Stacked Long Short-Term Memory Networks: A Story in 4 Papers

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

Group 2: Chris K., Jonathan L., Ivana M., Nima H.

Division of Biostatistics, UC Berkeley

Nov. 7, 2017

Paper 3: Towards End-to-End Speech Recognition with RNNs

- Traditional approaches for speech recognition required separate components and training for the pronunciation, acoustic and language model.
- End-to-end models jointly learn all the components of the speech recognizer.
- The first attempt of end-to-end ASR: deep bidirectional LSTM and Connectionist Temporal Classification layer.
- Modification to the objective function: train the network to minimize the expectation of an arbitrary transcription loss function. This allows a direct optimization of the word error rate, even without lexicon or language model.

Recall: RNNs and LSTMs

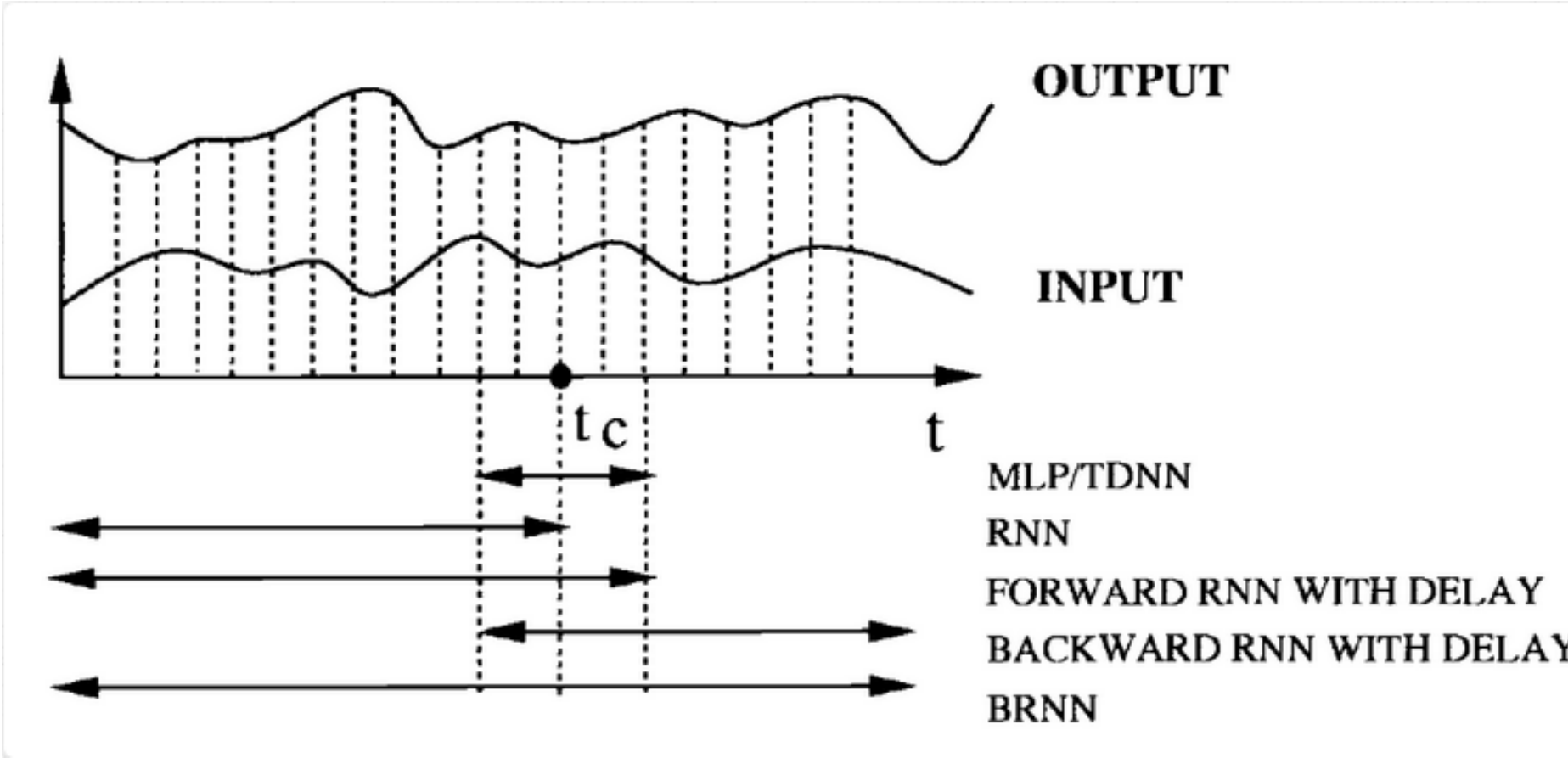
The basic structure of an RNN is as follows, featuring a single hidden layer:

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

Here we use the LSTM for each layer in the stack:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$
$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$
$$h_t = o_t tanh(c_t)$$

...learn from the future?



Bidirectional RNN

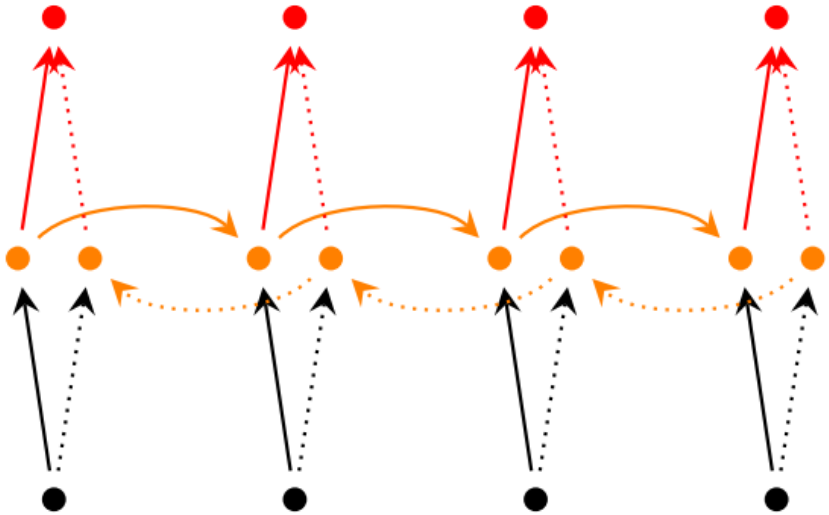
- Output at time t may not only depend on the previous elements in the sequence, but also future elements.
- Bidirectional RNN are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.
- If we wanted to predict the next word in a sentence, on a high level what a unidirectional LSTM will see is: **"The boys went to ..."** and will try to predict the next word only by this context.
- Bidirectional RNN allows you to see information further down the road.
  Forward RNN: **"The boys went to ..."**
  Backward RNN: **"... and then they got out of the pool"**
- Using the information from the future possibly makes it easier for the network to understand what the next word is.

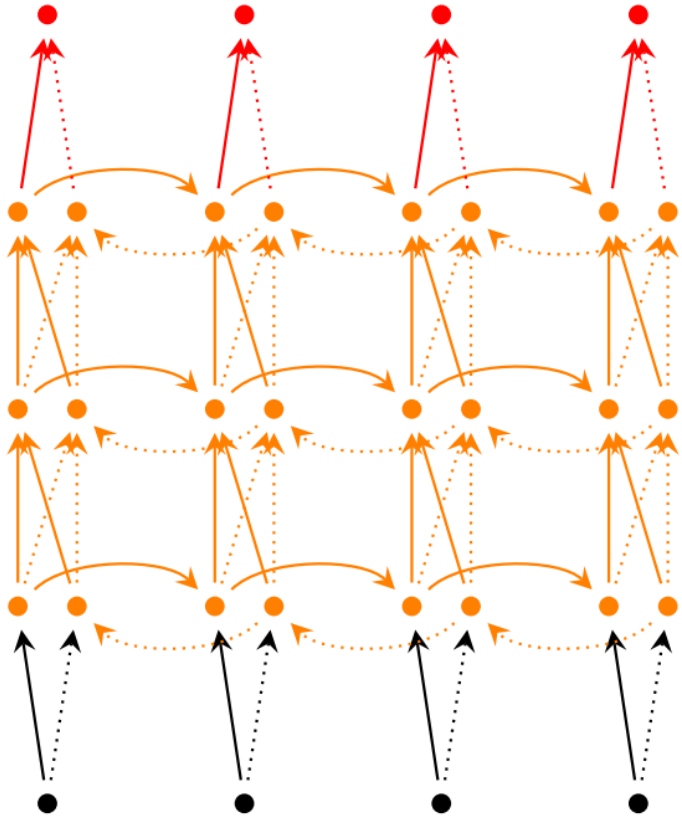
Bidirectional RNNs (BRNN)

- BRNN computes the *forward* hidden sequence $\overrightarrow{h}$, the *backward* hidden sequence $\overleftarrow{h}$ and the *output* sequence y by iterating the backward layer from $t = T$ to 1, the forward layer from $t = 1$ to T, and updating the output layer.

$$
\begin{aligned}
\overrightarrow{h}_t &= \mathcal{H}(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \\
\overleftarrow{h}_t &= \mathcal{H}(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \\
y_t &= W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_o)
\end{aligned}
$$

Bidirectional and Deep Bidirectional RNN





Connectionist temporal classification

- CTC allows RNNs to be trained in an end-to-end fashion without having a prior set alignment between input data and output targets.
- Suppose that for each input sequence x we have a label l, which is potentially shorter than the input sequence.
- General idea: generate a probability distribution at every timestep. We can then decode this probability distribution into a maximum likelihood label.
- Finally, we train out network by creating an objective function that coerces the maximum likelihood decoding for a given sequence x to corresponding desired label l.

Connectionist temporal classification: continued

- Define a blank token "-" to help construct alignments a.
- Crudely, $p(a|x) = \text{RNN}(x)$
- We can marginalize the probability and sum over all possible alignments:

$$p(l|x) = \sum_{a \in \mathcal{B}^{-1}(l)} p(a|x)$$

  operator $\mathcal{B}$ removes blanks and repeats from a sequence.
- The model can be optimized to maximize the likelihood $p(l|x)$ by marginalizing over all possible alignments using dynamic programming.
- Given an output transcription we have that:

$$CTC(x) = -\log(P(l|x))$$

Connectionist temporal classification: Pros and Cons

- Pros:
  - Can be used directly to model acoustics x into language characters l. As opposed to DNN-HMM models, CTC models have been shown to learn the pronunciation model directly, and not rely on an explicit pronunciation dictionary.

- Cons:
  - Assumes conditional independence between each output symbol
  - Cannot learn complicated multi-modal language distributions.

Expected Transcription Loss

- CTC objective function maximizes the log probability of getting the sequence transcription completely correct.
- Relative probabilities of the incorrect transcriptions are ignored, hence implying they are all equally bad.
- Standard measure is the word error rate (WER): edit distance between the true word sequence and the most probable word sequence emitted by the transcriber.
- Intuition: we want transcriptions with high WER to be more probable, for example. Network only receives the error term for changes to the alignment that alter the loss.

Expected Transcription Loss

- Train RNN to optimize the expected value of an arbitrary loss function defined over output transcriptions (such as WER).

$$\mathcal{L}(X) = \sum_{l} P(l|x)\mathcal{L}(x,l)$$
$$= \sum_{a} P(a|x)\mathcal{L}(x,\mathbb{B}(a))$$

- Approximate the loss: $\mathcal{L}(X) \sim \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x,\mathbb{B}(a^i))$
- Differentiate $\mathcal{L}(X)$ w.r.t. network outputs
- **Ex:** For transcription "WTRD ERROR RATE" gradient would encourage outputs changing the 2^{nd} output label to "O", discourage outputs making changes to the other 2 words and be close to 0 elsewhere.

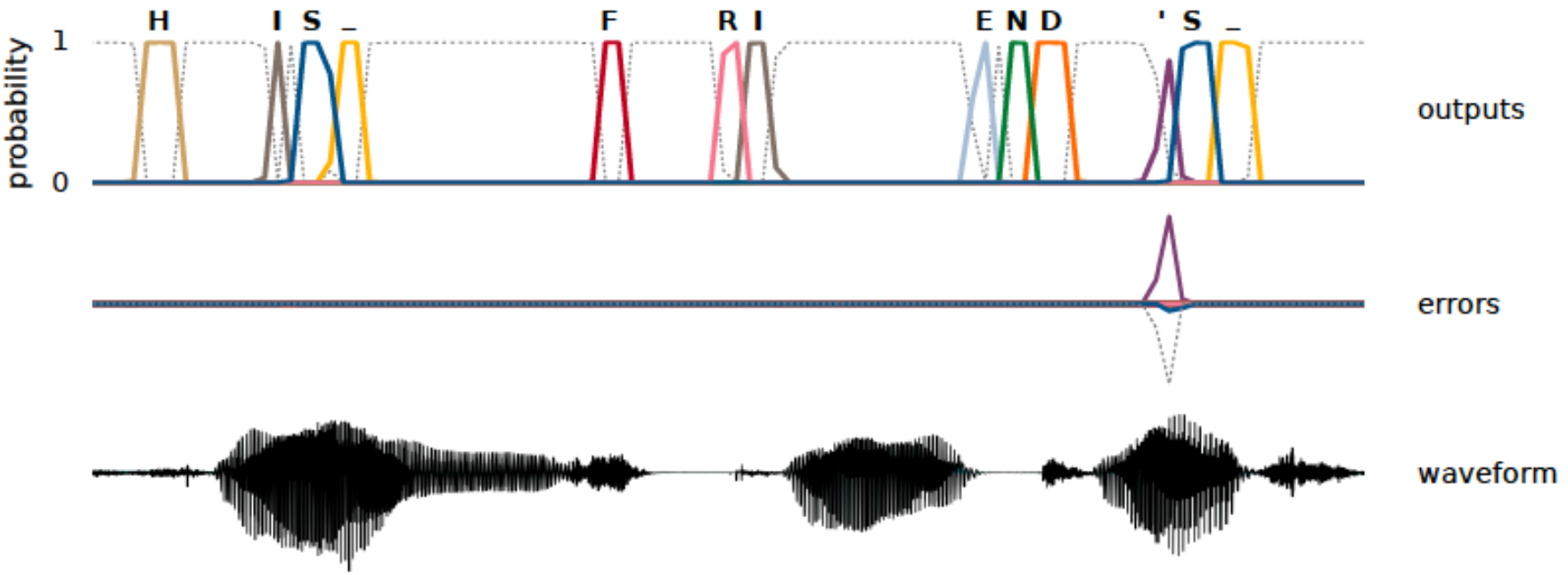
Decoding

- Decoding a CTC network = finding the most probable output transcription l for a given input sequence x.
- Pick the single most probable output at every time-step and return the corresponding transcription:

$$argmax_l P(l|x) \sim \mathcal{B}argmax_a P(a|x))$$

- More accurate decoding can be performed with a beam search algorithm, which makes it possible to integrate a language model.

General Idea



Experiments

- Experiments were carried out on the Wall Street Journal corpus.
- Trained on both the 14 hour and full 81 hour set. Input data were spectrograms derived from raw audio files.
- The network had 5 levels of bidirectional LSTM hidden layers, 500 cells in each layer: $\sim$ 26.5M weights!
- Comparison with baseline deep neural network-HMM hybrid, RNN trained with CTC, RNN trained to minimize the expected word error rate (5 alignment samples per sequence).
- Comparison between no dictionary or language model, 146K word dictionary followed by monogram, bigram and trigram language models.

Results

Table 1. **Wall Street Journal Results.** All scores are word error rate/character error rate (where known) on the evaluation set. 'LM' is the Language model used for decoding. '14 Hr' and '81 Hr' refer to the amount of data used for training.

SYSTEM	LM	14 HR	81 HR
RNN-CTC	NONE	74.2/30.9	30.1/9.2
RNN-CTC	DICTIONARY	69.2/30.0	24.0/8.0
RNN-CTC	MONOGRAM	25.8	15.8
RNN-CTC	BIGRAM	15.5	10.4
RNN-CTC	TRIGRAM	13.5	8.7
RNN-WER	NONE	74.5/31.3	27.3/8.4
RNN-WER	DICTIONARY	69.7/31.0	21.9/7.3
RNN-WER	MONOGRAM	26.0	15.2
RNN-WER	BIGRAM	15.3	9.8
RNN-WER	TRIGRAM	13.5	8.2
BASELINE	NONE	—	—
BASELINE	DICTIONARY	56.1	51.1
BASELINE	MONOGRAM	23.4	19.9
BASELINE	BIGRAM	11.6	9.4
BASELINE	TRIGRAM	9.4	7.8
COMBINATION	TRIGRAM	—	6.7

Results elaborated

- To provide character-level transcriptions, network must learn how to recognize speech sounds and transform them into letters- challenging for English language.
- Network makes phonetic mistakes (”shingle” vs. ”single”) and confuses homophones (”two” vs. ”to”).
- Unlike phonetic systems, the network makes lexical errors (”bootik” vs. ”boutique”).
- Promising direction: integrate the language model into CTC or expected transcription loss during training.

target: *ALL THE EQUITY RAISING IN MILAN GAVE THAT STOCK MARKET INDIGESTION LAST YEAR*

output: *ALL THE EQUITY RAISING IN MULONG GAVE THAT STACRK MARKET IN TO JUSTIAN LAST YEAR*

Paper 1:Speech Recognition

1. TIMIT dataset was used containing broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences.
2. End-to-end training avoids the problem of having incorrect alignments of training targets. Also enables training on a broader state space. End-to-end is essentially solving the loss function for all the parameters of the model and outputting the predictions directly from the inputs. In this case it is sounds to phonetic sequences.
3. Dynamics of speech make RNN's seem like a good match, hidden layers (depth) and function iteration (RNN) capture the flow of language.

Stacking and Training

The authors used 1 to 5 layers in the stack and the h's can be replaced by forward and backward counterparts.

$$h_t^n = \mathcal{H}(W h_{n-1} h^n h_t^{n-1} + W_{h^n h^n} h_{t-1}^n) + b_h^n$$
$$y_t = W_{h^N y} h_t^N + b_y$$

- We define a differentiable distribution of the outcomes given the weights in the matrices (two methods employed, CTC and Transducer)
- Use stochastic gradient descent and back propagation to update the weights.
- **Regularization**: Models are very complex so to avoid overfitting they applied early stopping and white noise added once per training sequence.

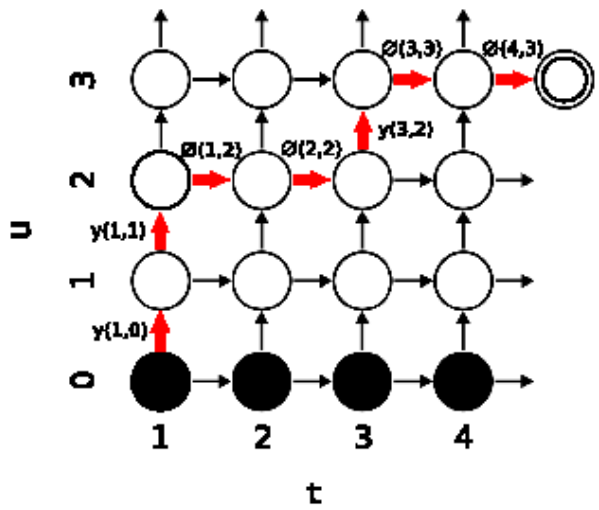
Transduction via RNN

- CTC network (transcription) is employed and another RNN (prediction) scans output and predicts each phoneme given the previous.
- the prediction RNN allows for language to be learned as well as acoustics of speech to phonemes.
- acoustic outputs at times, t, and linguistic choices at position, u, giving possible sequences as with CTC.
- **Improvement over previous Transducer**: Instead of multiplying probs of sound at time, t, with linguistic choice at position, u, and renormalizing, the acoustic hidden layer is fed to prediction hidden layer to form $h_{t,u}$ and then multinomial (softmax) probs are produced.

Sequences, Testing

Jumps to the right in time represent blanks. Vertical travel represents a sound being uttered.

Figure: display of a sequence in red



For testing (decoding) use beam search and don't bother normalizing by length

Results on TIMIT

Table

	NETWORK	WEIGHTS	EPOCHS	PER
1	CTC-3L-500H-TANH	3.7M	107	37.60%
2	CTC-1L-250H	0.8M	82	23.90%
3	CTC-1L-622H	3.8M	87	23.00%
4	CTC-2L-250H	2.3M	55	21.00%
5	CTC-3L-421H-UNI	3.8M	115	19.60%
6	CTC-3L-250H	3.8M	124	18.60%
7	CTC-5L-250H	6.8M	150	18.40%
8	TRANS-3L-250H	4.3M	112	18.30%
9	PRETRANS-3L-250H	4.3M	144	17.70%

method-number of layers-number of cells, Pre means pretrained
PER means phoneme error rate

Paper 2: Sequence Learning with LSTMs

"Sequence to Sequence Learning with Neural Networks"
(I. Sutskever, O. Vinyals, Q.V. Le, 2014)

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 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 II: LSTM 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: LSTM Architectures

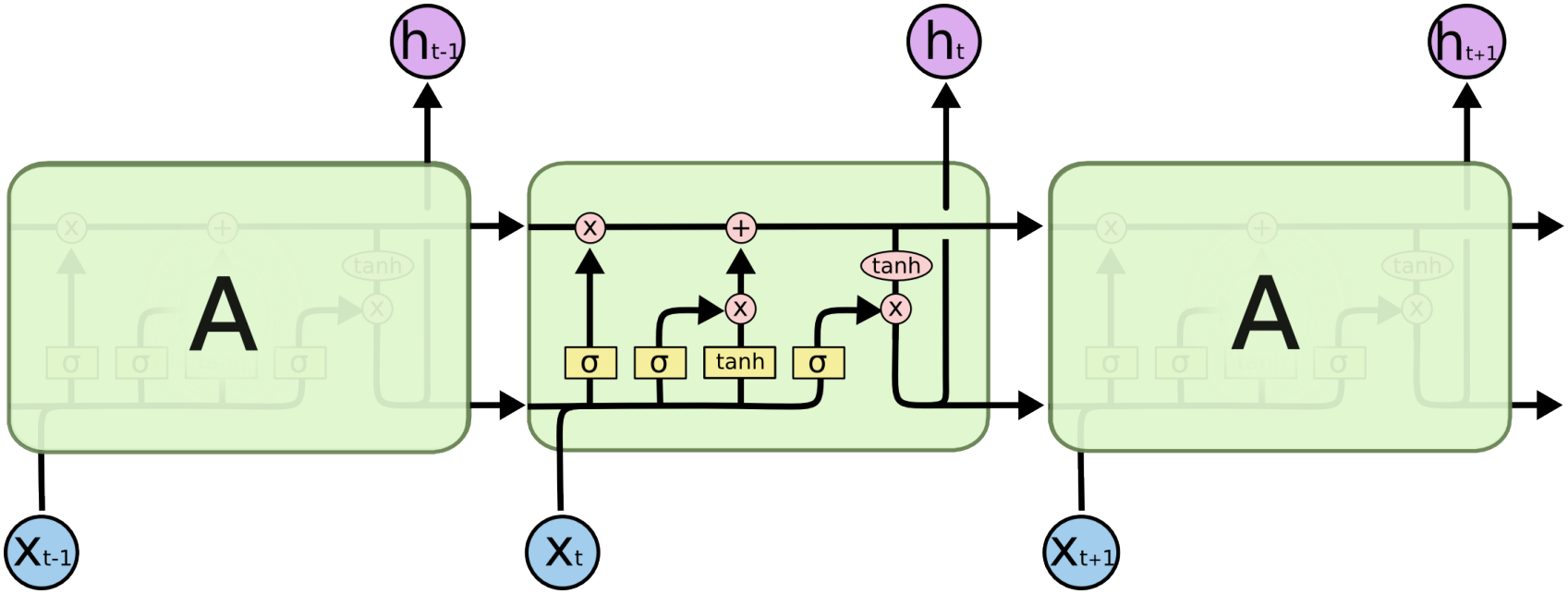


Figure: There are 4 interacting layers in the repeating module of an LSTM (source: C. Olah's blog)

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 II: Source Reversal

- A **key finding** was that reversing source sentences (without reversing target sentences) resulted in great gains.
- Dropped LSTM perplexity (from 5.8 to 4.7), and improved test BLEU score from 25.9 to 30.6!
- Unfortunately, there's no "complete explanation." 🤔
- How do such gains arise from a simple manipulation?
  1. "Minimal time lag" is lowered — when source and target sentences are concatenated, short-term dependencies are introduced (n.b., average distance unchanged).
  2. These can be exploited more easily by the LSTM. In fact, apparently, backpropagation has an easier time "establishing communication".
  3. LSTMs exhibited better performance on long sentences, perhaps better memory utilization.

The Process III: Training the Model

- Used deep LSTMs with 4 layers, 1000 cells at each layer, and 1000 dimensional word embeddings.
- Input vocab.: $160,000$; output vocab.: $80,000$.
- Deep LSTMs significantly outperformed shallow LSTMs, with each extra layer dropping 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 ~ 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 ~ 45).

Results II

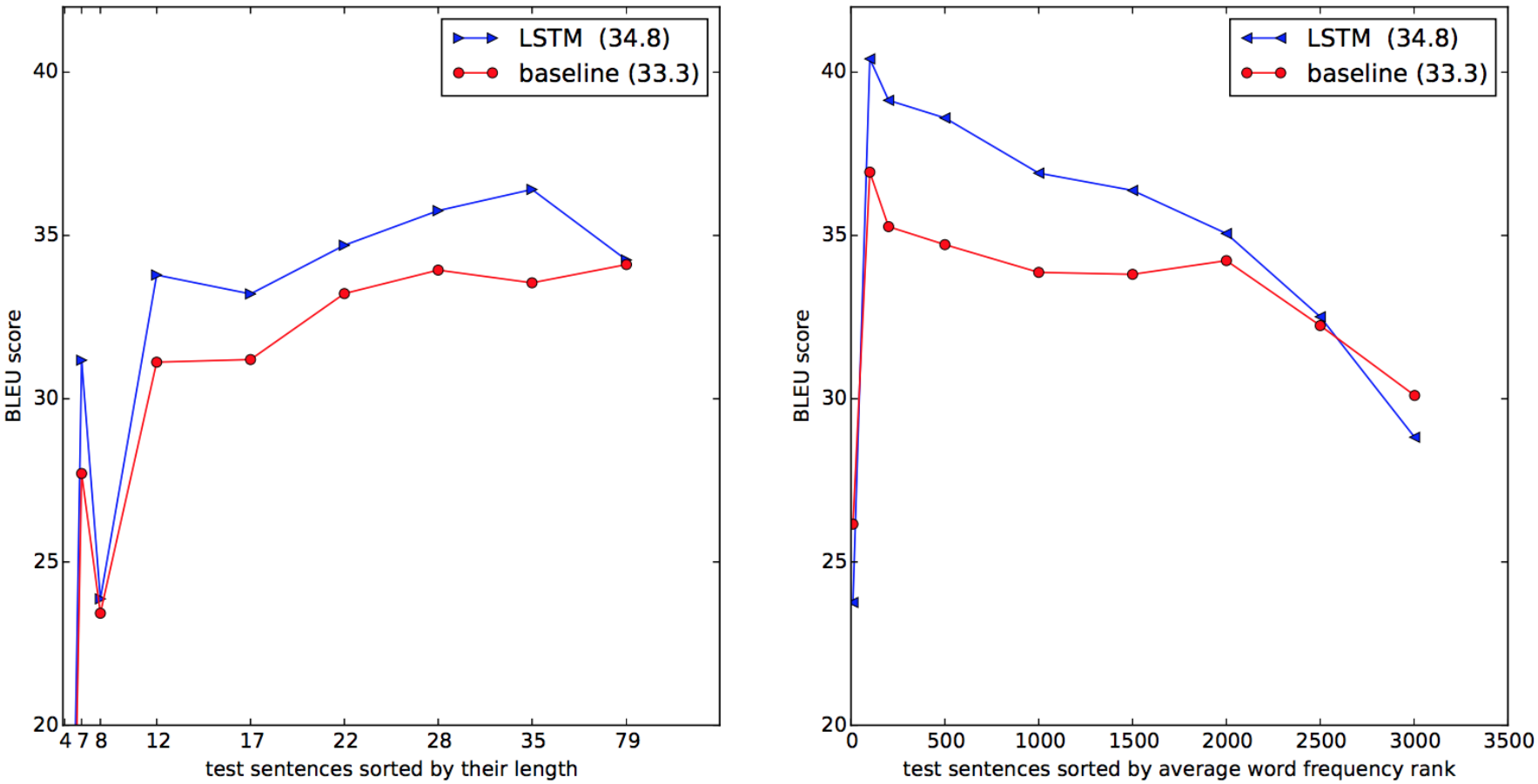


Figure: Performance of the LSTM model against the baseline (Figure 3 of Sutskever et al.)

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.

Paper 4

DRAW: A Recurrent Neural Network For Image Generation
Paper 4 on Stacked LSTMs
2015 paper by Google DeepMind: Karol Gregor, Ivo Danihelka, **Alex Graves**, Danilo Jimenez Rezende, Daan Wierstra

DRAW: Deep recurrent attentive writer

Two key components:

1. "**Spatial attention**" - focuses on parts of an image, like the human eye's "foveation".
   - **Foveation** refers to high resolution/detail at the region of focus, and a gradual blurring of areas (pixels) farther away from the fixation point.
2. "**Sequential variational auto-encoding framework**" - initially generates a blurry image and gradually sharpens sections at each iteration, yielding a realistic composition
   - Generative model comprised of two parts
   - **Encoder** RNN "compresses" the training images - learns a simpler latent representation
   - **Decoder** RNN "reconstitutes" a realistic image from that latent representation

Motivation

- Prior research suggests that visual structure may be better captured by looking at different parts of an image sequentially, rather examining the full image simultaneously.
- But how can the model learn how to examine subsets of an image iteratively?
- Policy gradients in reinforcement learning framework are one approach
- DRAW instead uses a differentiable attention model, allowing it to be trained by back-propagation.
- Big goal: Teaching a computer how to draw autonomously. Once trained, these RNN's draw things randomly based on their training. Here it is hand-written numbers then computer written and images (very difficult!)

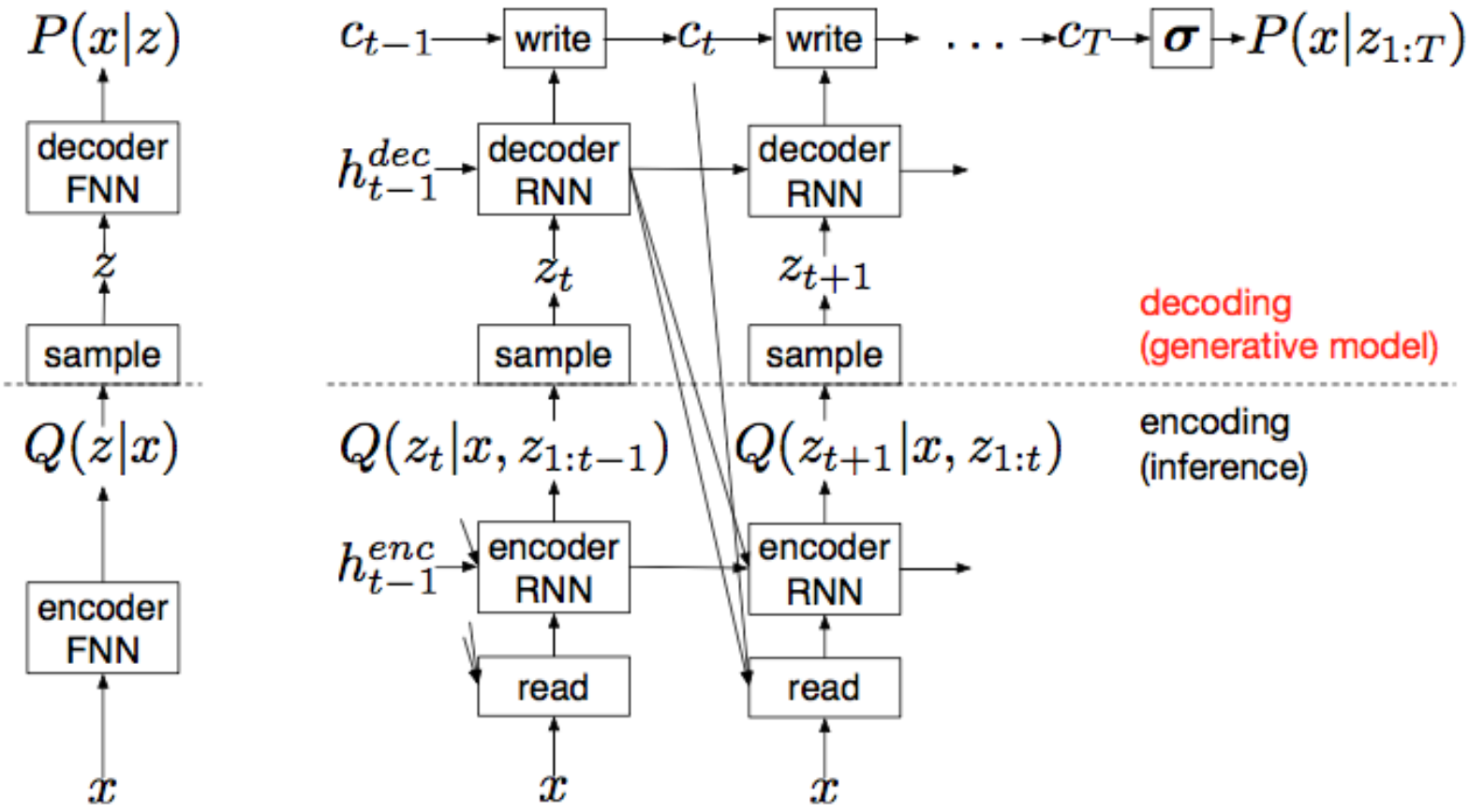
DRAW Network

- At each time step t, encoder is given the input image and previous decoder's hidden vector h_{t-1}^{dec}.
- The encoder's output parameterizes a distribution $Q(Z_t \mid h_t^{enc})$ over the latent vector z_t.
- Here the latent variables are diagonal (independent) Gaussians $\sim \mathcal{N}(Z_t \mid \mu_t, \sigma_t)$:

$$\mu_t = W(h_t^{enc})$$
$$\sigma_t = exp(W(h_t^{enc}))$$

Variational Auto-Encoders

(Left: Conventional, Right: DRAW)



Videotaped Demonstration

Youtube Demo

Network Architecture

We allow the read to evaluate the previous decoded hidden layer as well as the entirety of the image created thus far, much as an "artist" would proceed with drawing.

$$
\begin{aligned}
\hat{x}_t &= x - \sigma(c_{t-1}) \\
r_t &= read(x_t, \hat{x}_t, h_{t-1}^{dec}) \\
h_t^{enc} &= RNN^{enc}(h_{t-1}^{enc}, [r_t, h_{t-1}^{dec}]) \\
z_t &\sim Q(Z_t \mid h_t^{enc}) \\
h_t^{enc} &= RNN^{dec}(h_{t-1}^{dec}, z_t) \\
c_t &= c_{t-1} + write(h_t^{enc})
\end{aligned}
$$

The RNN's are LSTM's. Notice how the read network takes into account what you did not fill in on the canvas ($\hat{x}_t$) as well as the sequential input of data from the image.

Loss function

The encoder compresses input to real number values which have too much information (infinite, since it is real numbers). Thus we create a noisy gaussian version, $Q(Z_t \mid h_t^{enc})$, as a form of discretization of the input.

1. $L^z = \sum_{t=1}^{T} KL\left(Q(Z_t \mid h_t^{enc}) || P(Z_t)\right) = \sum_{t=1}^{T} log\left(\frac{Q(Z_t|h_t^{enc})}{P(Z_t)}\right)$ measures the amount of bits required for transmission to the decoder.
2. $L^x = \sum_{t=1}^{T} -\log D(x \mid c_T)$ coding cost to reconstruct x given z or more familiarly to a stats person, log-likelihood of the estimated distribution of x given the input z.
3. $L^x + L^z$ = total coding cost, differentiable, making SGD and back propagation straightforward. Forms an upper bound on $-log(p(x))$ or maximum likelihood.

Stochastic Data Generation

DRAW network can generate new images ($\tilde{x}$) by sampling from the prior Z_t, tantamount to sampling random standard normals, applying that to its internal decoder RNN (LSTM), generating a write operation, and then repeating for T time steps.

Formally:

- $\tilde{z}_t \sim P(Z_t)$
- $\tilde{h}_t^{dec} = RNN^{dec}(\tilde{h}_{t-1}^{dec}, \tilde{z}_t)$
- $\tilde{c}_t = \tilde{c}_{t-1} + write(\tilde{h}_t^{dec})$
- $\tilde{x} \sim D(X \mid \tilde{c}_T)$

Stacked LSTMs for Sequence Learning

Stochastic Data Generation

Stochastic Data Generation

DRAW network can generate new images ($\tilde{x}$) by sampling from the prior Z_t, tantamount to sampling random standard normals, applying that to its internal decoder RNN (LSTM), generating a write operation, and then repeating for T time steps.

Formally:

- $\tilde{z}_t \sim P(Z_t)$
- $\tilde{h}_t^{dec} = RNN^{dec}(\tilde{h}_{t-1}^{dec}, \tilde{z}_t)$
- $\tilde{c}_t = \tilde{c}_{t-1} + write(\tilde{h}_t^{dec})$
- $\tilde{x} \sim D(X \mid \tilde{c}_T)$

- $\tilde{z}_t$ is a latent sample from prior P
- $\tilde{h}_t^{dec}$ is the updated internal state of the decoder RNN (LSTM) based on the new data $(\tilde{z}_t)$ applied to its state in the previous time step $(\widetilde{h}_{t-1}^{dec})$.
- $\tilde{c}_t$ is the canvas (generated image) that is being progressively updated (written to) by the decoder network.
- $\tilde{x}$ is the final generated image.

Reading and Writing Without Attention

Without attention, we pass the entire image to the encoder at each time step, and the decoder writes an update to the entire canvas at each time step.

So the read and write operations are:

- $read(x, \hat{x}_t, h_{t-1}^{dec}) = [x, \hat{x}_t]$
- $write(h_t^{dec}) = W(h_t^{dec})$

Stacked LSTMs for Sequence Learning

└Reading and Writing Without Attention

Reading and Writing Without Attention

Without attention, we pass the entire image to the encoder at each time step, and the decoder writes an update to the entire canvas at each time step.

So the read and write operations are:

- $read(x, \hat{x}_t, h_{t-1}^{dec}) = [x, \hat{x}_t]$
- $write(h_t^{dec}) = W(h_t^{dec})$

- $[v, w]$ is a concatenation of vectors v and w into a single vector.
- We use the notation $b = W(a)$ to denote a linear weight matrix with bias from the vector a to the vector b.

Selective Attention Model

Five parameters for t are generated by the latent variables of the decoder network (H_{dec}) at t - 1.

1. g_X - x coordinate of the current focal point
2. g_Y - y coordinate of the current focal point
3. δ - stride (step size) of each patch
4. σ^2 - isotropic variance of the gaussian filters
5. γ - scalar intensity that multiplies filter response

δ, σ^2, γ are non-negative, so H_{dec} is interpreted as emitting log-transformed variables $(\log \delta, \log \sigma^2, \log \gamma)$ which can be negative. Those intermediate representations are then exponentiated to a non-negative scale.

Stacked LSTMs for Sequence Learning

Selective Attention Model

- These parameters are used to generate a grid of patches.
- The grid size N x N would be specified in advance as a hyperparameter.
- N x N might be 12x12 for example (MNIST) or 54x54 (SVHN).

Selective Attention Model

Five parameters for t are generated by the latent variables of the decoder network (H_{dec}) at t - 1.

1. g_X - x coordinate of the current focal point
2. g_Y - y coordinate of the current focal point
3. δ - stride (step size) of each patch
4. σ^2 - isotropic variance of the gaussian filters
5. γ - scalar intensity that multiplies filter response

δ, σ^2, γ are non-negative, so H_{dec} is interpreted as emitting log-transformed variables $(\log \delta, \log \sigma^2, \log \gamma)$ which can be negative. Those intermediate representations are then exponentiated to a non-negative scale.

Reading and Writing With Attention

With attention our read operation becomes a function of five parameters: F_X, F_Y, intensity γ, input image x, and error image $\hat{x}_t$.

$$read(x, \hat{x}_t, h_{t-1}^{dec}) = \gamma[F_Y x F_X^T, F_Y \hat{x} F_X]$$

For the write operation a separate set of attention parameters are used: $\hat{\gamma}, \hat{F}_X, \hat{F}_Y$

$$w_t = W(h_t^{dec})$$

$$write(h_t^{dec}) = \frac{1}{\hat{\gamma}} \hat{F}_Y^T w_t \hat{F}_X$$

Stacked LSTMs for Sequence Learning

Reading and Writing With Attention

Reading and Writing With Attention

With attention our read operation becomes a function of five parameters: F_X, F_Y, intensity γ, input image x, and error image $\hat{x}_t$.

$$read(x, \hat{x}_t, h_{t-1}^{dec}) = \gamma[F_Y x F_X^T, F_Y \hat{x} F_X]$$

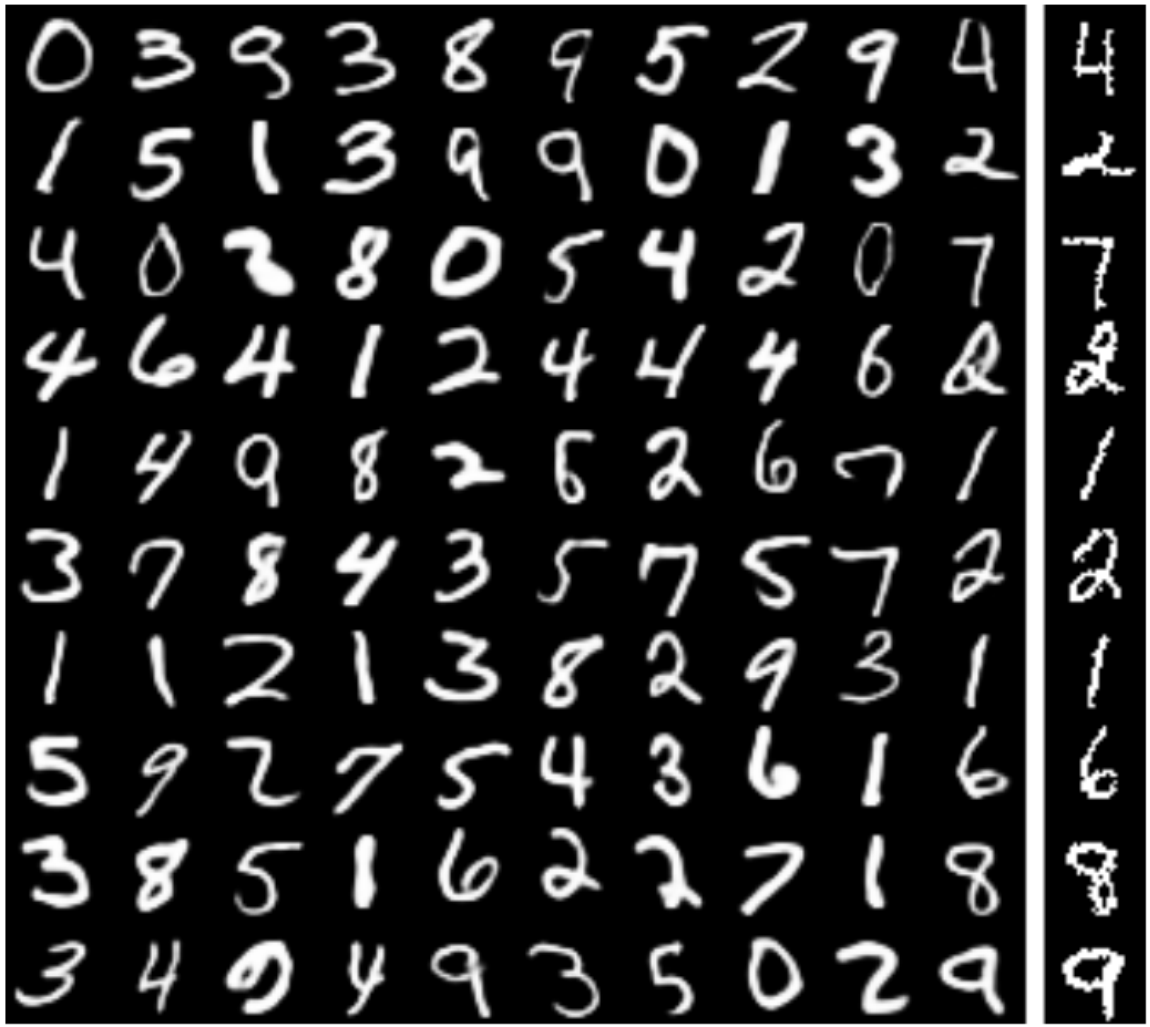
For the write operation a separate set of attention parameters are used: $\hat{\gamma}, \hat{F}_X, \hat{F}_Y$

$$w_t = W(h_t^{dec})$$

$$write(h_t^{dec}) = \frac{1}{\hat{\gamma}} \hat{F}_Y^T w_t \hat{F}_X$$

- Error image $\hat{x}_t$ is the residual between the true image and the current canvas prediction c_t.
- w_t is the N x N writing patch emitted by h_t^{dec}.

Results: MNIST



Stacked LSTMs for Sequence Learning

Results: MNIST

- Left box shows numbers generated by the network.
- Right column shows the image from the training data that is most similar to the column to its left.

Results: Streetview House Numbers

Stacked LSTMs for Sequence Learning

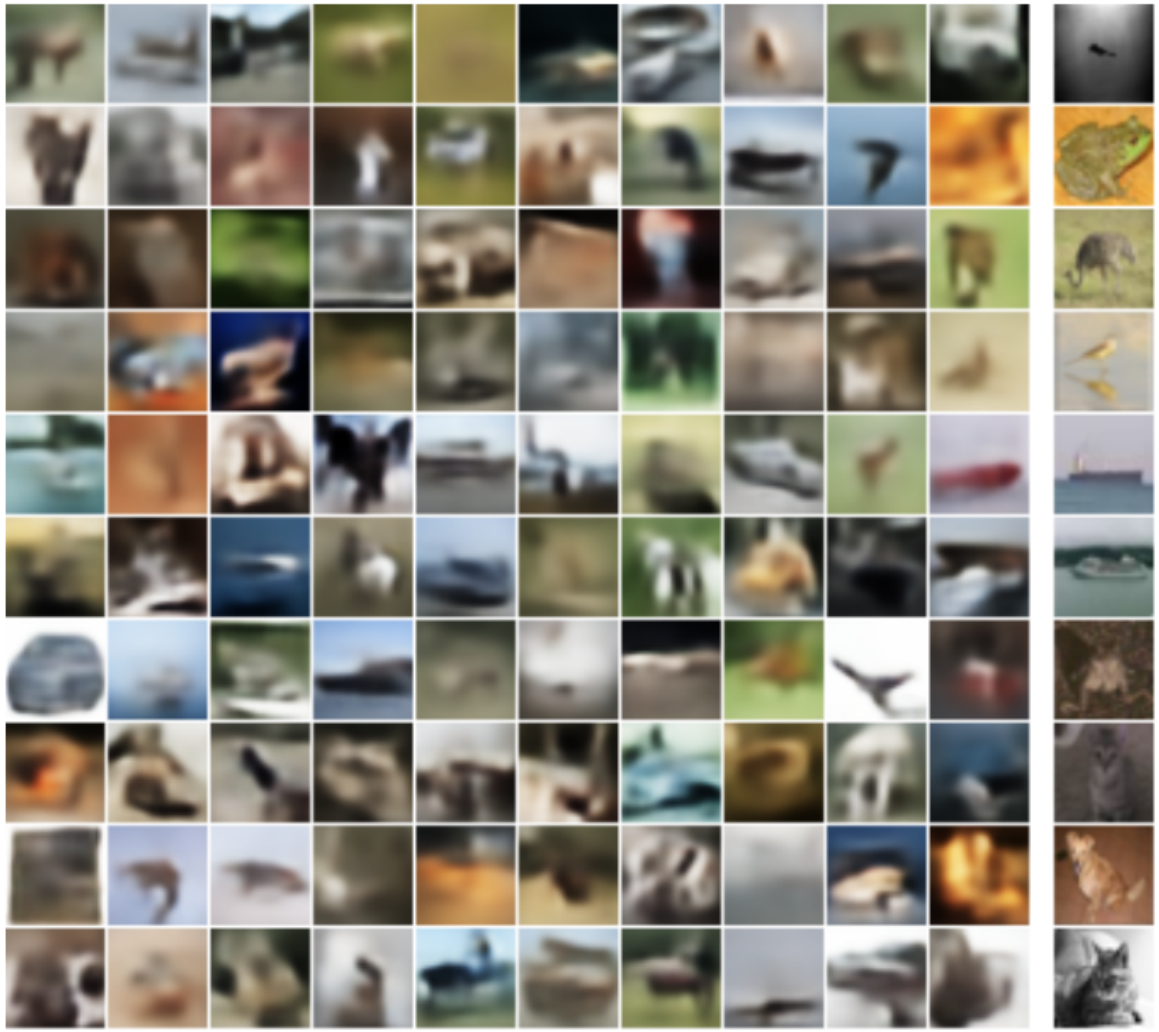
Results: Streetview House Numbers

Results: Streetview House Numbers

Same story:

- Left box shows images generated by the network.
- Right column shows the image from the training data that is most similar to the column to its left.

Results: CIFAR



Conclusion: let's get meta

- The task of this assignment has gotten us to mentally build an encoder-decoder with attention.
- We read an article multiple times, learning to represent raw text as mental latent variables (concepts) by focusing on the important parts and where our error rate is highest.
- We can then emit a shorter reconstruction of the article as a series of slides plus verbal annotation, based on that latent representation.
- After this has been done successfully on enough articles, we are able to generate new articles by sampling from the prior distribution of those latent variables.
- When those generated articles are sufficiently high-quality, we are given a PhD.