NAS, Zoph (2016)	54M	-	-	62.4
LSTM	10M	1	61.8	59.6
LSTM		2	63.0	60.8
LSTM		4	62.4	60.1
RHN		5	66.0	63.5
LSTM	24M	1	61.4	59.5
LSTM		2	62.1	59.6
LSTM		4	60.9	58.3
RHN		5	64.8	62.2

Table 1: Validation and test set perplexities on Penn Treebank for models with different numbers of parameters and depths. All results except those from Zaremba are with shared input and output embeddings. VD stands for Variational Dropout from Gal and Ghahramani (2016).

Summary

Long Range Dependencies

- The repeated multiplication of the recurrent weights V lead to vanishing (or exploding) gradients,
- additive gated architectures, such as LSTMs, significantly reduce this issue.

Deep RNNs

- Increasing the size of the recurrent layer increases memory capacity with a quadratic slow down,
- deepening networks in both dimensions can improve their representational efficiency and memory capacity with a linear complexity cost.

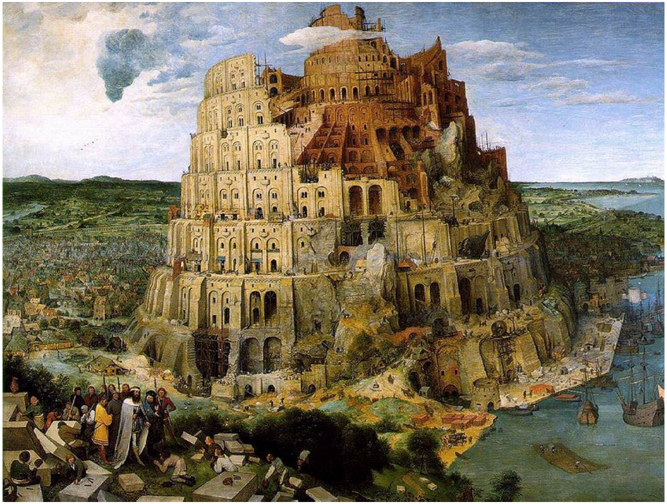
Large Vocabularies

- Large vocabularies, $V > 10^4$, lead to slow softmax calculations,
- reducing the number of vector matrix products evaluated, by factorising the softmax or sampling, reduces the training overhead significantly.
- Different optimisations have different training and evaluation complexities which should be considered.

Outline

Intro to MT

The confusion of tongues:



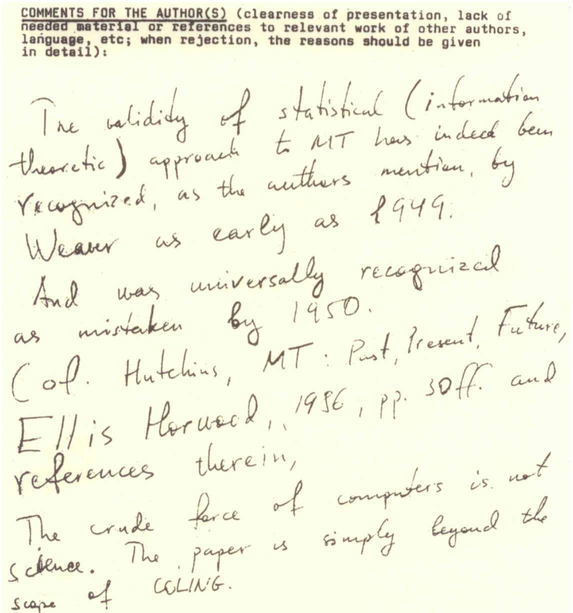
Parallel Corpora

MT History: Statistical MT at IBM

Fred Jelinek, 1988:

"Every time I fire a linguist, the performance of the recognizer goes up."

MT History: Statistical MT at IBM



COMMENTS FOR THE AUTHOR(S) (clearness of presentation, lack of needed material or references to relevant work of other authors, language, etc; when rejection, the reasons should be given in detail):

The validity of statistical (information theoretic) approach to MT has indeed been recognized, as the authors mention, by Weaver as early as 1949. And was universally recognized as mistaken by 1950. (cf. Hutchins, MT: Past, Present, Future, Ellis Horwood, 1986, pp. 30ff. and references therein).

The crude force of computers is not science. The paper is simply beyond the scope of COLING.

Models of translation

The Noisy Channel Model

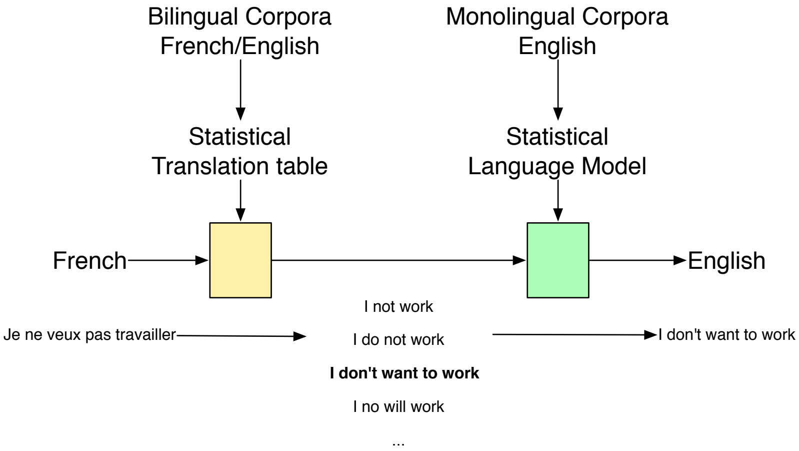
$$P(English|French) = \frac{P(English) \times P(French|English)}{P(French)}$$

$$\underset{\mathbf{e}}{\operatorname{argmax}} P(\mathbf{e}|\mathbf{f}) = \underset{\mathbf{e}}{\operatorname{argmax}} \left[P(\mathbf{e}) \times P(\mathbf{f}|\mathbf{e})\right]$$

- Bayes' rule is used to reverse the translation probabilities
- the analogy is that the French is English transmitted over a *noisy channel*
- we can then use techniques from statistical signal processing and decryption to translate

Models of translation

The Noisy Channel Model



IBM Model 1: The first translation attention model!

A simple generative model for $p(\mathbf{s}|\mathbf{t})$ is derived by introducing a latent variable $\mathbf{a}$ into the conditional probabiliy:

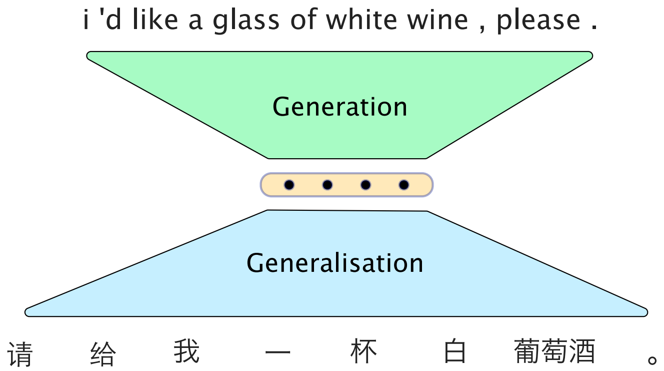
$$p(\mathbf{s}|\mathbf{t}) = \sum_{\mathbf{a}} \frac{p(J|I)}{(I+1)^J} \prod_{j=1}^{J} p(s_j|t_{a_j}),$$

where:

- $\mathbf{s}$ and $\mathbf{t}$ are the input (source) and output (target) sentences of length J and I respectively,
- $\mathbf{a}$ is a vector of length J consisting of integer indexes into the target sentence, known as the alignment,
- $p(J|I)$ is not importent for training the model and we'll treat it as a constant ϵ.

To learn this model we use the EM algorithm to find the MLE values for the parameters $p(s_j|t_{a_j})$.

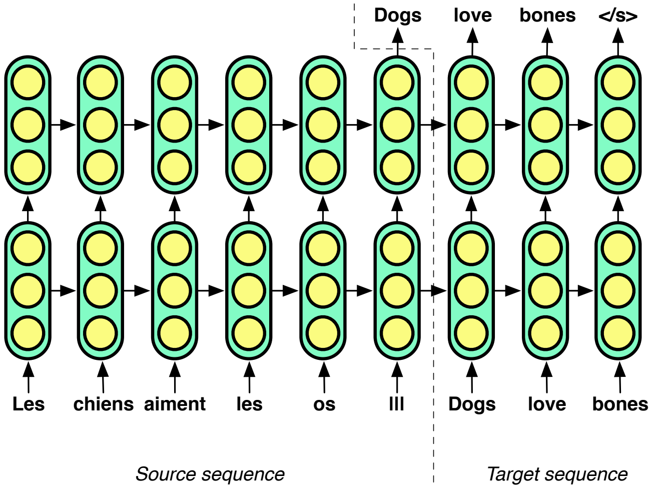
Encoder-Decoders[13]


i 'd like a glass of white wine , please .
Generation
Generalisation
请 给 我 一 杯 白 葡萄酒 。

13

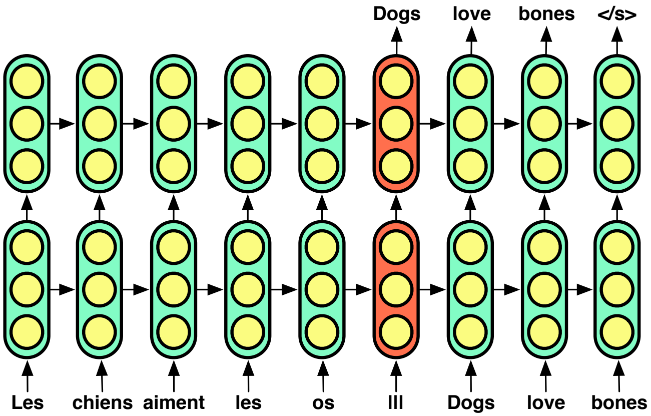
Recurrent Continuous Translation Models. Kalchbrenner and Blunsom, EMNLP'13
Sequence to Sequence Learning with Neural Networks. Sutskever et al., NIPS'14
Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Recurrent Encoder-Decoders for MT[14]



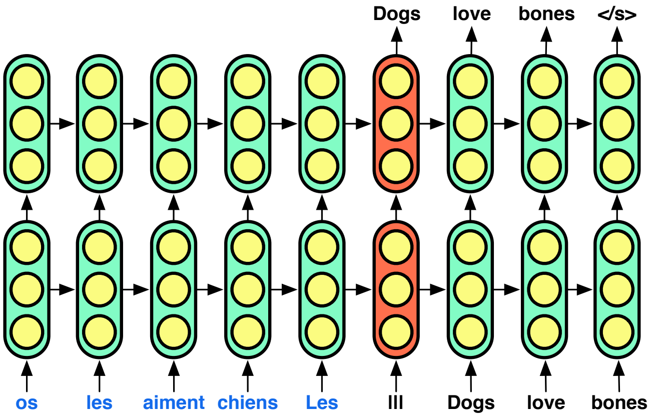
[14] Sequence to Sequence Learning with Neural Networks. Sutskever et al., NIPS'14

Recurrent Encoder-Decoders for MT[14]



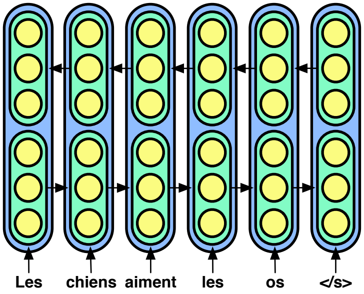
[14] Sequence to Sequence Learning with Neural Networks. Sutskever et al., NIPS'14

Recurrent Encoder-Decoders for MT[14]



[14] Sequence to Sequence Learning with Neural Networks. Sutskever et al., NIPS'14

Attention Models for MT[15]

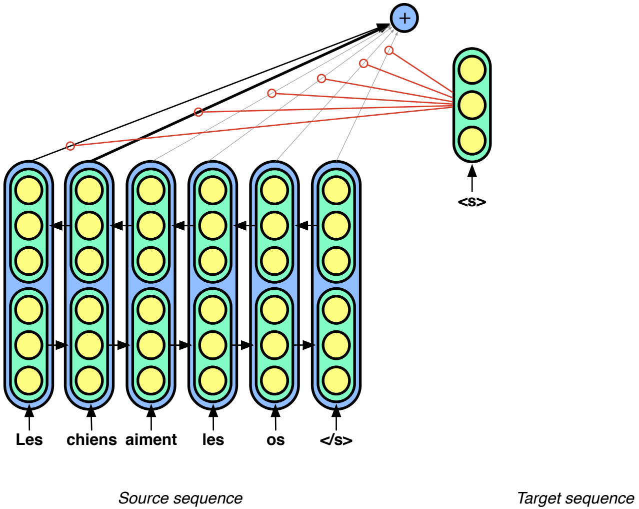


Source sequence

Target sequence

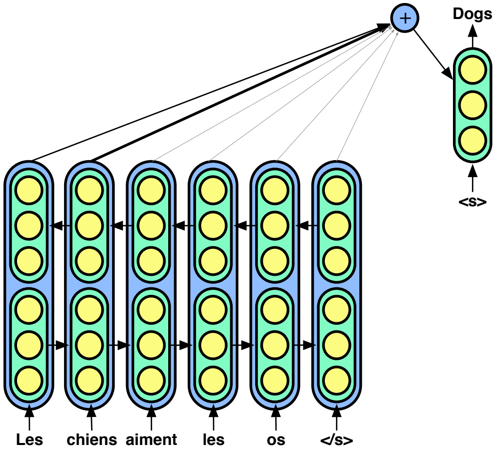
[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Attention Models for MT[15]



[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

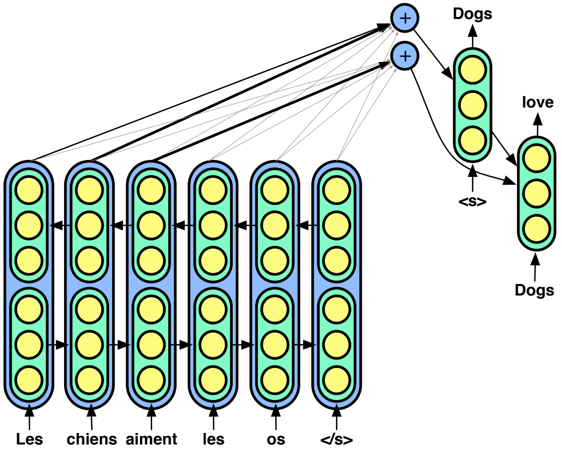
Attention Models for MT[15]



Source sequence *Target sequence*

[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Attention Models for MT[15]

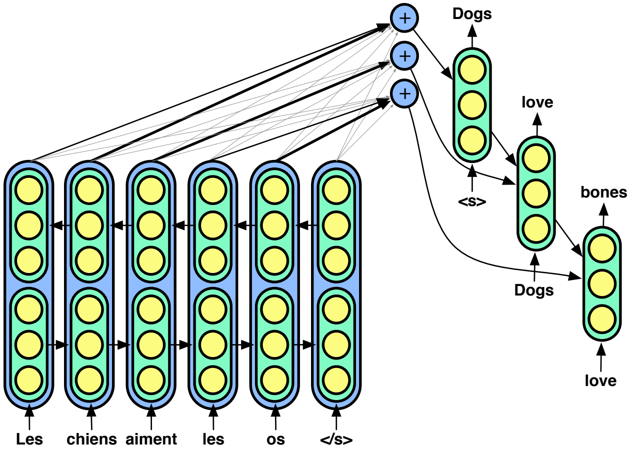


Source sequence

Target sequence

[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Attention Models for MT[15]

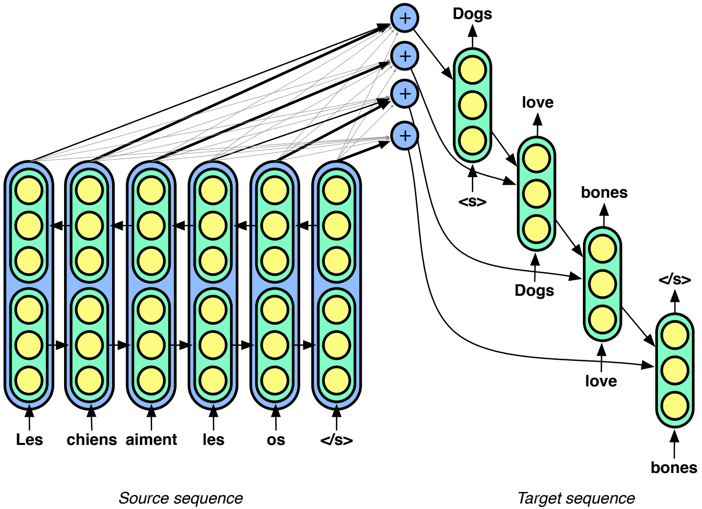


Source sequence

Target sequence

[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Attention Models for MT[15]



[15] Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al., ICLR'15

Returning to the Noisy Channel

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}) &= EncDecRNN(\mathbf{x}) \\ &= \prod_i p(y_i|\mathbf{x}, \mathbf{y}_{<i}) \end{aligned}$$

- Lots of (x,y) pairs $\rightarrow$ Great performance.
- Two serious problems with direct models:
  1. Can't use of unpaired x's and y's (and unpaired data is cheaper/and often naturally abundant)
  2. "Explaining away of inputs": models learn to ignore difficult input in favor of high probability continuations of partial input prefixes ("label bias")

Returning to the Noisy Channel

$$p(y|x) \propto \underbrace{p(y)}_{RNN-LM} \times \underbrace{p(x|y)}_{EncDecRNN}$$

Features:

- Models can be parameterised, trained, and even deployed separately.
- Make principled use of unpaired output data.
- Outputs have to explain the input: helps mitigate risks due to explaining away of inputs
- Training – straightforward.
- Decoding – hard.

Decoding

Searching for the best translation:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\mathrm{argmax}} p(\mathbf{y}|\mathbf{x})$$

Challenges:

- Hypothesis space is very large ($\Sigma*$ in fact)
- We need to factorise the search problem
- This is easier to do in the direct model than in the noisy channel model
- (And it's still a hard problem–we can only solve it approximately)

Decoding: Direct vs. Noisy Channel

Direct Model:

```
while $y_i \neq \text{STOP}$:
    $\hat{y}_i = \underset{y}{\text{argmax}}\, \underbrace{p(y|\mathbf{x}, \hat{\mathbf{y}}_{<i})}_{\text{Chain Rule}}$
    $i \leftarrow i + 1$
```

Greedy maximisation provides an reasonable approximation:

$$\hat{\mathbf{y}} \approx \underset{\mathbf{y}}{\text{argmax}} p(\mathbf{y}|\mathbf{x})$$

Decoding: Direct vs. Noisy Channel

Noisy Channel Model:

```
while $y_i \neq \text{STOP}$:
    $\hat{y}_i = \operatorname*{argmax}_y \underbrace{p(y \mid \hat{\mathbf{y}}_{<i}) \times p(\mathbf{x} \mid \hat{\mathbf{y}}_{<i}, y)}_{\text{This is not how probability works!}}$
    $i \leftarrow i + 1$
```

Decoding: Noisy Channel Model

Solution: We introduce an alignment latent variable $\mathbf{z}$ that determines when enough of the input has been read to produce another output:

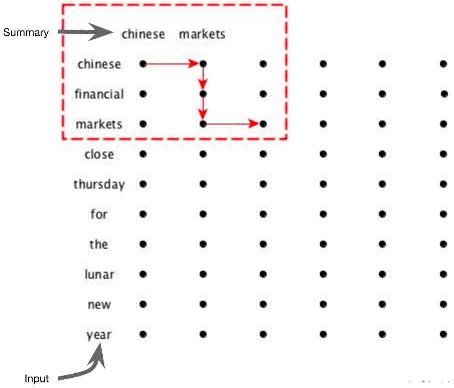
$$p(\mathbf{x}|\mathbf{y}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\mathbf{y})$$

$$p(\mathbf{x}, \mathbf{z}|\mathbf{y}) \approx \prod_{j=1}^{|\mathbf{x}|} p(z_j|z_{j-1}, \mathbf{y}_1^{z_j}, \mathbf{x}_1^{j-1}) p(x_j|\mathbf{y}_1^{z_j}, \mathbf{x}_1^{j-1})$$

z_j records how much of $\mathbf{y}$ we need to read to predict the j^{th} token of $\mathbf{x}$.

Segment to Segment Neural Transduction

- Introduced as a direct model by Yu et al. (2016),
- a strong online Encoder Decoder model,
- when reversed it is exactly what we need for a channel model,
- similar to Graves (2012).

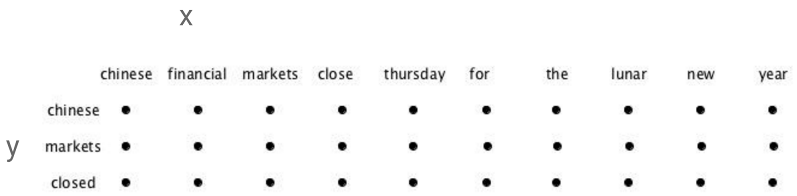


Noisy Channel Decoding

- Expensive to go through every token y_j in the vocabulary and calculate:

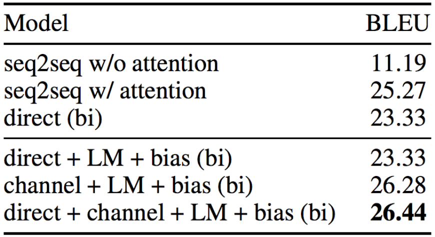
$$p(x_{1:i}|y_{1:j})p(y_{1:j})$$

- Use the direct model $p(y|x)$ to guide the search.



Relative Performance[16]

The noisy channel model performs strongly on sentence compression and morphological inflection. For MT it provide a principled way to incorporate large language models:



Model	BLEU
seq2seq w/o attention	11.19
seq2seq w/ attention	25.27
direct (bi)	23.33
direct + LM + bias (bi)	23.33
channel + LM + bias (bi)	26.28
direct + channel + LM + bias (bi)	**26.44**

BLEU scores from different models for the Chinese to English machine translation task.

[16] *Yu et al. The Neural Noisy Channel. ICLR 2017.*

The End