Deep Learning 4

Autoencoder, Attention (spatial transformer), Multi-modal learning, Neural Turing Machine, Memory Networks, Generative Adversarial Net

Jian Li

IIIS, Tsinghua

Autoencoder

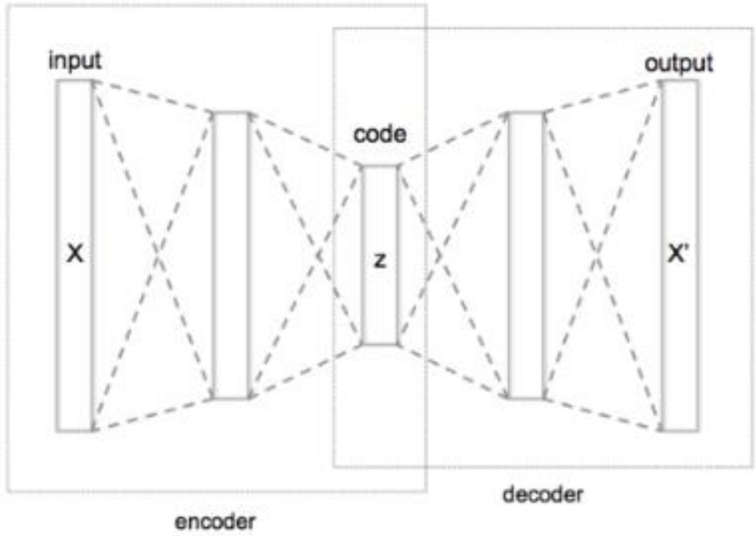
Autoencoder

- Unsupervised learning
    - Let the learning algorithm figure out the structure of the data (without supervised information)
        - Compact representation
        - Sparse representation
    - Representation learning (related to dictionary learning)

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2$$



Both the input and the output are x

Denoising Autoencoder

- Artificially add some noise to the input
  - The higher level representations are relatively stable and robust to the corruption of the input;
  - It is necessary to extract features that are useful for representation of the input distribution.

Sparse Autoencoder

- We can make the hidden layer larger, and at the same time encourage the **sparsity** of the code
  - By adding sparsity encouraging regularization term. E.g.

$$J_{\text{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} \mathrm{KL}(\rho \| \hat{\rho}_j),$$

Average activation of neuron j in the hidden layer

ρ : Bernoulli(0.05)

  - or manually zeroing all but the few strongest hidden unit activations

Variational autoencoder (VAE)

- Bayesian approach
- Perspective from variational inference

Prior of the code

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big(\log p_\theta(\mathbf{x}|\mathbf{z})\big)$$

The distr learnt by encoder to approximate the posterior distribution p(z|x)

Distr generated by the decoder

A quick intro to variational inference

- Typically, the posterior is hard to compute and sample from (MCMC approach can be pretty slow)
- We wish to use q (from some parametric family) to approximate the posterior p(z|x)

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathcal{Q}}{\arg\min}\ \mathrm{KL}\left(q(\mathbf{z}) \| p(\mathbf{z}\,|\,\mathbf{x})\right).$$

$$\mathrm{KL}\left(q(\mathbf{z}) \| p(\mathbf{z}\,|\,\mathbf{x})\right) = \mathbb{E}\left[\log q(\mathbf{z})\right] - \mathbb{E}\left[\log p(\mathbf{z}\,|\,\mathbf{x})\right]$$

$$\mathrm{KL}\left(q(\mathbf{z}) \| p(\mathbf{z}\,|\,\mathbf{x})\right) = \mathbb{E}\left[\log q(\mathbf{z})\right] - \mathbb{E}\left[\log p(\mathbf{z},\mathbf{x})\right] + \log p(\mathbf{x}).$$

Minimizing KL is equivalent to maximizing ELBO (since evidence logp(x) doesn’t depend on z)

ELBO: evidence lower bound
ELBO<=logp(x)

$$\mathrm{ELBO}(q) = \mathbb{E}\left[\log p(\mathbf{z},\mathbf{x})\right] - \mathbb{E}\left[\log q(\mathbf{z})\right]$$

$$\begin{aligned}\mathrm{ELBO}(q) &= \mathbb{E}\left[\log p(\mathbf{z})\right] + \mathbb{E}\left[\log p(\mathbf{x}\,|\,\mathbf{z})\right] - \mathbb{E}\left[\log q(\mathbf{z})\right] \\ &= \mathbb{E}\left[\log p(\mathbf{x}\,|\,\mathbf{z})\right] - \mathrm{KL}\left(q(\mathbf{z}) \| p(\mathbf{z})\right).\end{aligned}$$

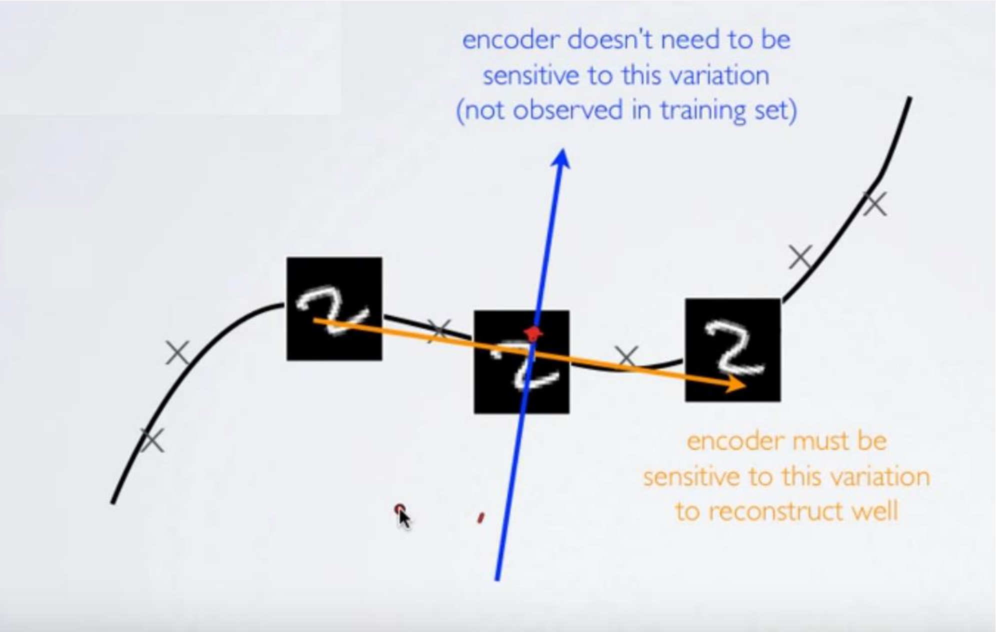
Contractive autoencoder (CAE)

- Perspective from manifold learning
- Encourage the encoding to be contractive

$$\mathcal{L}(\mathbf{x},\mathbf{x}') + \lambda \sum_i ||\nabla_x h_i||^2$$

Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input

$$||\nabla_{\mathbf{x}^{(t)}} \mathbf{h}(\mathbf{x}^{(t)})||_F^2 = \sum_j \sum_k \left(\frac{\partial h(\mathbf{x}^{(t)})_j}{\partial x_k^{(t)}}\right)^2$$



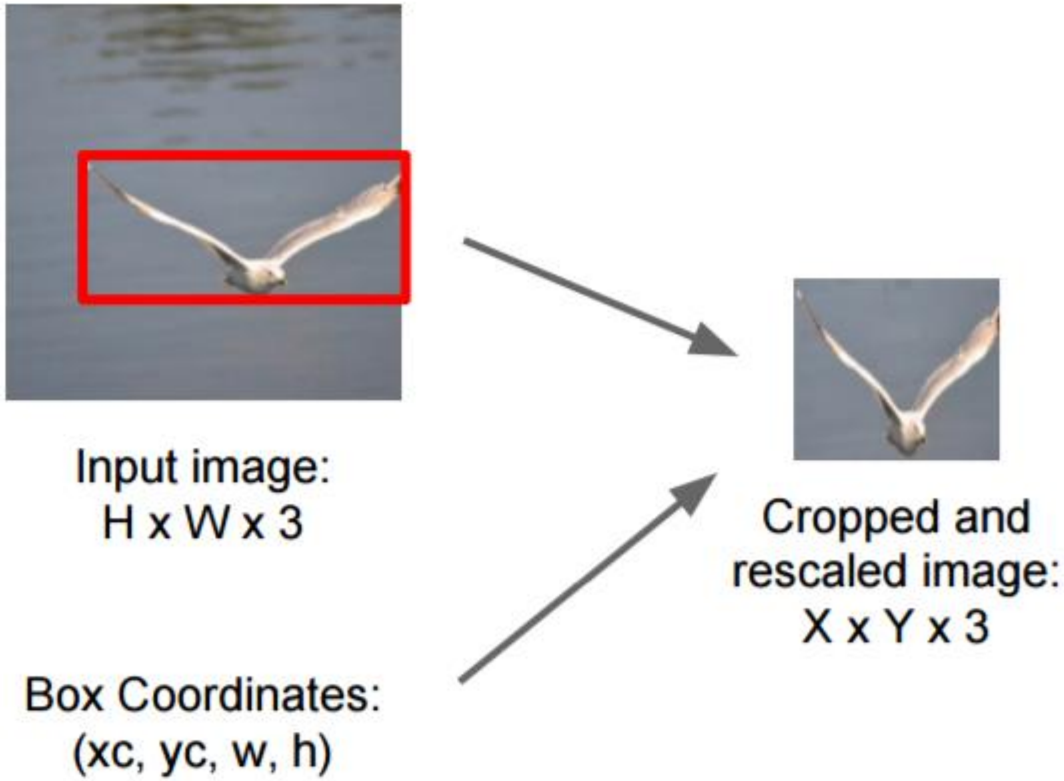
Related reading material: http://www.deeplearningbook.org/version-2015-10-03/contents/manifolds.html

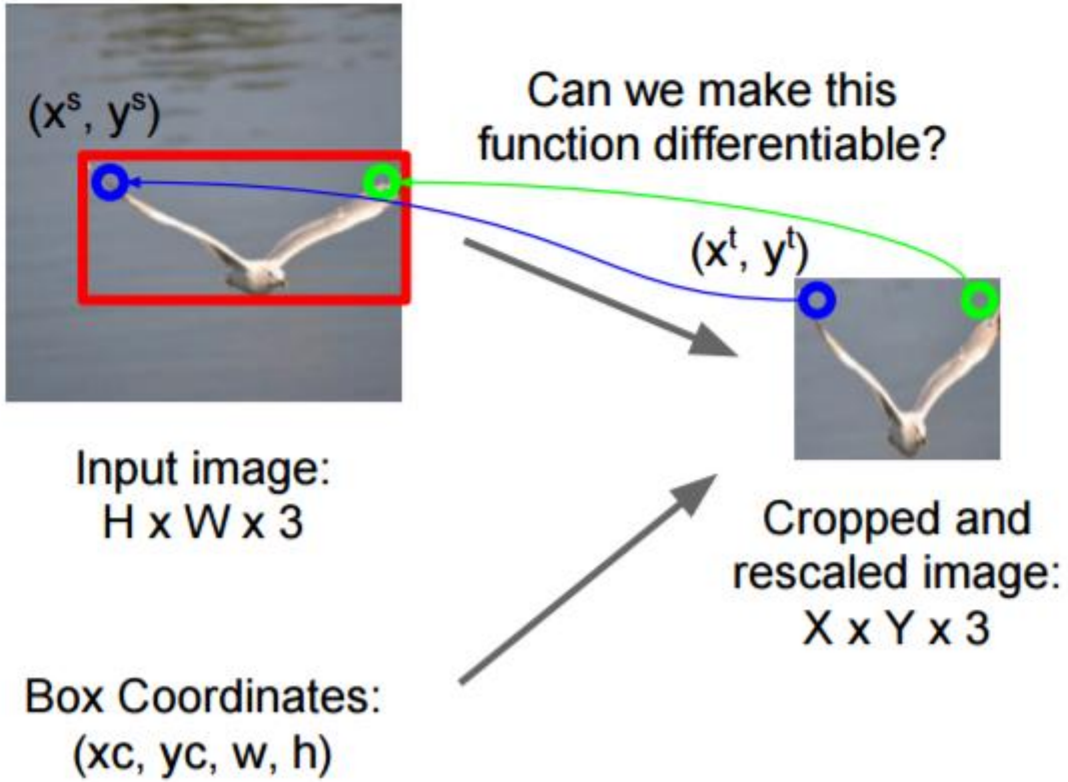
Spatial Transformer Networks

-an attention mechanism

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

- Would like to pay **attention** to certain areas of an image

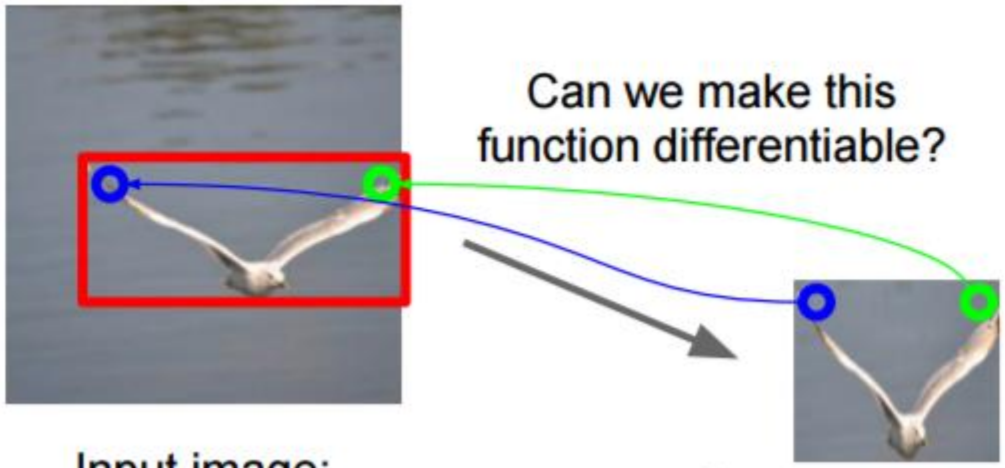




Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

Can we make this function differentiable?



Input image:
H x W x 3

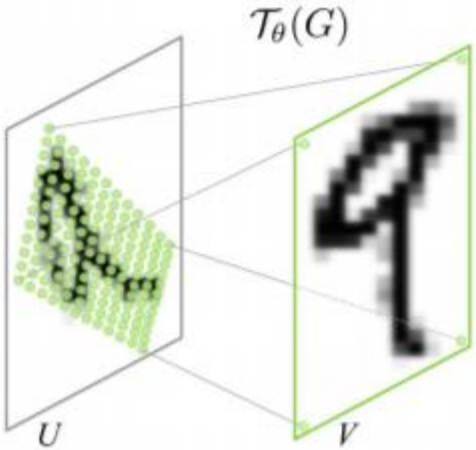
Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

θ: parameters we need to learn

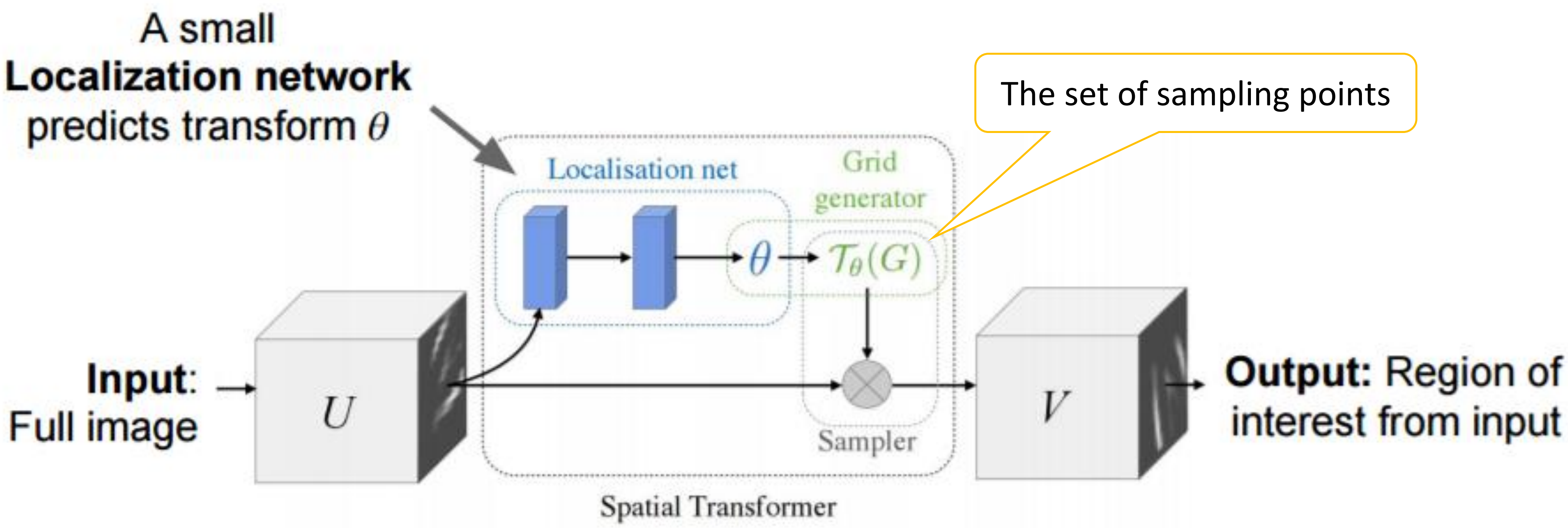
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



Repeat for all pixels in *output* to get a **sampling grid**

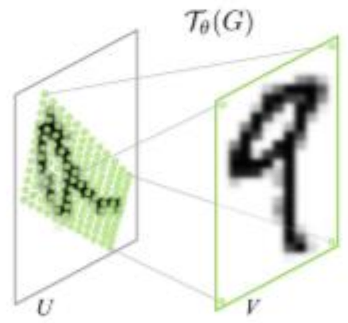
Affine transformation.
But it can be a more general transform

- A module can be inserted to any place of a network
  - Used several times in later deepmind papers



Grid generator uses θ to compute sampling grid

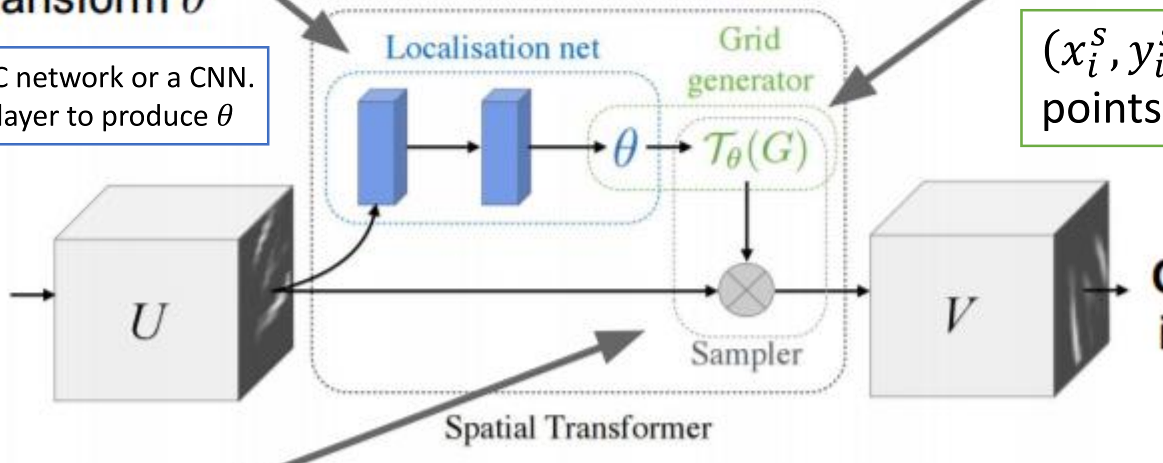
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



$(x_i^s, y_i^s) \in T_\theta(G)$ indicates which points in U we want to focus on

A small **Localization network** predicts transform θ

The localization network can be FC network or a CNN.
The last layer should a regression layer to produce θ



Input: Full image

Output: Region of interest from input

Sampler uses bilinear interpolation to produce output

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

Output V is determined by input U and sampling points $(x_i^s, y_i^s) \in T_\theta(G)$

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

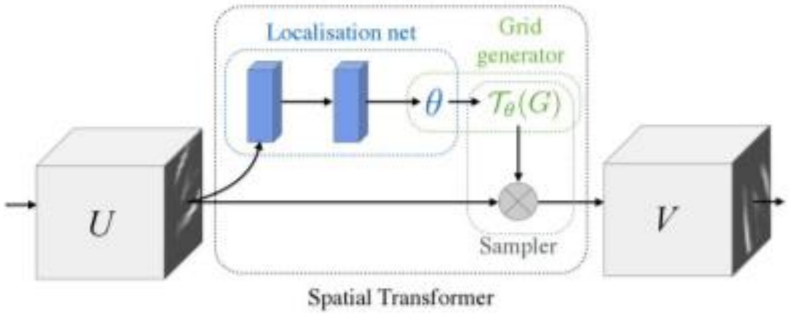
$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

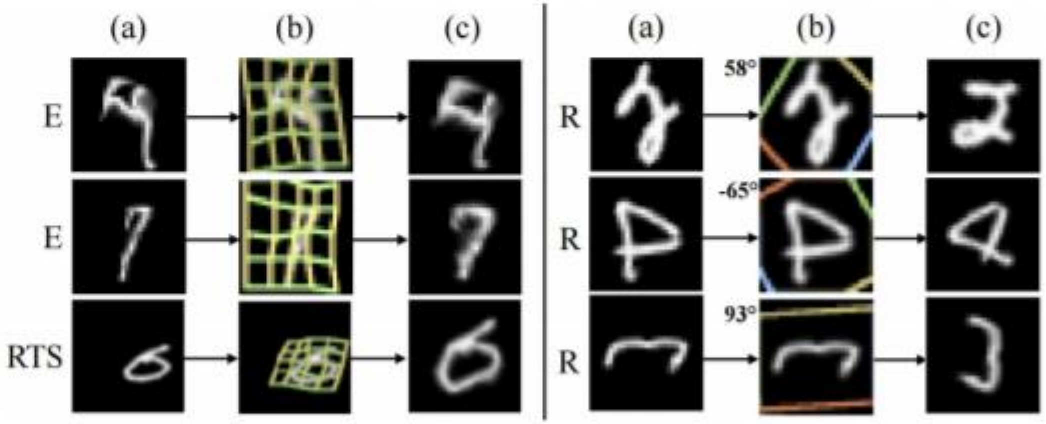
$\mathcal{T}_\theta = M_\theta B$, where B is a target grid representation

- We can even learn the target grid B (using "thin plate spline") (again through backprop)

Differentiable "attention / transformation" module



Insert spatial transformers into a classification network and it learns to attend and transform the input



Multimodal representation learning
---Image Caption 2

Kires et al. Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models

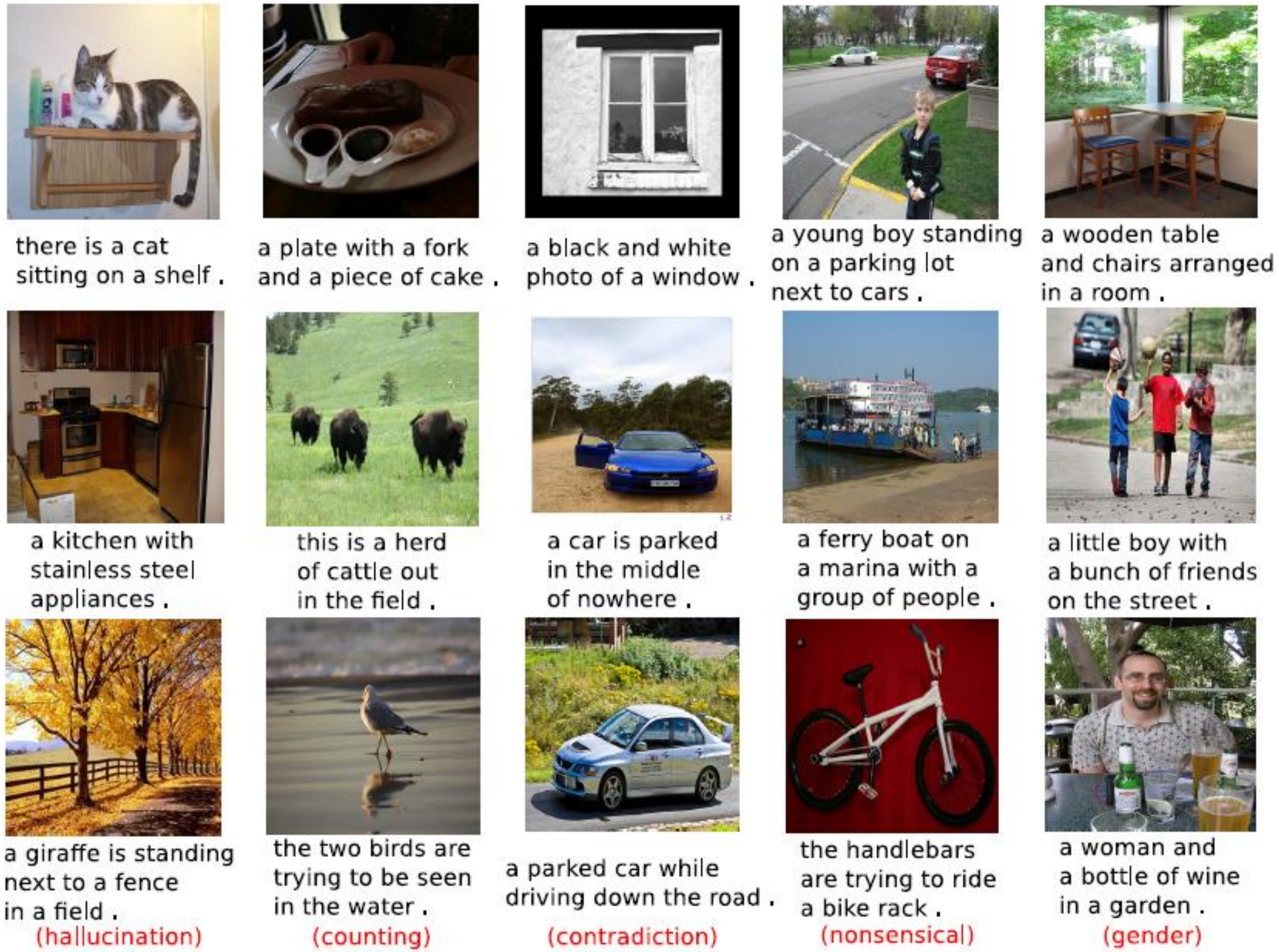
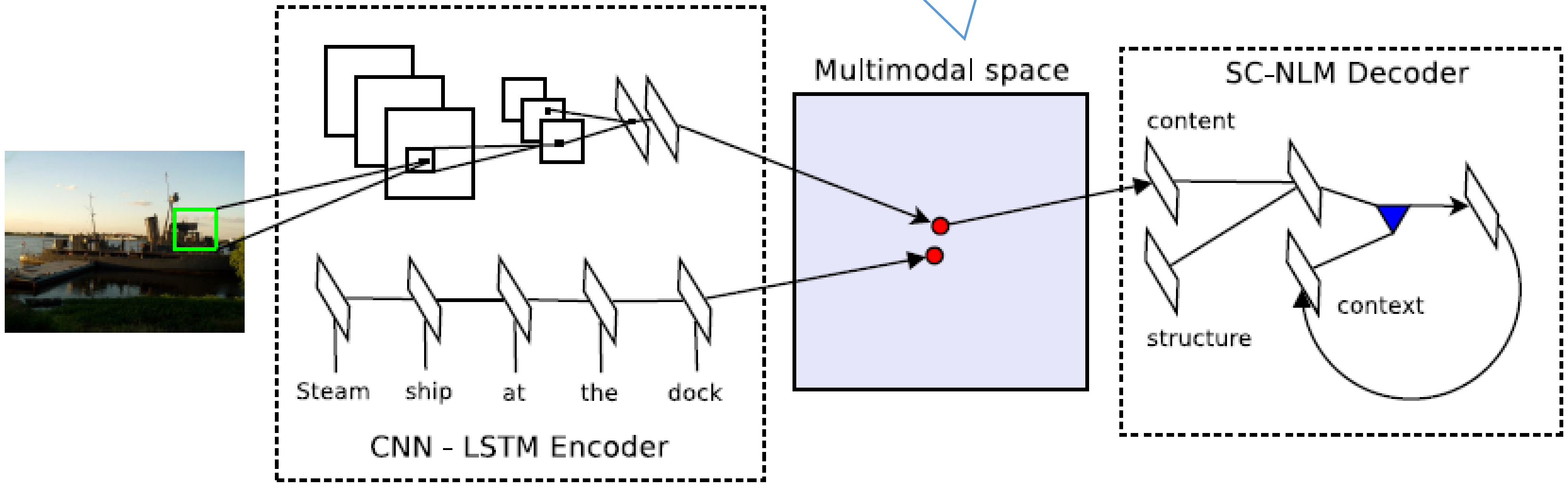


Figure 1: Sample generated captions. The bottom row shows different error cases. Additional results can be found at http://www.cs.toronto.edu/~rkiros/lstm_scnlm.html

Overview

Map CNN codes and RNN code to a common space



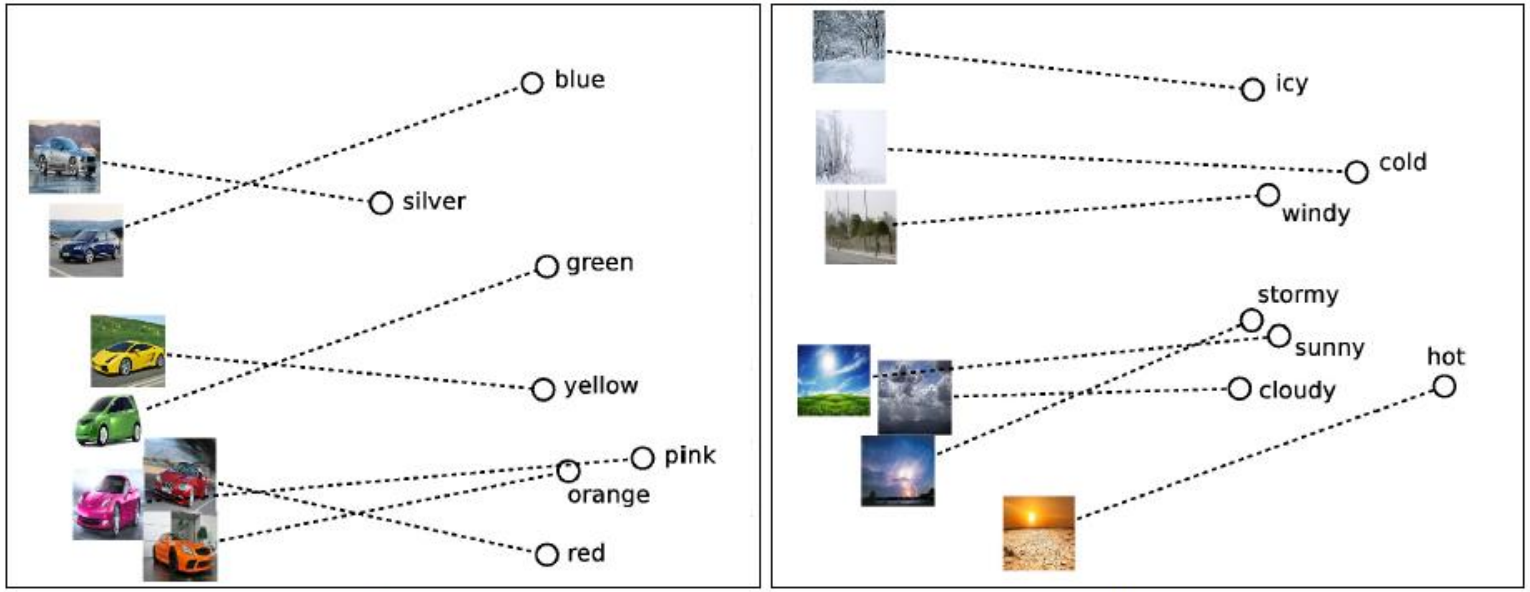
Details of SC-NLM. Please see the paper

Figure 2: **Encoder:** A deep convolutional network (CNN) and long short-term memory recurrent network (LSTM) for learning a joint image-sentence embedding. **Decoder:** A new neural language model that combines structure and content vectors for generating words one at a time in sequence.

(a) Simple cases (b) Colors (c) Image structure (d) Sanity check

Figure 4: Multimodal vector space arithmetic. Query images were downloaded online and retrieved images are from the SBU dataset.

$$
\begin{aligned}
\mathbf{v}_{car} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} \\
\mathbf{v}_{red} + \mathbf{v}_{car} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} + \mathbf{v}_{red} \\
\mathbf{I}_{rcar} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} + \mathbf{v}_{red}
\end{aligned}
$$



(a) Colors (b) Weather

Figure 5: PCA projection of the 300-dimensional word and image representations for (a) cars and colors and (b) weather and temperature.

Details

- LSTM notations used in this work

Let $\mathbf{X}_t$ denote a matrix of training instances at time t. In our case, $\mathbf{X}_t$ is used to denote a matrix of word representations for the t-th word of each sentence in the training batch. Let $(\mathbf{I}_t, \mathbf{F}_t, \mathbf{C}_t, \mathbf{O}_t, \mathbf{M}_t)$ denote the input, forget, cell, output and hidden states of the LSTM at time step t. The LSTM architecture in this work is implemented using the following equations:

$$
\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xi} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{C}_{t-1} \cdot \mathbf{W}_{ci} + \mathbf{b}_i) && (1)\\
\mathbf{F}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xf} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{C}_{t-1} \cdot \mathbf{W}_{cf} + \mathbf{b}_f) && (2)\\
\mathbf{C}_t &= \mathbf{F}_t \bullet \mathbf{C}_{t-1} + \mathbf{I}_t \bullet tanh(\mathbf{X}_t \cdot \mathbf{W}_{xc} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hc} + \mathbf{b}_c) && (3)\\
\mathbf{O}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xo} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{C}_t \cdot \mathbf{W}_{co} + \mathbf{b}_o) && (4)\\
\mathbf{M}_t &= \mathbf{O}_t \bullet tanh(\mathbf{C}_t) && (5)
\end{aligned}
$$

where (σ) denotes the sigmoid activation function, $(\cdot)$ indicates matrix multiplication and $(\bullet)$ indicates component-wise multiplication. [1]

Details

Let $\mathbf{q} \in \mathbb{R}^D$ denote an image feature vector

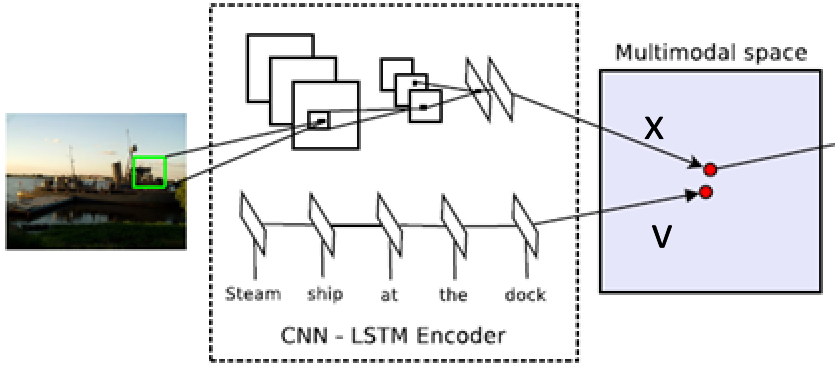
- D: length of the CNN code (CNN can be AlexNet, VggNet, or ResNet)

$\mathbf{x} = \mathbf{W}_I \cdot \mathbf{q} \in \mathbb{R}^K$ be the image embedding

image description $S = \{w_1, \ldots, w_N\}$ with words $w_1, \ldots, w_N$

$\{\mathbf{w}_1, \ldots, \mathbf{w}_N\}, \mathbf{w}_i \in \mathbb{R}^K, i = 1, \ldots, n$ denote the corresponding word representations to words $w_1, \ldots, w_N$ (entries in the matrix $\mathbf{W}_T$). The representation of a sentence $\mathbf{v}$ is the hidden state of the LSTM at time step N (i.e. the vector $\mathbf{m}_t$).

W_T: precomputed using e.g. word2vec (we don't learn it)



Details

- Optimize pairwise rank loss (θ:parameters needed to be learnt: W_I and LSTM parameters)

$$\min_{\boldsymbol{\theta}} \sum_{\mathbf{x}} \sum_{k} \max\{0, \alpha - s(\mathbf{x}, \mathbf{v}) + s(\mathbf{x}, \mathbf{v}_k)\} + \sum_{\mathbf{v}} \sum_{k} \max\{0, \alpha - s(\mathbf{v}, \mathbf{x}) + s(\mathbf{v}, \mathbf{x}_k)\}$$

Max-margin formulation. α margin

scoring function $s(\mathbf{x}, \mathbf{v}) = \mathbf{x} \cdot \mathbf{v}$,
$\mathbf{v}_k$ is a contrastive (non-descriptive) sentence for image embedding $\mathbf{x}$, and vice-versa with $\mathbf{x}_k$.

Neural Turing Machine [Graves et al.]

“Memory”

concept of working memory.

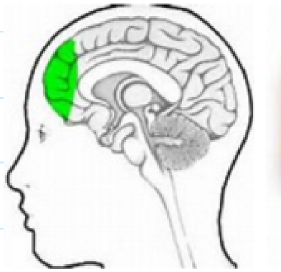
— in psychology: explain short term manipulation of information

a key concept is attention

focus attention on a few chuck of information and perform operation

— neuroscience: working mem is ascribed to prefrontal cortex and basal ganglia

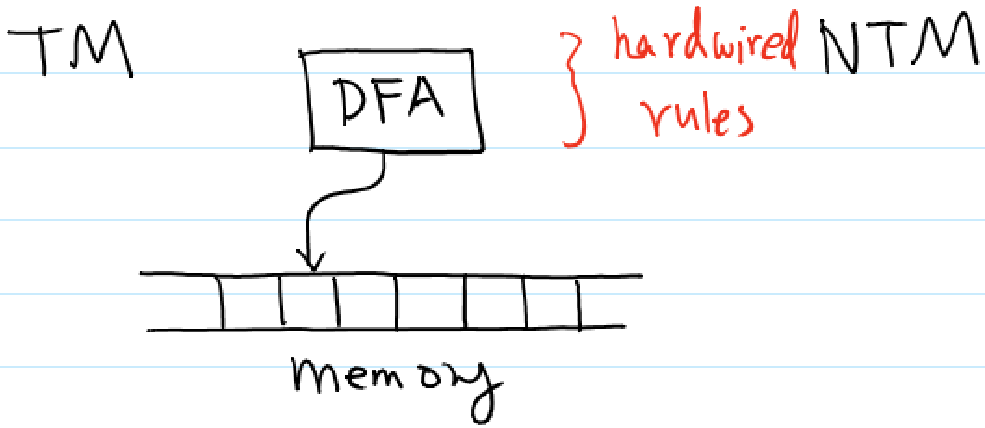
https://en.wikipedia.org/wiki/Prefrontal_cortex

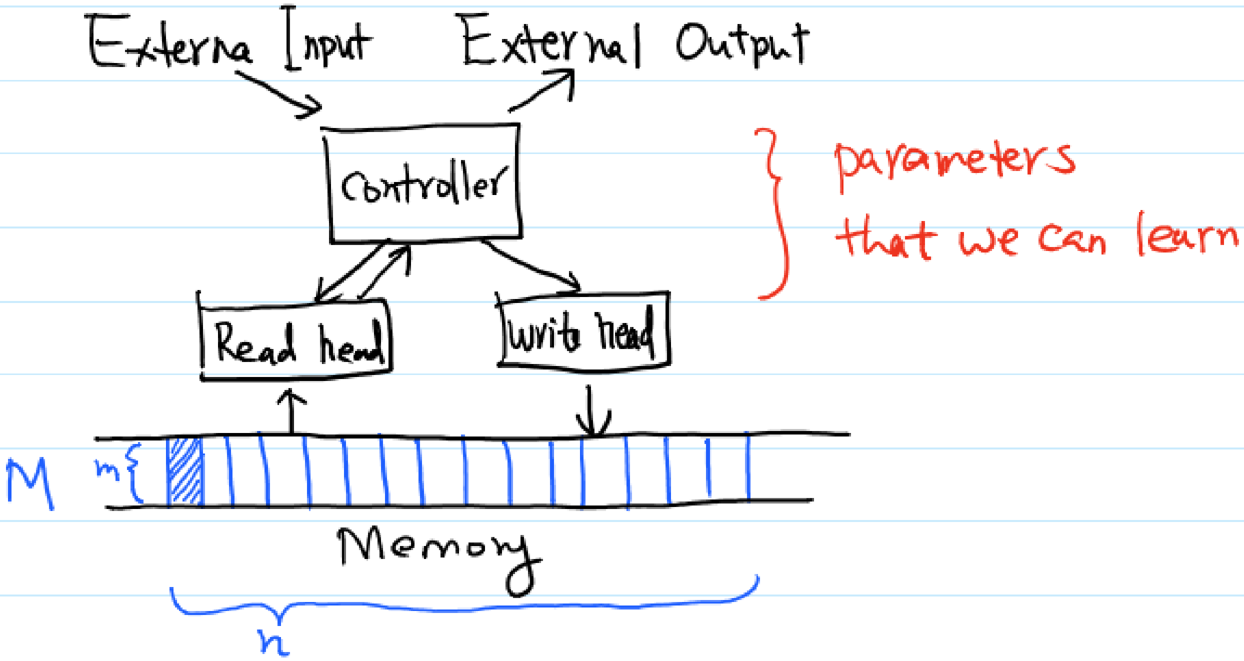


- experiment in Monkeys (neuron level)

Overview

NTM architechture





NTM can be through as a "continous" Turing Machine in which we can apply gradient decent.

Read

Read: • M_t : $N \times m$ — one mem location — memory at time t

w_t : N — weight vector (emit by read head)

$\underbrace{r_t}_{m} \leftarrow \sum_i w_t(i)\, M_t(i)$ — read vector

(if $w_t = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$, r_t is one row of M_t)

Write

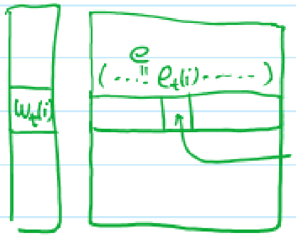
Write: first erase, then add.

w_t: N — weight vector (emit by write head)

Erase: $$\tilde{M}_t(i) \leftarrow M_{t-1}(i) \overset{\text{pointwise}}{\cdot} [\mathbf{1} - w_t(i)\, e_t]$$

$\mathbf{1}$: $\underbrace{(1, 1 \cdots 1)}_{m}$

e_t: erase vector $\underbrace{(0.1, 0, 0, 0.99, \cdots)}_{m}$ in $(0,1)$



it is erased if $w_t(i)$ and $e_t(i)$ are 1

Add: $$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)\, a_t$$

a_t: add vector $\underbrace{(\qquad\qquad)}_{m}$

Addressing Mechanism (overview)

Where to look at in the memory

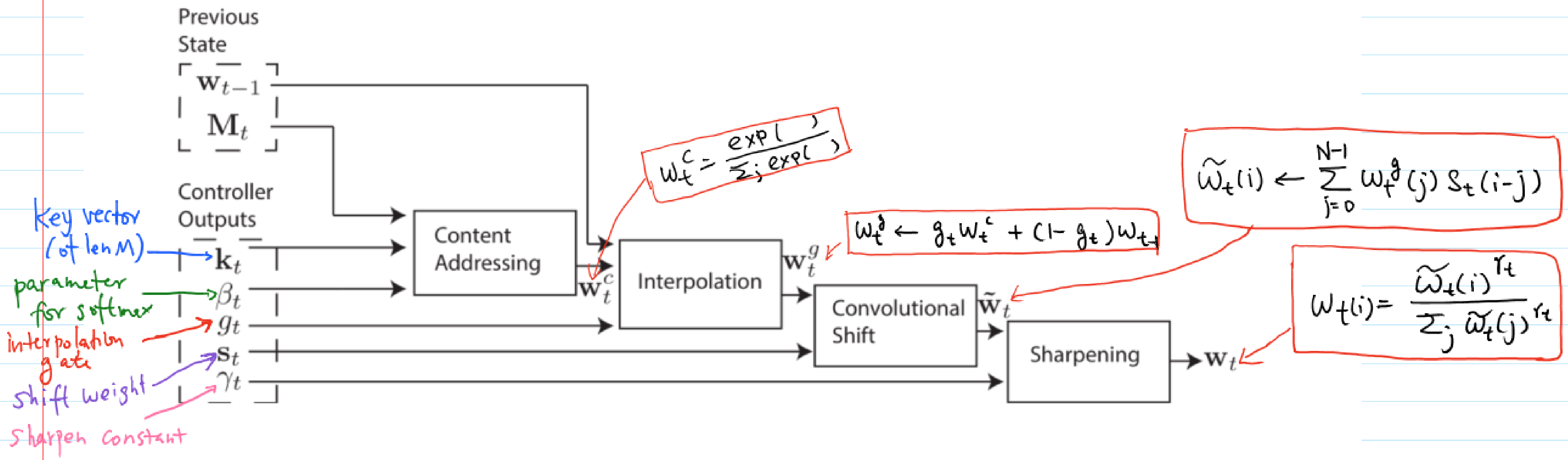


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

Addressing (details)

① Content addressing

key vector (of len M)

$$w_t^c(i) = \frac{\exp(\beta_t K[k_t, M_t(i)])}{\sum_j \exp(\beta_t K[k_t, M_t(j)])}$$

similarity $K[u,v] = \frac{\langle u, v \rangle}{\|u\| \|v\|}$

w_t^c : a normalized vector (recording the similarity of $M_t(i)$ and k_t)

Addressing (Details)

② location based addressing.

- interpolation gate $g_t \in (0,1)$

$$w_t^g \leftarrow g_t w_t^c + (1-g_t) w_{t-1}$$

(if $g=0$, we don't use content-based addressing)

- shift weight S_t e.g. $S_t = \begin{pmatrix} -1, & 0, & +1 \\ 0.9, & 0.05, & 0.05 \end{pmatrix}$

w.p. 0.9. shift by -1

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j)\, S_t(i-j)$$ ← Convolution

$w_t^g : (0, \quad , 0, 1, 0, \cdots 0)$ $\quad \tilde{w}_t = (0, \cdots, 0.9, 0.05, 0.05, 0 \cdots)$

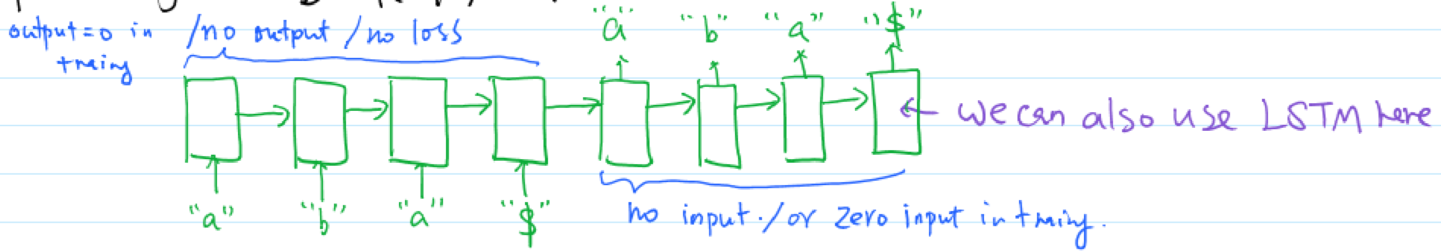
Sharpening :

$$w_t(i) \longleftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}} \quad (\gamma_t \geq 1)$$

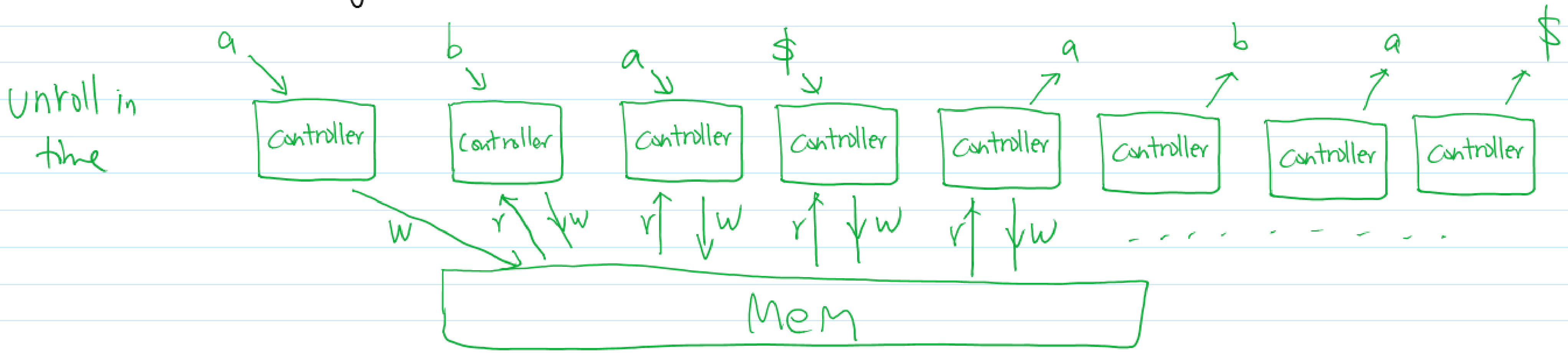
Go over the process

Go over the whole process (copy)

if we just use RNN/LSTM



Do the same using NTM



(Controller can be an ordinary feedforward NN (RNN/LSTM is also OK)

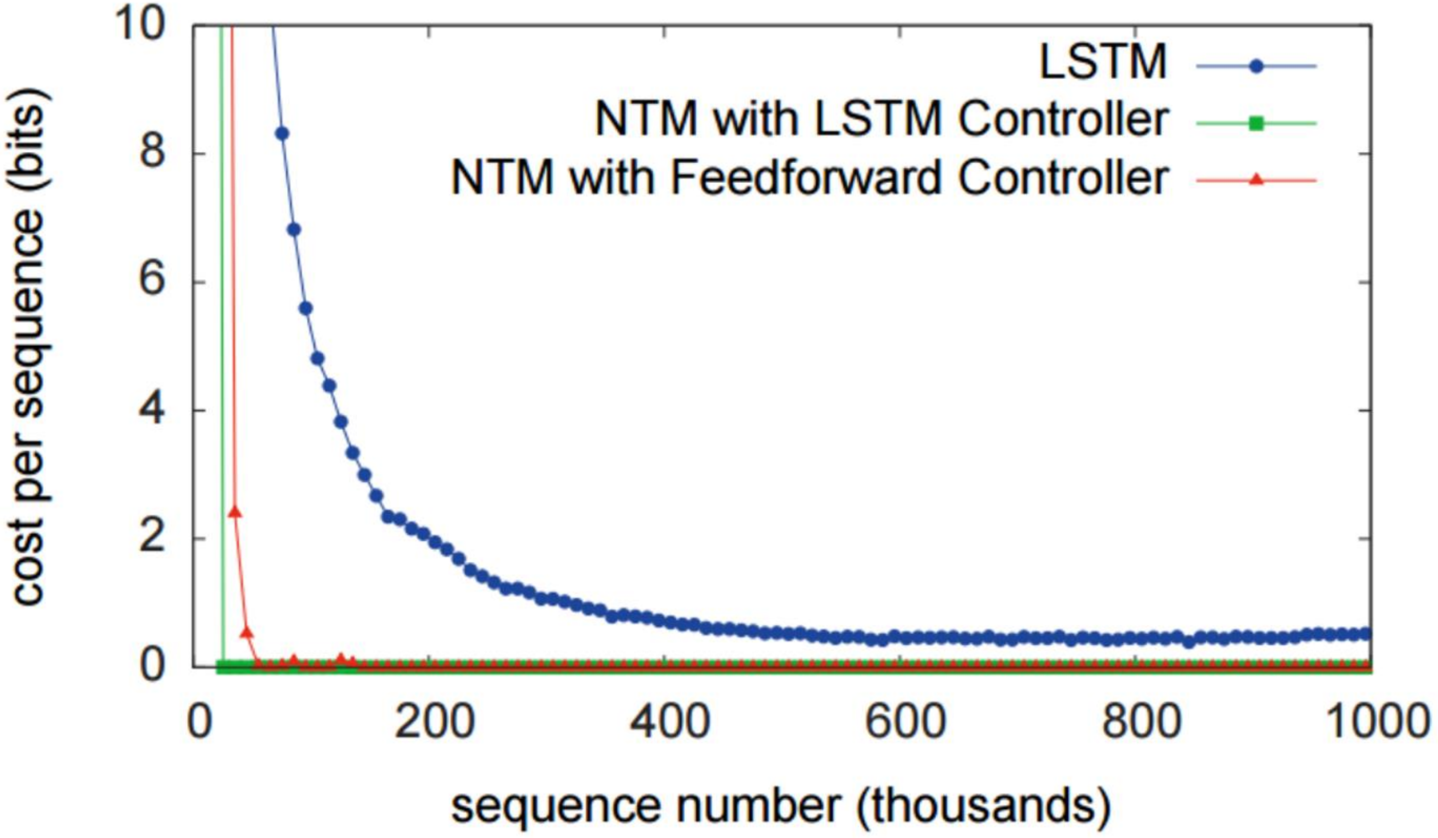


Figure 3: Copy Learning Curves.

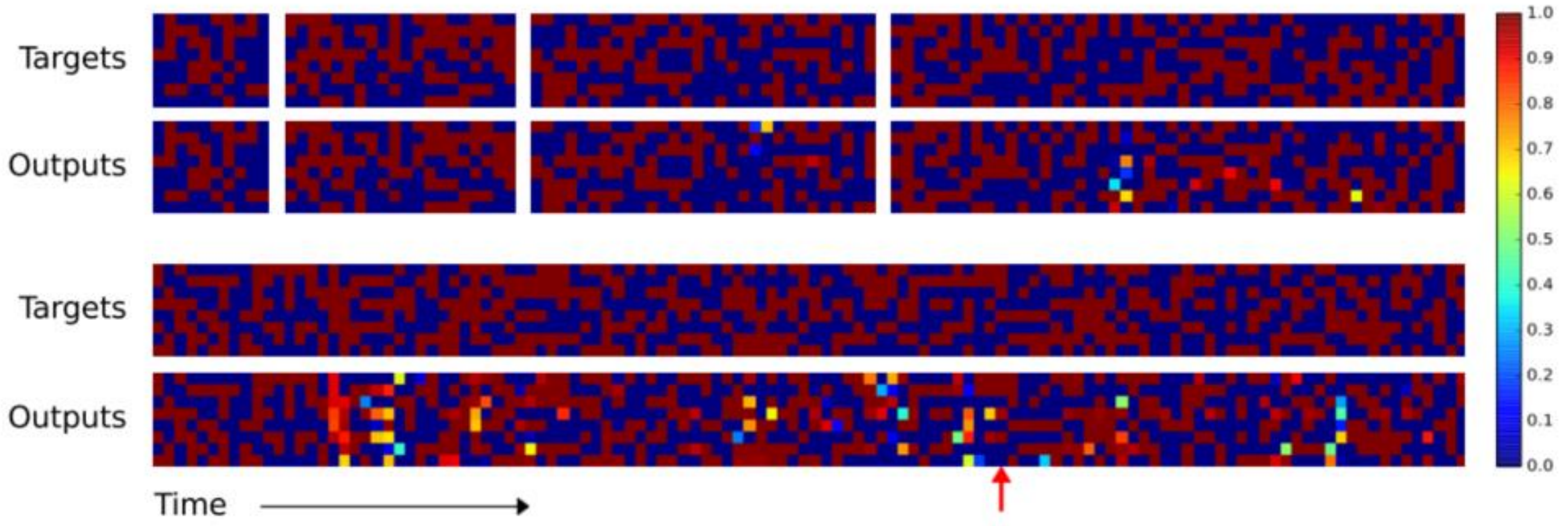


Figure 4: NTM Generalisation on the Copy Task. The four pairs of plots in the top row depict network outputs and corresponding copy targets for test sequences of length 10, 20, 30, and 50, respectively. The plots in the bottom row are for a length 120 sequence. The network was only trained on sequences of up to length 20. The first four sequences are reproduced with high confidence and very few mistakes. The longest one has a few more local errors and one global error: at the point indicated by the red arrow at the bottom, a single vector is duplicated, pushing all subsequent vectors one step back. Despite being subjectively close to a correct copy, this leads to a high loss.

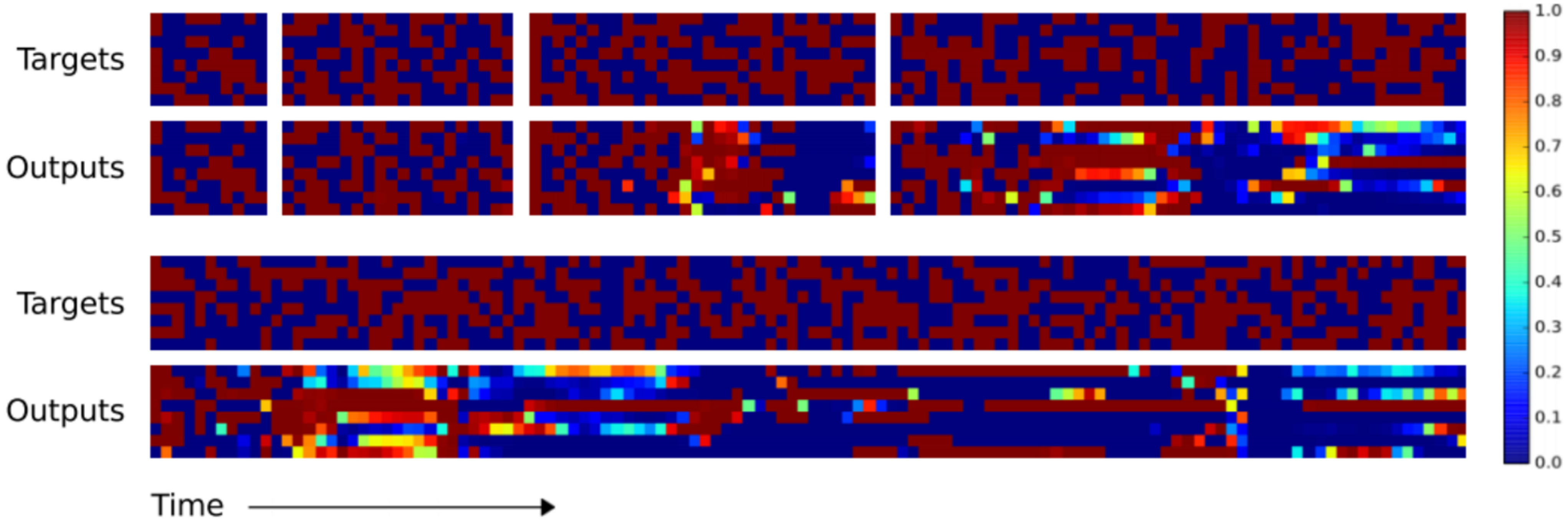


Figure 5: LSTM Generalisation on the Copy Task. The plots show inputs and outputs for the same sequence lengths as Figure 4. Like NTM, LSTM learns to reproduce sequences of up to length 20 almost perfectly. However it clearly fails to generalise to longer sequences. Also note that the length of the accurate prefix decreases as the sequence length increases, suggesting that the network has trouble retaining information for long periods.

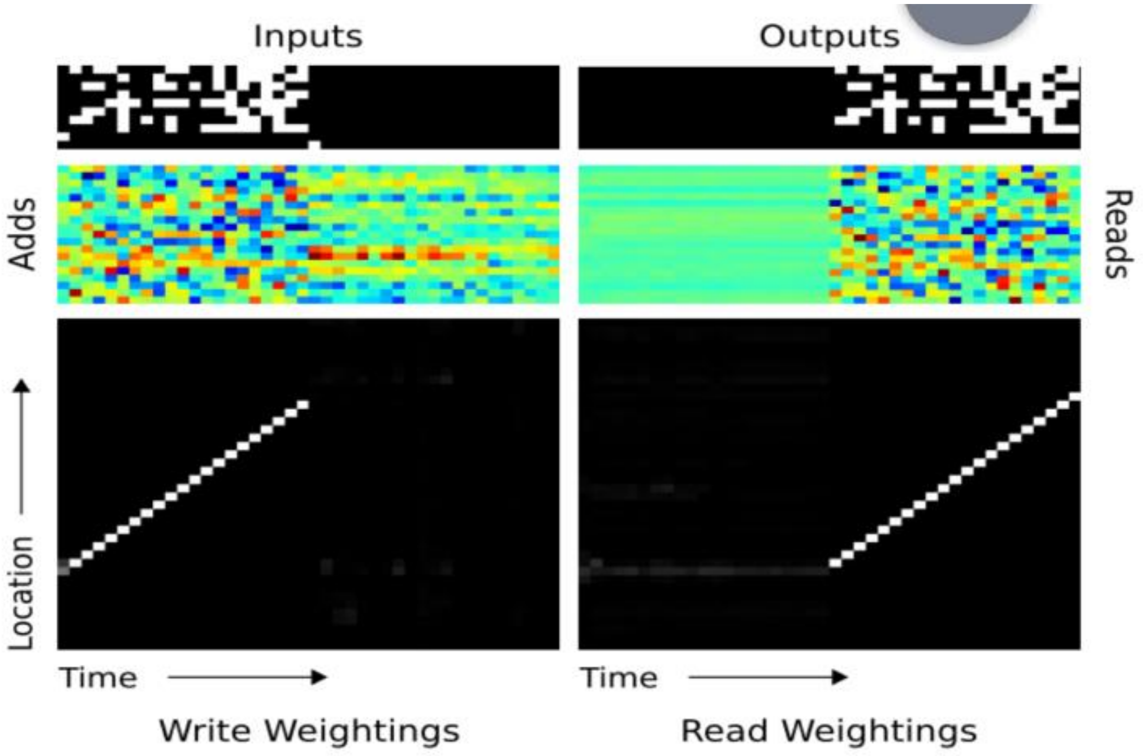


Figure 6: NTM Memory Use During the Copy Task. The plots in the left column depict the inputs to the network (top), the vectors added to memory (middle) and the corresponding write weightings (bottom) during a single test sequence for the copy task. The plots on the right show the outputs from the network (top), the vectors read from memory (middle) and the read weightings (bottom). Only a subset of memory locations are shown. Notice the sharp focus of all the weightings on a single location in memory (black is weight zero, white is weight one). Also note the translation of the focal point over time, reflects the network's use of iterative shifts for location-based addressing, as described in Section 3.3.2. Lastly, observe that the read locations exactly match the write locations, and the read vectors match the add vectors. This suggests that the network writes each input vector in turn to a specific memory location during the input phase, then reads from the same location sequence during the output phase.

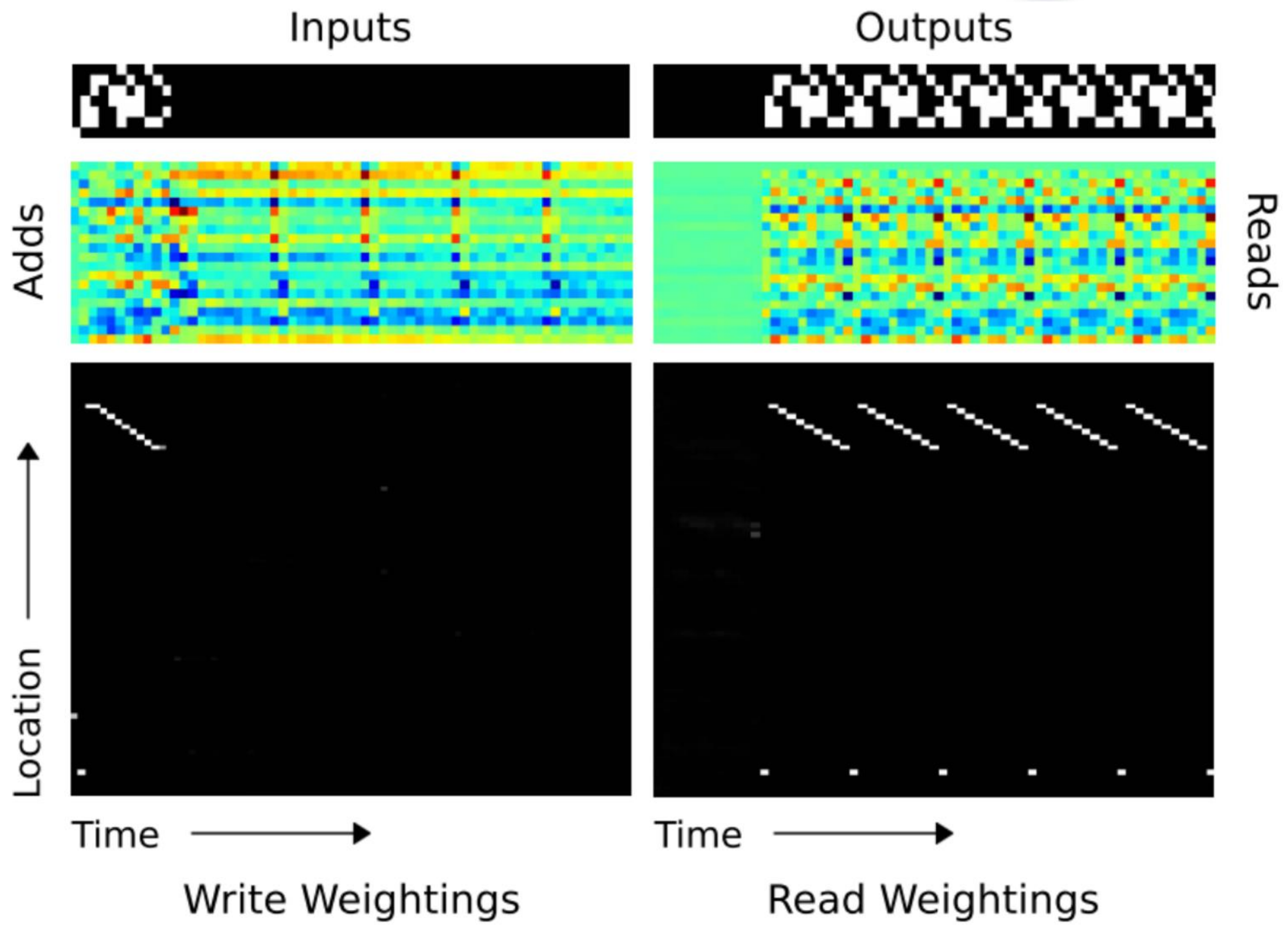


Figure 9: NTM Memory Use During the Repeat Copy Task. As with the copy task the network first writes the input vectors to memory using iterative shifts. It then reads through the sequence to replicate the input as many times as necessary (six in this case). The white dot at the bottom of the read weightings seems to correspond to an intermediate location used to redirect the head to the start of the sequence (The NTM equivalent of a *goto* statement).

a compressed representation,
written to memory.

generate the key accordingly
Use content-based addressing to locate

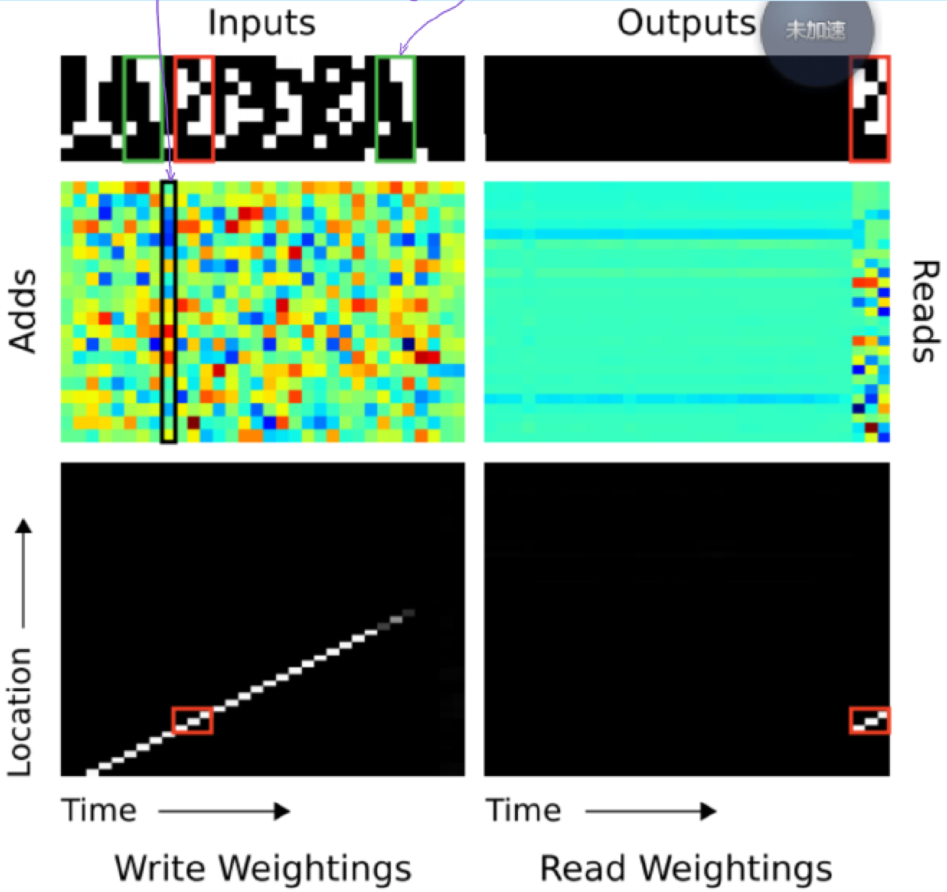
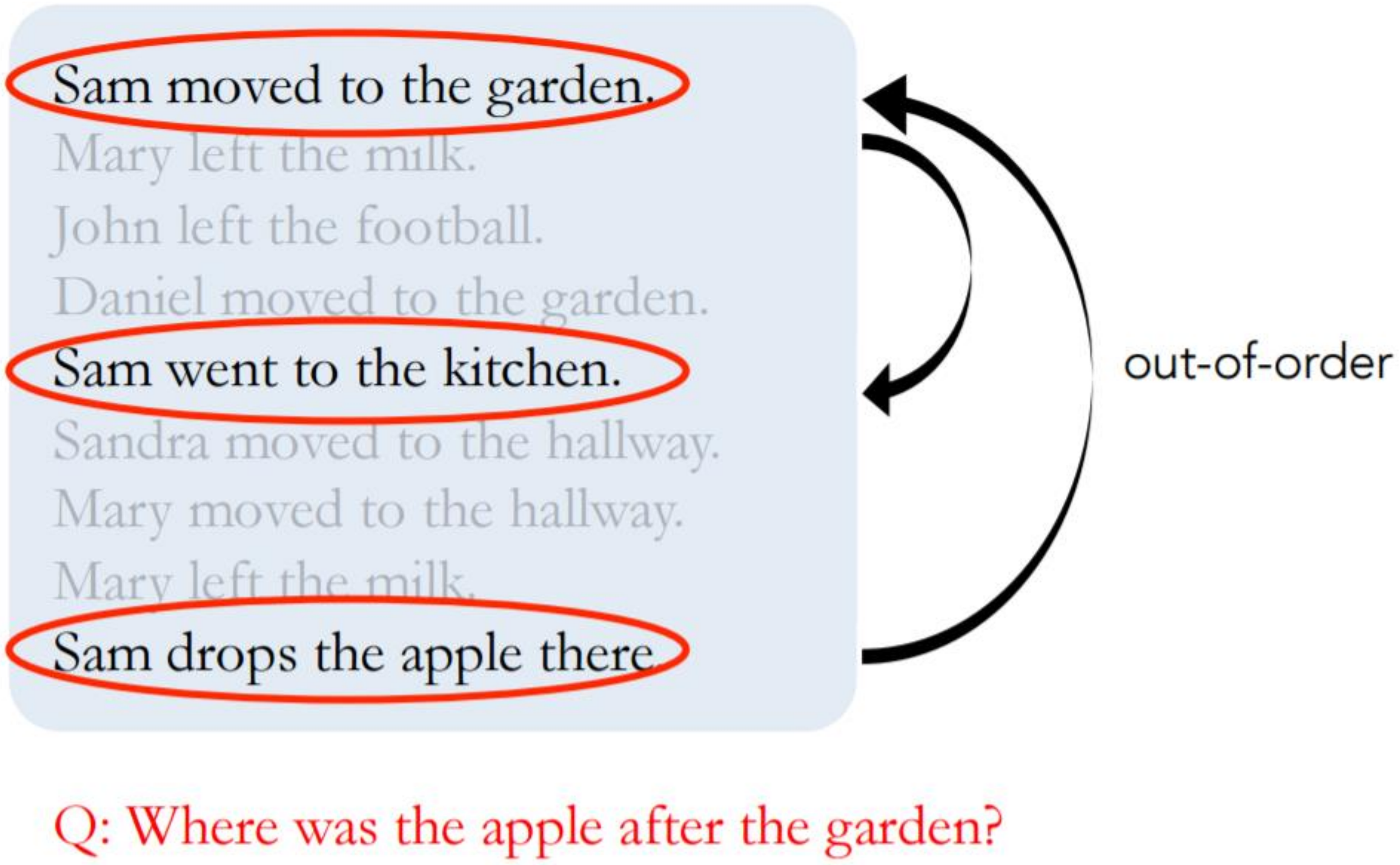


Figure 12: NTM Memory Use During the Associative Recall Task. In "Inputs," a sequence of items, each composed of three consecutive binary random vectors is propagated to the controller. The distinction between items is designated by delimiter symbols (row 7 in "Inputs"). After several items have been presented, a delimiter that designates a query is presented (row 8 in "Inputs"). A single query item is presented (green box), and the network target corresponds to the subsequent item in the sequence (red box). In "Outputs," we see that the network correctly produces the target item. The red boxes in the read and write weightings highlight the three locations where the target item was written and then read. The solution the network finds is to form a compressed representation (black box in "Adds") of each item that it can store in a single location. For further analysis, see the main text.

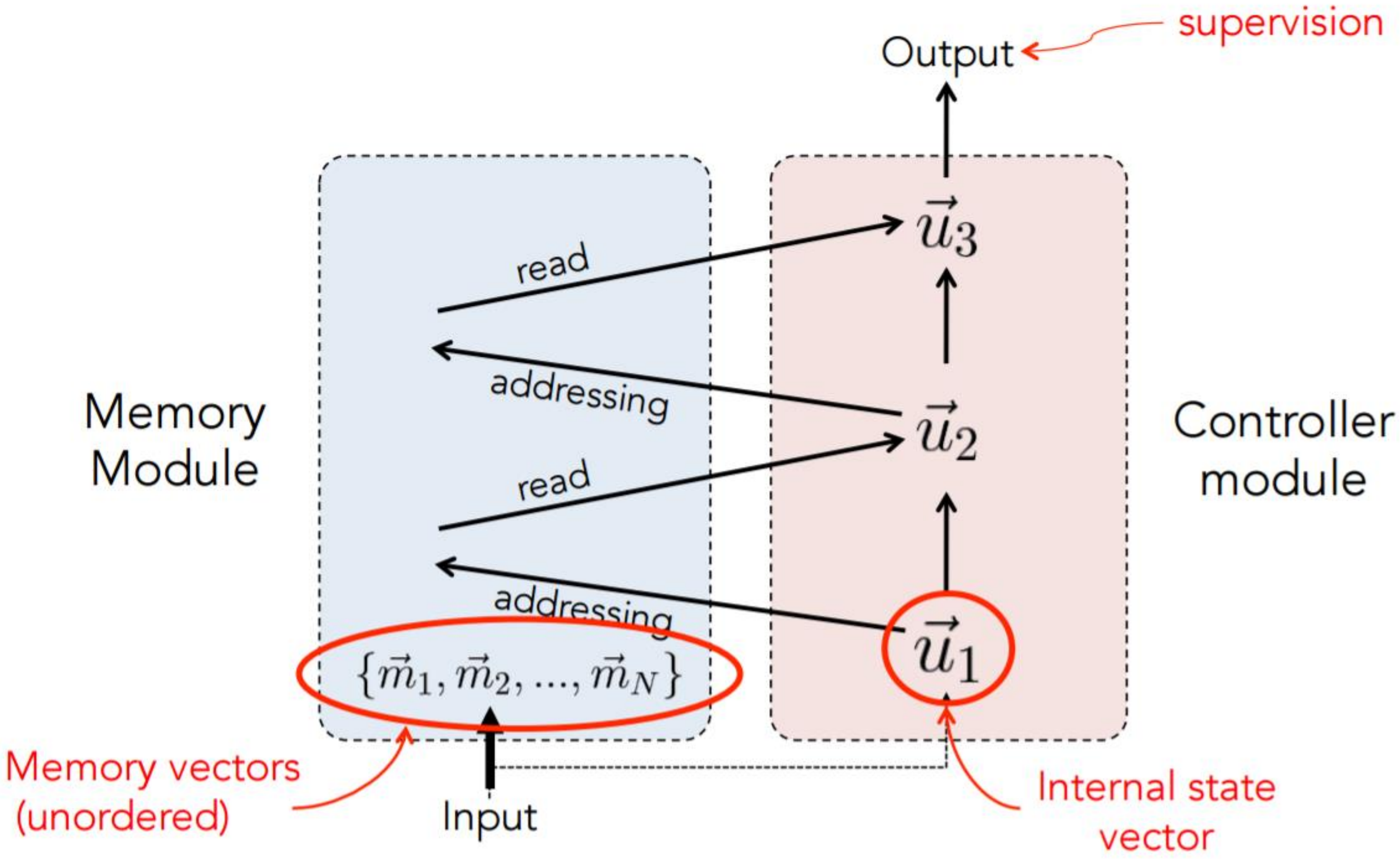
Memory Network

[Weston et al.][Sukhbaatar et al.]

Memory Network [Weston et al.][Sukhbaatar et al.]

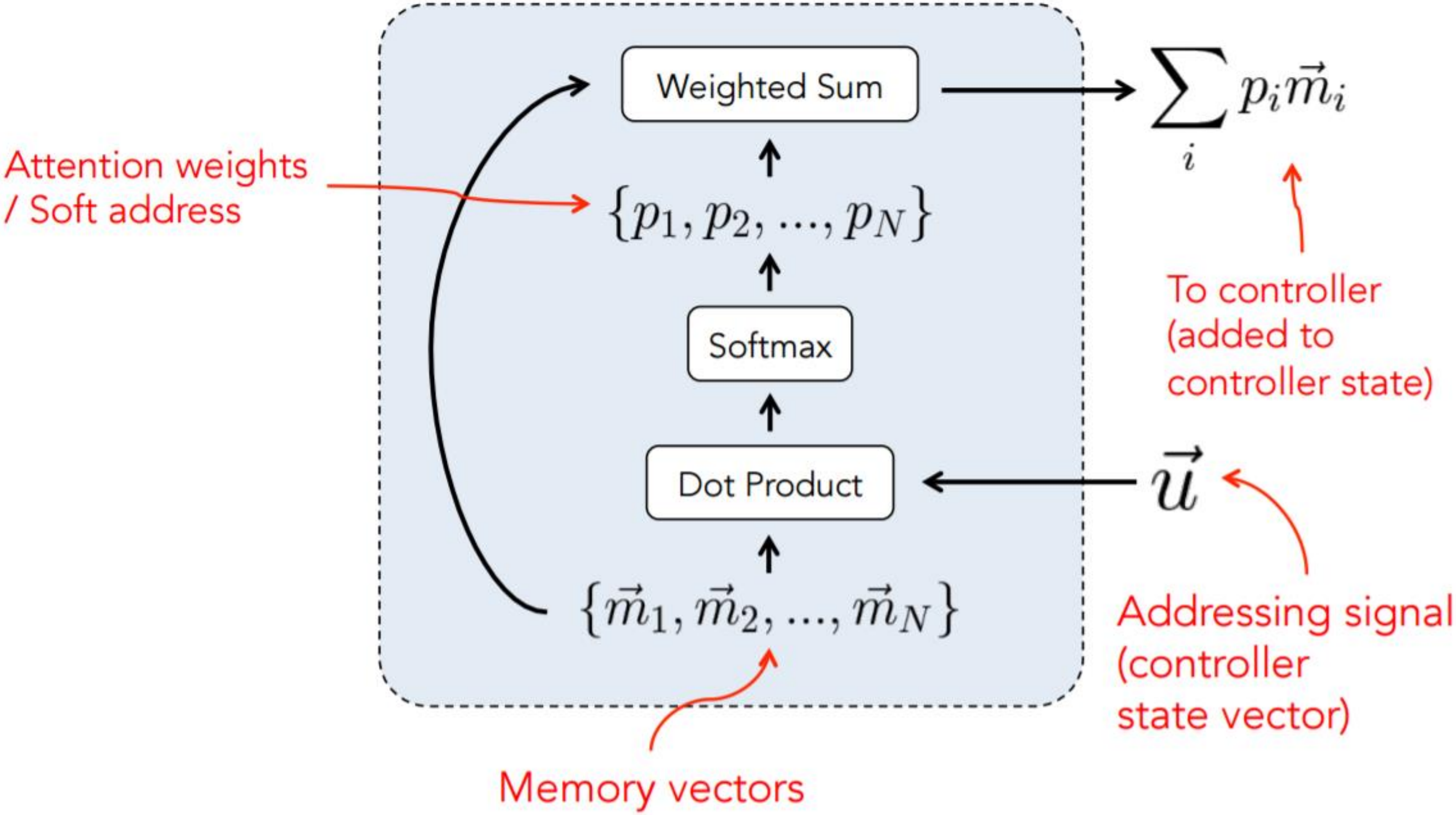


Overview



supervision
Output
$\vec{u}_3$
read
addressing
Memory
Module
$\vec{u}_2$
Controller
module
read
addressing
$\{\vec{m}_1, \vec{m}_2, ..., \vec{m}_N\}$
$\vec{u}_1$
Memory vectors
(unordered)
Input
Internal state
vector

Memory Module



Memory Vectors

E.g.) constructing memory vectors with Bag-of-Words (BoW)

1. Embed each word
2. Sum embedding vectors

$$\text{“Sam drops apple”} \rightarrow \underbrace{\vec{v}_{\text{Sam}} + \vec{v}_{\text{drops}} + \vec{v}_{\text{apple}}}_{\text{Embedding Vectors}} = \vec{m}_i$$

Memory Vector

E.g.) **temporal structure:** special words for time and include them in BoW

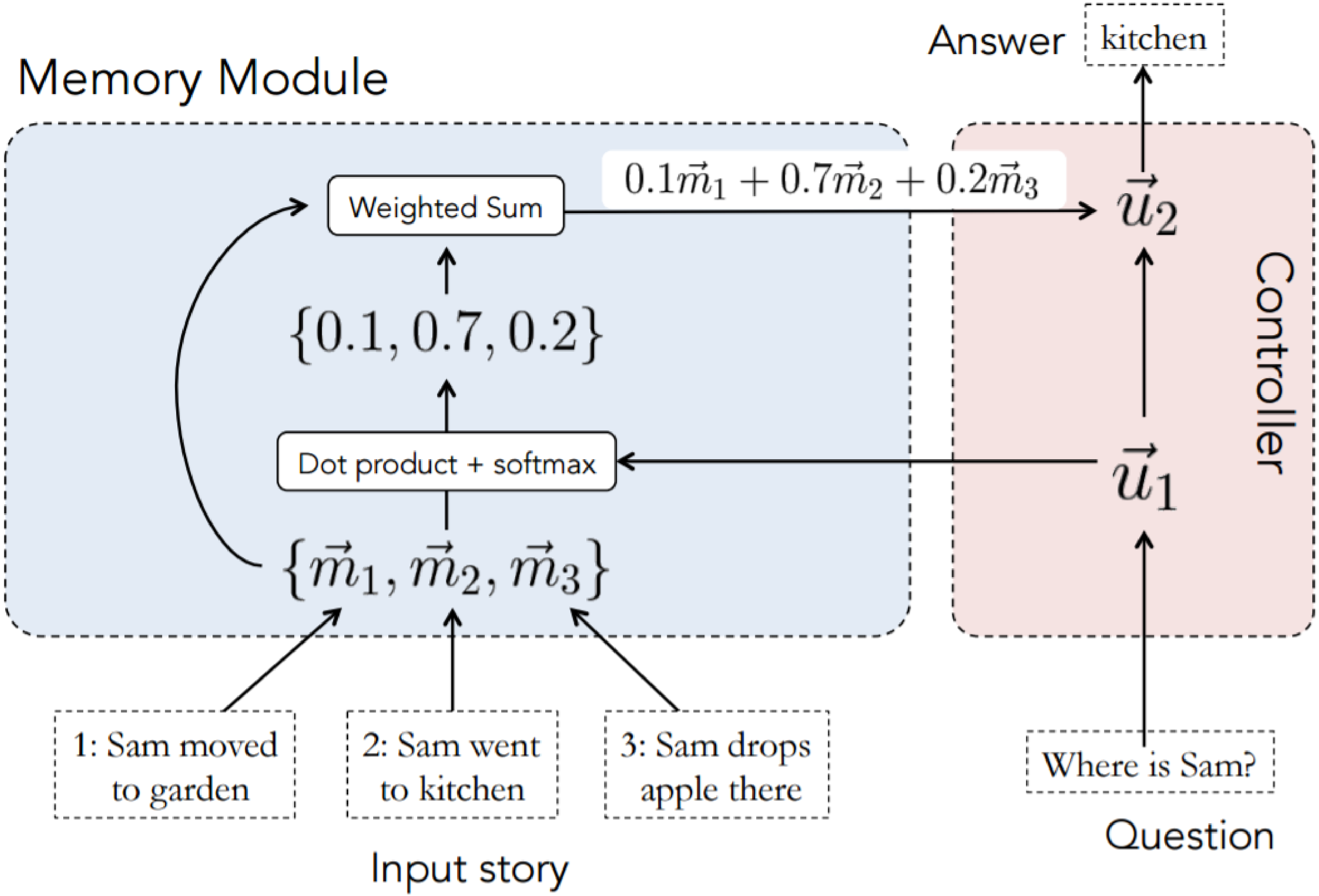
1: “Sam moved to garden”

2: “Sam went to kitchen”

3: “Sam drops apple” $\rightarrow v_{\text{Sam}} + v_{\text{drops}} + v_{\text{apple}} + v_3 = m_3$

Time embedding

Q&A Example



Answer
kitchen
Memory Module
Weighted Sum
$0.1\vec{m}_1 + 0.7\vec{m}_2 + 0.2\vec{m}_3$
$\vec{u}_2$
$\{0.1, 0.7, 0.2\}$
Controller
Dot product + softmax
$\vec{u}_1$
$\{\vec{m}_1, \vec{m}_2, \vec{m}_3\}$
1: Sam moved to garden
2: Sam went to kitchen
3: Sam drops apple there
Where is Sam?
Question
Input story

Generative Adversarial Nets (GAN)

[Goodfellow et al.]

Generative Models

- Most work on deep generative models focused on provided a parametric specification of a probability distribution function (like Deep Belief Net, PixelCNN, PixelRNN)
- Train these models by maximizing the log likelihood

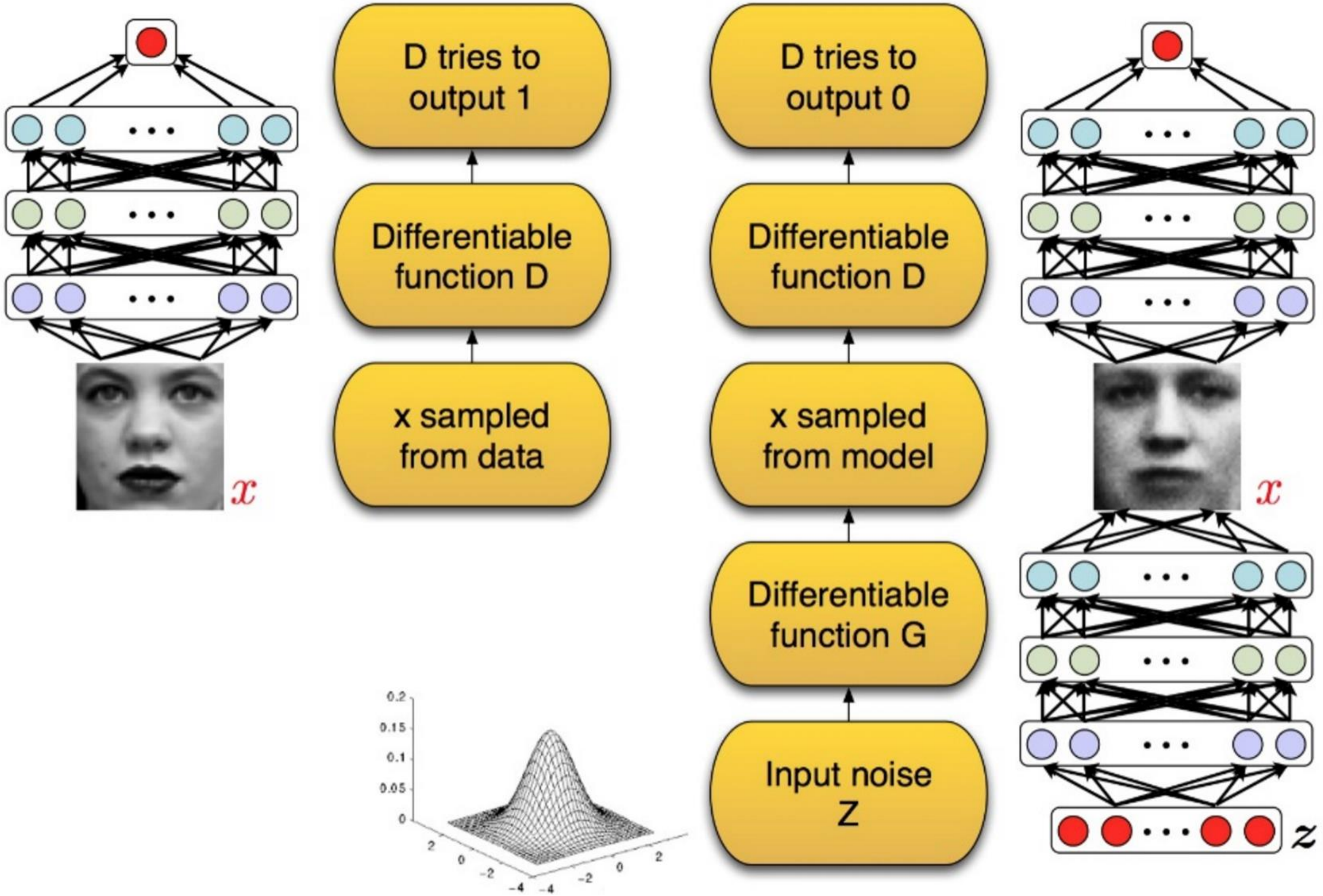
$$\text{Max } \sum_i \log P(x^{(i)}, y^{(i)})$$

- Difficulty: Intractable probabilistic computations

Generative Adversarial Nets

- Two neural networks: a generative model and a discriminative model
  - A two-player minimax game
  - One network for generation (e.g., generating images), one for classification (distinguishing the true data from the generated data)
  - Hence, if the generative model produces the same distribution as the true data distribution, the discriminative model wouldn’t be able to distinguish them. This is an equilibrium point!
    - But in practice, we can’t achieve this point. The discriminative model is a bit too strong for the current generative model.

The discriminative model D tries to distinguish whether x is from the original data distribution or from the generated distribution.



Generative Adversarial Nets

$p_{\boldsymbol{z}}(\boldsymbol{z})$: Prior noise (e.g., Gaussian) for the generative model

$D(\boldsymbol{x}; \theta_d)$: the discriminative model outputs a single scalar, which is the Prob[x is from the data (rather than from the generative model)]

Generative model wants to minimize $\log[1 - D(G(z))]$

Discriminative model wants to assign correct labels (from g or from data)

The value of the minmax game:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

D wants it large
G has no control on this

D wants it small
G wants it large

- The goal is to reach an equilibrium.

Training

A variant of best-response to reach an equilibrium

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by <u>ascending</u> its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

D: maximization

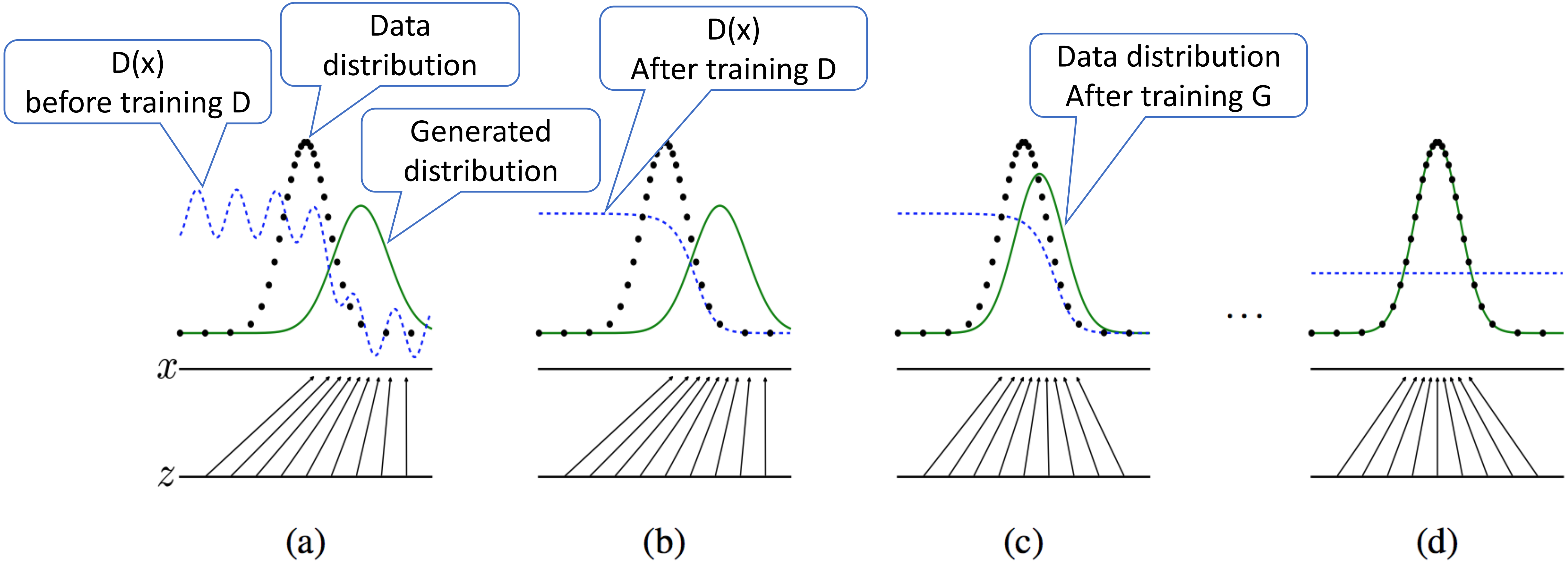
  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by <u>descending</u> its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

G: minimization

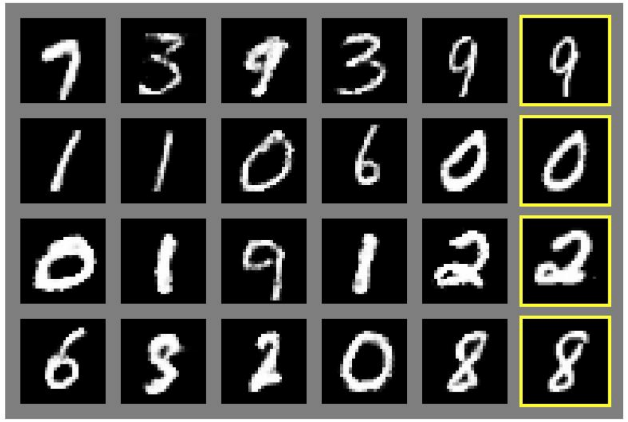
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training Process

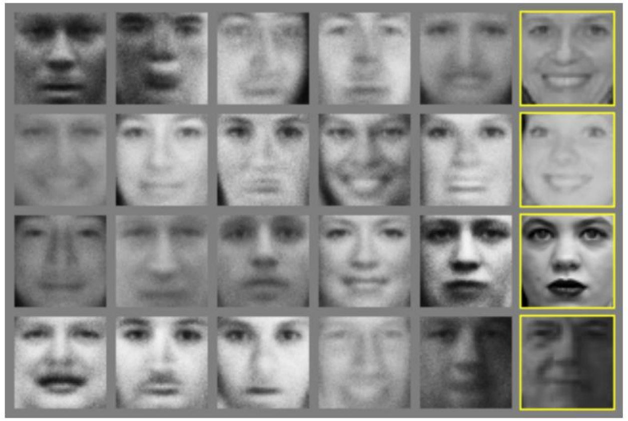


Another visualized training process, see http://cs.stanford.edu/people/karpathy/gan/

Experiments



a)



b)

c)

d)

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [5]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Final Notes

- We have covered the basics, and several recent "end products". There are many important ideas developed by many researchers that lead to those cool stuffs you see today
- A fast growing area (2000+ppl in NIPS 2013, now 8000+ ppl this year NIPS)
- In many cases, the design is more of an art than a science
  - But it doesn't mean that DL is just "tuning parameters"
- Important Things We didn't cover
  - Things related to graphical models, Bayesian Approaches
  - Deep Belief Net (Restricted Boltzmann Machine)
  - Autoencoder (Variational Autoencoder, Stacked Autoencoder)
  - Stacking traditional "shallow" models ……
  - Lots of applications in NLP (word2vec, topic models)
  - Unsupervised learning
  - Transfer learning
  - Theoretical results
  - …..

- Deep Reinforcement Learning
  - Play games
  - [Playing Atari with Deep Reinforcement Learning]
  - https://github.com/kuz/**Deep**Mind-**Atari**-**Deep**-Q-**Learn**er

  - Search technique (Monte-Carlo Tree Search) – AlphaGo
    - Open Source facebook Go engine:
    - https://github.com/facebookresearch/darkforestGo

- Some slides borrowed from cs231n at Stanford, slides for “End-To-End Memory Networks” by Sukhbaatar et al. and from wiki
- Variational inference, Blei.
- Thank Jianbo Guo for preparing some slides of GAN