“Sequence to Sequence Learning with Neural Networks” (I. Sutskever et al., 2014)

for the seminar *Deep Time-Series Learning with Finance Applications*, organized by L. El Ghaoui & F. Belletti, Fall 2017, UC Berkeley

Nima Hejazi

Division of Biostatistics,
University of California, Berkeley

07 November 2017

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Preview

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

The Data and Objective

- The WMT'14 English to French dataset was used.
- Models were trained on a "selected" subset of 12M sentences, consisting of 348M French words and 304M English words.
- This translation task and the specific subset was chosen based on the availability of a tokenized training and test set and other benchmarks.
- A *fixed* vocabulary was used for both languages — that is, a set of $160,000$ of the most frequent words for the source language and $80,000$ for the target language were used.

The Data and Objective

- The WMT'14 English to French dataset was used.
- Models were trained on a "selected" subset of 12M sentences, consisting of 348M French words and 304M English words.
- This translation task and the specific subset was chosen based on the availability of a tokenized training and test set and other benchmarks.
- A *fixed* vocabulary was used for both languages — that is, a set of $160,000$ of the most frequent words for the source language and $80,000$ for the target language were used.

The Data and Objective

- The WMT'14 English to French dataset was used.
- Models were trained on a "selected" subset of 12M sentences, consisting of 348M French words and 304M English words.
- This translation task and the specific subset was chosen based on the availability of a tokenized training and test set and other benchmarks.
- A *fixed* vocabulary was used for both languages — that is, a set of $160,000$ of the most frequent words for the source language and $80,000$ for the target language were used.

The Data and Objective

- The WMT'14 English to French dataset was used.
- Models were trained on a "selected" subset of 12M sentences, consisting of 348M French words and 304M English words.
- This translation task and the specific subset was chosen based on the availability of a tokenized training and test set and other benchmarks.
- A *fixed* vocabulary was used for both languages — that is, a set of $160,000$ of the most frequent words for the source language and $80,000$ for the target language were used.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \text{sigm}(W^{hx} x_t + W^{hh} h_{t-1})$
  2. $y_t = W^{yh} h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \mathrm{sigm}(W^{hx} x_t + W^{hh} h_{t-1})$
  2. $y_t = W^{yh} h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \mathrm{sigm}(W^{hx} x_t + W^{hh} h_{t-1})$
  2. $y_t = W^{yh} h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \text{sigm}(W^{hx} x_t + W^{hh} h_{t-1})$
  2. $y_t = W^{yh} h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \mathrm{sigm}(W^{hx} x_t + W^{hh} h_{t-1})$
  2. $y_t = W^{yh} h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model I: Recurrent Neural Networks

- RNNs are a straightforward generalization of feedforward neural networks for sequences.
- For a sequence of inputs $(x_1, \ldots, x_T)$, an RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating over
  1. $h_t = \mathrm{sigm}(W^{hx}x_t + W^{hh}h_{t-1})$
  2. $y_t = W^{yh}h_t$
- Note from the above that it is not immediately clear how to apply this procedure over inputs and outputs of differing lengths — in fact, such a procedure would **not** be simple.
- This problem is made even more severe when considering that inputs and outputs may have a complex and (very likely) non-monotonic relationship.

The Model II: Long Short-Term Memory Networks

- LSTMs overcome the problems faced by RNNs, providing a relatively simple way to learn in settings with long-range temporal dependencies.

- The LSTM operates by estimating $p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T)$, where the lengths T and T' need not be identical.

- The approach employed here works in two simple steps:
  1. Obtain a fixed-dimensional representation v of $(x_1, \ldots, x_T)$ (the last hidden state of the LSTM).
  2. Compute the probability of $y_1, \ldots, y_{T'}$ by a standard LSTM-LM formulation:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_1, \ldots, y_{t-1}),$$

     where the distribution in the likelihood is represented by a softmax over all the words in the vocabulary.

The Model II: Long Short-Term Memory Networks

- LSTMs overcome the problems faced by RNNs, providing a relatively simple way to learn in settings with long-range temporal dependencies.

- The LSTM operates by estimating $p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T)$, where the lengths T and T' need not be identical.

- The approach employed here works in two simple steps:
  1. Obtain a fixed-dimensional representation v of $(x_1, \ldots, x_T)$ (the last hidden state of the LSTM).
  2. Compute the probability of $y_1, \ldots, y_{T'}$ by a standard LSTM-LM formulation:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_1, \ldots, y_{t-1}),$$

  where the distribution in the likelihood is represented by a softmax over all the words in the vocabulary.

The Model II: Long Short-Term Memory Networks

- LSTMs overcome the problems faced by RNNs, providing a relatively simple way to learn in settings with long-range temporal dependencies.
- The LSTM operates by estimating $p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T)$, where the lengths T and T' need not be identical.
- The approach employed here works in two simple steps:
  1. Obtain a fixed-dimensional representation v of $(x_1, \ldots, x_T)$ (the last hidden state of the LSTM).
  2. Compute the probability of $y_1, \ldots, y_{T'}$ by a standard LSTM-LM formulation:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_1, \ldots, y_{t-1}),$$

     where the distribution in the likelihood is represented by a softmax over all the words in the vocabulary.

The Model II: Long Short-Term Memory Networks

- LSTMs overcome the problems faced by RNNs, providing a relatively simple way to learn in settings with long-range temporal dependencies.
- The LSTM operates by estimating $p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T)$, where the lengths T and T' need not be identical.
- The approach employed here works in two simple steps:
  1. Obtain a fixed-dimensional representation v of $(x_1, \ldots, x_T)$ (the last hidden state of the LSTM).
  2. Compute the probability of $y_1, \ldots, y_{T'}$ by a standard LSTM-LM formulation:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_1, \ldots, y_{t-1}),$$

     where the distribution in the likelihood is represented by a softmax over all the words in the vocabulary.

The Model II: Long Short-Term Memory Networks

- LSTMs overcome the problems faced by RNNs, providing a relatively simple way to learn in settings with long-range temporal dependencies.

- The LSTM operates by estimating $p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T)$, where the lengths T and T' need not be identical.

- The approach employed here works in two simple steps:
  1. Obtain a fixed-dimensional representation v of $(x_1, \ldots, x_T)$ (the last hidden state of the LSTM).
  2. Compute the probability of $y_1, \ldots, y_{T'}$ by a standard LSTM-LM formulation:

  $$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_1, \ldots, y_{t-1}),$$

  where the distribution in the likelihood is represented by a softmax over all the words in the vocabulary.

The Model III: Long Short-Term Memory Architectures

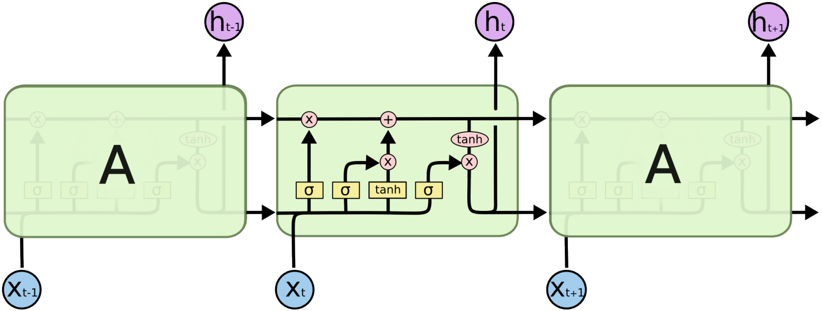

Figure 1: There are 4 interacting layers in the repeating module of an LSTM (source: C. Olah's blog)

The Process I: Decoding and Rescoring

- The experiment centered on training a large LSTM on French-English sentence pairs, with training performed by maximizing the log-probability of a correct translation.
- The training objective, over a training set $\mathcal{S}$, was

$$\frac{1}{\|\mathcal{S}\|} \sum_{(T,S) \in \mathcal{S}} \log p(T \mid S)$$

- After training, the LSTM was used to produce translations:

$$\hat{T} = \arg\max_{T} p(T \mid S)$$

- Beam search is used to find the most likely translation. A beam of size 1 gave good performance while a beam of size 2 provided nearly the same utility as full beam search.

The Process I: Decoding and Rescoring

- The experiment centered on training a large LSTM on French-English sentence pairs, with training performed by maximizing the log-probability of a correct translation.
- The training objective, over a training set $\mathcal{S}$, was

$$\frac{1}{\|\mathcal{S}\|} \sum_{(T,S)\in\mathcal{S}} \log p(T \mid S)$$

- After training, the LSTM was used to produce translations:

$$\hat{T} = \arg\max_{T} p(T \mid S)$$

- Beam search is used to find the most likely translation. A beam of size 1 gave good performance while a beam of size 2 provided nearly the same utility as full beam search.

The Process I: Decoding and Rescoring

- The experiment centered on training a large LSTM on French-English sentence pairs, with training performed by maximizing the log-probability of a correct translation.
- The training objective, over a training set $\mathcal{S}$, was

$$\frac{1}{\|\mathcal{S}\|} \sum_{(T,S)\in\mathcal{S}} \log p(T \mid S)$$

- After training, the LSTM was used to produce translations:

$$\hat{T} = \arg\max_{T} p(T \mid S)$$

- Beam search is used to find the most likely translation. A beam of size 1 gave good performance while a beam of size 2 provided nearly the same utility as full beam search.

The Process I: Decoding and Rescoring

- The experiment centered on training a large LSTM on French-English sentence pairs, with training performed by maximizing the log-probability of a correct translation.
- The training objective, over a training set $\mathcal{S}$, was

$$\frac{1}{\|\mathcal{S}\|} \sum_{(T,S) \in \mathcal{S}} \log p(T \mid S)$$

- After training, the LSTM was used to produce translations:

$$\hat{T} = \arg\max_{T} p(T \mid S)$$

- Beam search is used to find the most likely translation. A beam of size 1 gave good performance while a beam of size 2 provided nearly the same utility as full beam search.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process II: Source Reversal

- A **key finding** of this work was that reversing source sentences (without reversing target sentences) provided great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there is no "complete explanation" for this. 🤔
- How do such great gains arise from a simple manipulation?
  1. The "minimal time lag" is lowered — that is, when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance remains unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication" under this sort of dependency structure.
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.
- "The deepers" got me here — quite (read: too) empirical.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings (recall v from before).
- Input vocab.: $160,000$ words; output vocab.: $80,000$ words.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping model perplexity by nearly 10%.
- Mini-batches of size 128 sequences were used for the gradient, and the problem of exploding gradients was avoided using "clipping": $s = \|g\|_2$ was computed (where g is the gradient divided by minibatch size), and we set $g = 5 \cdot g/s$ if $s > 5$.
- Minibatches were set to have the same proportion of short and long sentences to help in training.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings (recall v from before).
- Input vocab.: $160,000$ words; output vocab.: $80,000$ words.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping model perplexity by nearly 10%.
- Mini-batches of size 128 sequences were used for the gradient, and the problem of exploding gradients was avoided using "clipping": $s = \|g\|_2$ was computed (where g is the gradient divided by minibatch size), and we set $g = 5 \cdot g/s$ if $s > 5$.
- Minibatches were set to have the same proportion of short and long sentences to help in training.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings (recall v from before).
- Input vocab.: $160,000$ words; output vocab.: $80,000$ words.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping model perplexity by nearly 10%.
- Mini-batches of size 128 sequences were used for the gradient, and the problem of exploding gradients was avoided using "clipping": $s = \|g\|_2$ was computed (where g is the gradient divided by minibatch size), and we set $g = 5 \cdot g/s$ if $s > 5$.
- Minibatches were set to have the same proportion of short and long sentences to help in training.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings (recall v from before).
- Input vocab.: $160,000$ words; output vocab.: $80,000$ words.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping model perplexity by nearly 10%.
- Mini-batches of size 128 sequences were used for the gradient, and the problem of exploding gradients was avoided using "clipping": $s = \|g\|_2$ was computed (where g is the gradient divided by minibatch size), and we set $g = 5 \cdot g/s$ if $s > 5$.
- Minibatches were set to have the same proportion of short and long sentences to help in training.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings (recall v from before).
- Input vocab.: $160,000$ words; output vocab.: $80,000$ words.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping model perplexity by nearly 10%.
- Mini-batches of size 128 sequences were used for the gradient, and the problem of exploding gradients was avoided using "clipping": $s = \|g\|_2$ was computed (where g is the gradient divided by minibatch size), and we set $g = 5 \cdot g/s$ if $s > 5$.
- Minibatches were set to have the same proportion of short and long sentences to help in training.

The Process IV: Parallelization

- A C++ implementation on a single GPU processes $\sim$ 1700 words/second — not fast enough!
- The aforementioned model was trained on a machine with 8 GPUs, where each layer of the LSTM was executed on a different GPU, with activations communicated to the next GPU once complete.
- The remaining 4 GPUs were used to compute the softmax, a matrix multiplication procedure involving a matrix of dimension 1000 by 20000.
- This parallelized architecture improved the speed of the implementation up to 6300 words/second when minibatches of size 128 were used — training took 10 days. 😞

The Process IV: Parallelization

- A C++ implementation on a single GPU processes $\sim$ 1700 words/second — not fast enough!
- The aforementioned model was trained on a machine with 8 GPUs, where each layer of the LSTM was executed on a different GPU, with activations communicated to the next GPU once complete.
- The remaining 4 GPUs were used to compute the softmax, a matrix multiplication procedure involving a matrix of dimension 1000 by 20000.
- This parallelized architecture improved the speed of the implementation up to 6300 words/second when minibatches of size 128 were used — training took 10 days. 😞

The Process IV: Parallelization

- A C++ implementation on a single GPU processes $\sim$ 1700 words/second — not fast enough!
- The aforementioned model was trained on a machine with 8 GPUs, where each layer of the LSTM was executed on a different GPU, with activations communicated to the next GPU once complete.
- The remaining 4 GPUs were used to compute the softmax, a matrix multiplication procedure involving a matrix of dimension 1000 by 20000.
- This parallelized architecture improved the speed of the implementation up to 6300 words/second when minibatches of size 128 were used — training took 10 days. 😞

The Process IV: Parallelization

- A C++ implementation on a single GPU processes $\sim$ 1700 words/second — not fast enough!
- The aforementioned model was trained on a machine with 8 GPUs, where each layer of the LSTM was executed on a different GPU, with activations communicated to the next GPU once complete.
- The remaining 4 GPUs were used to compute the softmax, a matrix multiplication procedure involving a matrix of dimension 1000 by 20000.
- This parallelized architecture improved the speed of the implementation up to 6300 words/second when minibatches of size 128 were used — training took 10 days. 😞

Results I

- Surprisingly, good for long sentences as well as short ones.
- LSTM was *sensitve* to the order of words in the sentence.
- LSTM was *insensitive* to active versus passive voice.
- The best model trained was an ensemble of 5 LSTMs, with a beam of size 12, which achieved a BLEU score of 34.81 (reducing the beam to size 2 gave a score of 34.50).
- Rescoring of the baseline using the same ensemble of 5 LSTMs produced a BLEU score of 36.5 while the state of the art achieved 37.0 (n.b., the oracle score was $\sim$ 45).

Results I

- Surprisingly, good for long sentences as well as short ones.
- LSTM was *sensitve* to the order of words in the sentence.
- LSTM was *insensitive* to active versus passive voice.
- The best model trained was an ensemble of 5 LSTMs, with a beam of size 12, which achieved a BLEU score of 34.81 (reducing the beam to size 2 gave a score of 34.50).
- Rescoring of the baseline using the same ensemble of 5 LSTMs produced a BLEU score of 36.5 while the state of the art achieved 37.0 (n.b., the oracle score was ~ 45).

Results I

- Surprisingly, good for long sentences as well as short ones.
- LSTM was *sensitve* to the order of words in the sentence.
- LSTM was *insensitive* to active versus passive voice.
- The best model trained was an ensemble of 5 LSTMs, with a beam of size 12, which achieved a BLEU score of 34.81 (reducing the beam to size 2 gave a score of 34.50).
- Rescoring of the baseline using the same ensemble of 5 LSTMs produced a BLEU score of 36.5 while the state of the art achieved 37.0 (n.b., the oracle score was ~ 45).

Results I

- Surprisingly, good for long sentences as well as short ones.
- LSTM was *sensitve* to the order of words in the sentence.
- LSTM was *insensitive* to active versus passive voice.
- The best model trained was an ensemble of 5 LSTMs, with a beam of size 12, which achieved a BLEU score of 34.81 (reducing the beam to size 2 gave a score of 34.50).
- Rescoring of the baseline using the same ensemble of 5 LSTMs produced a BLEU score of 36.5 while the state of the art achieved 37.0 (n.b., the oracle score was ~ 45).

Results I

- Surprisingly, good for long sentences as well as short ones.
- LSTM was *sensitve* to the order of words in the sentence.
- LSTM was *insensitive* to active versus passive voice.
- The best model trained was an ensemble of 5 LSTMs, with a beam of size 12, which achieved a BLEU score of 34.81 (reducing the beam to size 2 gave a score of 34.50).
- Rescoring of the baseline using the same ensemble of 5 LSTMs produced a BLEU score of 36.5 while the state of the art achieved 37.0 (n.b., the oracle score was ~ 45).

Results II

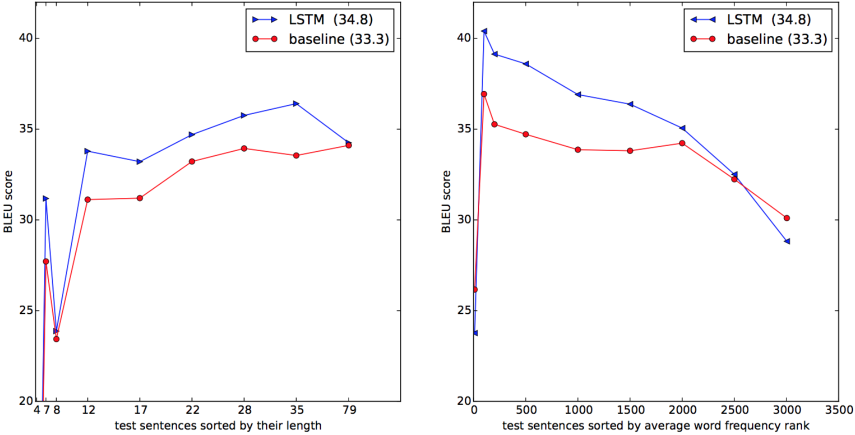

Figure 2: Performance of the LSTM model against the baseline (Figure 3 of Sutskever et al.)

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

Review

- LSTM architecture solves sequence to sequence problems.
- RNNs are not sufficient since the dimensionality of the inputs and outputs needs to be known *a priori* and fixed.
- Architecture: 2 LSTMs — (1) read input sequence, a single timestep at a time, to obtain fixed-dimensional vector representations, and (2) extract output sequence.
- Approach obtains a BLEU score of 34.81 — the best ever achieved by a neural net system.
- Use of deep LSTMs significantly outperformed that of shallow LSTMs, at a nearly negligible computational cost.
- **Key** finding: reversing the order of words in the input led to significantly better results.

References I

Hardt, M. (2017). "The Deepers": A community of empirics in Deep Learning. Personal Communication (in CS 294-139: "Fairness in Machine Learning," Fall 2017, UC Berkeley).

Olah, C. (2015). Understanding LSTM networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs`.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.