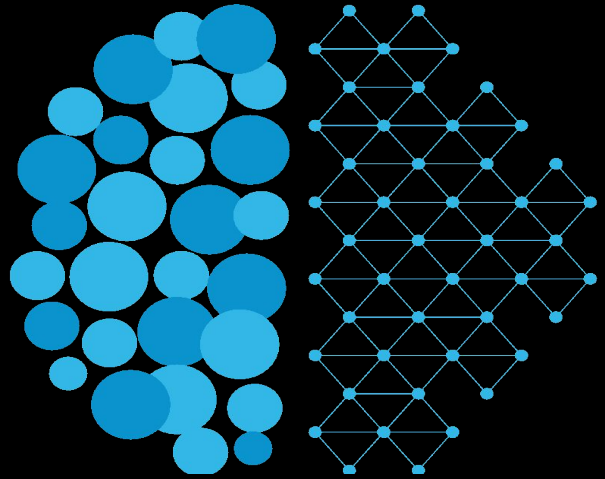


Institut
des algorithmes
d'apprentissage
de Montréal

MILA

# Professor Forcing: A new algorithm for training Recurrent Nets

Anirudh Goyal*, Alex Lamb*, Ying Zhang, Saizheng Zhang, Aaron Courville, Yoshua Bengio

* Equal contribution

# Introduction

- Discrepancy between how the model is used at training and inference.
- Mistakes made early in the sequence generation process can be quickly amplified (compounding errors)
  - Model might be in a part of the state space it has never seen at training time.

# Related Work

- Generative Adversarial Networks.
- Adversarial Domain Adaptation
- Scheduled Sampling (related to SEARN and DAGGER)
- Training Boltzmann Machines.

# Contributions

- Training generative RNNs to improve long-term sequence sampling from recurrent networks.
- Learn a generative model over **discrete random variables.**
- In some domains the sequences available at training time are shorter then the sequences that we want to generate at test time. Professor Forcing can help a lot with this.
- Can act as a regularizer for recurrent networks.

# Sequence Generation

-Sequence of random variables X[0], X[1] ……...

- Data is generated by a process that applies to all time steps:

$$p(x[n] \mid x[n-1], x[n-2], \ldots, x[0])$$

- Lets us generalize to arbitrary length sequences

- Applications: machine translation, speech synthesis, demand forecasting, weather/climate forecasting, text generation, video generation

# How do we learn this process?

- Learn an **open loop generator** to do one-step-ahead prediction.
- Recursively compose these outputs to do multi-step prediction **(closed-loop)**.
- Example time series: [1,2,4,8,16,32]. Model is: y[t] = y[t-1] * w.
- Train on the examples: 2 = 1 * w, 4 = 2 * w, 8 = 4 * w, .....
- Learn a model that conditions on the observed inputs.
- Use this model to generate as far forwards as we want:

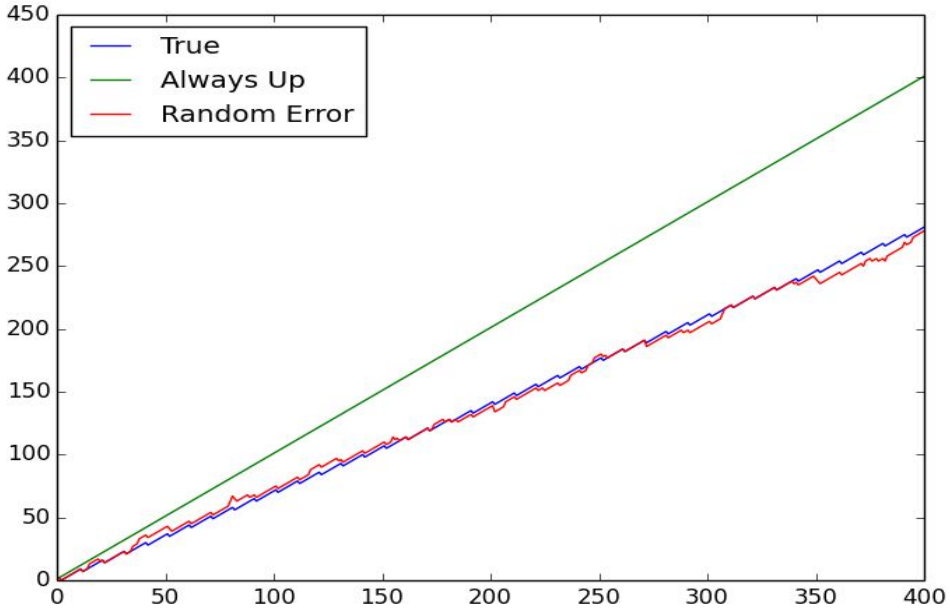
$$Y[t] = y[t - 1] * 2 \Rightarrow 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 \ldots\ldots$$

**“Generalizes to new lengths not seen during training”**

# Toy Example

- Goes up for 9 steps, then goes down for one step. Cycle repeats.
- Two models to consider:
  - Always go up.
  - Random up/down jump 20% of time
- Both have 20% error for one-step-ahead

Over 400 time steps: error is 43.6% vs. 2.6%

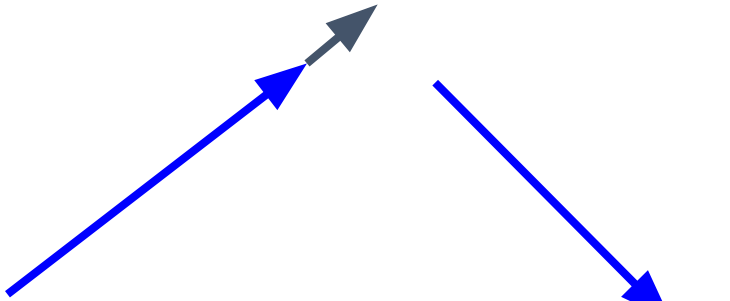


# Why does it go off the tracks?

- The *best* multi-step model is also the *best* one-step-ahead model (proof by induction)
- But merely *good* one-step-ahead models can be bad multi-step models.
- Accumulating errors vs. cancelling errors.
- The one-step ahead model is penalized equally for accumulating and cancelling errors.

# Why not train on closed-loop network?

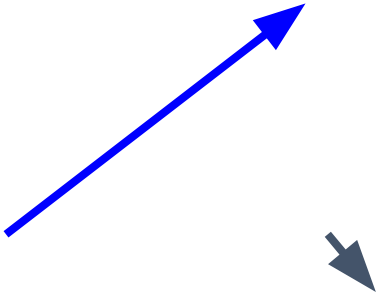
- Definitely not the right thing to do (but it can help in practice if used in moderation)



50% chance

50% chance

Real data



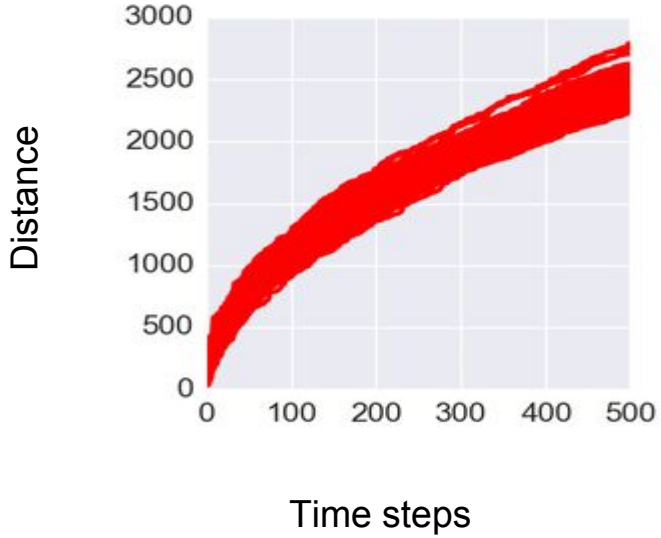
**Scheduled Sampling** trains our model to think that this can happen (blue arrow is sampled and green arrow is target.)

**Scheduled Sampling can greatly overestimate the variance in your distribution**

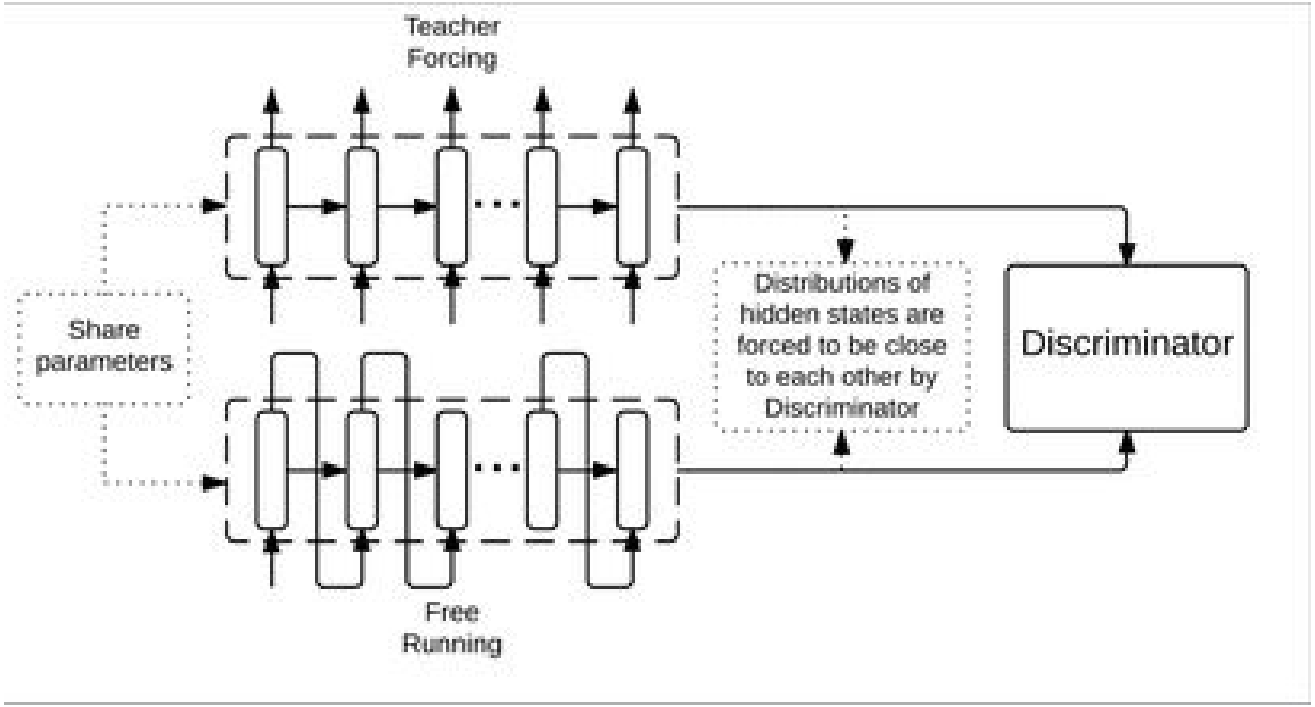
# What goes wrong?

Centroid distance b/w free running and Teacher forcing hiddens after taking T-SNE representation as a function of time steps.

(over 100 examples)



# Professor Forcing Description



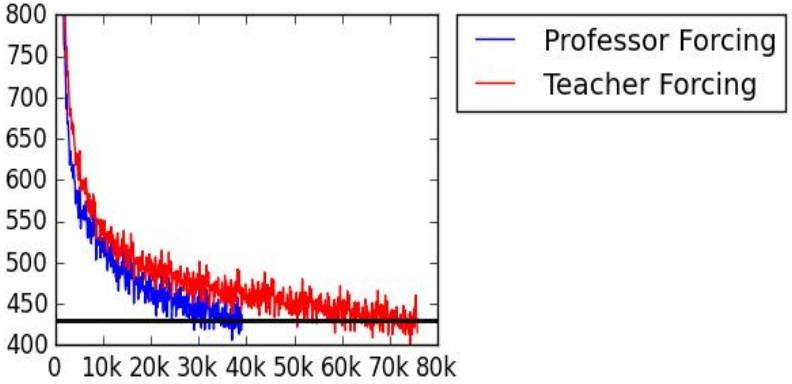
# Professor Forcing

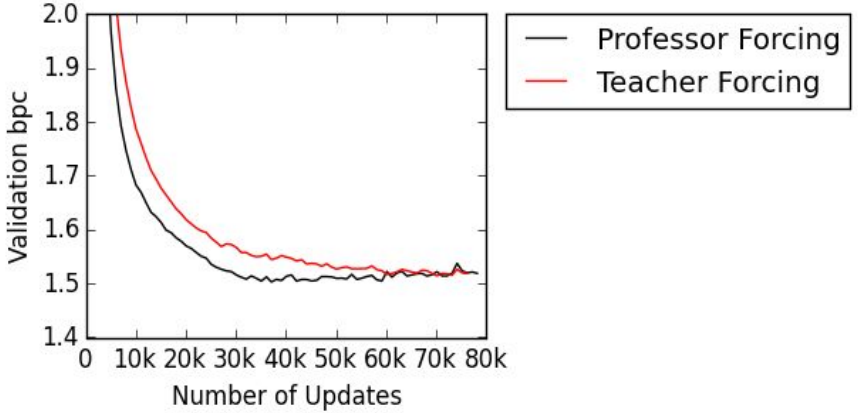
- Uses adversarial framework where we have two generators:
  - Closed loop - Open loop

- Train a classifier to learn: open loop vs. closed loop as a function of the sequence of hidden states

- Optimize the closed loop generator to fool the classifier.
- Optimize the open loop generator with teacher forcing.
- The closed loop and open loop generators share all parameters

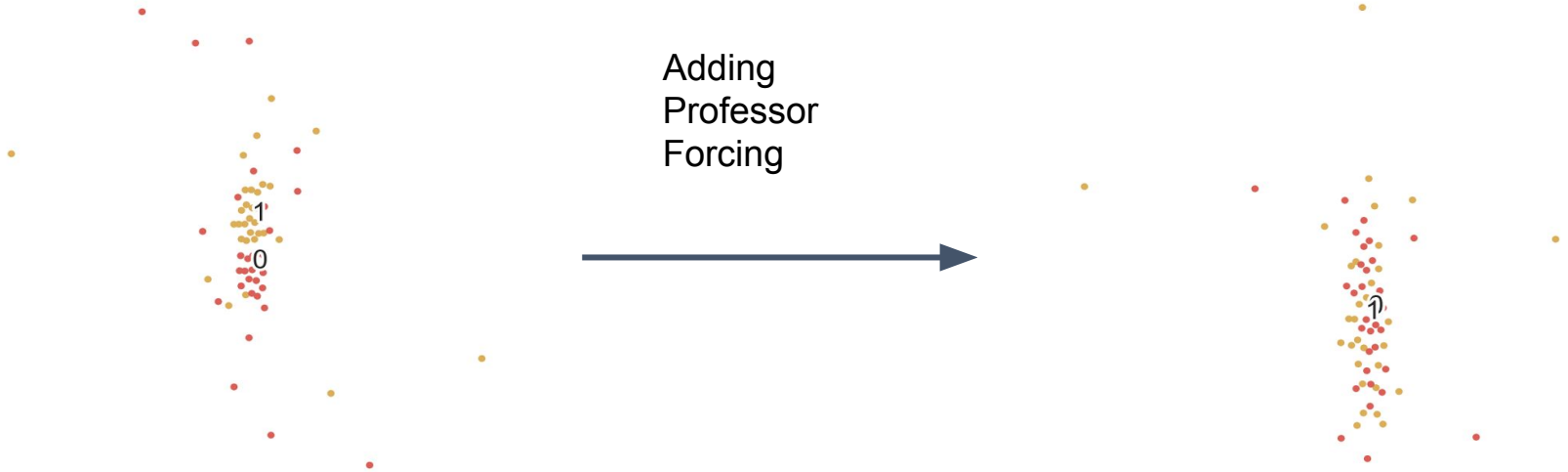
# Char Level Penn TreeBank




- Takes less number of updates during training to reach the similar log likelihood.
- Improves the bpc(bits per character) on validation/test set.
- Improved bpc due to better long term structure.

# Results - tsne




At t = 500, the closed-loop and open-loop hidden states clearly occupy distinct regions

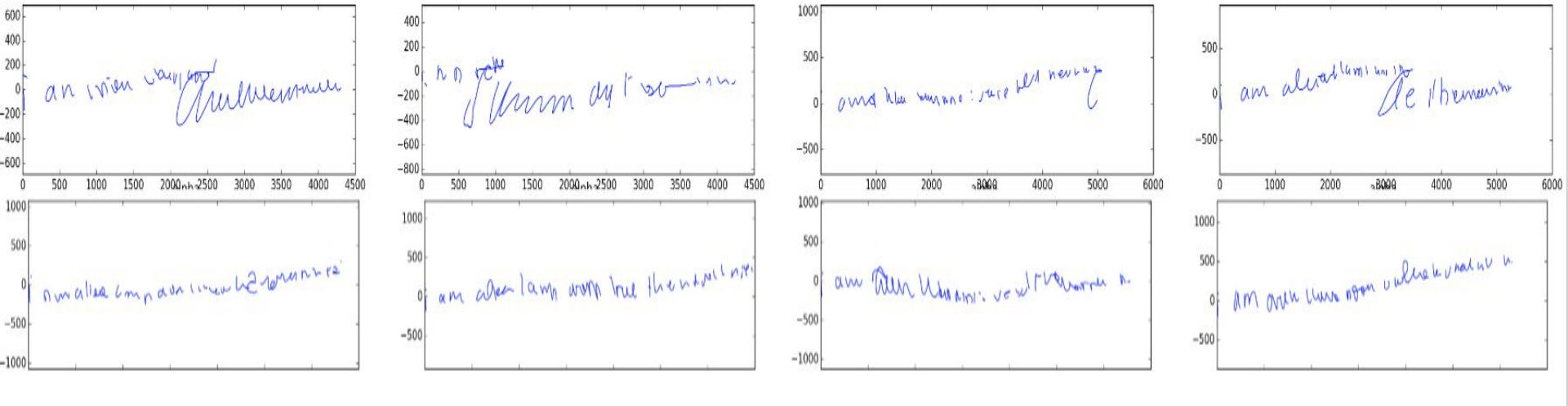
With professor forcing, these regions now largely overlap

# Sequential MNIST

Adding Professor forcing!

| Method | NLL (negative log likelihood) |
|---|---|
| Teacher Forcing (Ours) | 81.5 |
| **Professor Forcing** | 79.58 |
| PixelRNN(SOTA) | **79.2** |

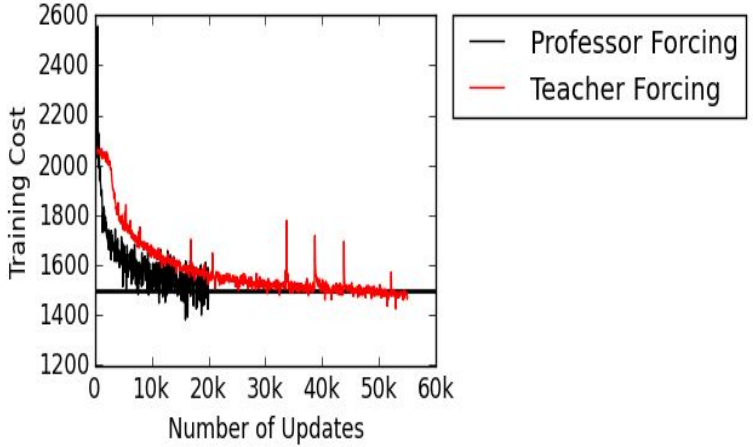
# Handwriting Generation

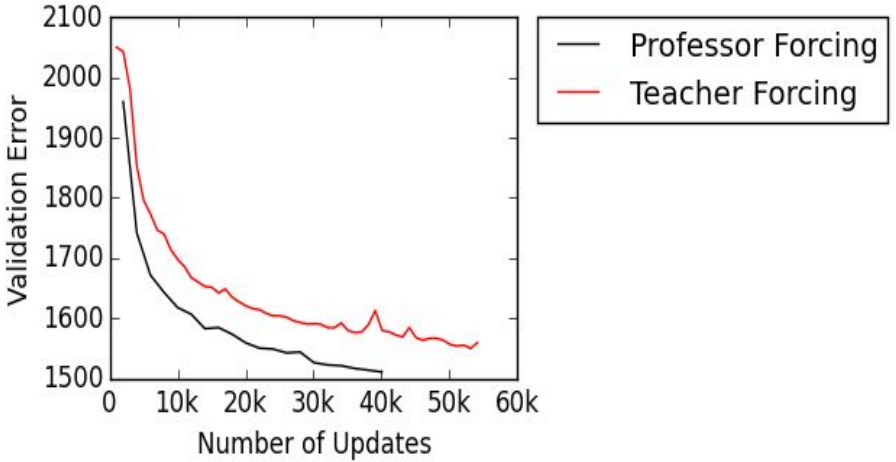


Handwriting samples with teacher forcing (top) and professor forcing (bottom)

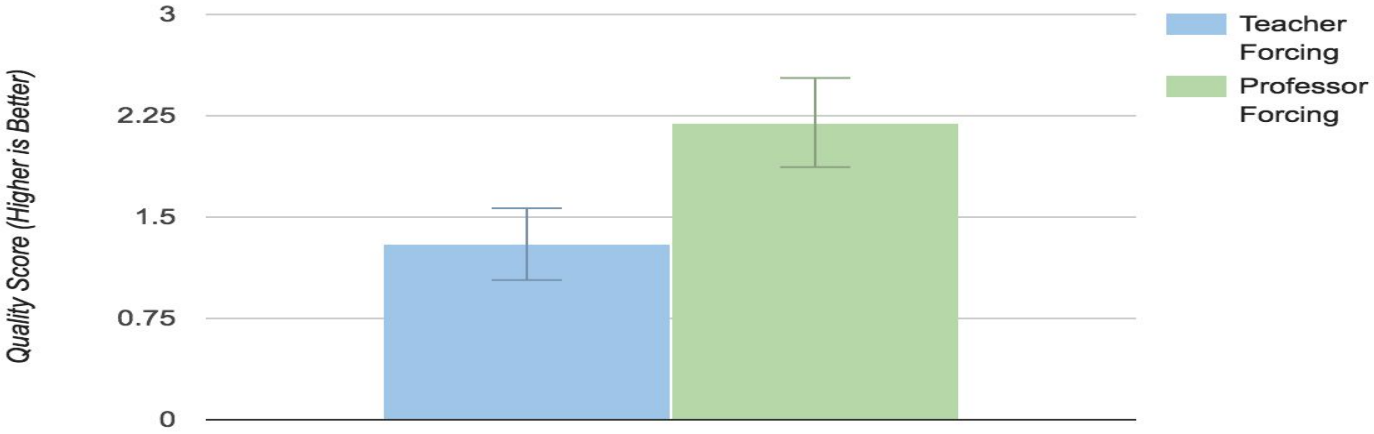
# Music Synthesis on Raw Waveforms

- 3 hours of monk chanting audio(https://www.youtube.com/watch?v=9-pD28iSiTU)



# Human Evaluation of generated music



# Connection to Boltzmann Machines

- Matching the behaviour of the model in
    - Free running - Constrained by observed data (clamped on “visible units”)
- Zeroing the maximum likelihood gradient on undirected graphical models with latent variables (such as Boltzmann machines)
- Training Boltzmann machines requires to match the statistics (which summarize the behavior of the model) in
    - Positive phase (teacher forcing) - Negative Phase (free running)

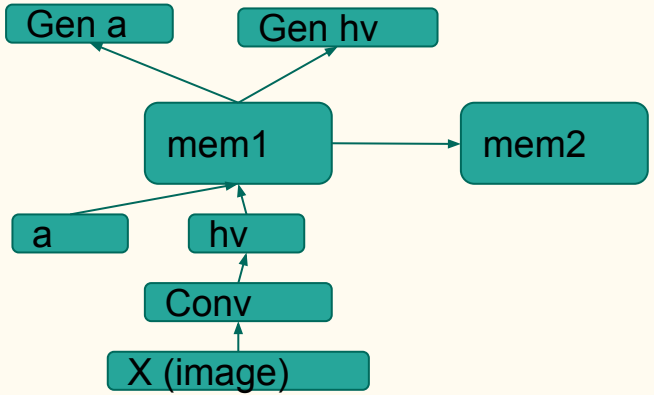
# Results

- Professor Forcing makes open-loop and closed-loop hidden representations occupy similar regions.  This effect grows with the length of the sequence.
- Professor Forcing improves the accuracy of one-step ahead predictions as well as optimization time.
- More coherent long term samples.

# Future Work - Ideas

- One perspective on this work is that adversarial networks learn to try to make two sets similar.
- Professor Forcing uses this to try to make open-loop and closed-loop networks have similar dynamics.
- Can we think about this as a more general tool for deep neural networks? Can we try to encourage all of the layers in a deep neural network to have the same dynamics, which might let us train deeper networks?
- One thing that's always bothered me about RNNs is that we use the same weights for all time steps, but the different time steps could have hidden states with very different distributions!

# Idea for a simple Supervised Starcraft “AI”




-Once we have such a network, how can we extend it by using reinforcement learning (to play “better” than the observed players)?

- RNN takes last input command and screen image as input.
- Estimates joint distribution over command vector and image-representation for next time step.
- Professor forcing could help ensure that we really have a good way to estimate the hidden representation of the image.
- Essentially we simulate the game environment.

# Questions ?