Accuracy and Time in RNN Attention Models

Raphael Palefsky-Smith
rpalefsk@stanford.edu

March 22, 2017

1 Introduction

Attention, attention! Neural network attention models allow networks to ”zoom in” on salient input regions, ignoring noise and focusing their processing power on areas that matter. This is a powerful tool for applied Artificial Intelligence - it has proven effective on Natural Language Processing [1] and Computer Vision [2] tasks - but it is also exciting for cognitive psychology. Attention is explicitly biologically inspired, taking cues from the foveation of the eyes. It offers a chance to poke and prod at an artificial mechanism with clear neurological parallels; if all goes well, neural network attention models can teach us about human attention, and visa versa.

In this project, I altered the internal structure of one such attention model, Gregor et al's Deep Recurrent Attention Writer (DRAW) [3], to more closely model human attention. As my modification is fairly low-level, it is important to understand how these attention mechanisms work.

1.1 Background

Attention models have shown promise for text processing, but I focused on standard image-based tasks. Whereas traditional convolutional neural networks examine the entire input image, attention-based networks move a dynamically-sized window over subregions of the image. The attention model must learn both its ”top-level” task (often classification or image reconstruction) and how to move the window to most effectively perform this task. In Mnih et al's RAM model [4], the window movement strategy is trained via reinforcement learning. This model is more performant than a baseline network, but the reinforcement learning component is clunky and inelegant. Gregor et al's DRAW, by contrast, is fully differentiable. Both DRAW's window movement strategy and classification/reconstruction ability are trained simultaneously via backpropagation, evolving together to minimize loss.

1.2 DRAW Mechanics

Gregor et al apply DRAW to different tasks including image generation, but in the scenario that I examined, the network is tasked with classifying a single MNIST digit randomly placed on a noisy background. DRAWs core is an LSTM recurrent network. At each timestep, the network places a window on the input image, with both its location and zoom level determined via learned connections from the image and previous hidden state. It then outputs a classification of the contents of the window.

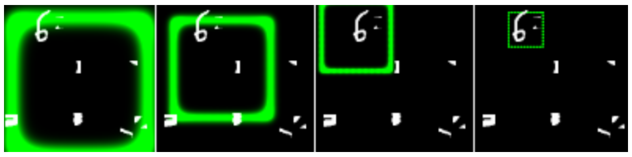


Figure 1: Movement of the attention window (Gregor et al)

The network is given a fixed number of timesteps (glimpses), with only its final classification included in the loss. This key point - that only the final glimpses classification matters - was the subject of my investigation.

1.3 My Modification

While considering only the final timestep might be computationally efficient, it is not how human attention works.

This procedure is akin to telling a subject to stare at an image for exactly ten seconds and only then report a classification. Rather, humans prioritize time as well as accuracy, attempting to classify visual stimuli as soon as possible without too much error.

To better model this prioritization of time, I modified DRAW to average the error at all timesteps into its loss, rather than only including the final one. I refer to this altered model as All-Step attention, and the original model as Last-Step attention. I hypothesized that All-Step would drive the network to minimize its classification error as soon as possible, rather than waiting until the final step, perhaps at the cost of overall classification accuracy. As far as I know, this is the first attempt to make an attention model's loss more human-like.

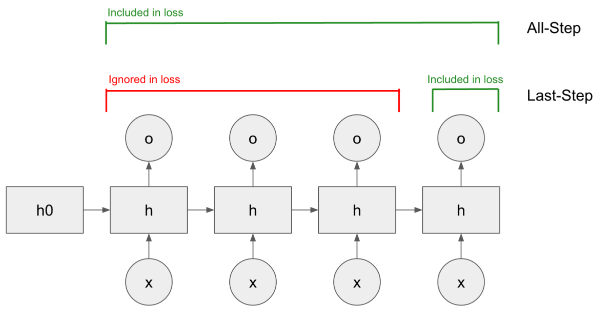


Figure 2: Diagram of my modification, the All-Step model

1.4 Goals

I hoped to answer three main questions. First, how does All-Step's classification accuracy compare to Last-Step? Does the additional loss inputs over-constrain the network, and ruin its performance? Second, how quickly do the two models train? Does All-Step attention significantly slow things down? Finally, would All-Step's window movement strategy, once visualized on images from the dataset, differ qualitatively from Last-Step?

While I had initially hoped to compare the network's performance to human attention studies, I was unable to find data (either eye foveation or time-vs-accuracy plots) that matched the network's architecture. However, there was more than enough analysis to be done on the network, and I do not believe that the human comparison is essential.

2 Methods

My implementation consisted of modifications to Jack Lindsey's DRAM model [5], which uses code from Eric Jang's TensorFlow implementation of DRAW [6]. DRAM is slightly simplified and allows one to train using digit classification loss, rather than the image reconstruction loss used by the original paper.

2.1 Technical Details

The network takes as input 100 by 100 pixel images, which consist of 28 by 28 pixel MNIST [7] images modified to add distortion and random translation. The attention mechanism slides a parameterized filterbank over the image and produces a 12 by 12 pixel patch. This is fed through a 256-unit LSTM encoder, a 10-unit fully-connected layer, another 256-unit LSTM decoder, two fully-connected layers (256 and 10 units respectively), and finally a softmax function to produce a classification output. This process is repeated over 10 timesteps (glimpses), with the hidden of the previous timestep's decoder fed into the current timestep's encoder.

Each network was trained for 5 epochs over 60,000 training examples using a batch size of 1, for a total of 300,000 iterations. The loss was minimized with the ADAM optimizer using a learning rate of 10e-4 and a beta1 parameter of 0.5. Gradients were clipped to a norm of 5. The weights were checkpointed and test accuracy computed every 1000 iterations. Training was performed on an NVIDIA 1080 GPU.

At the implementation level, I slightly modified the network code to allow for instrumentation. At each time step, the network would additionally output its attention window coordinates, LSTM cell states, and computed 12 by 12 pixel patches.

2.2 Attention Visualizer

Inspired by the interactive tools from our homeworks, I created an online program to visualize the two attention models. The application runs entirely in the browser, with no ssh or X11 required. It allows one to load images from the test set, compare the two attention models' windows and probability outputs, and view the computed 12 by 12 pixel glimpse at each time step. Additionally, one can evaluate different model checkpoints on the same test image, allowing one to analyze the evolution of the network's behavior over the training process. I've created a demonstration video of the tool which can be viewed here: `https://www.youtube.com/watch?v=lDag7iP6BKQ`. I hope it finds use beyond this single project!

3 Results

After the lengthy training process, I conducted a series of tests on the two trained models. Due to the time cost of re-training the models, I focused on runtime experiments. Here, I present the top-level quantitative and qualitative results from these tests, with the nitty-gritty "why" analysis saved for Section 4.

3.1 Accuracy

For both models, I evaluated test-time accuracy using the classification at the final time step, ignoring the previous steps. I expected the Last-Step model to significantly outperform the All-Step model, as Last-Step attention is specifically optimized for performance on this final step. However, to my huge surprise, All-Step attention beat Last-Step with a noticeable improvement.

All-Step	0.822
Last-Step	0.770
Difference	0.052
Percent Improvement	6.75%

Table 1: Classification accuracy on noisy MNIST test set

While these numbers fall short of the 0.95+ accuracies [7] that have been attained on MNIST, we must remember that this is on noisy, distorted MNIST images. Running through some of the test set, I was frequently unable to classify the digits myself! Due to distortions and obstructions, many of the digits lose their distinctive shape, and so these accuracy figures seem reasonable.

In addition to its overall accuracy boost, the All-Step network successfully learned to predict the digit as soon as possible. When accuracy is plotted at each time step, and not just at the end, it is immediately clear that the All-Step model's loss function has taken effect.

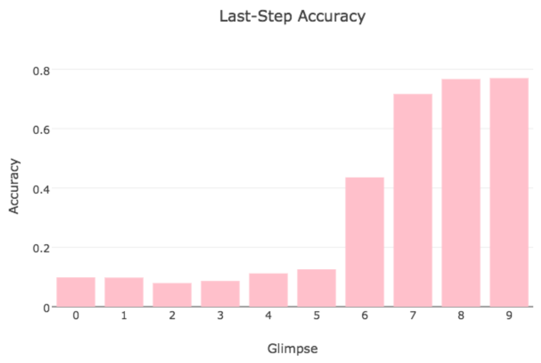


Figure 3: Classification accuracy of Last-Step model at each glimpse

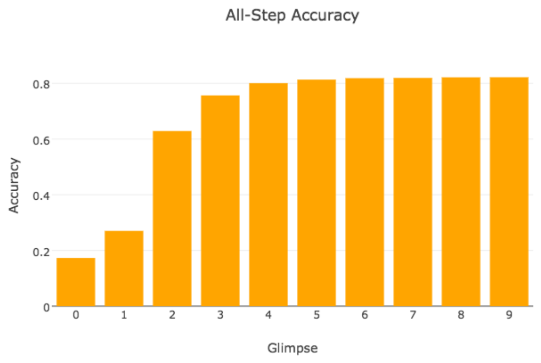


Figure 4: Classification accuracy of All-Step model at each glimpse. It works!

This result is not especially surprising, since the network is entirely reliant on its loss function and will happily contort itself to attain a lower loss. Nevertheless, the improvement in overall accuracy is puzzling, and was a focus of my analysis.

3.2 Training Time

I evaluated the test set accuracy (again, using the final step) every 1000 iterations, and plotted both models to compare:

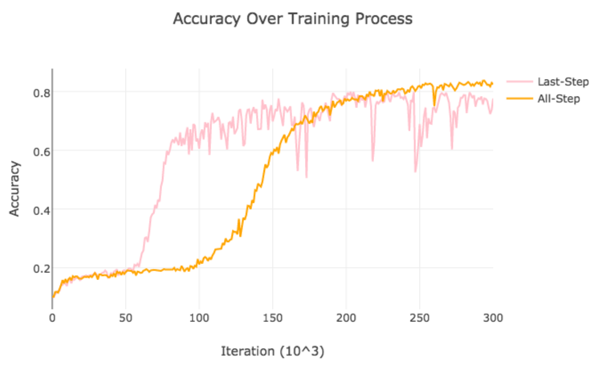


Figure 5: Classification accuracy over the training process

Immediately, two trends jump out. First, that both models roughly follow an S-curve, starting at low accuracy and moving steeply up to high accuracy. But the Last-Step model begins its climb much sooner, starting around iteration 6,500. The All-Step model starts climbing around iteration 110,000 and its climb is much more gradual. Second, that the Last-Step model's accuracy wildly fluctuates. All-Step is relatively stable, but Last-Step is all over the place, dropping below 0.6 accuracy and swinging back up.

Both of these trends seem to point towards the All-Step loss acting as a regularizer, a theory I will revisit in the Analysis section.

3.3 Window Movement

Finally, I loaded the test set into the Attention Visualizer (Section 2.2) and observed the window movement behavior and classification output. With few exceptions, I observed the trend illustrated on the following page.

The All-Step model "locks on" to the image by Glimpse 5, and keeps its window almost identically placed throughout the following steps. The window is not particularly tight around the digit, covering a small to medium sized area. Mirroring the window lock, the model chooses the correct class with very high probability starting at Glimpse 5. The classification predictions fluctuate very slightly, but it is remarkably consistent.

The Last-Step model, in contrast, exhibits very odd behavior. Rather than locking on early and holding window size consistent, it zooms its window smaller and smaller until Glimpse 8. At this point, its window is much smaller than the All-Step's. And until this point, its classification probabilities are all roughly equal, with no clear pick.

After this point, however, it rapidly chooses the correct digit, and predicts it with high probability in Glimpses 9 and 10. And strangely enough, its window actually grows larger, moving without explanation. To anthropomophize the model, it appears that it searches until Glimpse 8, declining to make a prediction until it is sure. Then, it suddenly makes up its mind, and convinced of its prediction, it no longer needs to accurately control its window and allows it to drift. It seems likely that this is the result of the LSTM's advanced memory/forgetting abilities.

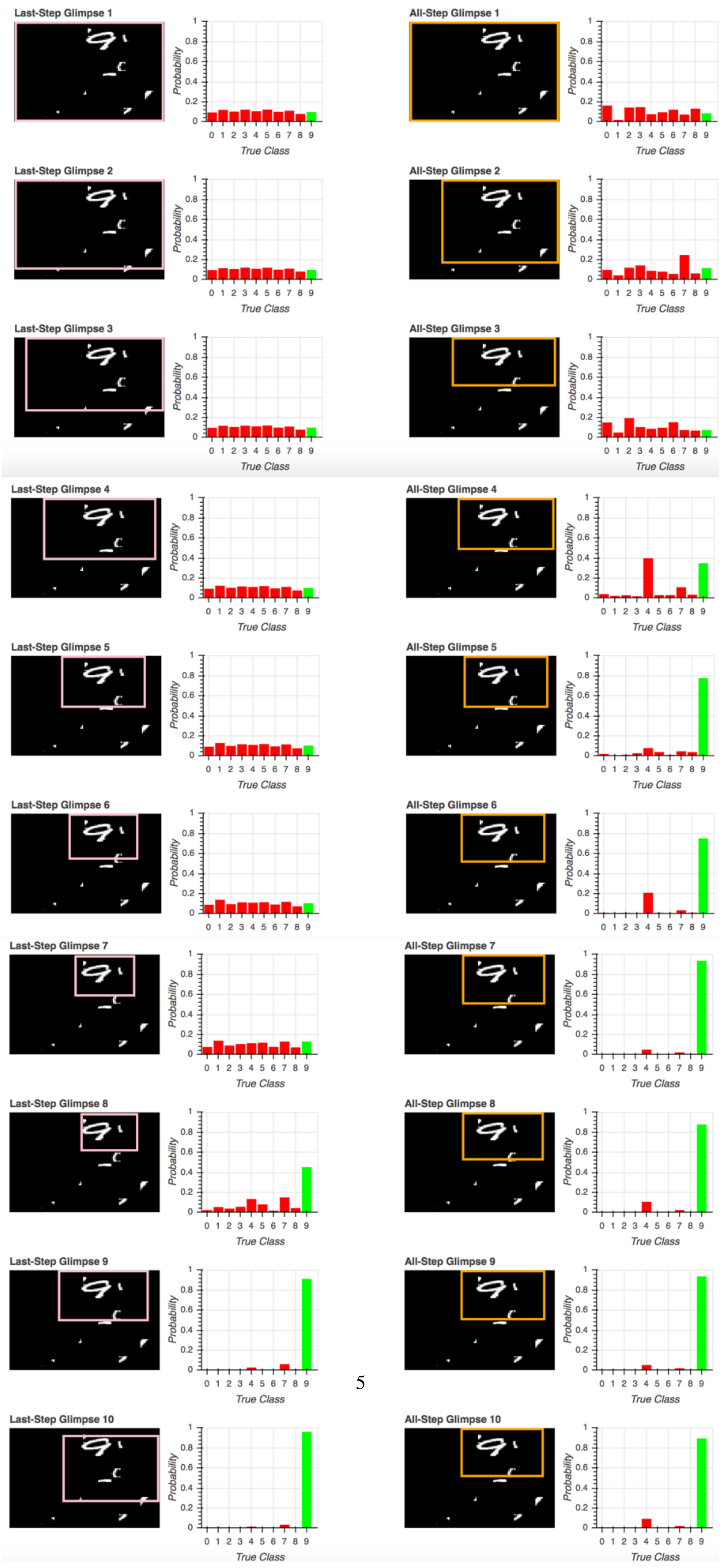
4 Analysis

Due to the models' long training time, I didn't get as much time for analysis as I would have liked. So, the following analyses are more explorations and suggestions for future inquiry than definitive, final answers.

4.1 Accuracy and Training Time

After seeing the All-Step model outperform the Last-Step's accuracy, I set out to explain the discrepancy. My hunch was that the All-Step loss acts as a regularizer. Since the loss is the mean of 10 different classification errors, it should act almost like a minibatch and smooth out the gradient updates. This would also explain the All-Step model's slower "accuracy climb" - regularization leads to higher test accuracy, but can slow down training with its retardant effect.

I wanted to see if my regularization theory had any legs, so I turned to visualizing the models' weights over time. Since the models have thousands of parameters, it is impractical to analyze them individually. Instead, I iterated through each checkpoint, and computed the Euclidean distance between each weight matrix and its value at the last checkpoint. I then divided these raw distances


Last-Step Glimpse 1
All-Step Glimpse 1
Last-Step Glimpse 2
All-Step Glimpse 2
Last-Step Glimpse 3
All-Step Glimpse 3
Last-Step Glimpse 4
All-Step Glimpse 4
Last-Step Glimpse 5
All-Step Glimpse 5
Last-Step Glimpse 6
All-Step Glimpse 6
Last-Step Glimpse 7
All-Step Glimpse 7
Last-Step Glimpse 8
All-Step Glimpse 8
Last-Step Glimpse 9
All-Step Glimpse 9
Last-Step Glimpse 10
All-Step Glimpse 10
Probability
True Class
0 1 2 3 4 5 6 7 8 9
0 0.2 0.4 0.6 0.8 1

by the size of the matrices to yield a per-parameter distance measure invariant to the dimensions of the weight. Large distances would mean a significant update, and small distances would mean those parameters stayed relatively constant.

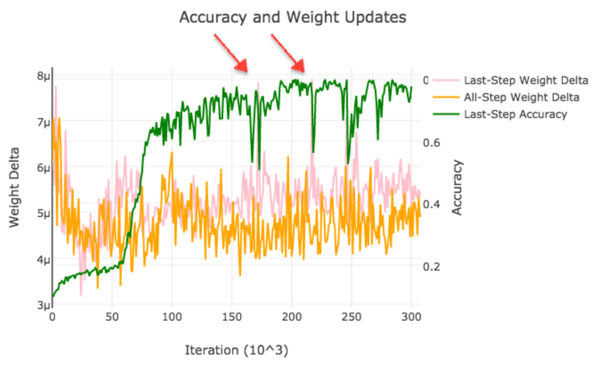


Figure 7: Euclidean distance to previous checkpoint's LSTM decoder weights, plotted against the Last-Step model's accuracy. Red arrows indicate weight update spikes that correlate with Last-Step accuracy dips.

After plotting the weight deltas, I found that the Last-Step's LSTM decoder weights had huge spikes. And after plotting them against the Last-Step's accuracy, it was clear that large decoder updates occurred right before large dips in accuracy. So, it looks like the accuracy fluctuations described in Section 3.2 are highly correlated with - though not necessarily caused by! - large decoder weight updates. I plotted the All-Step's weight updates on the same scale, and it does not exhibit the same spikes. This provides even more evidence that the All-Step loss is a regularizer.

Finally, I was curious if the Last-Step's reduction in accuracy was uniform across all digit classes, or was focused on a few digits. I computed confusion matrices for both models, and then subtracted the All-Step's matrix from the Last-Step's. This means that positive values in the matrix are predicted by the Last-Step more than the All-Step, and negative values are predicted more frequently by the All-Step.

Interestingly, the accuracy dip appears strongly focused on 1, 3 and to some degree, 5. When compared to the All-Step, the Last-Step model makes fewer correct (diagonal)

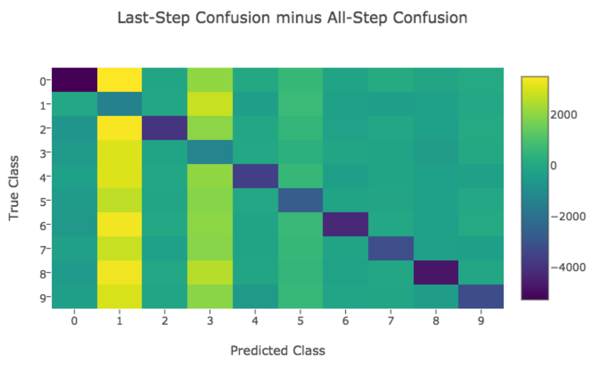


Figure 8: Last-Step confusion matrix minus All-Step confusion matrix

predictions, and overpredicts 1, 3, and 5. It is unclear why this is the case. 1, 3, and 5 are not particularly similar in appearance, so it seems unlikely that the correlation is based on their visual structure. I attempted to trace the overprediction back to the weights themselves, but there are so many different moving parts - the encoder, the decoder, and the three different hidden layers - that I was unable to find conclusive evidence. This would be a great candidate for further exploration; clearly, there is a pattern to the Last-Step model's error, but its cause eludes me.

4.2 Window Movement

As described in Section 3.3, the Last-Step model exhibits a strange window movement pattern: it behaves normally until around Glimpse 8, gradually reducing its window size and homing in on the digit while outputting relatively uniform classifications. After Glimpse 8, however, it strongly predicts a single digit and counterintuitively increases the size of its window.

I hypothesized that this behavior stems from the LSTM's memory-management functionality. The LSTM cell can learn to selectively remember or forget stored information, and weight this stored information with the current input. It can also control how much of this past information is included in its output. Unfortunately, TensorFlow's LSTM implementation does not expose the low-level input, forget, and output gates. (In future explorations, the LSTM could be re-implemented and its individual parameters explored). So, I chose to examine the LSTM's cell vector, which controls its memory access. Much like my analyses of weights over time, I plotted the Euclidean distance between cell vectors at each timestep. A large value means that the cell state changed significantly between glimpses, and therefore, the LSTM's memory-management "strategy" has shifted.

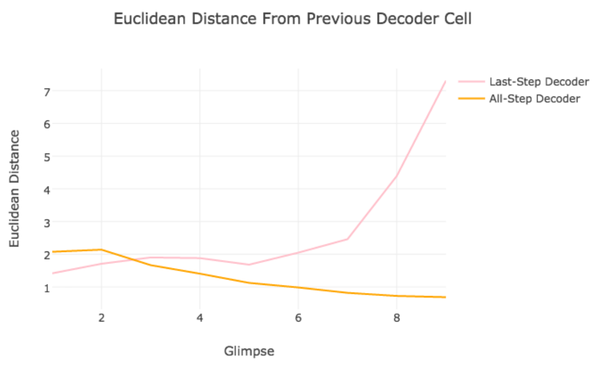


Figure 9: Euclidean distance to previous glimpse's LSTM decoder cell

Here, we can see the Last-Step's decoder state changes rapidly at Glimpses 7, 8, and 9. In contrast, the All-Step's decoder state barely changes, with its rate of change actually decreasing as time progresses. This does not prove anything, but it highly supports the theory that the Last-Step's odd behavior change has its basis in the LSTM cell state. For good measure, I also plotted each model's encoder change over time (Figure 10).

However, I do not see as clear of a pattern or difference in the encoder state deltas. Once again, the Last-Step model has more change in its LSTM cell state, but there is no dramatic uptick as in the decoder. This makes sense, for it is the decoder's output that feeds directly into both the classification and the next glimpse's window location. The encoder is separated from both of these outputs by several matrix multiplications.

Without a thorough analysis of the currently-inaccessible input, forget, and output gates it is hard to describe how, exactly, the Last-Step's decoder LSTM is behaving. But given that its memory-controlling cell state fluctuates rapidly right when the model's behavior changes, a link seems very likely.

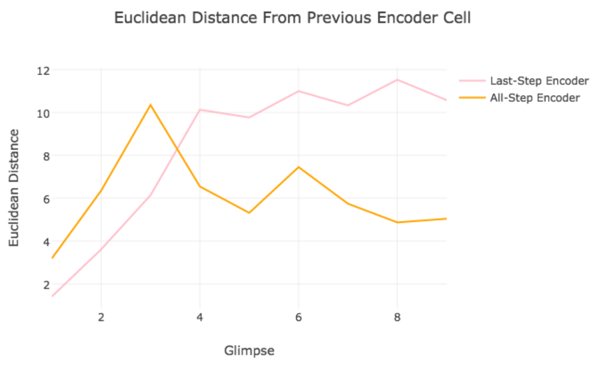


Figure 10: Euclidean distance to previous glimpse's LSTM encoder cell

5 Discussion

While there are still many unknowns buried deep in the LSTM weight matrices, let us return to the three simple questions I posed in Section 1.4.

5.1 Questions Answered

How does the modified loss function affect accuracy? Not only does it successfully force the model to predict the digit sooner, it increases overall accuracy. The latter property is likely due to implicit regularization over the timesteps, and a similar effect could probably be achieved in fewer timesteps using explicit regularization.

How does the modification affect training time? Again due to this (probable) regularization, the All-Step model takes longer to converge. However, it appears that the wait is worth it, as it ends up at a higher accuracy than the Last-Step model. The Last-Step model suffers from extreme accuracy fluctuations, which are highly correlated with large weight updates that the All-Step's regularization cancels out.

Finally, how is window movement behavior impacted? The All-Step model's behavior is actually the more "rational" one, locking its window to the digit and staying put. The Last-Step model exhbits an odd strategy of zooming in on the digit, making a strong classification, and then moving its window haphazardly. This seems to be the result of its LSTM decoder's memory behavior.

5.2 Limitations and Future Work

I see three main limitations to my work. First, that all experiments were performed at test-time. With more GPUs and training time, it would be interesting to try various weightings of the timestep losses, rather than just a simple mean. Could the network be trained to predict even sooner? It would also be worth investigating ”learning when to stop.” My implementation fixes the number of glimpses at 10, but it appears that many of these timesteps are unnecessary, as the All-Step model simply repeats the classification from the previous steps. Dynamic stopping would be even more human-like, as people do not usually keep staring at an image after they've figured out what it contains. It might be tricky to perform this operation in a fully-differentiable manner, but it is definitely worth looking into.

Second, my analyses did not dive deep enough into the raw weights. The Last-Step model's preference for 1, 3, and 5 remains elusive, as does the exact workings of the decoder LSTM that leads to odd window movement. With more time and more powerful tools, the accuracy and behavior issues could be traced to the source.

Finally, I did not get the chance to compare the model to real human performance. Given the aforementioned problem of fixed-length sequences, I doubt the model in its current state could be easily evaluated with regards to human experimental data. However, if the model were ”taught how to stop,” its accuracy-over-time could likely be compared to humans performing the same task.

5.3 Conclusion

The All-Step model not only models human attention better than the Last-Step model, it enjoys a significant accuracy boost. There are many questions yet to answer, but one thing is clear: this model is worthy of our attention.

6 Acknowledgements

I am enormously grateful to Jay McClelland, who gave me the idea for this project and provided invaluable feedback. I am also indebted to Steven Hansen for his analysis suggestions and TensorFlow debugging expertise. Finally, a huge thank-you to Jack Lindsey and Eric Jang for their well-written and easily-extendable code.

References

[1] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[2] Feng Wang and David M. J. Tax. Survey on the attention based RNN model and its applications in computer vision. *CoRR*, abs/1601.06823, 2016.

[3] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[4] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[5] Jack Lindsey. Dram. `https://github.com/jlindsey15/DRAM`, 2016.

[6] Eric Jang. draw. `https://github.com/ericjang/draw`, 2016.

[7] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.