Attention and Memory in Deep Learning

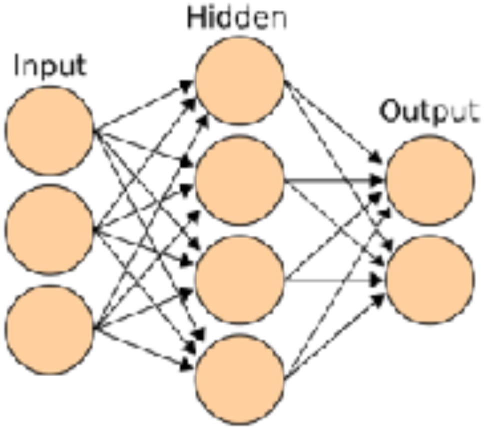
Alex Graves

Attention, memory and Cognition

The ability to focus on one thing and ignore others has a vital role in guiding cognition.

Not only does this allow us to pick out salient information from noisy data (the cocktail party problem) it also allows us to pursue one thought at a time, remember one event rather than all events...

Neural Networks



Neural nets are parametric, nonlinear function approximations that can be fit to data to learn functions from input vectors (e.g. photographs) to output vectors (e.g. distributions over class labels)

What does that have to do with **attention**?

Implicit Attention in Neural Networks

Deep nets naturally learn a form of **implicit attention** where they respond more strongly to some parts of the data than others

To a first approximation, we can visualise this by looking at the network **Jacobian** — sensitivity of the network outputs with respect to the inputs

Neural Network Jacobian

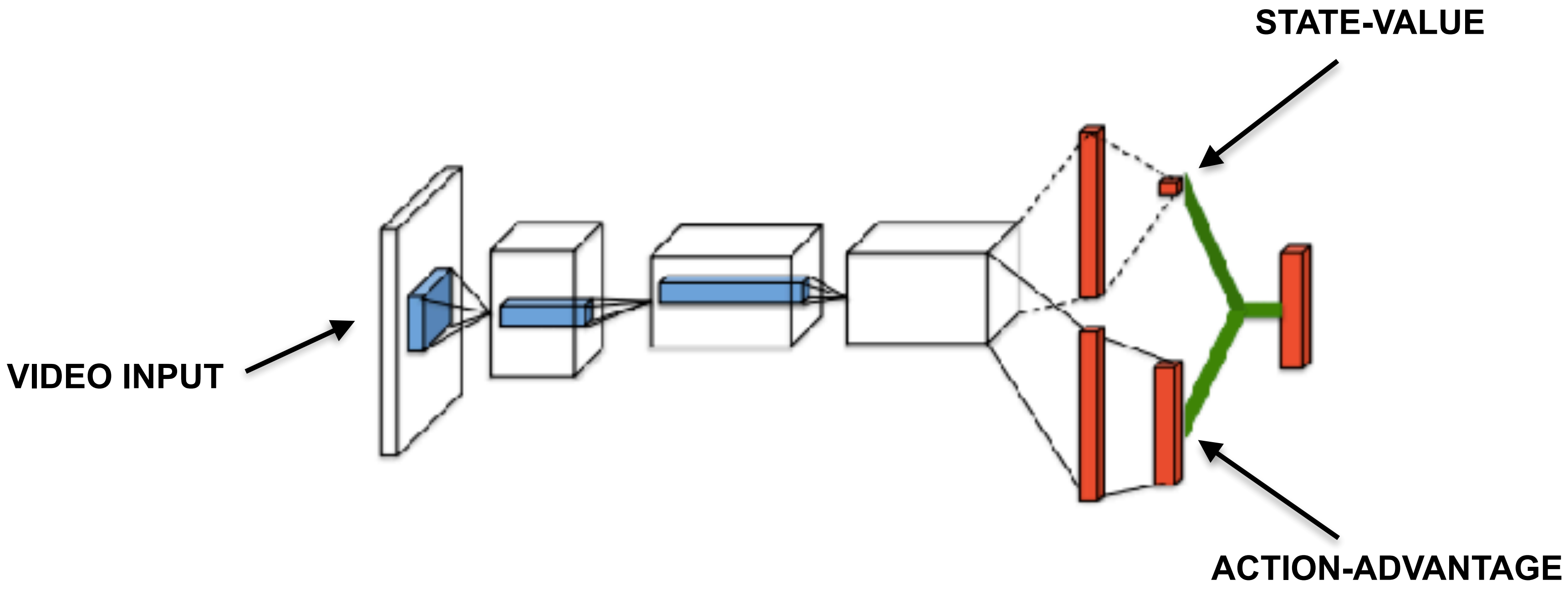
x = size k input vector
y = size m output vector
Jacobian J = m x k matrix

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_k} \end{bmatrix}$$

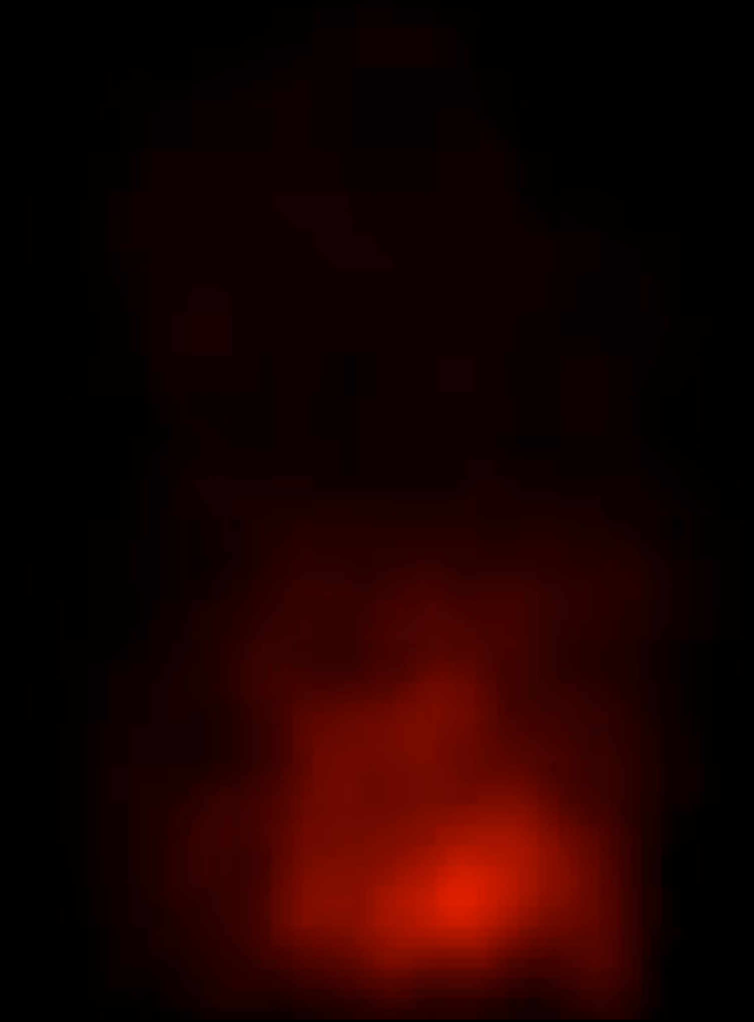
$$J_{ij} = \frac{\partial y_i}{\partial x_j}$$

Can compute with ordinary backdrop
(just set output ‘errors’ = output activations)

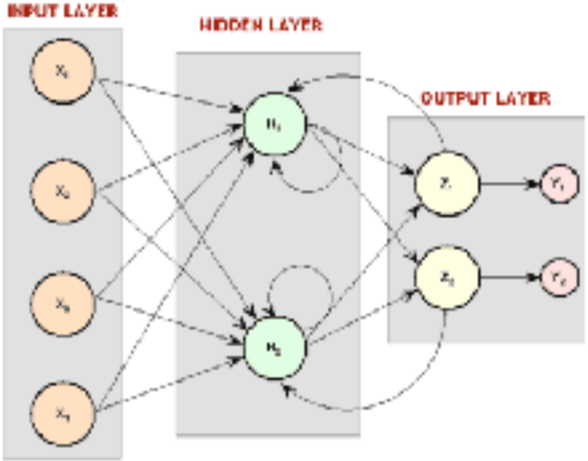
Jacobian in Action: Duelling Network



Dueling Network Architectures for Deep Reinforcement Learning, Wang et. al. (2015)



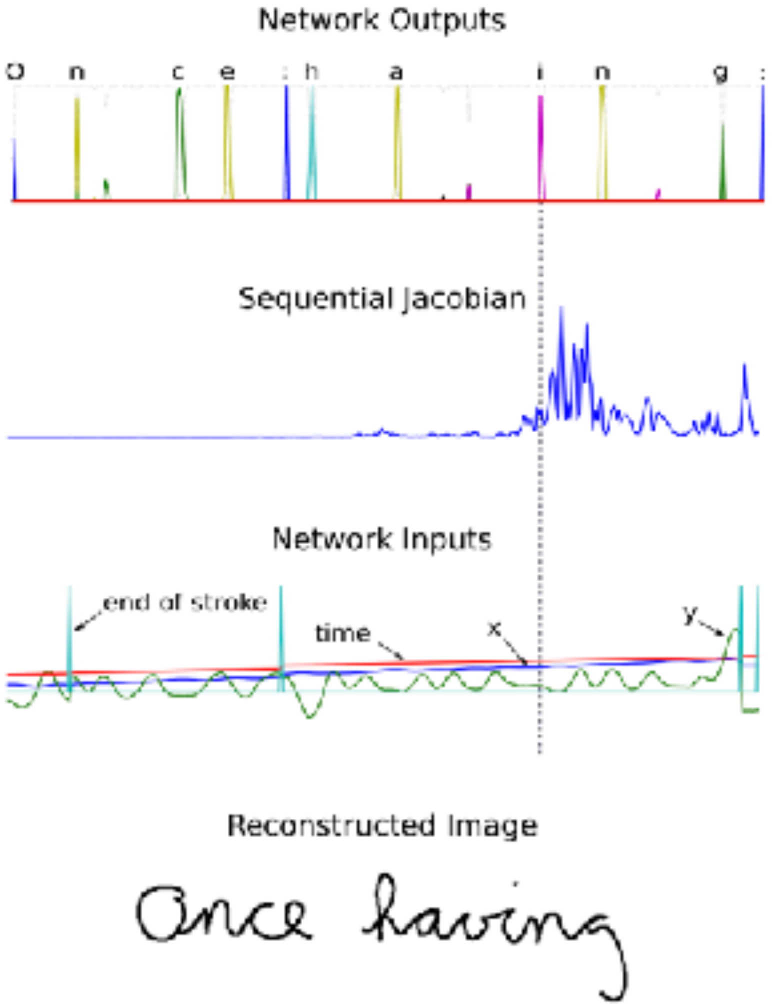
Attention and memory in Recurrent Networks (RNNs)



“Hello World”

RNNs contain a recursive hidden state and learn functions from sequences of inputs (e.g. a speech signal) to sequences of outputs (e.g. words)

The **sequential Jacobian** shows which past inputs they **remember** when predicting current outputs.



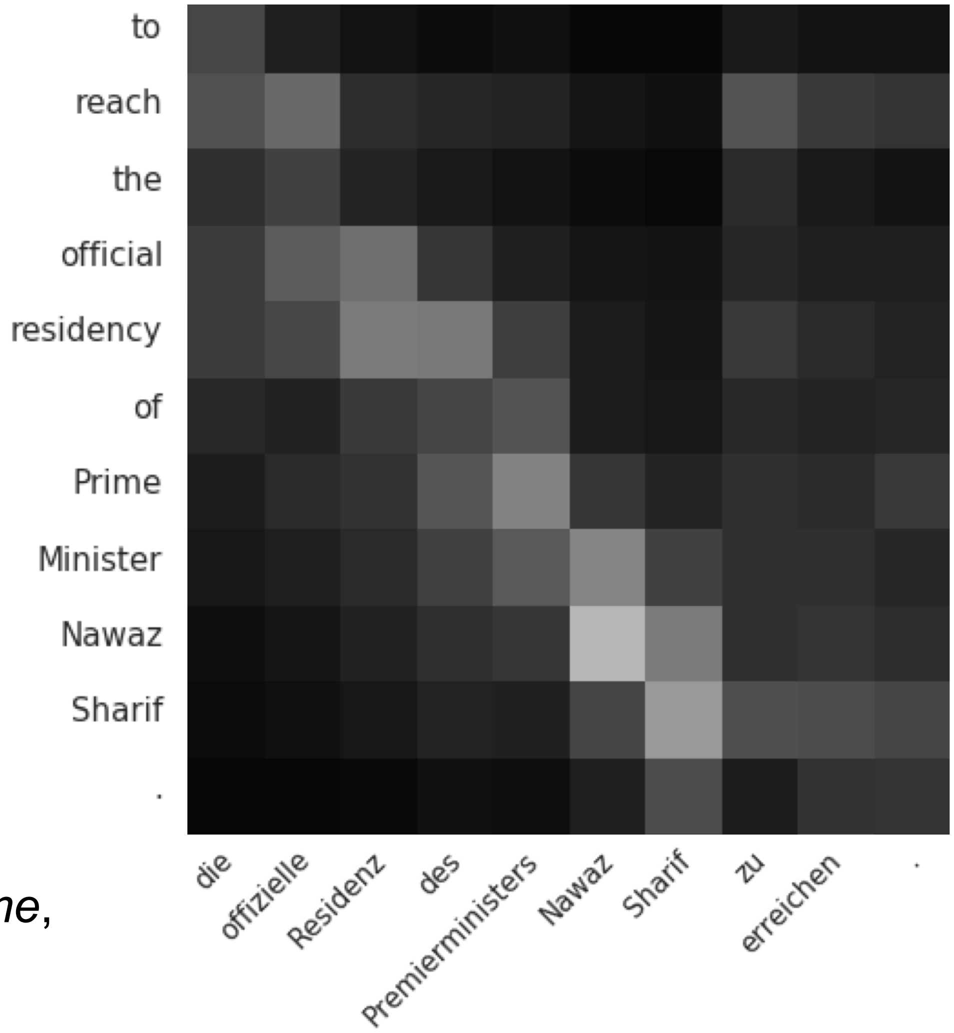
- The Sequential Jacobian is the set of derivatives of one network output with respect to all the inputs

$$J_k^t = \left(\frac{\partial y_k^t}{\partial \mathbf{x}^1}, \frac{\partial y_k^t}{\partial \mathbf{x}^2} \dots \right)$$

- It shows how the network responds to widely separated, but related, inputs, such as the delayed dot of the 'i' in 'having'

Implicit Attention allows reordering in machine translation:

“to reach” -> “zu erreichen”



Neural Machine Translation in Linear Time, Kalchbrenner et. al. (2016)

Explicit Attention

Implicit attention is great, but there are still advantages to an **explicit attention** mechanism that limits the data presented to the network in some way:

- Computational **efficiency**
- **Scalability** (e.g. fixed sized glimpse for any size image)
- **Sequential processing** of static data (e.g. moving gaze)
- **Easier to interpret**

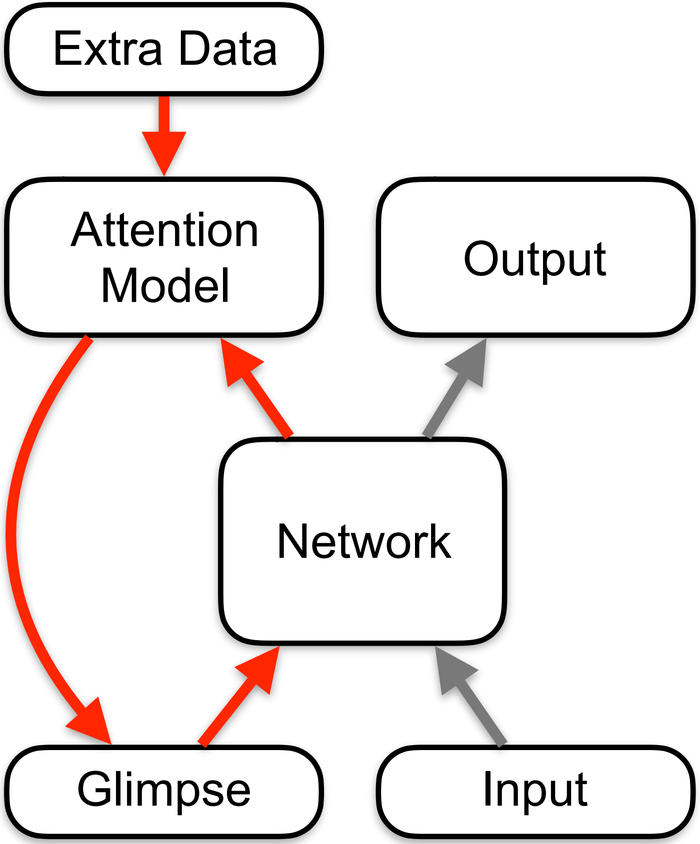
Neural Attention Models

The **network** receives **input** and produces **output** as usual.

It also produces an extra set of outputs used to parameterise an **attention model**

The attention model then operates on some **extra data** (image, audio sample, text to be translated…) to create a fixed-size “**glimpse**” that is passed to the network as an extra input at the next time step

The complete system is **recurrent**, even if the network isn’t



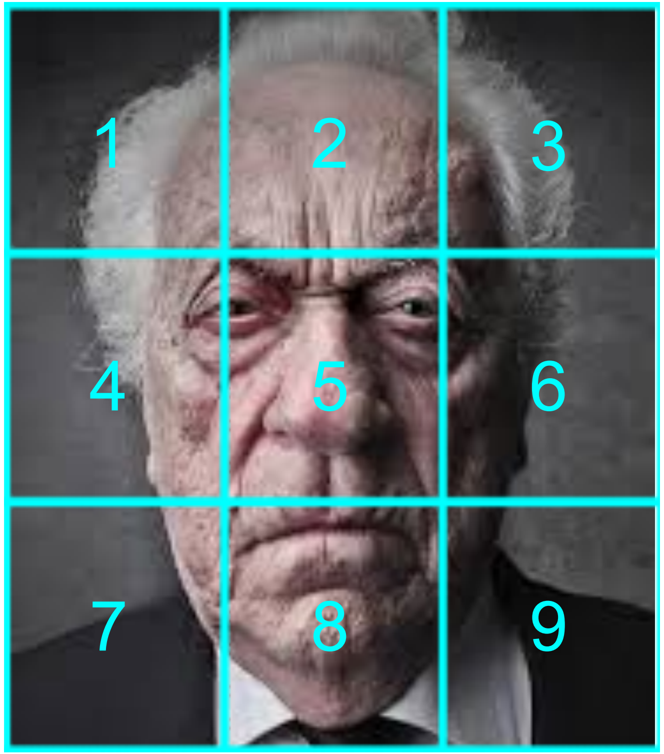
Glimpse Distribution

Attention models generally work by defining a probability distribution over glimpses **g** of the data **x** and some set of attention outputs **a** from the network:

$$\Pr(\mathbf{g}|\mathbf{a})$$

simplest case: **a** just assigns probabilities to a set of discrete glimpses:

$$\Pr(\mathbf{g}_k|\mathbf{a}) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$



Attention with RL

We can treat the distribution over glimpses **g** as a **stochastic policy** $\pi_{\mathbf{a}}$, sample from it, and use **RL** techniques (with reward R = task loss L induced by the glimpse) to train the attention model

$$\pi_{\mathbf{a}} = \Pr(\mathbf{g}_k | \mathbf{a})$$

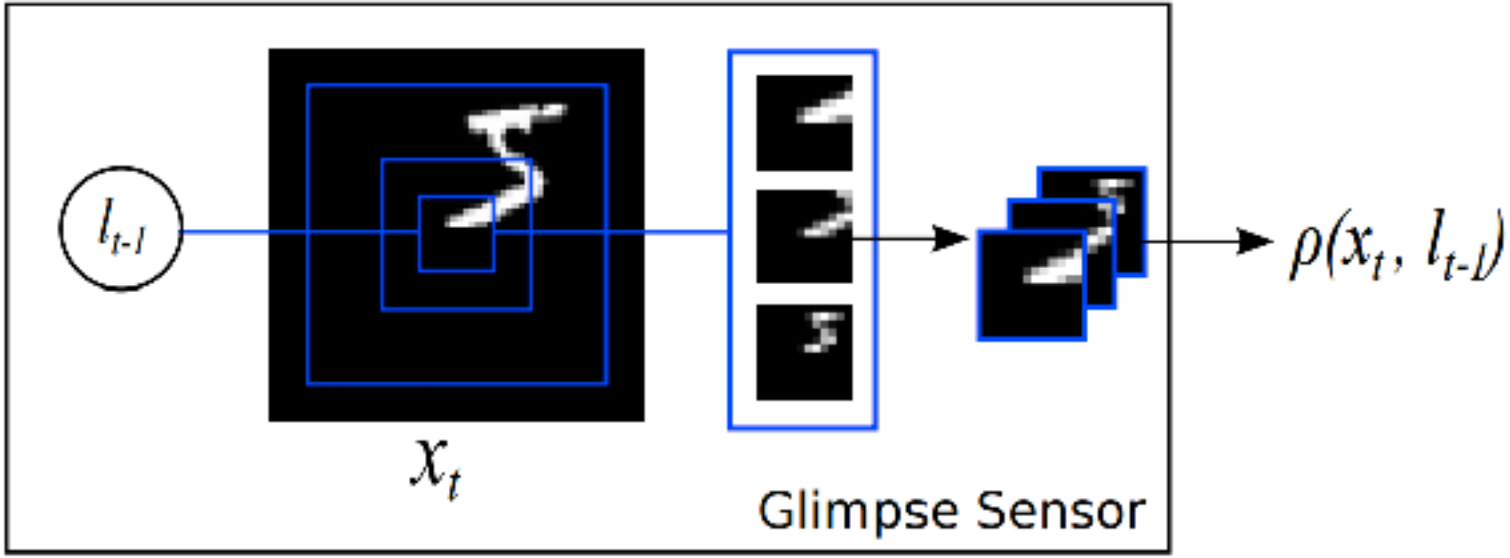
$$R = \mathbb{E}_{\mathbf{g} \sim \pi_{\mathbf{a}}} [\log \pi_{\mathbf{a}} L(\mathbf{g})]$$

$$\nabla_{\mathbf{a}} R = \mathbb{E}_{\mathbf{g} \sim \pi_{\mathbf{a}}} [\nabla_{\mathbf{a}} \log \pi_{\mathbf{a}} L(\mathbf{g})]$$

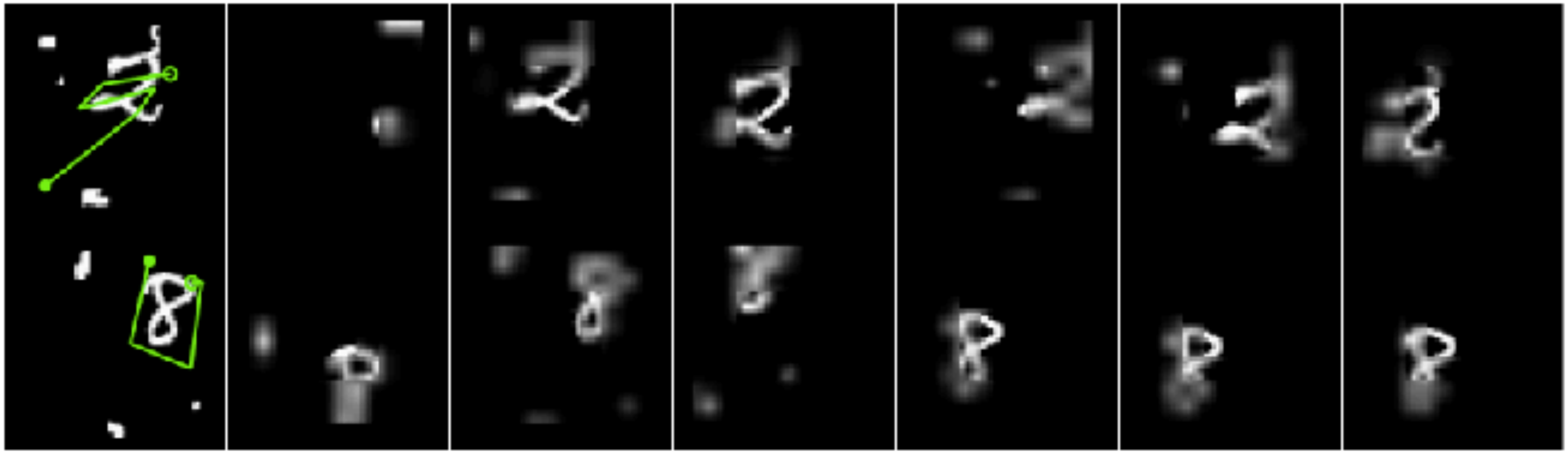
In general we can use RL methods for supervised tasks any time some module in the network is **non-differentiable**

Complex Glimpses

Generally the glimpse distribution is more complex than just a softmax (e.g. Gaussian over co-ordinates, width, height…) and the glimpses are more complex than image tiles (e.g. foveal models)

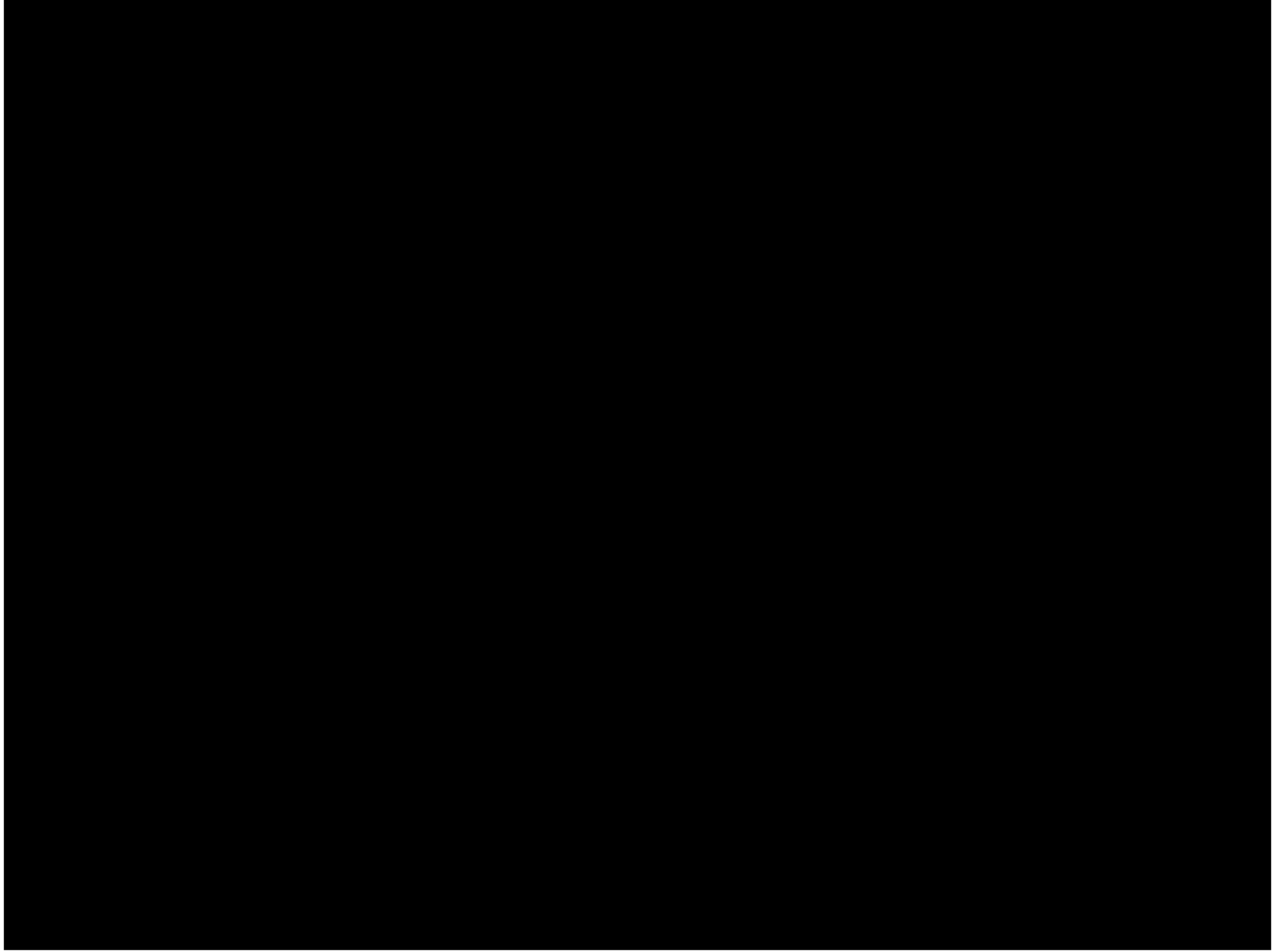


6 point “Glimpse path” (in green) while trying to classify image



6 Foveal Glimpses seen by the network

Recurrent Models of Visual Attention, Mnih et. al. (2014)



Multiple Object Recognition with Visual Attention, Ba et. al. (2014)

Soft Attention

The last examples used **hard attention**: fixed size attention windows moved around the image, trained with RL techniques.

Robots have to look left or right, but in many cases attention doesn't need to be hard: we just want to focus more on certain regions and less on others.

If we do this in a differentiable way, we get **soft attention** which we can train **end-to-end** with **backprop**

Generally easier than using RL, but more expensive to compute

Soft Attention

Basic template: we use the attention parameters **a** to determine a distribution Pr(**g**|**a**) as before, only now we take an **expectation** over all possible glimpses instead of a **sample**

$$\mathbf{g} = \sum_{\mathbf{g}' \in \mathbf{x}} \mathbf{g}' \Pr(\mathbf{g}'|\mathbf{a})$$

This is differentiable w.r.t. **a** as long as Pr(**g**|**a**) is:

$$\nabla_{\mathbf{a}} \mathbf{g} = \sum_{\mathbf{g}' \in \mathbf{x}} \mathbf{g}' \nabla_{\mathbf{a}} \Pr(\mathbf{g}'|\mathbf{a})$$

Location Attention

Window vector (input to net)

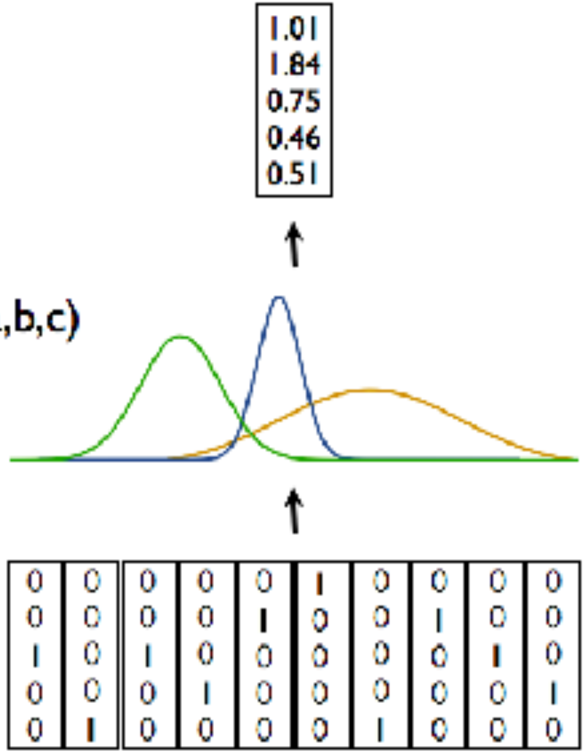
$$v^{t+1} = \sum_{i=1}^{S} w_i^t s_i$$

Window weights (net outputs for a,b,c)

$$w_i^t = \sum_{k=1}^{K} a_k^t \exp\left(-b_k^t [c_k^t - i]^2\right)$$

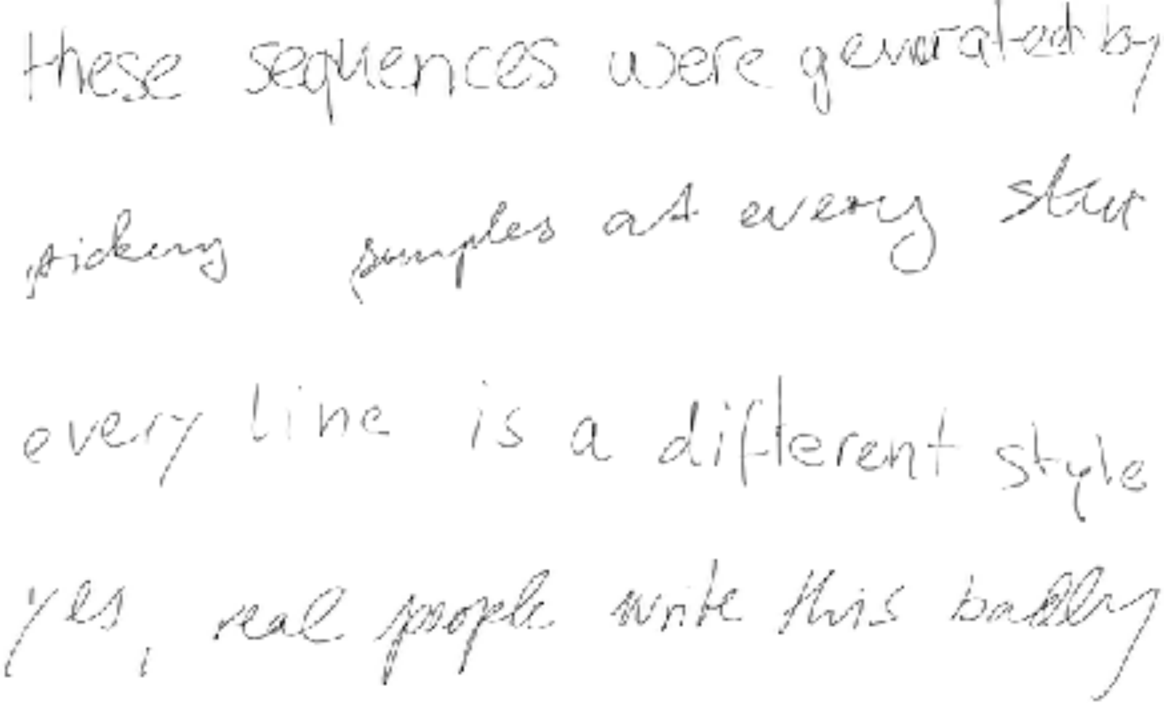
Input vectors (one-hot)

$$(s_1, \ldots, s_S)$$

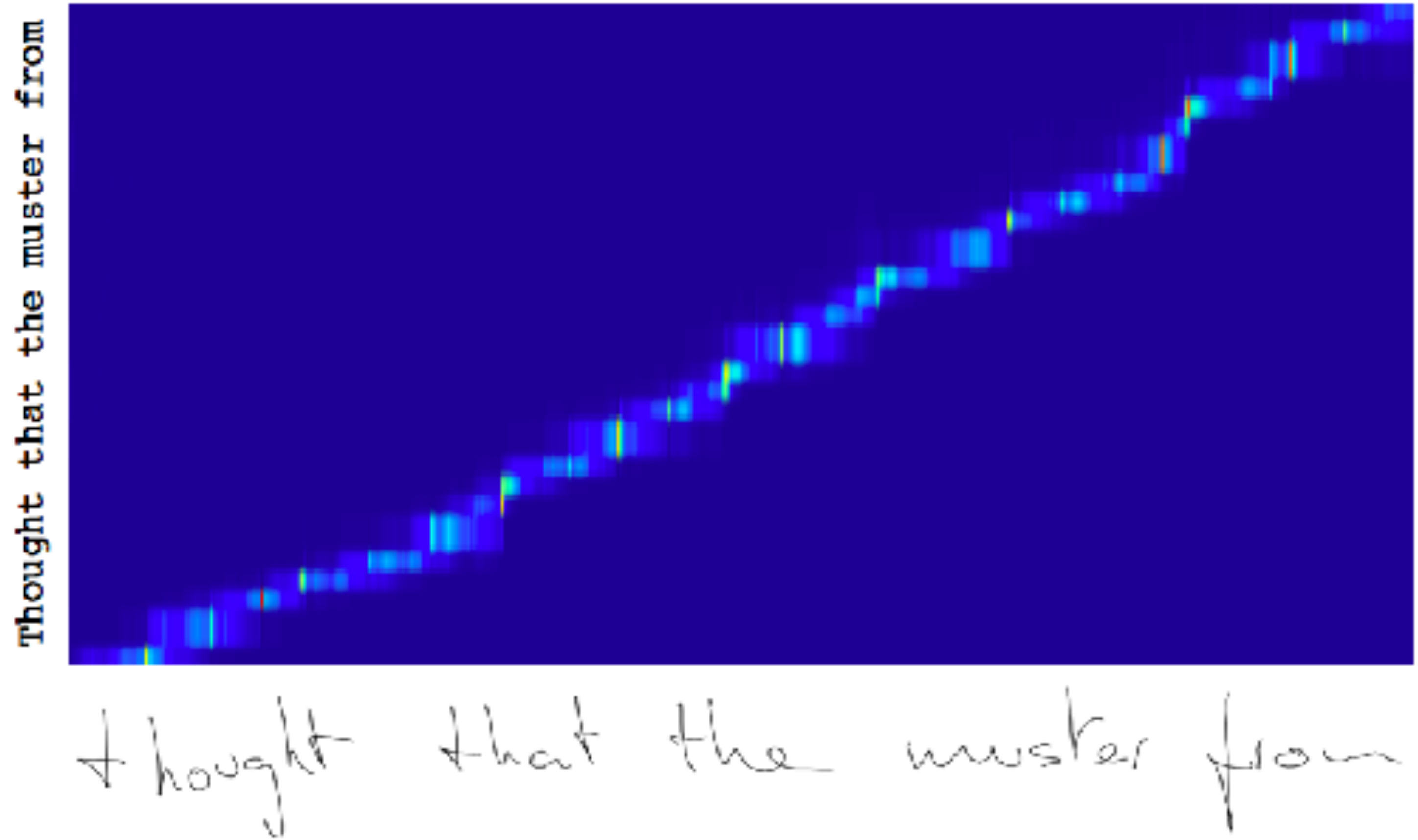


Generating Sequences with Recurrent Neural Networks, Graves (2013)

Handwriting Synthesis With Soft Reading



Alignment



Associative Attention

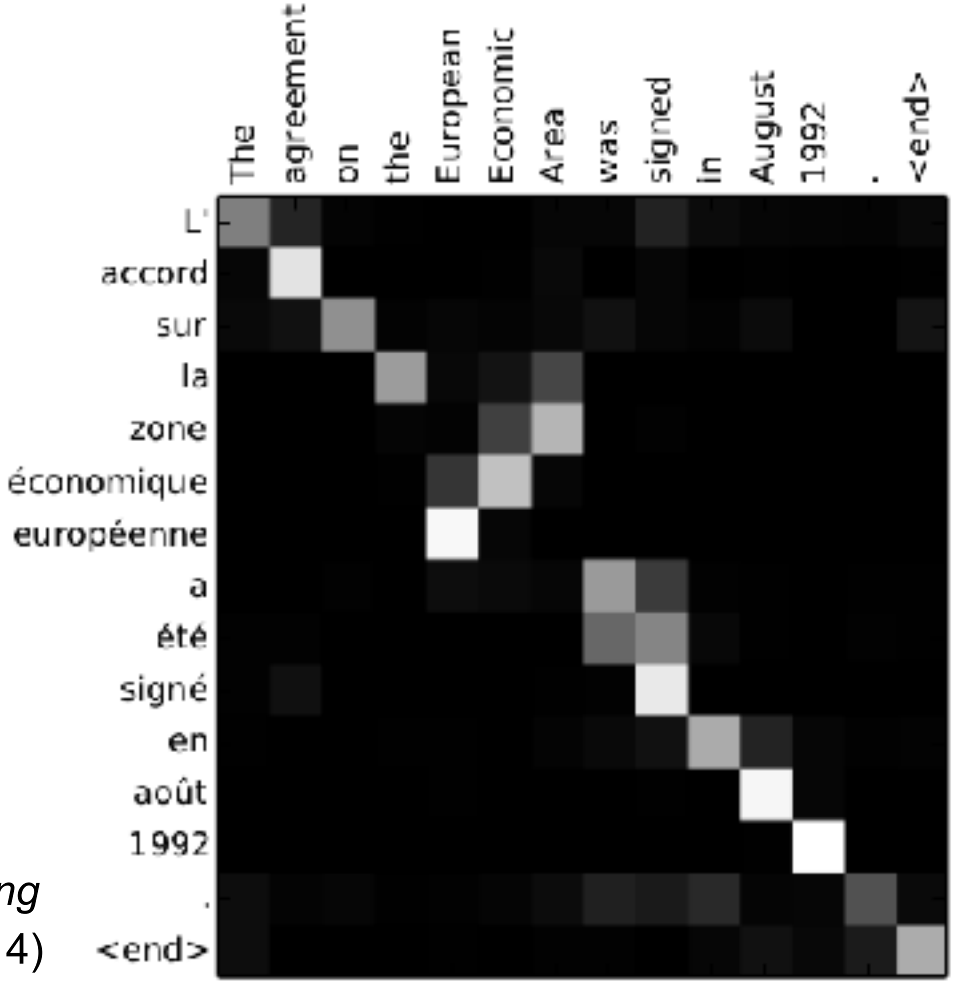
Instead of attending by position, we can attend by **content**. In this setting **a** = a **key vector** that is compared to *all* glimpses **g** using some **similarity function** *S*. The scores are normalised and used to define Pr(**g**|**a**)

$$\Pr(\mathbf{g}|\mathbf{a}) = \frac{\exp(S(\mathbf{g}, \mathbf{a}))}{\sum_{\mathbf{g}' \in \mathbf{x}} \exp(S(\mathbf{g}', \mathbf{a}))}$$

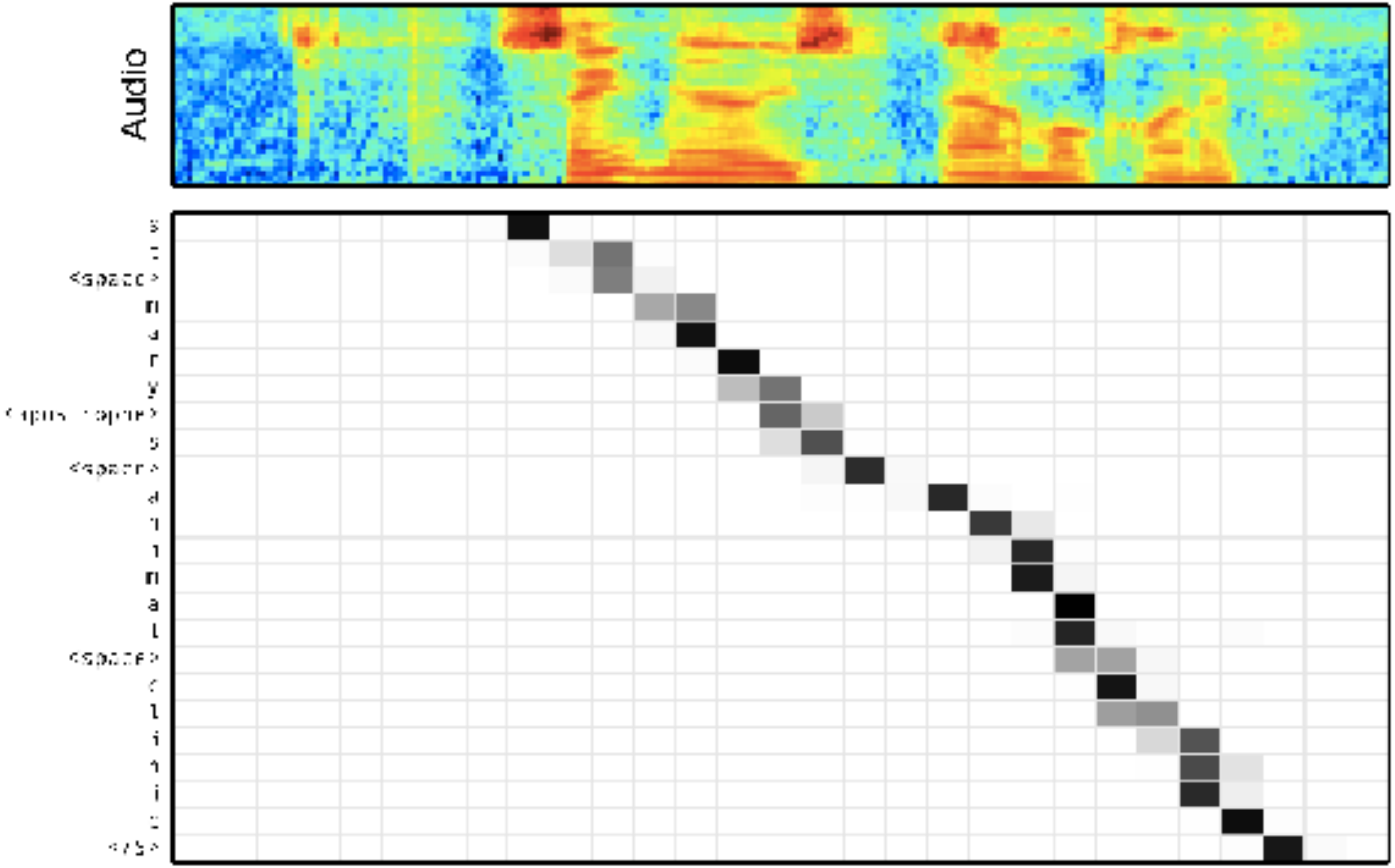
S can be learned (MLP, linear operator...) or fixed (dot product, cosine similarity…). Yields a **Multidimensional, feature-based** lookup: natural way to search data

Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et. al. (2014)

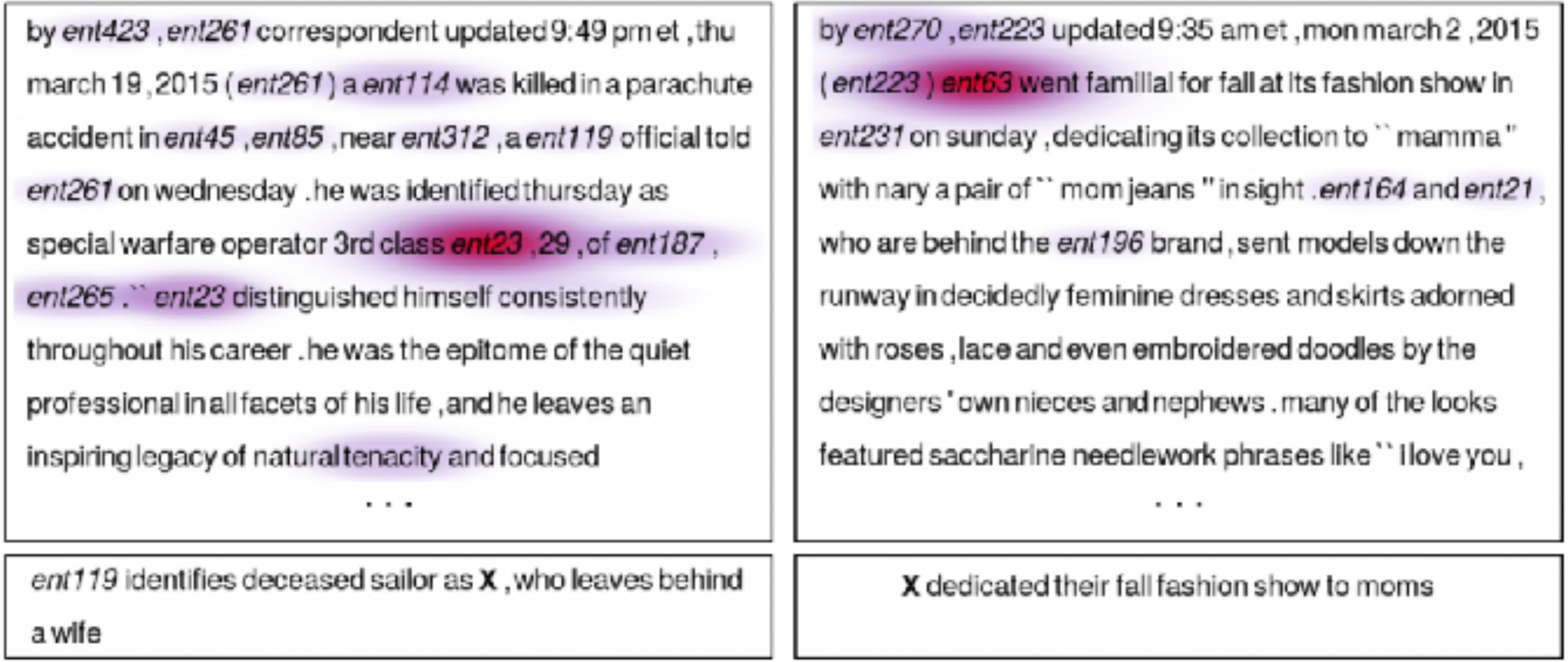
Reordering in machine translation using associative attention



Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et. al. (2014)



Listen, Attend and Spell, Chan et. al. (2015)



Teaching Machines to Read and Comprehend, Hermann et. al. (2015)

Introspective Attention

So far we have looked at attention to **external data**

Also useful to selectively attend to the network's internal state or memory: **introspective attention**

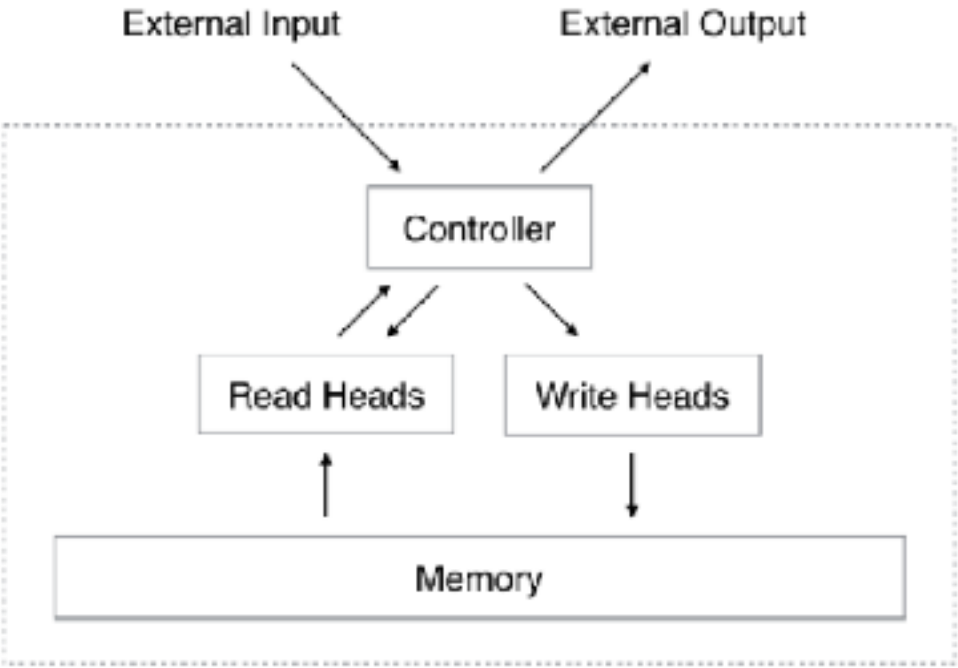
With internal information we can do selective ***writing*** as well as ***reading***, allowing the network to **iteratively modify** its state

Neural Turing Machines

The **Controller** is a **neural network** (recurrent or feedforward)

The **Heads** **select** portions of the memory and **read** or **write** to them

The **Memory** is a real-valued **matrix**



Neural Turing Machines, Graves et. al. (2014)

Addressing by Content

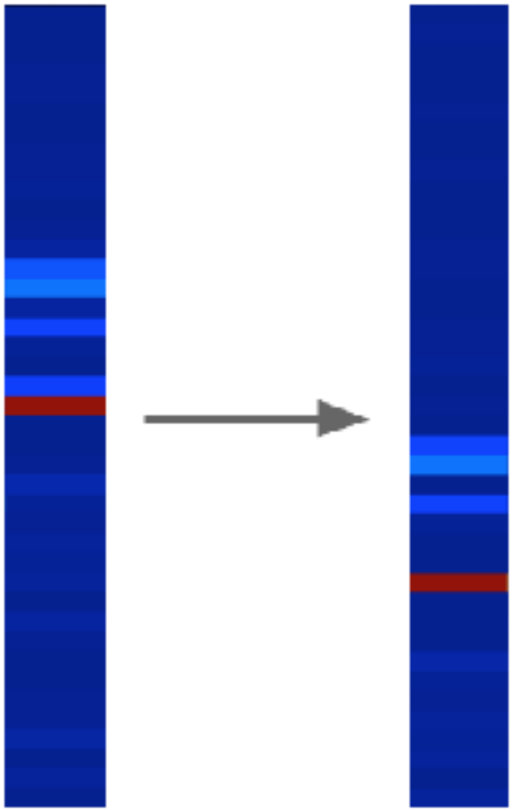
A key vector ***k*** is emitted by the controller and compared to the content of each memory location **M**[i] using a similarity measure $S(\cdot,\cdot)$ (e.g. **cosine distance**) then normalised with a **softmax**. A 'sharpness' β is used to narrow the focus. Finds the memories **'closest'** to the key

$$\mathbf{w}[i] = \frac{\exp\left(\beta S(\mathbf{k}, \mathbf{M}[i])\right)}{\sum_j \exp\left(\beta S(\mathbf{k}, \mathbf{M}[j])\right)}$$

Addressing by Location

The controller outputs a shift kernel **s** (e.g. a softmax on [-n,n]) which is convolved with a weighting **w** to produce a shifted weighting **ŵ**.

$$\hat{\mathbf{w}}[i] = \sum_j \mathbf{w}[j]\mathbf{s}(i-j)$$



Data Structure and Accessors

The combination of addressing mechanisms allows the controller to interact with the memory in several distinct modes, corresponding to different data structures and accessors.

Content key only — memory is accessed like an associative map

Content and location — key finds an array, shift indexes into it

Location only — shift iterates from the last focus

Reading and Writing

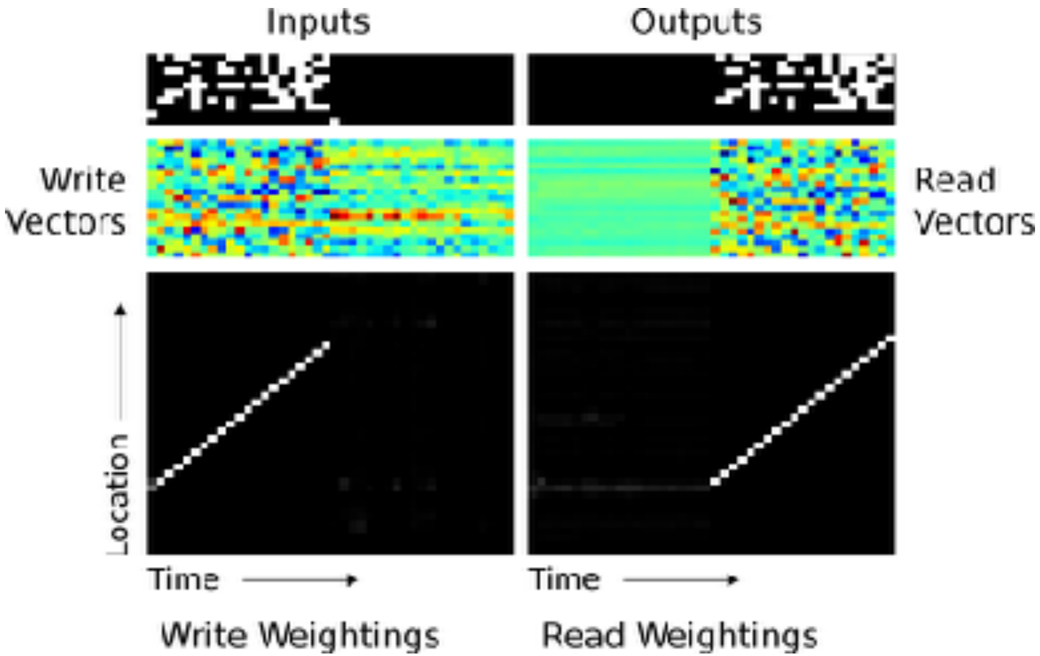
Once the weightings are defined, each read head returns a read vector **r** as input to the controller at the next timestep

$$\mathbf{r} = \sum_i \mathbf{w}[i]\mathbf{M}[i]$$

Each write head receives an erase vector **e** and an add vector **a** from the controller and resets then writes to modify the memory (like **LSTM**)

$$\mathbf{M}[i] \leftarrow \mathbf{M}[i](\mathbf{1} - \mathbf{w}[i]\mathbf{e}) + \mathbf{w}[i]\mathbf{a}$$

The NTM Copy Algorithm

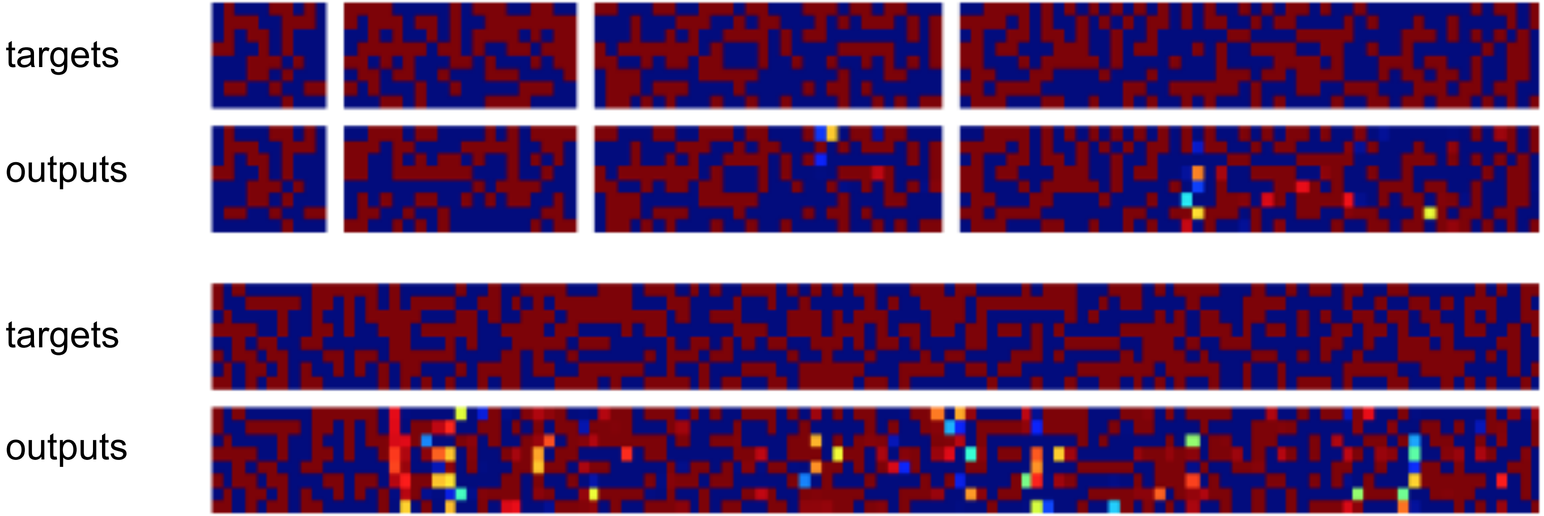


NTM++

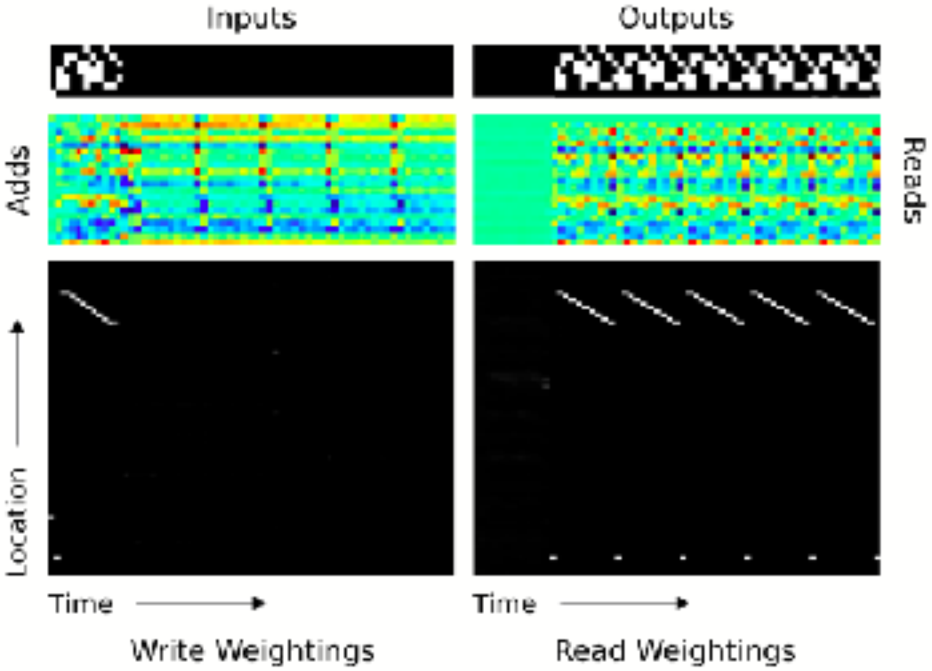
initialize: move head to start location
while input delimiter not seen **do**
    receive input vector
    write input to head location
    increment head location by 1
end while
return head to start location
while true **do**
    read output vector from head location
    emit output
    increment head location by 1
end while

pseudocode

Copy Generalisation: length 10 to 120

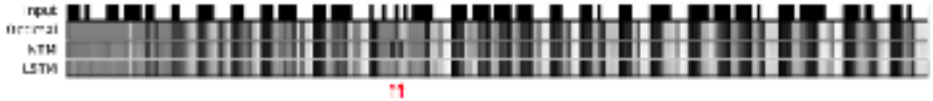


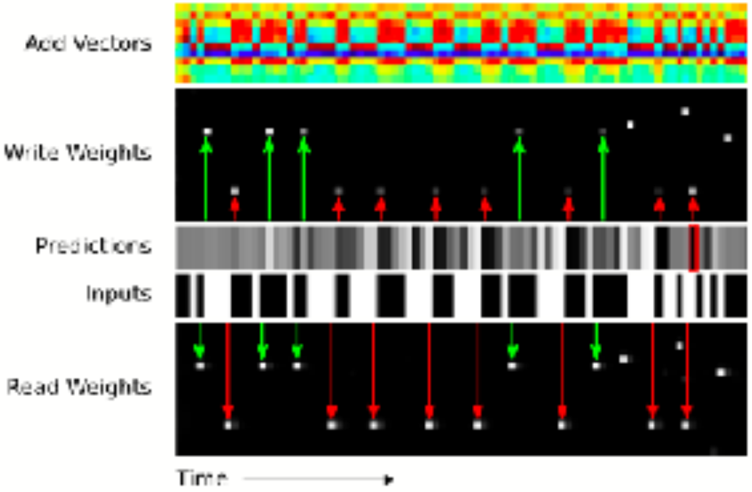
Copy *N* Times



NTM learns its first **for-loop**, using **content** to jump, **iteration** to step, and a **variable** to count to *N*.

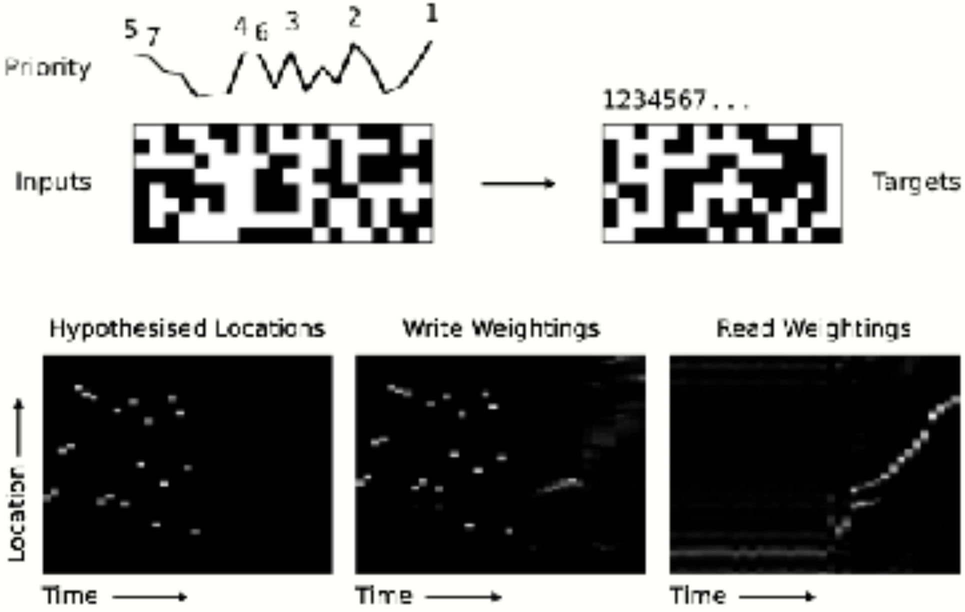
N-Gram Inference





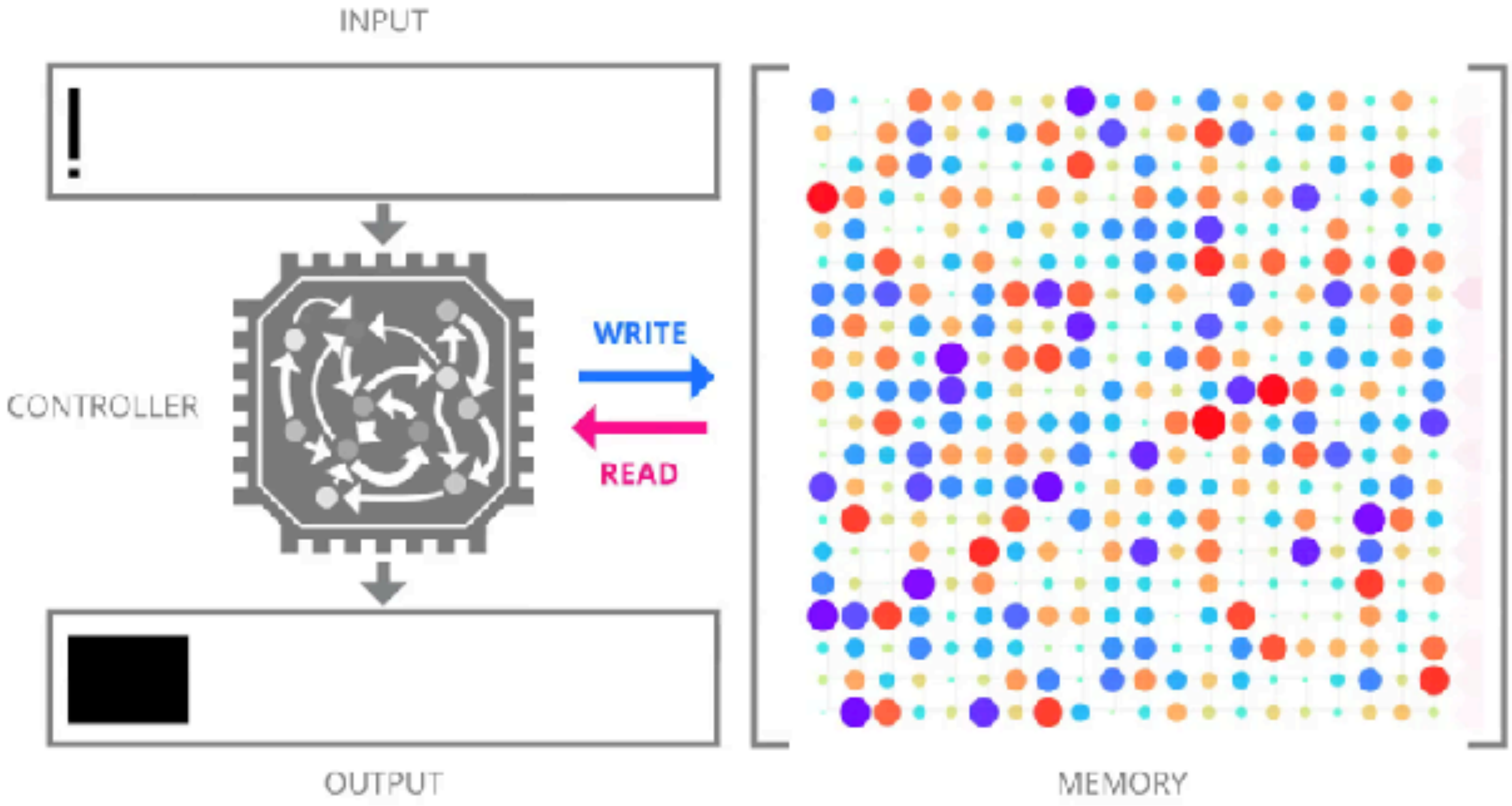
Specific memory locations store **variables** that **count** the **occurrences** of particular N-Grams

Priority Sort



The network maps from **priorities** to **write locations**, then **iterates** through the memory to return the sorted list

TRAINING

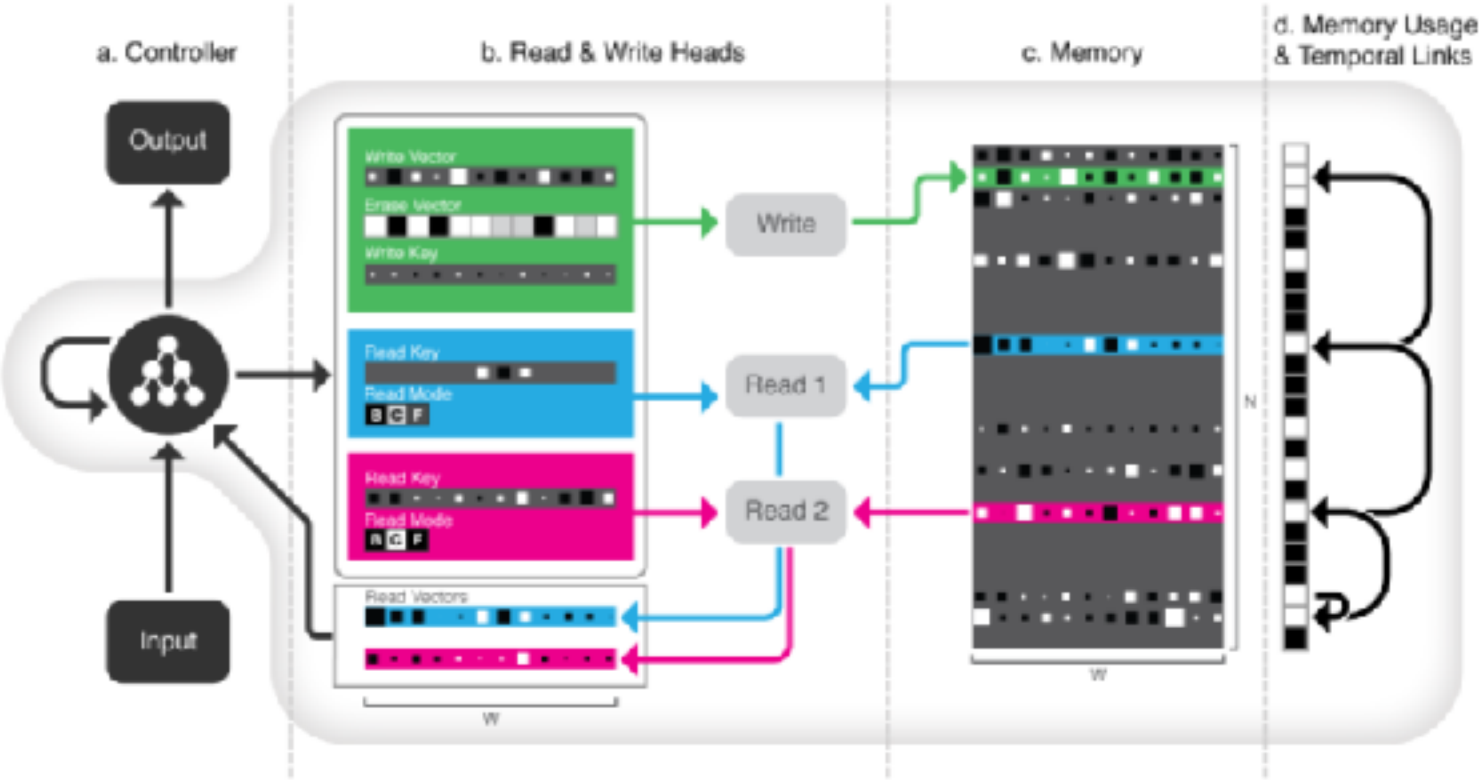


Differentiable Neural Computers

DNC is a successor architecture to Neural Turing Machines with new attention mechanisms for memory access

hybrid computing using a neural network with dynamic external memory, Graves et. al. (2016)

Overall Architecture



Allocating Memory

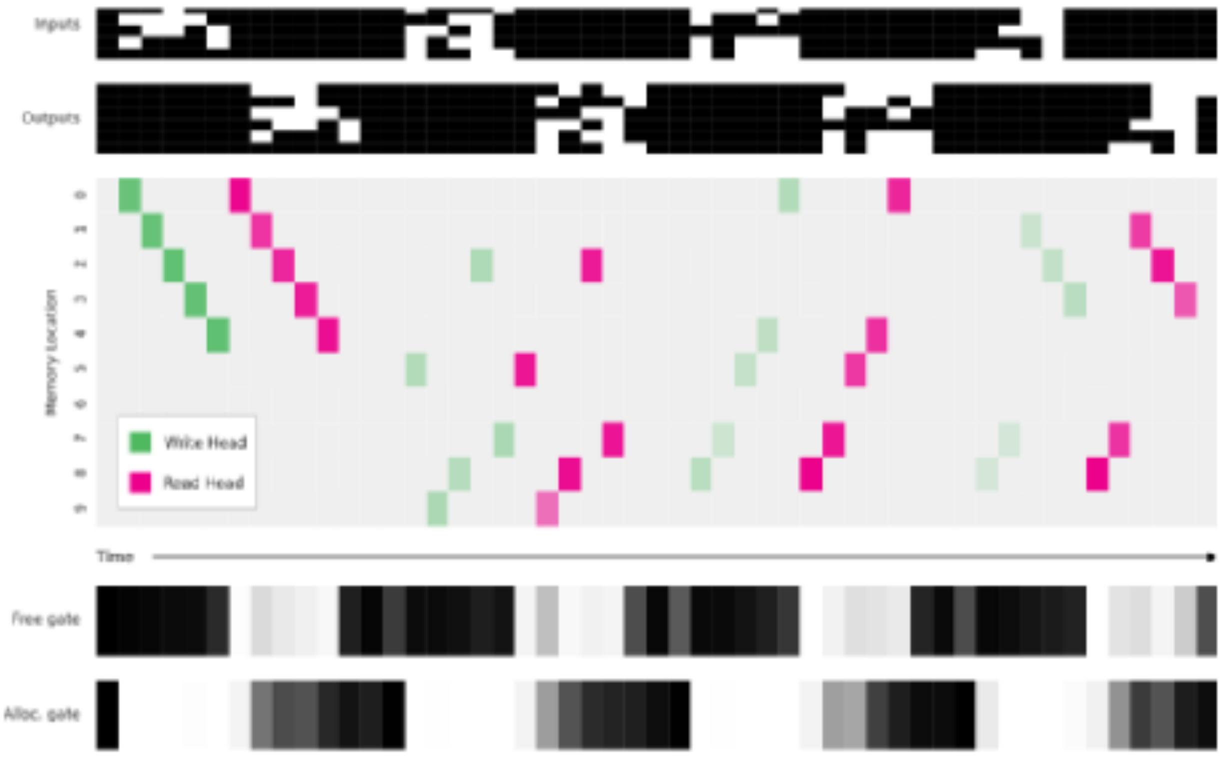
- NTM could only 'allocate' memory in contiguous blocks, leading to memory management problems
- DNC defines a differentiable *free list* tracking the *usage* ($\boldsymbol{u}_t$) of each memory location
- Usage is automatically *increased* after each write ($\boldsymbol{w}^{w}_{t}$) and optionally *decreased* after each read ($\boldsymbol{w}^{r,i}_{t}$) by *free gates* (f^{i}_{t})

$$\boldsymbol{u}_t = (\boldsymbol{u}_{t-1} + \mathbf{w}^{\mathrm{w}}_{t-1} - \boldsymbol{u}_{t-1} \circ \mathbf{w}^{\mathrm{w}}_{t-1}) \circ \prod_{i=1}^{R} \left(\mathbf{1} - f^{i}_{t} \mathbf{w}^{\mathrm{r},i}_{t-1}\right)$$

- The controller then uses an *allocation gate* (g^{a}_{t}) to interpolate between writing to a newly allocated ($\boldsymbol{a}_t$) location, or an existing one found by content ($\boldsymbol{c}^{w}_{t}$)

$$\mathbf{w}^{w}_{t} = g^{w}_{t}\left(g^{a}_{t}\mathbf{a}_t + (1 - g^{a}_{t})\mathbf{c}^{w}_{t}\right)$$

Memory Allocation Test



Searching By Time

- NTM was able to retrieve memories in order of their *index* but not in order in which they were written
- Preserving temporal order is necessary for many tasks (e.g. sequence of instructions) and appears to play an important role in human cognition
- We wanted DNC to be able to iterate through memories in the order they were written (or rewritten)
- A *precedence weighting* ($\boldsymbol{p}_t$) keeps track of which locations were most recently written to:

$$\boldsymbol{p}_t = \left(1 - \sum_i \boldsymbol{w}_t^{\mathrm{w}}[i]\right)\boldsymbol{p}_{t-1} + \boldsymbol{w}_t^{\mathrm{w}}$$

Searching By Time

$\boldsymbol{p}_t$ is then used to update a *temporal link matrix* (L_t), defining the order locations were written in:

$$L_t[i,j] = (1 - \boldsymbol{w}_t^{\mathrm{w}}[i] - \boldsymbol{w}_t^{\mathrm{w}}[j])L_{t-1}[i,j] + \boldsymbol{w}_t^{\mathrm{w}}[i]\boldsymbol{p}_{t-1}[j]$$

The controller can use L_t to retrieve the write before (b^i_t) or after (f^i_t) the last read location ($w^{r,i}_{t-1}$), allowing it to iterate forwards or backwards in time

$$\boldsymbol{b}_t^i = \hat{L}_t^\top \boldsymbol{w}_{t-1}^{\mathrm{r},i} \qquad \boldsymbol{f}_t^i = \hat{L}_t \boldsymbol{w}_{t-1}^{\mathrm{r},i}$$

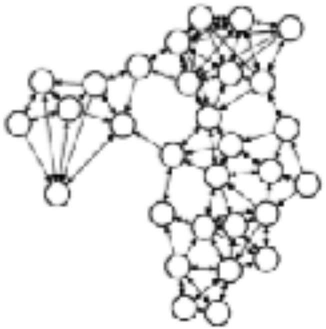
Finally three-way gates ($\boldsymbol{\pi}^i_t$) are used to interpolate among iterating forwards, backwards, or reading by content:

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

Graph Experiments

Training Data

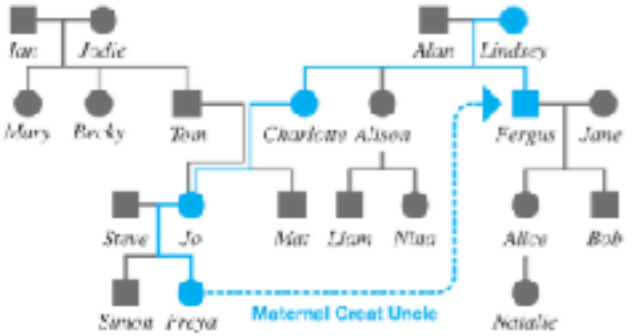
a. Random Graph



Test Examples

b. London Underground

c. Family Tree



Underground Input:

(OxfordCircus, TottenhamCtRd, Central)
(TottenhamCtRd, OxfordCircus, Central)
(BakerSt, Marylebone, Circle)
(BakerSt, Marylebone, Bakerloo)
(BakerSt, OxfordCircus, Bakerloo)

...

(LeicesterSq, CharingCross, Northern)
(TottenhamCtRd, LeicesterSq, Northern)
(OxfordCircus, PiccadillyCircus, Bakerloo)
(OxfordCircus, NottingHillGate, Central)
(OxfordCircus, Euston, Victoria)

- 84 edges in total

Traversal Question:

(OxfordCircus, _, Central), (_, _, Circle)
(_, _, Circle), (_, _, Circle),
(_, _, Bakerloo), (_, _, Victoria),
(_, _, Victoria), (_, _, Circle),
(_, _, Bakerloo), (_, _, Jubilee)

Answer:

(OxfordCircus, NottingHillGate, Central)
(NottingHillGate, Paddington, Circle)
...
(Embankment, Waterloo, Bakerloo)
(Waterloo, GreenPark, Jubilee)

Shortest Path Question:

(Moorgate, PiccadillyCircus, _)

Answer:

(Moorgate, Bank, Northern)
(Bank, Holborn, Central)
(Holborn, LeicesterSq, Piccadilly)
(LeicesterSq, PiccadillyCircus, Piccadilly

Family Tree Input:

(Charlotte, Alan, Father)
(Simon, Steve, Father)
(Steve, Simon, Son1)
(Melanie, Alison, Mother)
(Lindsey, Fergus, Son1)

...

(Bob, Jane, Mother)
(Natalia, Alice, Mother)
(Mary, Ian, Father)
(Jane, Alice, Daughter1)
(Mel, Charlotte, Mother)

- 54 edges in total

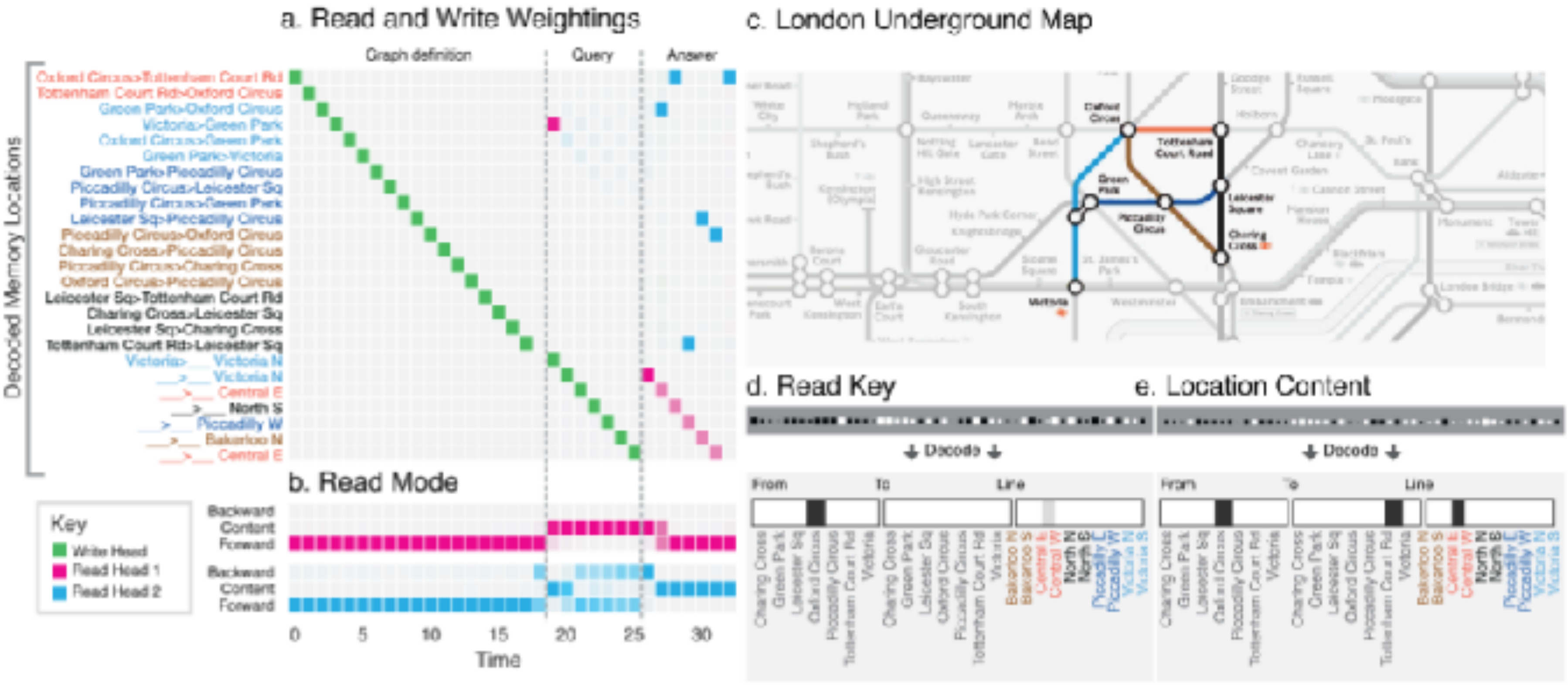
Inference Question:

[Freya, _, MaternalGreatUncle]

Answer:

(Freya, Fergus, MaternalGreatUncle)

How DNC Reads Graphs



Differentiable Visual Attention

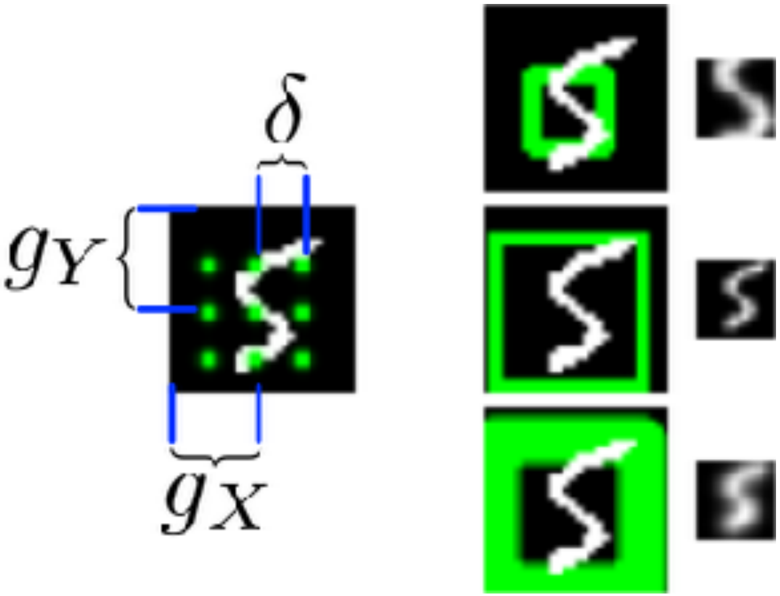
DRAW (Gregor et. al. 2015) uses a grid of Gaussian filters to **read** from input images and **draw** to a **canvas** image:

σ^2 — filter variance

g_X, g_Y — grid centre

δ — grid stride

γ — intensity

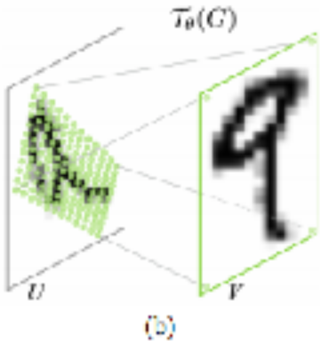


Reading MNIST

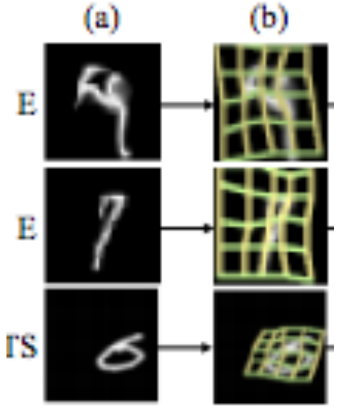
Spatial Transformers

The attention mechanism in **Spatial Transformer Networks** (Jaderberg et. al. 2015) extends DRAW by allowing more complex transformations of the **sampling grid**

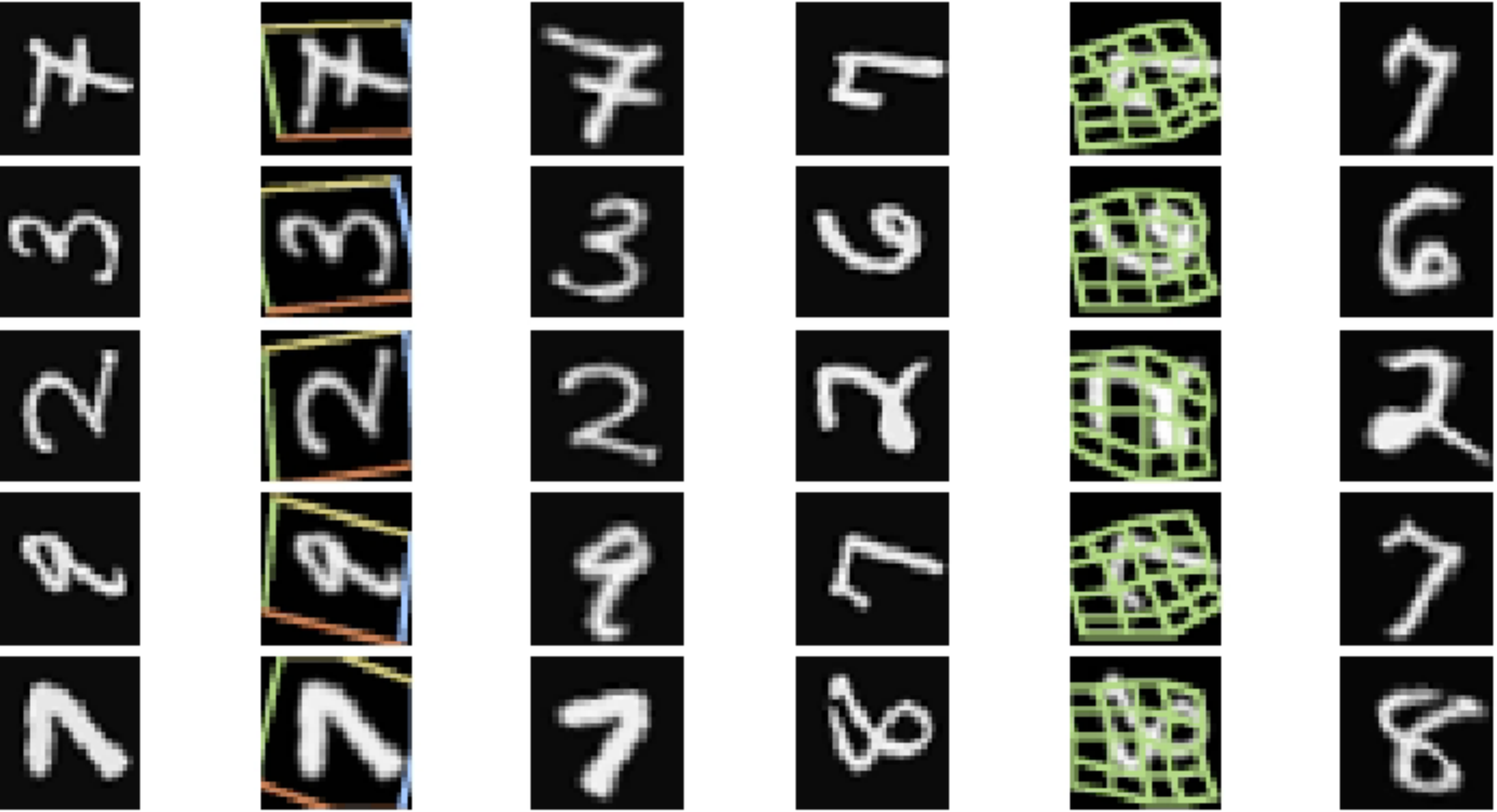
E.g. affine transformations

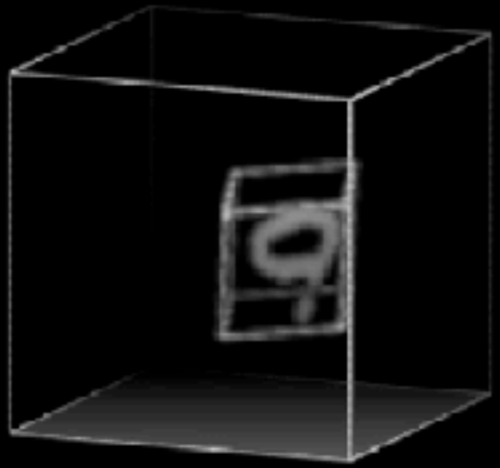


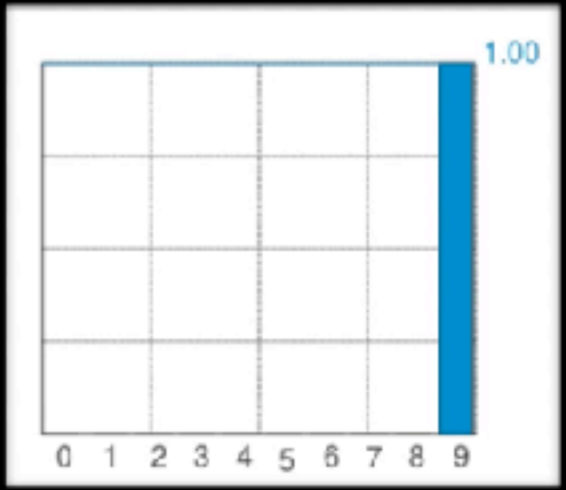
splines



Can also extend to 3d sampling grids






1.00
0
1
2
3
4
5
6
7
8
9

Summary

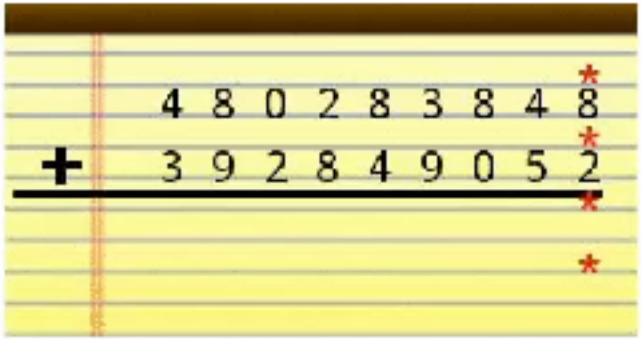
- Selective attention appears to be as useful for deep learning as it is for people
- Neural nets always have **implicit attention**, but we can also add **explicit attention** mechanisms
- These can be **stochastic** and trained with **reinforcement learning**
- Or **differentiable** and trained with ordinary **backprop**
- We can use attention to decide where to **write**, as well as where to **read**
- Many types of neural attention model (**content**, **location**, **visual**, **temporal**…) have been defined, and more are appearing all the time

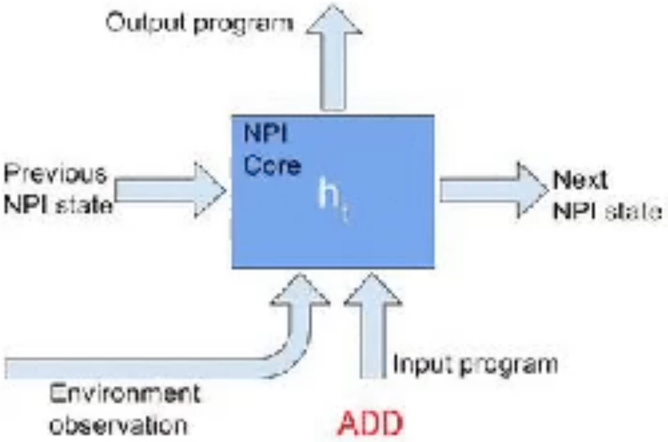
NPI inference

Generated commands

ADD

Addition scratch pad





Neural Programmer-Interpreters, Reed and de Freitas (2015)