# Generating Sequences with Recurrent Neural Networks

Alex Graves
CIFAR Global Scholar
University of Toronto

# Why Generate Sequences?

- To improve classification?
- To create synthetic training data?
- Practical tasks like speech synthesis?
- To simulate situations?
- <u>To understand the data</u>

# Generation and Prediction

- Obvious way to generate a sequence: repeatedly predict what will happen next

$$\Pr(\mathbf{x}) = \prod_t \Pr(x_t | x_{1:t-1})$$

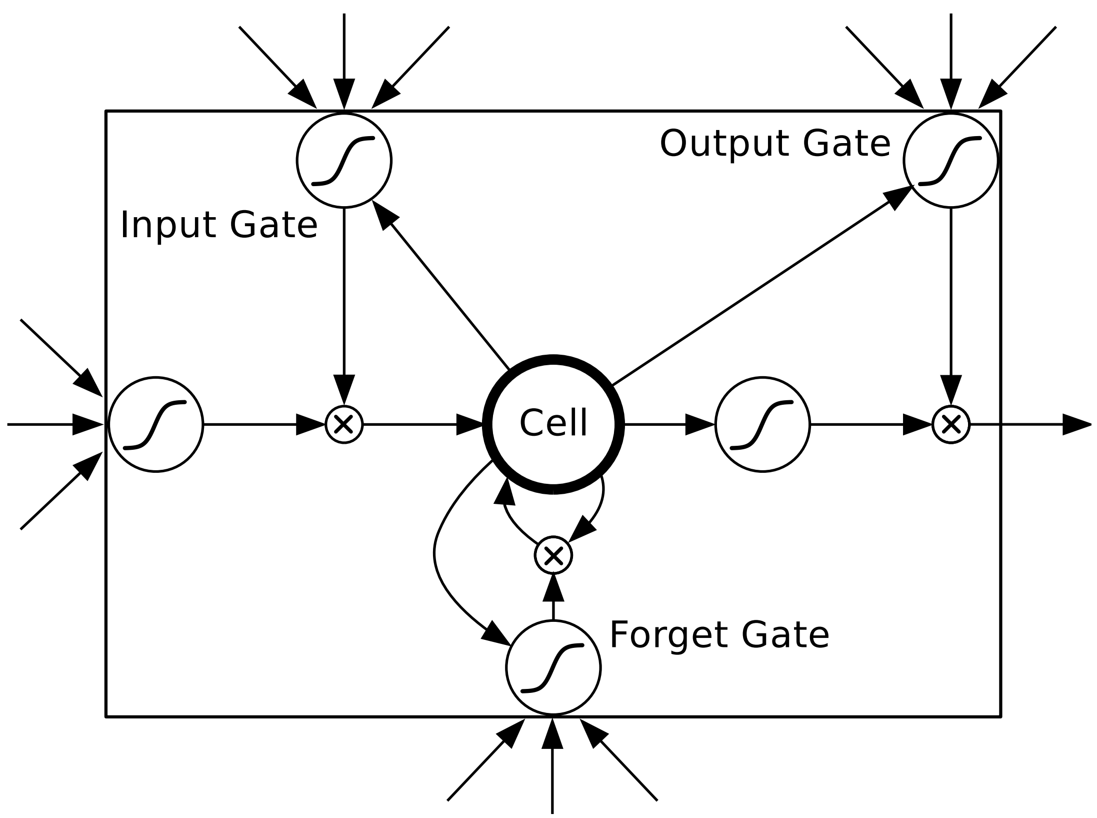
- Best to split into smallest chunks possible: more flexible, fewer parameters, avoids 'blurred edges'

# The Role of Memory

- Need to remember the past to predict the future
- Having a longer memory has several advantages:
  - can store and generate longer range patterns
  - especially ‘disconnected’ patterns like balanced quotes and brackets
  - more robust to ‘mistakes’

# Long Short-Term Memory

- LSTM is an RNN architecture designed to have a better memory. It uses linear memory cells surrounded by multiplicative gate units to store read, write and reset information



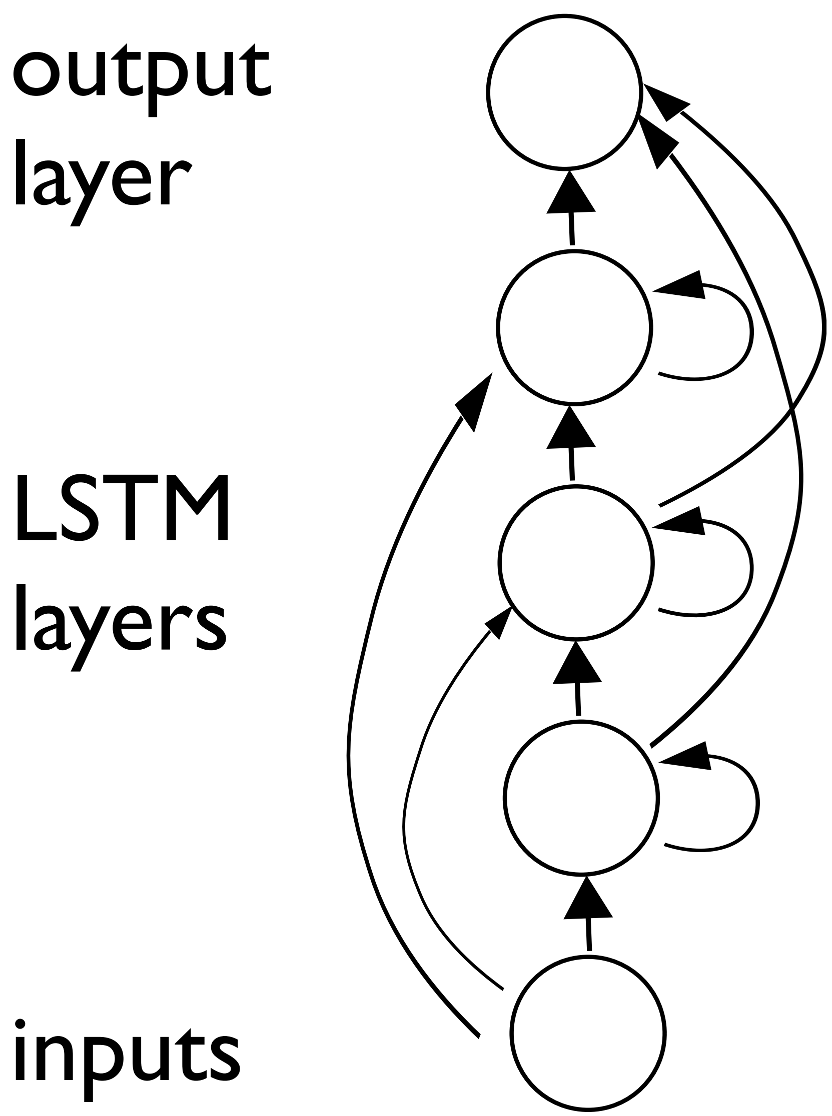

Input gate: scales input to cell (write)

Output gate: scales output from cell (read)

Forget gate: scales old cell value (reset)

- S. Hochreiter and J. Schmidhuber, "*Long Short-term Memory*" Neural Computation 1997

# Basic Architecture




- Deep recurrent LSTM net with skip connections
- Inputs arrive one at a time, outputs determine predictive distribution over next input
- Train by minimising log-loss:

$$\sum_{t=1}^{T} -\log \Pr(x_t | x_{1:t-1})$$

- Generate by sampling from output distribution and feeding into input

# Text Generation

- Task: generate text sequences one character at a time
- Data: raw wikipedia markup from Hutter challenge (100 MB)
- 205 inputs (unicode bytes), 205 way softmax output layer, 5 hidden layers of 700 LSTM cells, ~21M weights
- Split into length 100 sequences, no resets in between
- Trained with SGD, learn rate 0.0001, momentum 0.9
- Took forever!

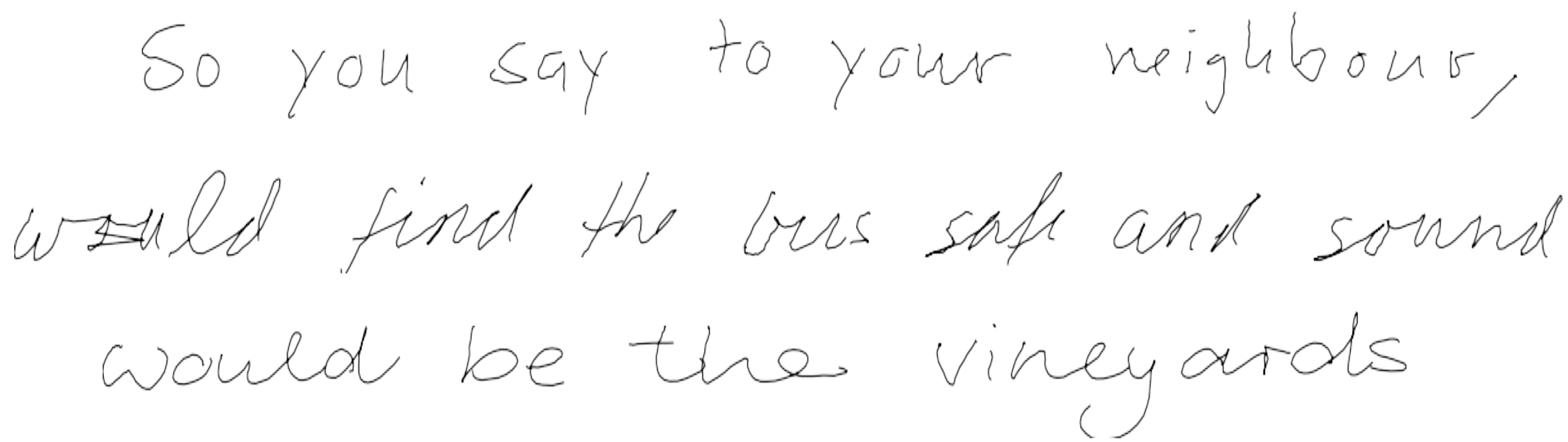
# Compression Results

| Method | Bits per Character |
|---|---|
| bzip2 | 2.32 |
| M-RNN[1] | 1.6 (text only) |
| **deep LSTM** | **1.42** (1.33 validation) |
| PAQ-8[2] | 1.28 |

1) I. Sutskever et. al. "*Generating Text with Recurrent Neural Networks*" ICML, 2011
2) M. Mahoney, *"Adaptive Weighing of Context Models for Lossless Data Compression"*, Florida Tech. CS-2005-16, 2005

# Handwriting Generation

- Task: generate pen trajectories by predicting one (x,y) point at a time
- Data: IAM online handwriting, 10K training sequences, many writers, unconstrained style, captured from whiteboard




- First problem: how to predict real-valued coordinates?

# Recurrent Mixture Density Networks

- Can model continuous sequences with RMDNs
- Suitably squashed output units parameterise a mixture distribution (usually Gaussian)
- <u>Not</u> just fitting Gaussians to data: every output distribution conditioned on all inputs so far

$$\Pr(o_t) = \sum_i w_i(x_{1:t}) \mathcal{N}\left(o_t | \sigma_i(x_{1:t}), \Sigma_i(x_{1:t})\right)$$

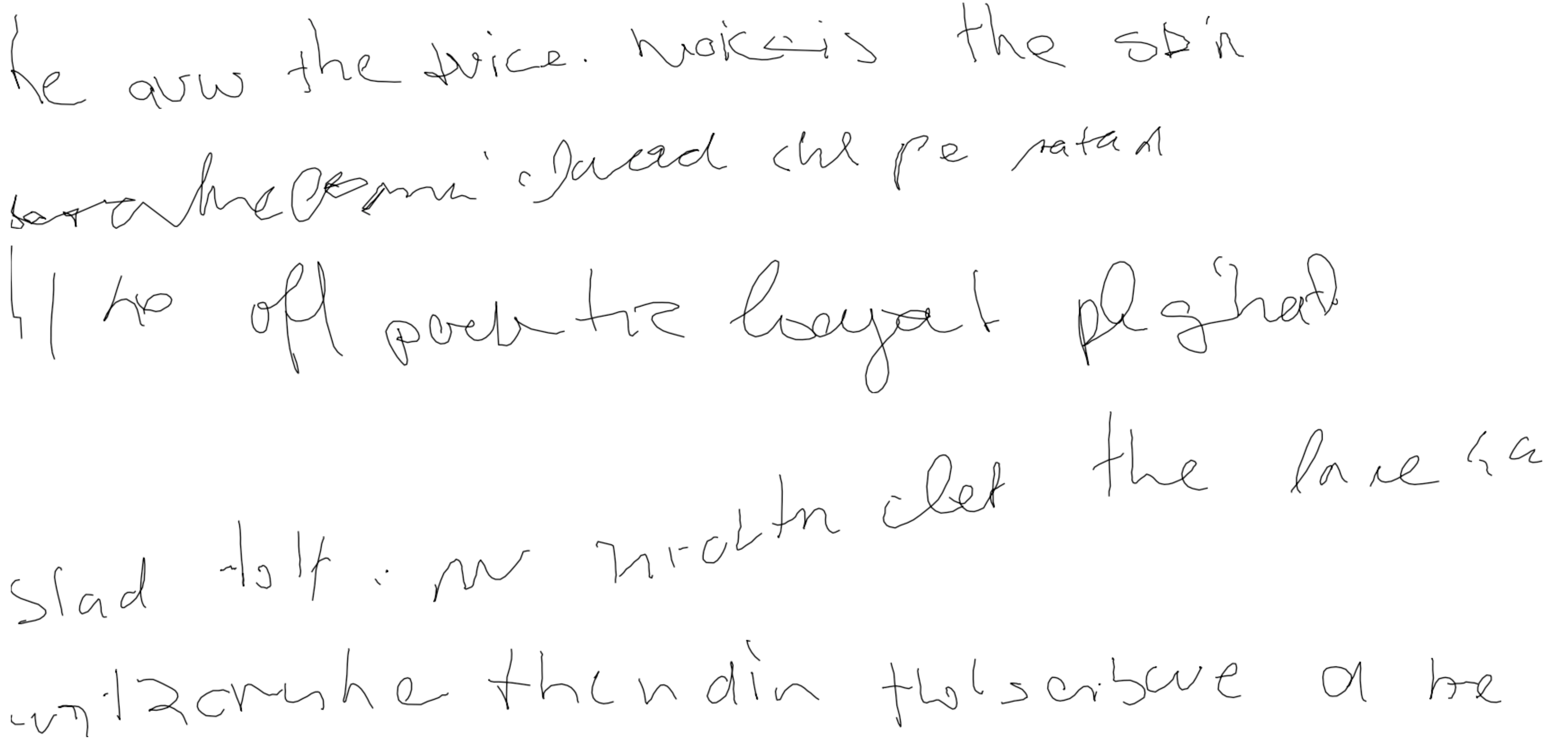
- For prediction, number of components is number of *choices* for what comes next


- M. Schuster, "Better Generative Models for Sequential Data Problems: Bidirectional Recurrent Mixture Density Networks", NIPS 1999

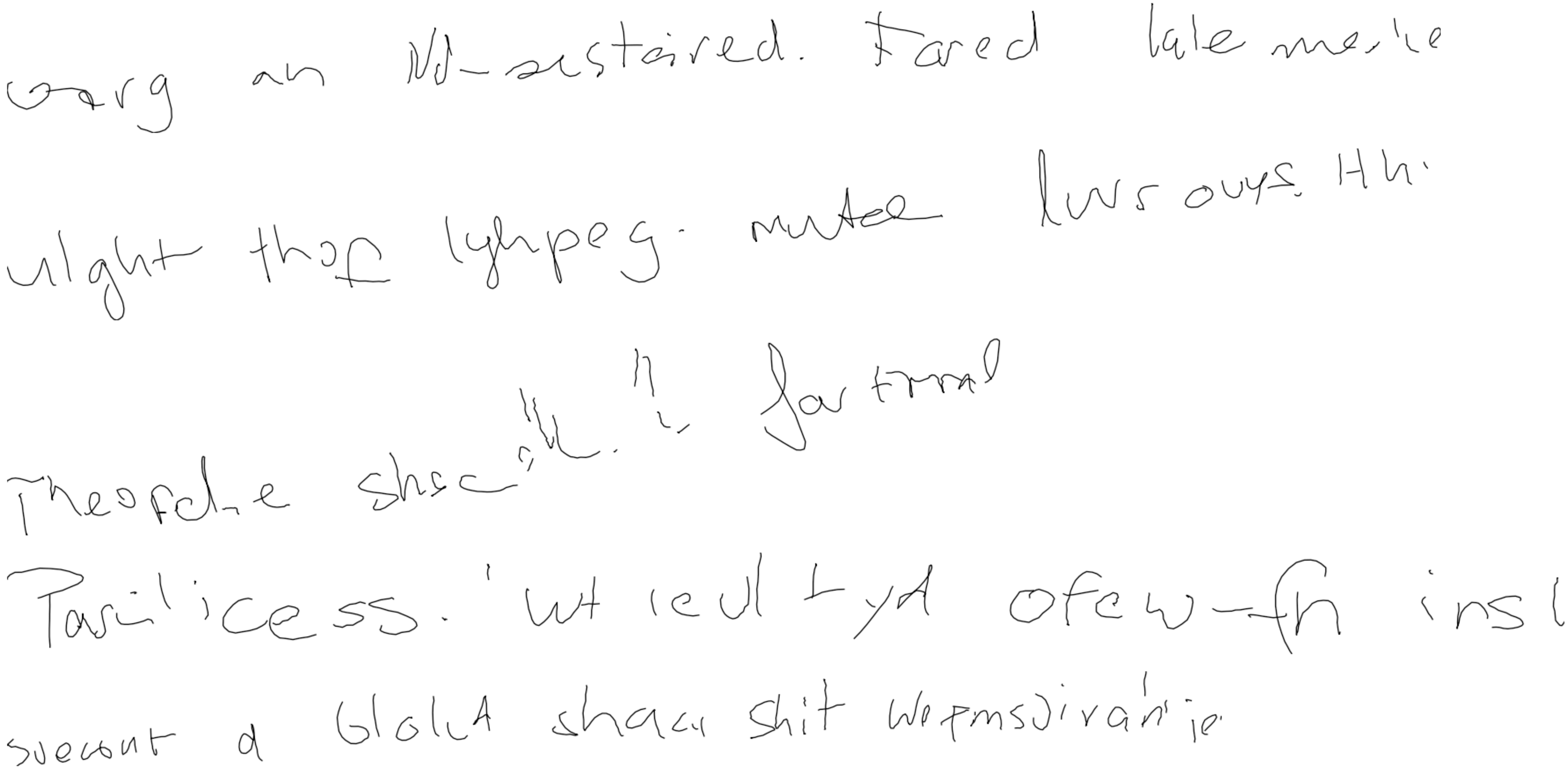
# Network Details

- 3 inputs: $\Delta x$, $\Delta y$, pen up/down
- 121 output units
  - 20 two dimensional Gaussians for x,y = 40 means (linear) + 40 std. devs (exp) + 20 correlations (tanh) + 20 weights (softmax)
  - 1 sigmoid for up/down
- 3 hidden Layers, 400 LSTM cells in each
- 3.6M weights total
- Trained with RMSprop, learn rate 0.0001, momentum 0.9
- Error clipped during backward pass (<u>lots</u> of numerical problems)
- Trained overnight on fast multicore CPU

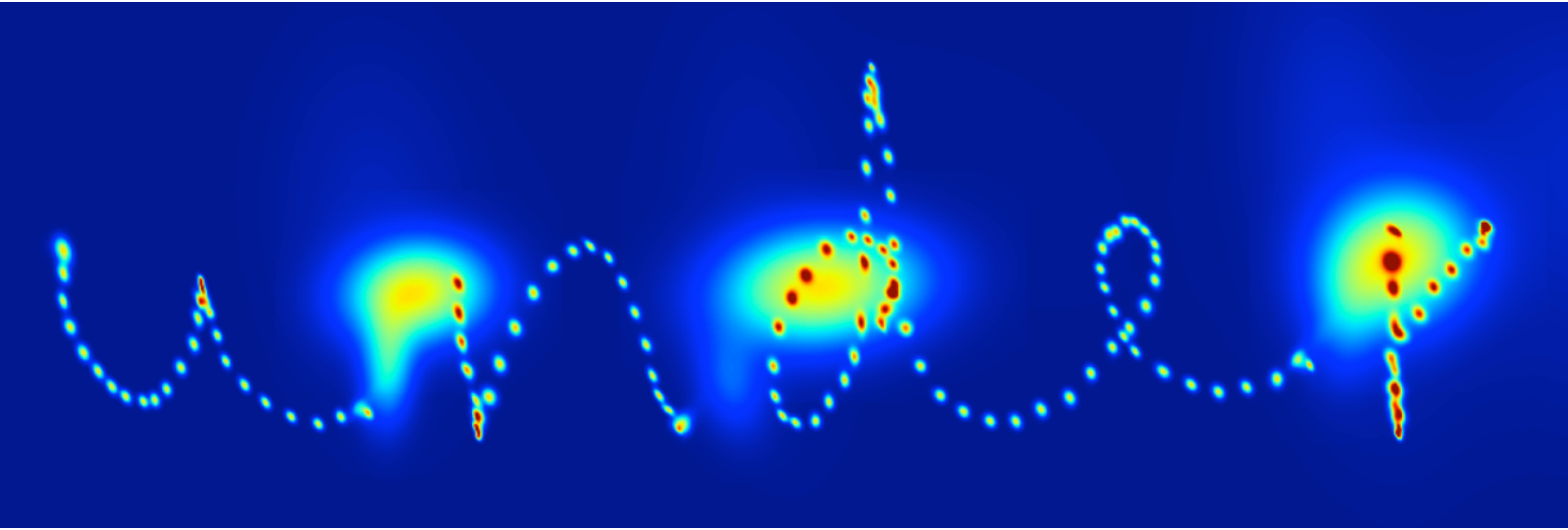
# Samples

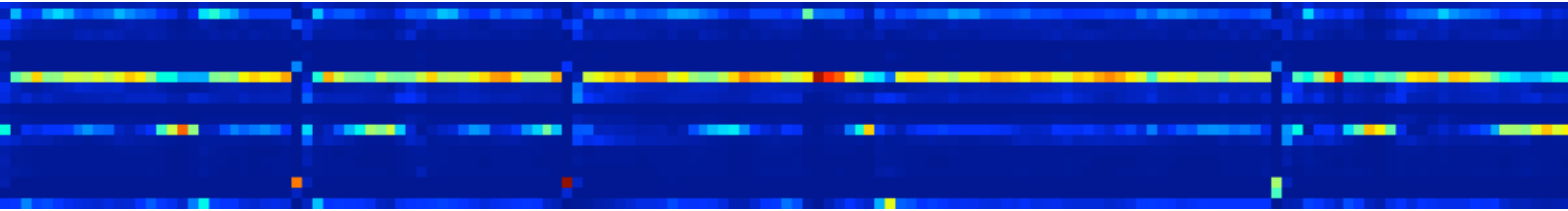


# Samples



# Output Density





# Handwriting Synthesis

- Want to tell the network *what* to write without losing the distribution over *how* it writes
- Can do this by conditioning the predictions on a text sequence
- Problem: alignment between text and writing unknown
- Solution: before each prediction, let the network decide *where* it is in the text sequence

# Soft Windows

window vector (input to net)

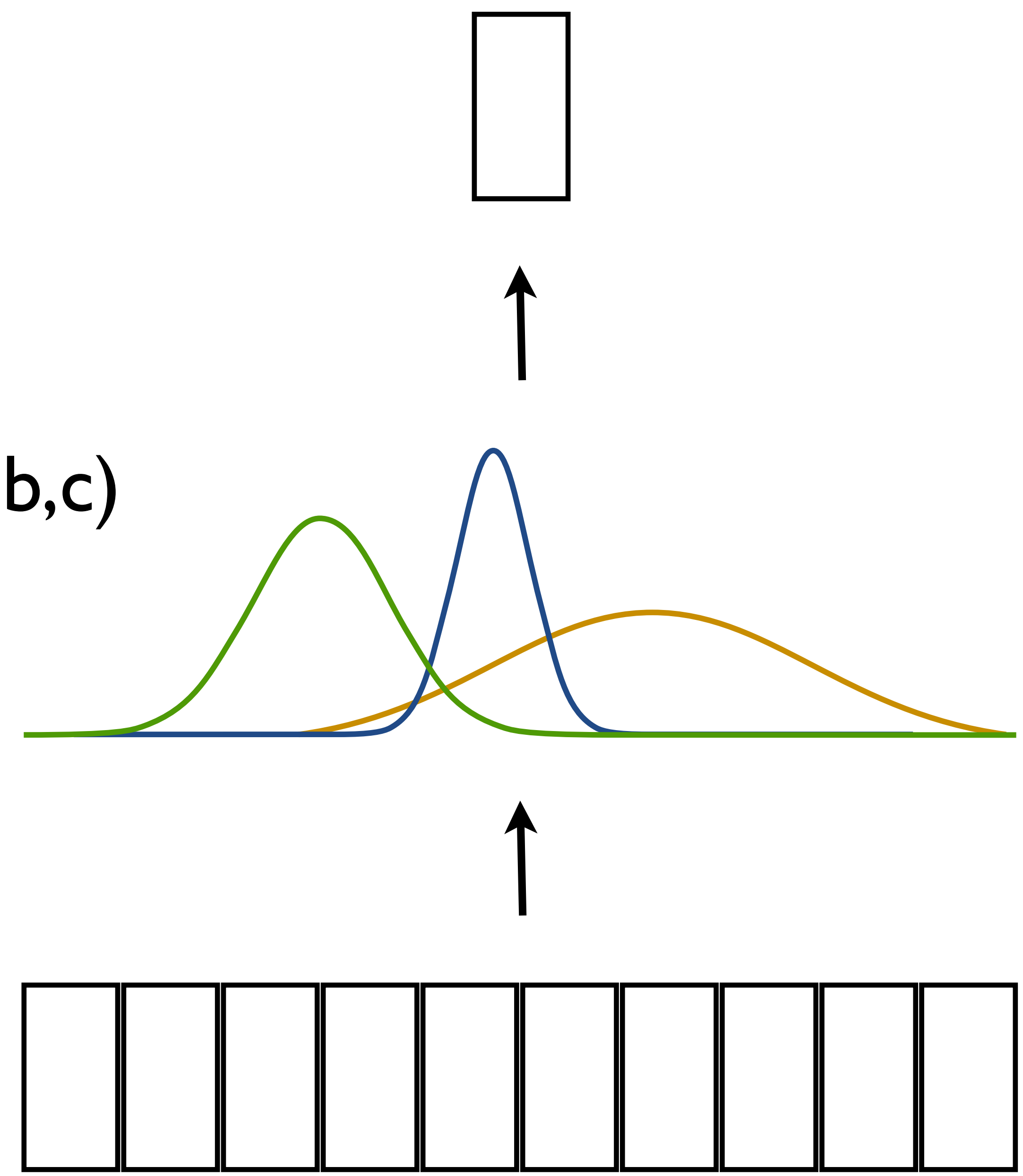
$$v^{t+1} = \sum_{i=1}^{S} w_i^t s_i$$

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

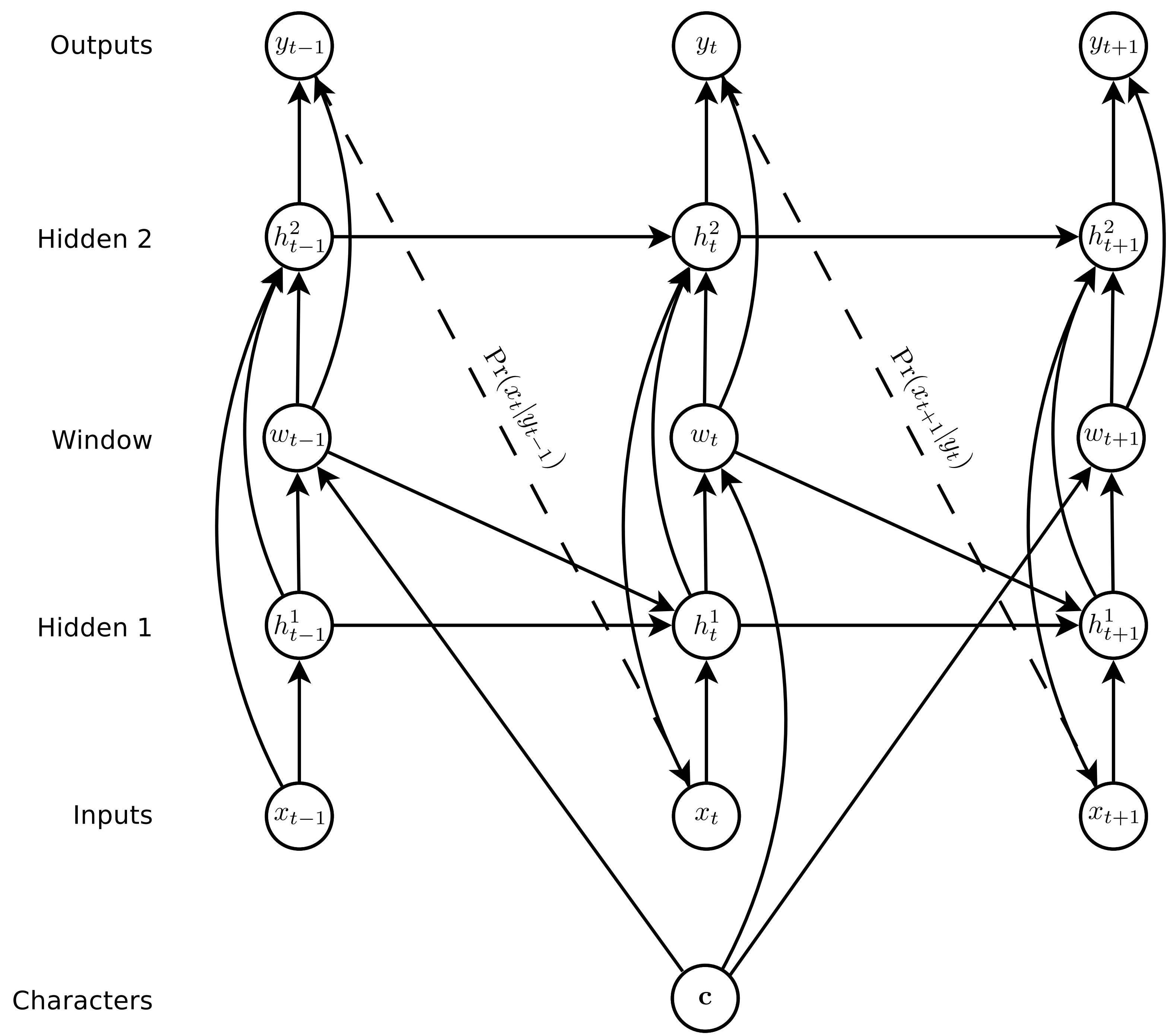
$$w_i^t = \sum_{k=1}^{K} a_k^t \exp\left(-b_k^t [c_k^t - i]^2\right)$$

input vectors (text)

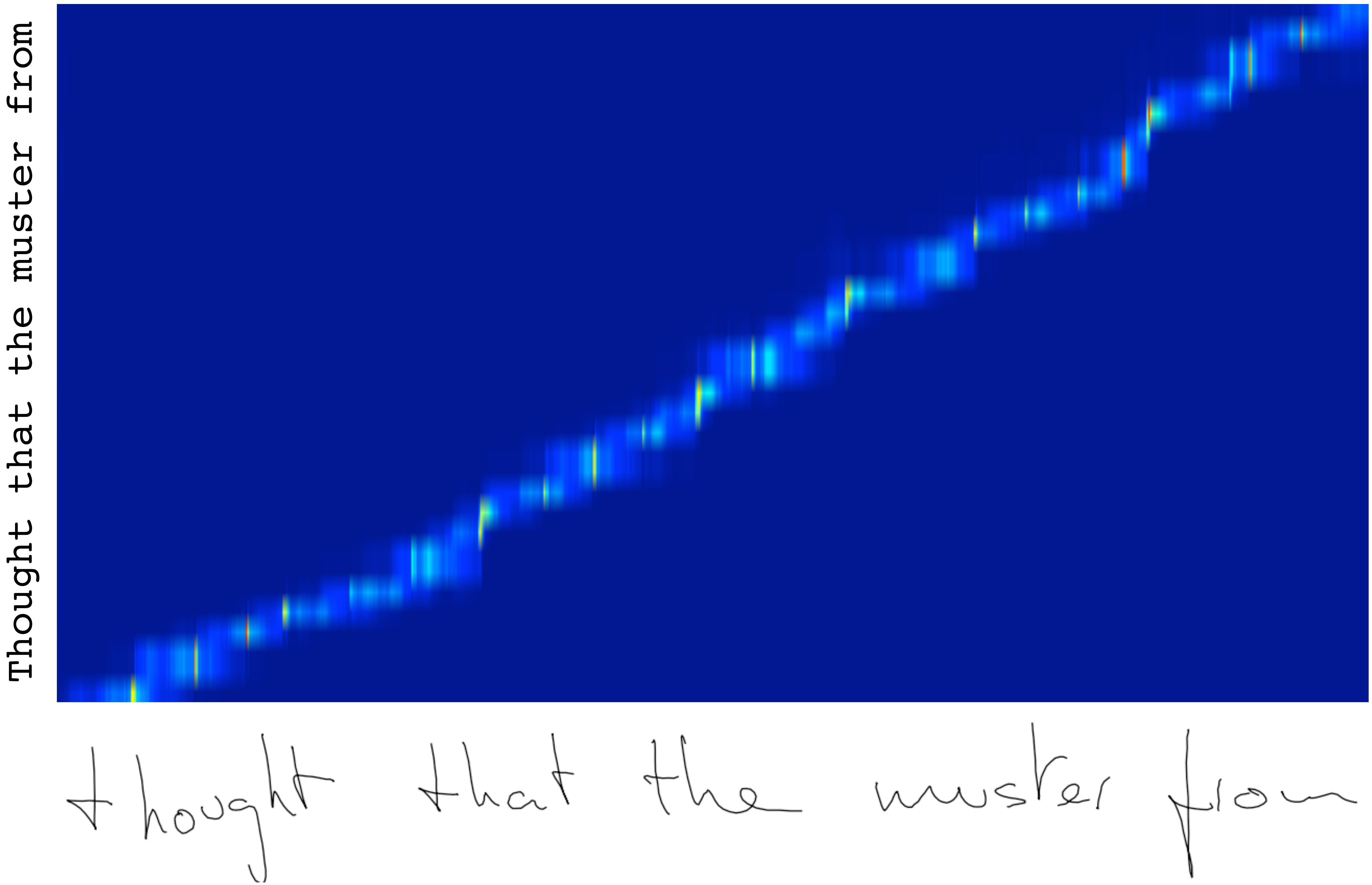
$$(s_1, \ldots, s_S)$$



# Network Architecture



# Alignment



# Which is Real?

that a doctor should be

that a doctor should be

thaet a docaur should ae

that a doctor should be

that a doctor sholoel be

that a doctor should be

# Which is Real?

of presentlee altty inremtoohering

of preseng reality in qeumming

of present reality i rem=mbering

of present reality in remembering

of present reality in remembering

of present reality in remembering

# Which is Real?

was an occasicn wothy of his

was an occasion worthy of his

was an ocascionsamthy of his

was an occasion worthy op his

wss an occasion vurthy of his

was an occarion woorthy of his

# Unbiased Sampling

these sequences were generated by

picking samples at every step

every line is a different style

yes, real people write this badly

# Biased Sampling

when the samples are biased

towards more probable sequences

they get easier to read

but less interesting to look at.

# Primed Sampling

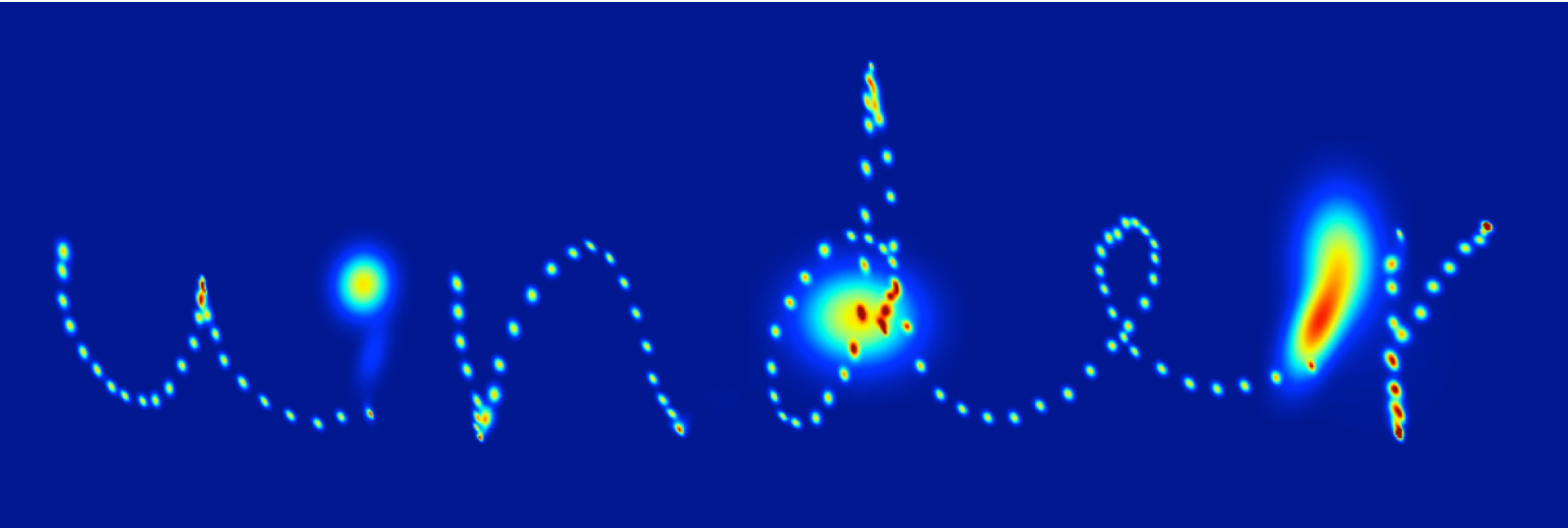
when the sample starts with real data

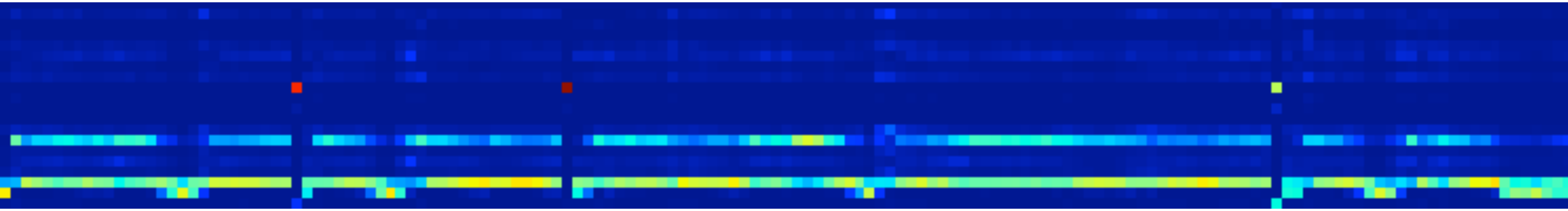
(prison welfare Officer complement)

it continues in the same style

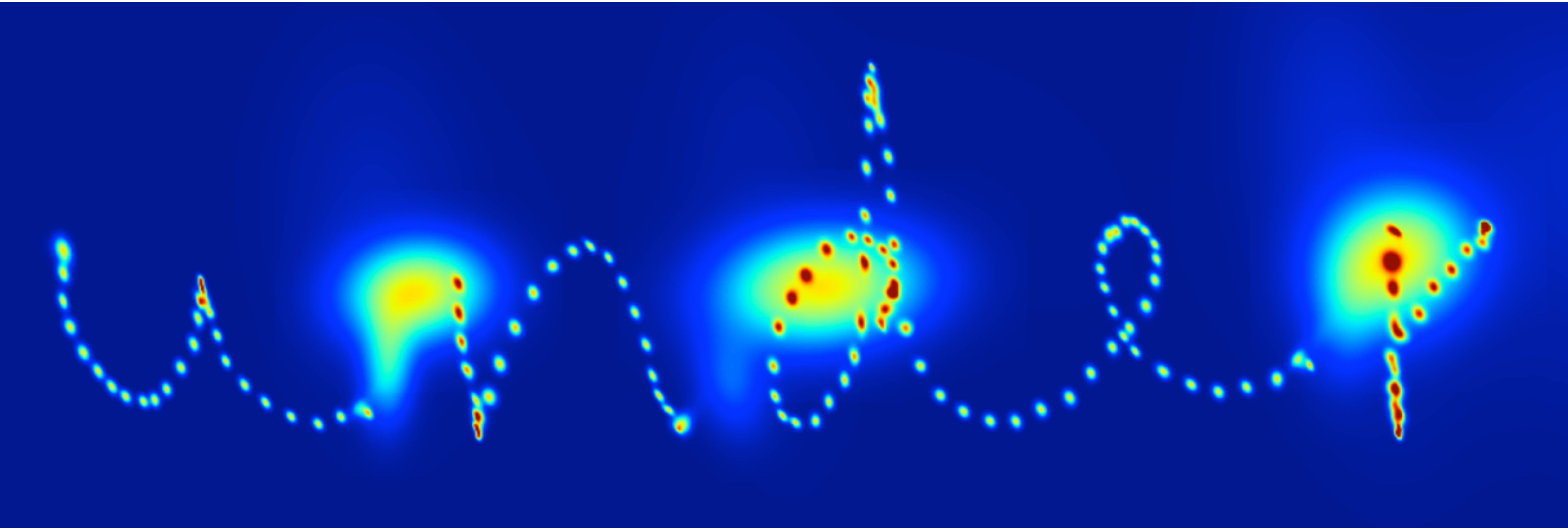
(He dismissed the idea)

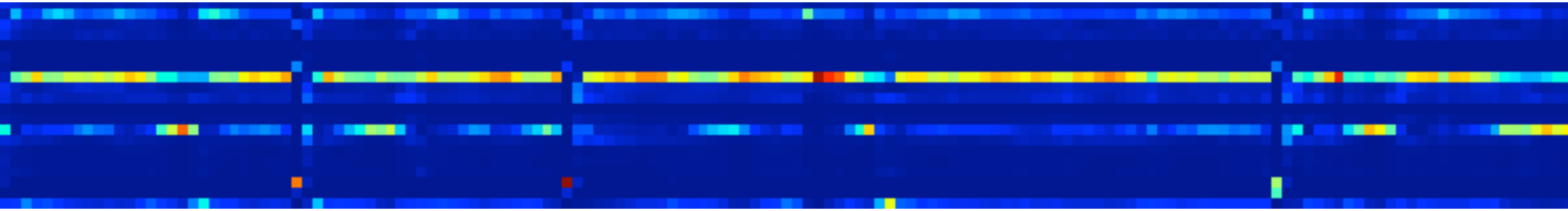
# Synthesis Output Density





# Prediction Output Density





# Some Numbers

| Network | Δ Nats |
|---|---|
| 3 layer tanh prediction | +1139(!) |
| 1 layer prediction | +15 |
| 3 layer prediction (baseline) | 0 |
| 3 layer synthesis | -56 |
| 3 layer synthesis + var. Bayes | -86 |
| 3 layer synthesis + text | -25 |

# Where Next?

- Speech synthesis
- Better understanding of internal representation
- Learn high level features (strokes, letters, words...) rather than adding them manually

Thank You!