Speech Synthesis, External Memory Networks

Milan Straka

May 15, 2023

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Our goal is to model speech, using a convolutional auto-regressive model

$$P(\boldsymbol{x}) = \prod_t P(x_t | x_{t-1}, \ldots, x_1).$$

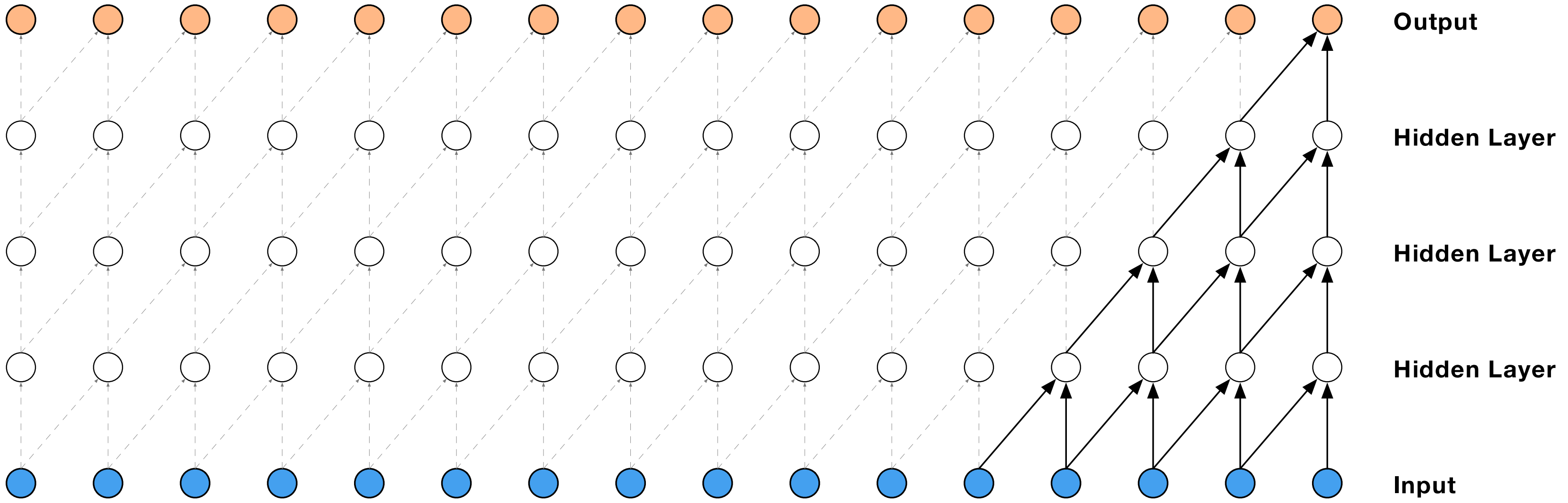


Figure 2: Visualization of a stack of causal convolutional layers.

Figure 2 of "WaveNet: A Generative Model for Raw Audio", https://arxiv.org/abs/1609.03499

However, to achieve larger receptive field, we utilize **dilated** (or **atrous**) convolutions:

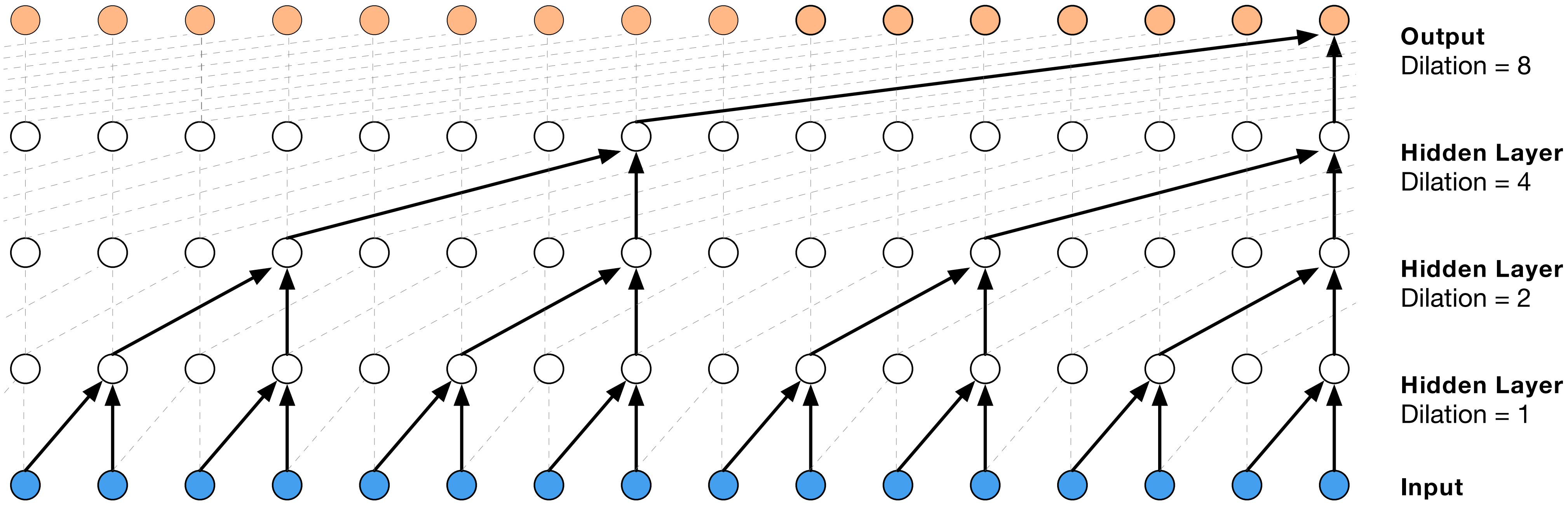
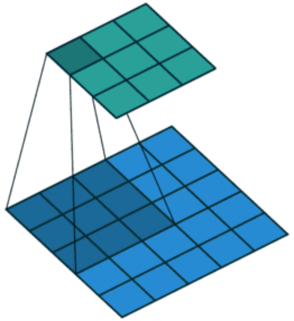
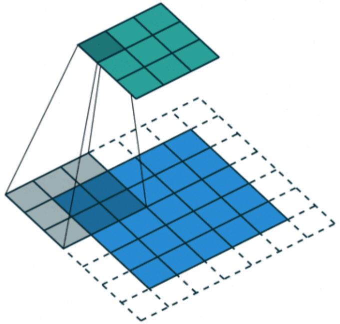
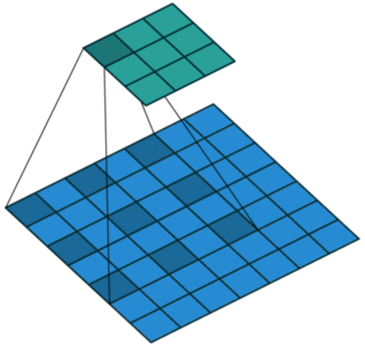
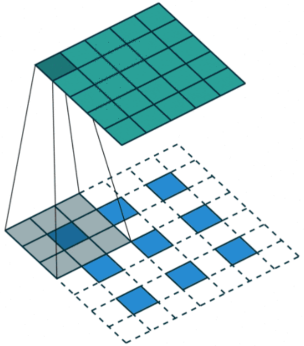


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Figure 3 of "WaveNet: A Generative Model for Raw Audio", https://arxiv.org/abs/1609.03499

Dilated Versus Regular Versus Strided Convolutions

Regular Convolution	Strided Convolution
 *https://github.com/vdumoulin/conv_arithmetic*	 *https://github.com/vdumoulin/conv_arithmetic*
Dilated Convolution	**Transposed Strided Convolution**
 *https://github.com/vdumoulin/conv_arithmetic*	 *https://github.com/vdumoulin/conv_arithmetic*

WaveNet – Output Distribution

Output Distribution

WaveNet generates audio with 16kHz frequency and 16-bit samples.

However, classification into $65\,536$ classes would not be efficient. Instead, WaveNet adopts the μ-law transformation, which passes the input samples in $[-1, 1]$ range through the μ-law encoding

$$\operatorname{sign}(x)\frac{\log(1+255|x|)}{\log(1+255)},$$

and the resulting $[-1, 1]$ range is linearly quantized into 256 buckets.

The model therefore predicts each samples using classification into 256 classes, and then uses the inverse of the above transformation on the model predictions.

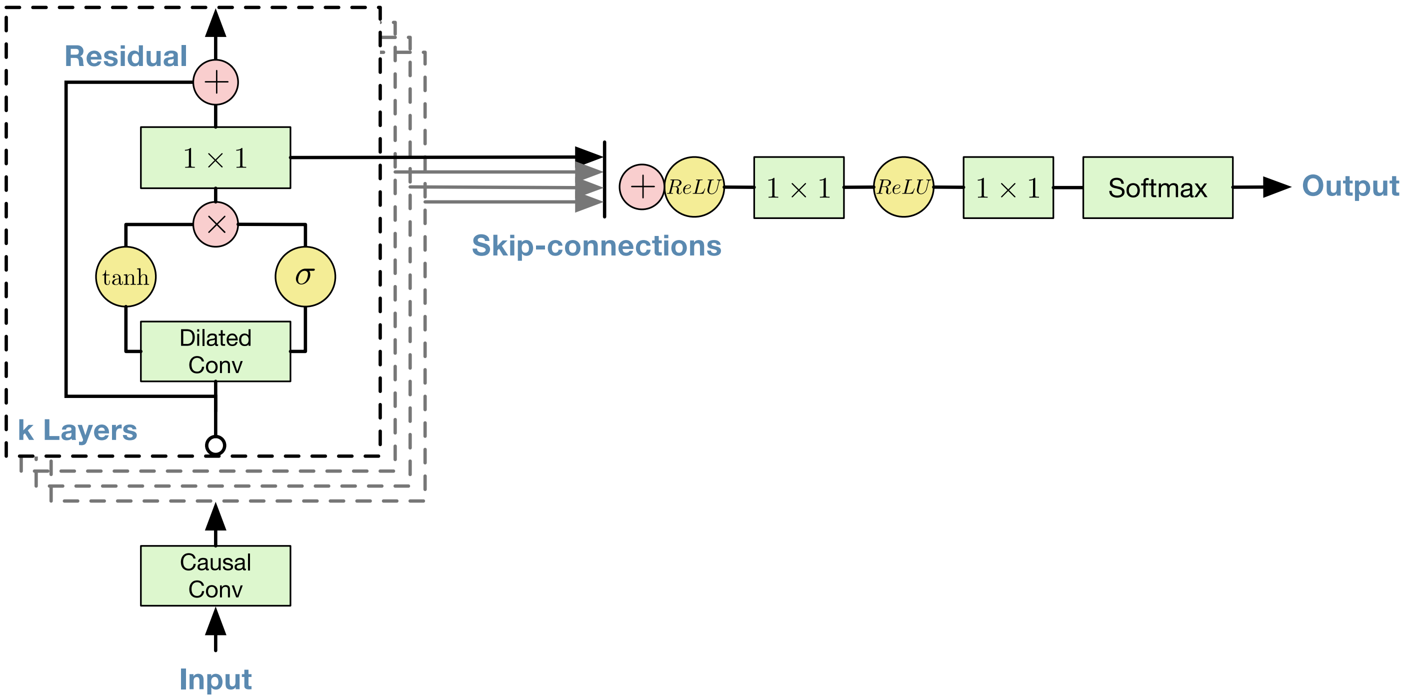


Figure 4: Overview of the residual block and the entire architecture.

Figure 4 of "WaveNet: A Generative Model for Raw Audio", https://arxiv.org/abs/1609.03499

The outputs of the dilated convolutions are passed through the *gated activation unit*:

$$\boldsymbol{z} = \tanh(\boldsymbol{W}_f * \boldsymbol{x}) \odot \sigma(\boldsymbol{W}_g * \boldsymbol{x}).$$

Global Conditioning

Global conditioning is performed by a single latent representation $\boldsymbol{h}$, changing the gated activation function to

$$\boldsymbol{z} = \tanh(\boldsymbol{W}_f * \boldsymbol{x} + \boldsymbol{V}_f \boldsymbol{h}) \odot \sigma(\boldsymbol{W}_g * \boldsymbol{x} + \boldsymbol{V}_g \boldsymbol{h}).$$

Local Conditioning

For local conditioning, we are given a time series $\boldsymbol{h}$, possibly with a lower sampling frequency. We first use transposed convolutions $\boldsymbol{y} = f(\boldsymbol{h})$ to match resolution and then compute analogously to global conditioning

$$\boldsymbol{z} = \tanh(\boldsymbol{W}_f * \boldsymbol{x} + \boldsymbol{V}_f * \boldsymbol{y}) \odot \sigma(\boldsymbol{W}_g * \boldsymbol{x} + \boldsymbol{V}_g * \boldsymbol{y}).$$

The original paper did not mention hyperparameters, but later it was revealed that:

- 30 layers were used
  - grouped into 3 dilation stacks with 10 layers each
  - in a dilation stack, dilation rate increases by a factor of 2, starting with rate 1 and reaching maximum dilation of 512
- filter size of a dilated convolution is 2 (and extended to 3 in Parallel WaveNet)
- residual connection has dimension 512
- gating layer uses 256+256 hidden units
- the 1×1 convolutions in the output step produce 256 filters
- trained for $1\,000\,000$ steps using Adam with a fixed learning rate of 2e-4

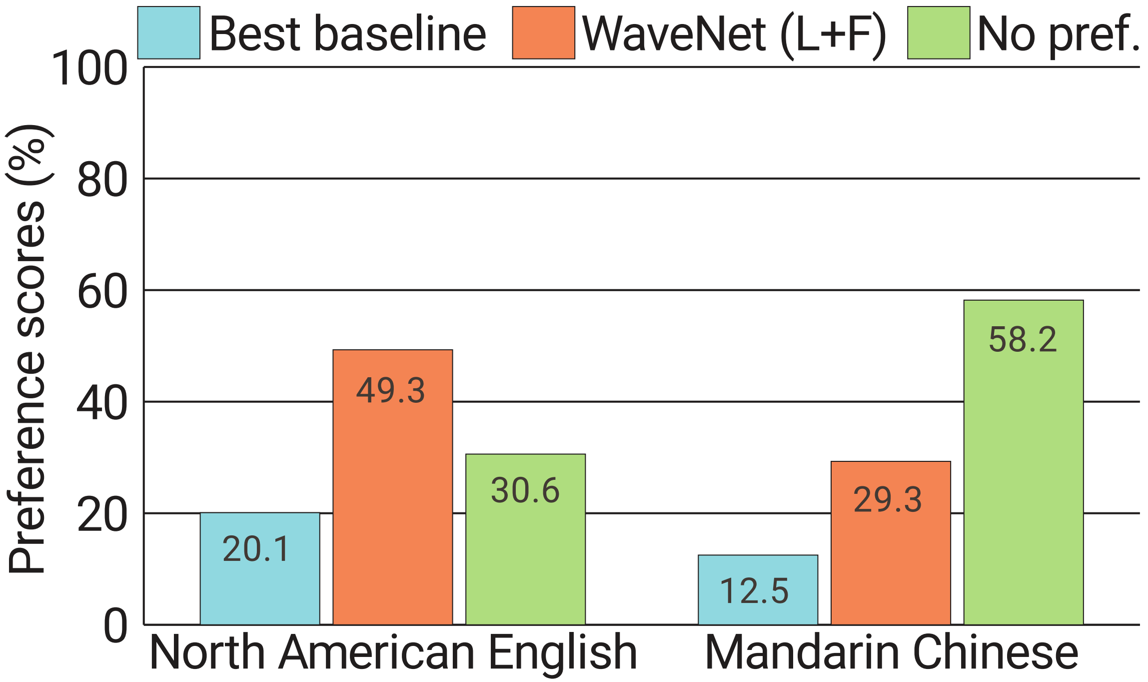


Figure 5: Subjective preference scores (%) of speech samples between (top) two baselines, (middle) two WaveNets, and (bottom) the best baseline and WaveNet. Note that LSTM and Concat correspond to LSTM-RNN-based statistical parametric and HMM-driven unit selection concatenative baseline synthesizers, and WaveNet (L) and WaveNet (L+F) correspond to the WaveNet conditioned on linguistic features only and that conditioned on both linguistic features and log F_0 values.

Figure 5 of "WaveNet: A Generative Model for Raw Audio", https://arxiv.org/abs/1609.03499

Gated Activations in Transformers

Similar gated activations seem to work the best in Transformers, in the FFN module.

Activation Name	Formula	$\mathrm{FFN}(x; \boldsymbol{W}_1, \boldsymbol{W}_2)$
ReLU	$\max(0, x)$	$\max(0, \boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2$
GELU	$x\Phi(x)$	$\mathrm{GELU}(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2$
Swish	$x\sigma(x)$	$\mathrm{Swish}(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2$

There are several variants of the new gated activations:

Activation Name	Formula	$\mathrm{FFN}(x; \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{W}_2)$
GLU (Gated Linear Unit)	$\sigma(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \odot (\boldsymbol{x}\boldsymbol{V} + \boldsymbol{c})$	$(\sigma(\boldsymbol{x}\boldsymbol{W}) \odot \boldsymbol{x}\boldsymbol{V})\boldsymbol{W}_2$
ReGLU	$\max(0, \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \odot (\boldsymbol{x}\boldsymbol{V} + \boldsymbol{c})$	$(\max(0, \boldsymbol{x}\boldsymbol{W}) \odot \boldsymbol{x}\boldsymbol{V})\boldsymbol{W}_2$
GEGLU	$\mathrm{GELU}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \odot (\boldsymbol{x}\boldsymbol{V} + \boldsymbol{c})$	$(\mathrm{GELU}(\boldsymbol{x}\boldsymbol{W}) \odot \boldsymbol{x}\boldsymbol{V})\boldsymbol{W}_2$
SwiGLU	$\mathrm{Swish}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \odot (\boldsymbol{x}\boldsymbol{V} + \boldsymbol{c})$	$(\mathrm{Swish}(\boldsymbol{x}\boldsymbol{W}) \odot \boldsymbol{x}\boldsymbol{V})\boldsymbol{W}_2$

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
$\text{FFN}_{\text{Swish}}$	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
$\text{FFN}_{\text{GEGLU}}$	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$\text{FFN}_{\text{Bilinear}}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
$\text{FFN}_{\text{SwiGLU}}$	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
$\text{FFN}_{\text{ReGLU}}$	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59

Table 2 of "GLU Variants Improve Transformer", https://arxiv.org/abs/2002.05202

	EM	F1
FFN_{ReLU}	83.18	90.87
FFN_{GELU}	83.09	90.79
$\text{FFN}_{\text{Swish}}$	83.25	90.76
FFN_{GLU}	82.88	90.69
$\text{FFN}_{\text{GEGLU}}$	83.55	91.12
$\text{FFN}_{\text{Bilinear}}$	**83.82**	91.06
$\text{FFN}_{\text{SwiGLU}}$	83.42	91.03
$\text{FFN}_{\text{ReGLU}}$	83.53	**91.18**

Table 4 of "GLU Variants Improve Transformer", https://arxiv.org/abs/2002.05202

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
GeLU	$223M$	$11.1T$	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**	26.47
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**	**26.75**
ELU	$223M$	$11.1T$	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	$223M$	$11.1T$	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34	**27.12**
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**	**26.87**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**	**27.02**
SeLU	$223M$	$11.1T$	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	$223M$	$11.1T$	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**	**27.02**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**	26.53
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02	26.30
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**	**26.89**

Table 1 of "Do Transformer Modifications Transfer Across Implementations and Applications?", https://arxiv.org/abs/2102.11972

Parallel WaveNet is an improvement of the original WaveNet by the same authors.

First, the output distribution was changed from 256 μ-law values to a Mixture of Logistic (suggested in another paper – PixelCNN++, but reused in other architectures since):

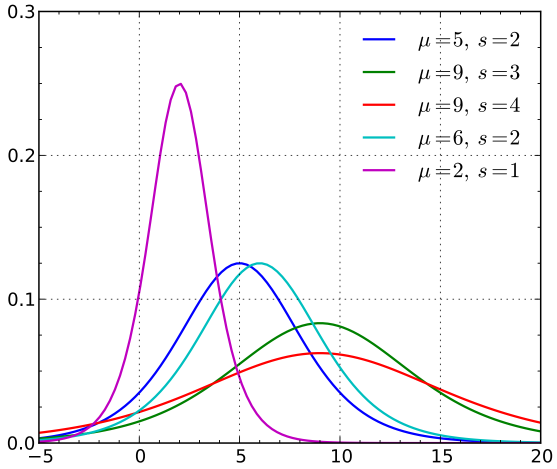
$$x \sim \sum_i \pi_i \operatorname{Logistic}(\mu_i, s_i).$$

The logistic distribution is a distribution with a σ as cumulative density function (where the mean and scale is parametrized by μ and s). Therefore, we can write

$$P(x|\boldsymbol\pi, \boldsymbol\mu, \boldsymbol s) = \sum_i \pi_i \left[\sigma\left(\frac{x+0.5-\mu_i}{s_i}\right) - \sigma\left(\frac{x-0.5-\mu_i}{s_i}\right)\right],$$

where we replace -0.5 and 0.5 in the edge cases by $-\infty$ and ∞.

In Parallel WaveNet teacher, 10 mixture components are used.



https://commons.wikimedia.org/wiki/File:Logisticpdfunction.svg

Parallel WaveNet

Auto-regressive (sequential) inference is extremely slow in WaveNet.

Instead, we model $P(x_t)$ as $P(x_t|\boldsymbol{z}_{<t}) = \operatorname{Logistic}\big(x_t; \mu^1(\boldsymbol{z}_{<t}), s^1(\boldsymbol{z}_{<t})\big)$ for a *random* $\boldsymbol{z}$ drawn from a logistic distribution $\operatorname{Logistic}(\mathbf{0}, \mathbf{1})$. Therefore, using the reparametrization trick,

$$x_t^1 = \mu^1(\boldsymbol{z}_{<t}) + z_t \cdot s^1(\boldsymbol{z}_{<t}).$$

Usually, one iteration of the algorithm does not produce good enough results – consequently, 4 iterations were used by the authors. In further iterations,

$$x_t^i = \mu^i(\boldsymbol{x}_{<t}^{i-1}) + x_t^{i-1} \cdot s^i(\boldsymbol{x}_{<t}^{i-1}).$$

After N iterations, $P(\boldsymbol{x}_t^N|\boldsymbol{z}_{<t})$ is a logistic distribution with location $\boldsymbol{\mu}^{\mathrm{tot}}$ and scale $\boldsymbol{s}^{\mathrm{tot}}$:

$$\mu_t^{\mathrm{tot}} = \sum_{i=1}^N \mu^i(\boldsymbol{x}_{<t}^{i-1}) \cdot \Big(\prod\nolimits_{j>i}^N s^j(\boldsymbol{x}_{<t}^{j-1})\Big) \text{ and } s_t^{\mathrm{tot}} = \prod_{i=1}^N s^i(\boldsymbol{x}_{<t}^{i-1}),$$

where we have denoted $\boldsymbol{z}$ as $\boldsymbol{x}^0$ for convenience.

Parallel WaveNet

The consequences of changing the model to

$$x_t^1 = \mu^1(\boldsymbol{z}_{<t}) + z_t \cdot \boldsymbol{s}^1(\boldsymbol{z}_{<t})$$
$$x_t^i = \mu^i(\boldsymbol{x}_{<t}^{i-1}) + x_t^{i-1} \cdot \boldsymbol{s}^i(\boldsymbol{x}_{<t}^{i-1})$$

are:

- During inference, the prediction can be computed in parallel, because x_t^i depends only on $\boldsymbol{x}_{<t}^{i-1}$, not on $x_{<t}^i$.
- However, we cannot perform training in parallel. If we try maximizing the log-likelihood of an input sequence $\boldsymbol{x}^1$, we need to find out which $\boldsymbol{z}$ sequence generates it.
  - The z_1 can be computed using x_1^1.
  - However, z_2 depends on only on x_1^1 and x_2^1, but also on z_1; generally, z_t depends on $\boldsymbol{x}^1$ and also on all $\boldsymbol{z}_{<t}$, and can be computed only sequentially.

Therefore, WaveNet can perform parallel training and sequential inference, while the proposed model can perform parallel inference but sequential training.

The authors propose to train the network by a **probability density distillation** using a teacher WaveNet (producing a mixture of logistic with 10 components) with KL-divergence as a loss.

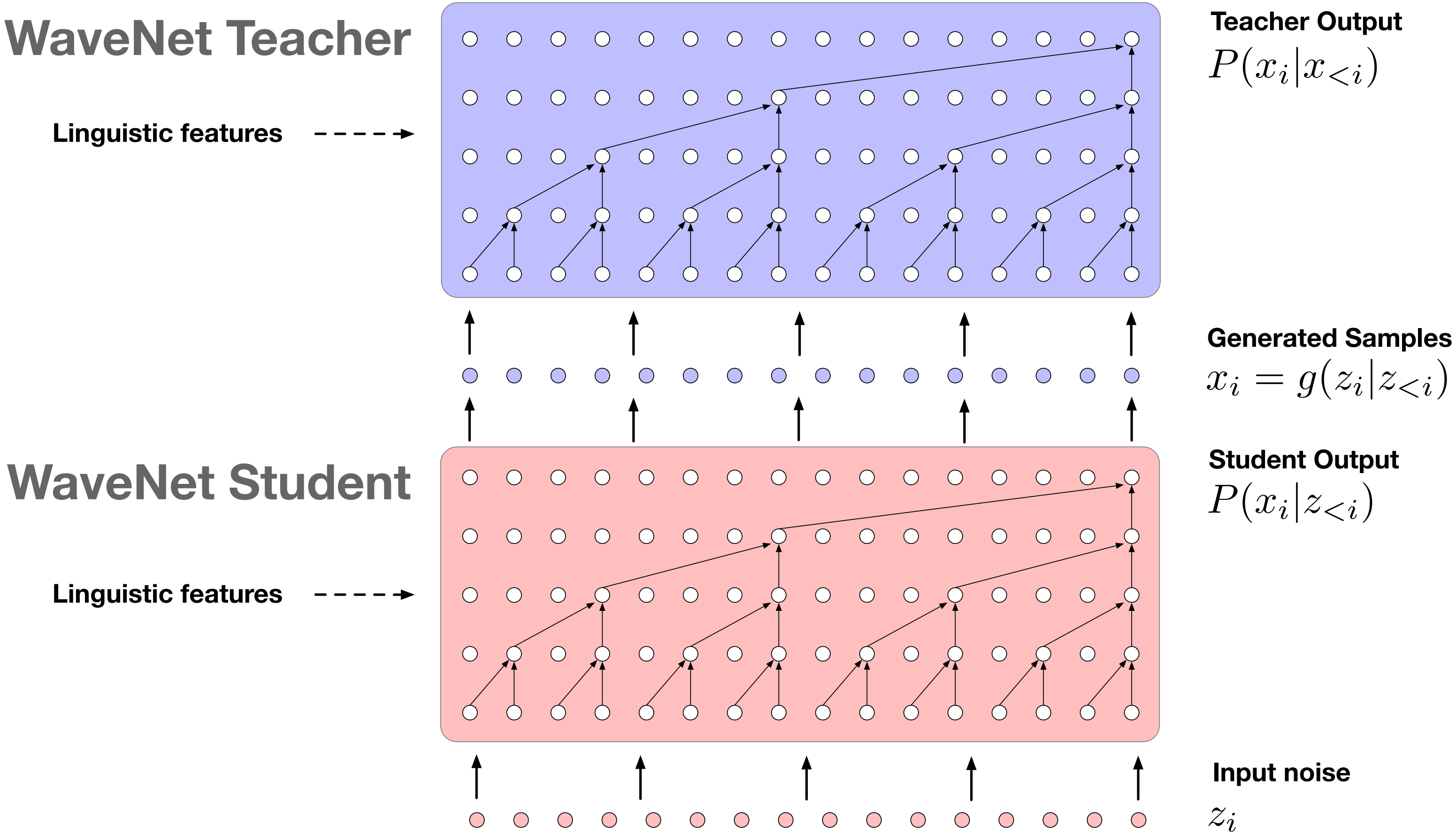


Figure 2 of "Parallel WaveNet: Fast High-Fidelity Speech Synthesis", https://arxiv.org/abs/1711.10433

Probability Density Distillation

Therefore, instead of computing $\boldsymbol{z}$ from some gold $\boldsymbol{x}_g$, we

- sample a random $\boldsymbol{z}$;
- generate the output $\boldsymbol{x}$;
- use the teacher WaveNet model to estimate the log-likelihood of $\boldsymbol{x}$;
- update the student to match the log-likelihood of the teacher.

Denoting the teacher distribution as P_T and the student distribution as P_S, the loss is

$$D_{\mathrm{KL}}(P_S||P_T) = H(P_S, P_T) - H(P_S).$$

Therefore, we do not only minimize cross-entropy, but we also try to keep the entropy of the student as high as possible – it is indeed crucial not to match just the mode of the teacher.

- Consider a teacher generating white noise, where every sample comes from $\mathcal{N}(0,1)$ – in this case, the cross-entropy loss of a constant $\mathbf{0}$, complete silence, would be maximal.

In a sense, probability density distillation is similar to GANs. However, the teacher is kept fixed, and the student does not attempt to fool it but to match its distribution instead.

Probability Density Distillation Details

Because the entropy of a logistic distribution $\operatorname{Logistic}(\mu, s)$ is $\log s + 2$, the entropy term $H(P_S)$ can be rewritten as follows:

$$\begin{aligned} H(P_S) &= \mathbb{E}_{z\sim\operatorname{Logistic}(0,1)}\left[\sum_{t=1}^{T} -\log p_S(x_t|\boldsymbol{z}_{<t})\right] \\ &= \mathbb{E}_{z\sim\operatorname{Logistic}(0,1)}\left[\sum_{t=1}^{T} \log s(\boldsymbol{z}_{<t}, \boldsymbol{\theta})\right] + 2T. \end{aligned}$$

Therefore, this term can be computed without having to generate $\boldsymbol{x}$.

Probability Density Distillation Details

However, the cross-entropy term $H(P_S, P_T)$ requires sampling from P_S to estimate:

$$
\begin{aligned}
H(P_S, P_T) &= \int_{\boldsymbol{x}} -P_S(\boldsymbol{x}) \log P_T(\boldsymbol{x}) \\
&= \sum_{t=1}^{T} \int_{\boldsymbol{x}} -P_S(\boldsymbol{x}) \log P_T(x_t|\boldsymbol{x}_{<t}) \\
&= \sum_{t=1}^{T} \int_{\boldsymbol{x}} -P_S(\boldsymbol{x}_{<t}) P_S(x_t|\boldsymbol{x}_{<t}) P_S(\boldsymbol{x}_{>t}|\boldsymbol{x}_{\leq t}) \log P_T(x_t|\boldsymbol{x}_{<t}) \\
&= \sum_{t=1}^{T} \mathbb{E}_{P_S(\boldsymbol{x}_{<t})} \left[\int_{x_t} -P_S(x_t|\boldsymbol{x}_{<t}) \log P_T(x_t|\boldsymbol{x}_{<t}) \underbrace{\int_{\boldsymbol{x}_{>t}} P_S(\boldsymbol{x}_{>t}|\boldsymbol{x}_{\leq t})}_{1} \right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{P_S(\boldsymbol{x}_{<t})} H\Big(P_S(x_t|\boldsymbol{x}_{<t}), P_T(x_t|\boldsymbol{x}_{<t})\Big).
\end{aligned}
$$

$$H(P_S, P_T) = \sum_{t=1}^T \mathbb{E}_{P_S(\boldsymbol{x}_{<t})} H\Big(P_S(x_t|\boldsymbol{x}_{<t}), P_T(x_t|\boldsymbol{x}_{<t})\Big)$$

We can therefore estimate $H(P_S, P_T)$ by:

- drawing a single sample $\boldsymbol{x}$ from the student P_S *[a Logistic($\boldsymbol{\mu}^\mathrm{tot}$, $\boldsymbol{s}^\mathrm{tot}$)]*,
- compute all $P_T(x_t|\boldsymbol{x}_{<t})$ from the teacher in parallel *[mixture of logistic distributions]*,
- and finally evaluate $H(P_S(x_t|\boldsymbol{x}_{<t}), P_T(x_t|\boldsymbol{x}_{<t}))$ by sampling multiple different x_t from the $P_S(x_t|\boldsymbol{x}_{<t})$.

The authors state that this unbiased estimator has a much lower variance than naively evaluating a single sequence sample under the teacher using the original formulation.

Finally, analogously to the normal distribution, the logistic distribution offers the **reparametrization trick**. Therefore, we can differentiate $\log P_T(x_t|\boldsymbol{x}_{<t})$ with respect to both x_t and $\boldsymbol{x}_{<t}$ (while the categorical distribution is differentiable only with respect to $\boldsymbol{x}_{<t}$).

Parallel WaveNet

With the 4 iterations, the Parallel WaveNet generates over 500k samples per second, compared to ~170 samples per second of a regular WaveNet – more than a 1000 times speedup.

Method	Subjective 5-scale MOS
16kHz, 8-bit μ-law, 25h data:	
LSTM-RNN parametric [27]	3.67 ± 0.098
HMM-driven concatenative [27]	3.86 ± 0.137
WaveNet [27]	4.21 ± 0.081
24kHz, 16-bit linear PCM, 65h data:	
HMM-driven concatenative	4.19 ± 0.097
Autoregressive WaveNet	4.41 ± 0.069
Distilled WaveNet	4.41 ± 0.078

Table 1 of "Parallel WaveNet: Fast High-Fidelity Speech Synthesis", https://arxiv.org/abs/1711.10433

Parallel WaveNet – Additional Losses

To generate high-quality audio, the probability density distillation is not entirely sufficient. The authors therefore introduce additional losses:

- **power loss**: ensures the power in different frequency bands is on average similar between the generated speech and human speech. For a conditioned training data $(\boldsymbol{x}, \boldsymbol{c})$ and WaveNet student g, the loss is

$$\left\| \operatorname{STFT}(g(\boldsymbol{z}, \boldsymbol{c})) - \operatorname{STFT}(\boldsymbol{x}) \right\|^2.$$

- **perceptual loss**: apart from the power in frequency bands, we can use a pre-trained classifier to extract features from generated and human speech and add a loss measuring their difference. The authors propose the loss as squared Frobenius norm of differences between Gram matrices (uncentered covariance matrices) of features of a WaveNet-like classifier predicting phones from raw audio.
- **contrastive loss**: to make the model respect the conditioning instead of generating outputs with high likelihood independent on the conditioning, the authors propose a contrastive distillation loss ($\gamma = 0.3$ is used in the paper):

$$D_{\mathrm{KL}}\big(P_S(\boldsymbol{c}_1) || P_T(\boldsymbol{c}_1)\big) - \gamma D_{\mathrm{KL}}\big(P_S(\boldsymbol{c}_1) || P_T(\boldsymbol{c}_2)\big).$$

Method	Preference Scores versus baseline concatenative system Win - Lose - Neutral
Losses used	
KL + Power	60% - 15% - 25%
KL + Power + Perceptual	66% - 10% - 24%
KL + Power + Perceptual + Contrastive (= default)	65% - 9% - 26%

Table 3: Performance with respect to different combinations of loss terms. We report preference comparison scores since their mean opinion scores tend to be very close and inconclusive.

Table 3 of "Parallel WaveNet: Fast High-Fidelity Speech Synthesis", https://arxiv.org/abs/1711.10433

Tacotron 2 model presents end-to-end speech synthesis directly from text. It consists of two components trained separately:

- a seq2seq model processing input characters and generating mel spectrograms;
- a Parallel WaveNet generating the speech from Mel spectrograms.

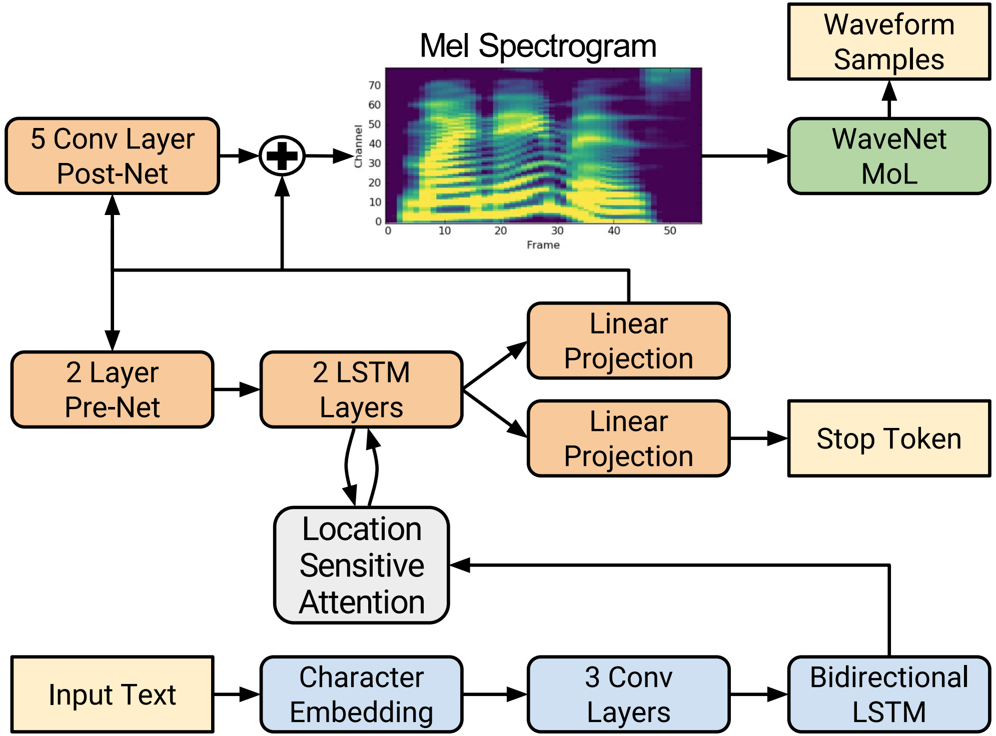


Figure 1 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

The Mel spectrograms are computed using STFT (short-time Fourier transform).

- The authors propose a frame size of 50ms, 12.5ms frame hop, and a Hann window.
- STFT magnitudes are transformed into 80-channel Mel scale spanning 175Hz to 7.6kHz, followed by a log dynamic range compression (clipping input values to at least 0.01).

To make sequential processing of input characters easier, Tacotron 2 utilizes *location-sensitive attention*, which is an extension of the additive attention. While the additive (Bahdanau) attention computes

$$\boldsymbol{\alpha}_i = \operatorname{Attend}(\boldsymbol{s}_{i-1}, \boldsymbol{h}), \quad \alpha_{ij} = \operatorname{softmax}\left(\boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + \boldsymbol{b})\right),$$

the location-sensitive attention also inputs the previous time step attention weights into the current attention computation:

$$\boldsymbol{\alpha}_i = \operatorname{Attend}(\boldsymbol{s}_{i-1}, \boldsymbol{h}, \boldsymbol{\alpha}_{i-1}).$$

In detail, the previous attention weights are processed by a 1-D convolution with kernel $\boldsymbol{F}$:

$$\alpha_{ij} = \operatorname{softmax}\left(\boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + (\boldsymbol{F} * \boldsymbol{\alpha}_{i-1})_j + \boldsymbol{b})\right).$$

System	MOS
Parametric	3.492 ± 0.096
Tacotron (Griffin-Lim)	4.001 ± 0.087
Concatenative	4.166 ± 0.091
WaveNet (Linguistic)	4.341 ± 0.051
Ground truth	4.582 ± 0.053
Tacotron 2 (this paper)	$\mathbf{4.526 \pm 0.066}$

Table 1 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

Tacotron 2

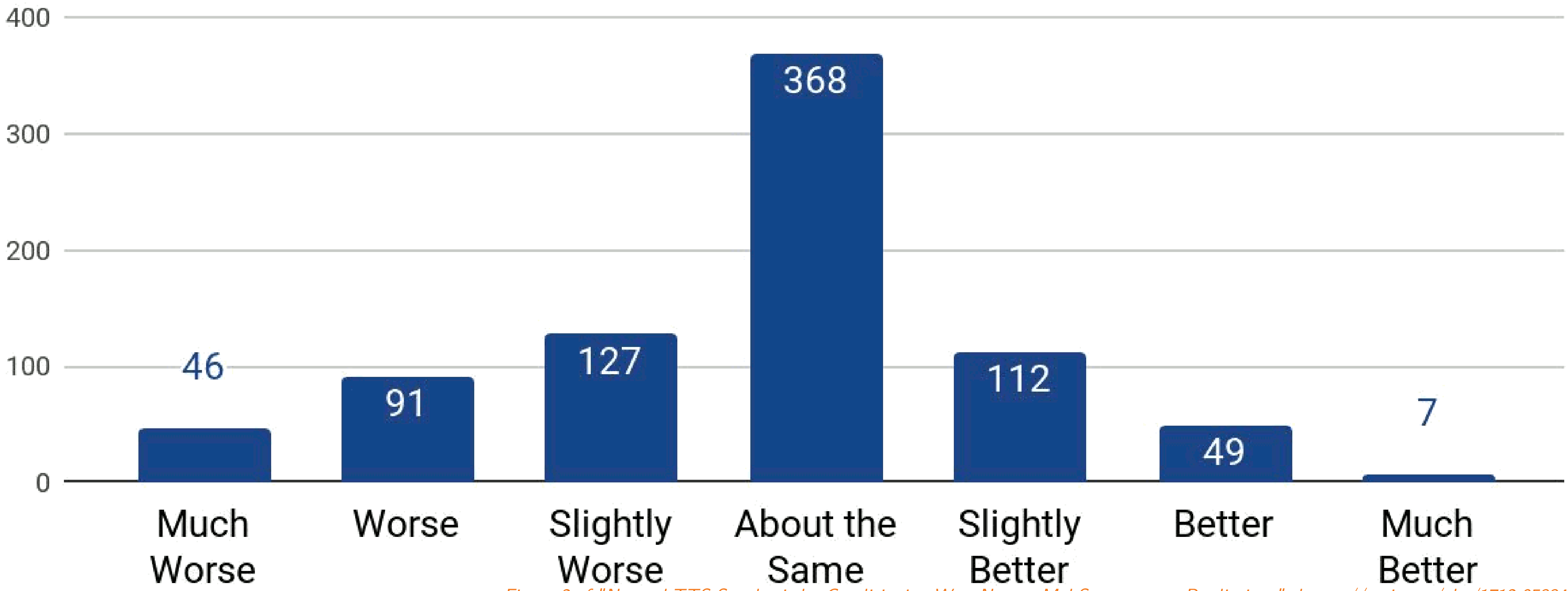


Figure 2 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

You can listen to samples at https://google.github.io/tacotron/publications/tacotron2/

Training	Synthesis: Predicted	Synthesis: Ground truth
Predicted	4.526 ± 0.066	4.449 ± 0.060
Ground truth	4.362 ± 0.066	4.522 ± 0.055

Table 2. Comparison of evaluated MOS for our system when WaveNet trained on predicted/ground truth mel spectrograms are made to synthesize from predicted/ground truth mel spectrograms.

Table 2 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

System	MOS
Tacotron 2 (Linear + G-L)	3.944 ± 0.091
Tacotron 2 (Linear + WaveNet)	4.510 ± 0.054
Tacotron 2 (Mel + WaveNet)	$\mathbf{4.526 \pm 0.066}$

Table 3. Comparison of evaluated MOS for Griffin-Lim vs. WaveNet as a vocoder, and using 1,025-dimensional linear spectrograms vs. 80-dimensional mel spectrograms as conditioning inputs to WaveNet.

Table 3 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

Total layers	Num cycles	Dilation cycle size	Receptive field (samples / ms)	MOS
30	3	10	6,139 / 255.8	4.526 ± 0.066
24	4	6	505 / 21.0	4.547 ± 0.056
12	2	6	253 / 10.5	4.481 ± 0.059
30	30	1	61 / 2.5	3.930 ± 0.076

Table 4. WaveNet with various layer and receptive field sizes.

Table 4 of "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", https://arxiv.org/abs/1712.05884

Neural Turing Machines

So far, all input information was stored either directly in network weights, or in a state of a recurrent network.

However, mammal brains seem to operate with a **working memory** – a capacity for short-term storage of information and its rule-based manipulation.

We can therefore try to introduce an external memory to a neural network. The memory $\boldsymbol{M}$ will be a matrix, where rows correspond to memory cells.

The network will control the memory using a controller which reads from the memory and writes to is. Although the original paper also considered a feed-forward (non-recurrent) controller, usually the controller is a recurrent LSTM network.

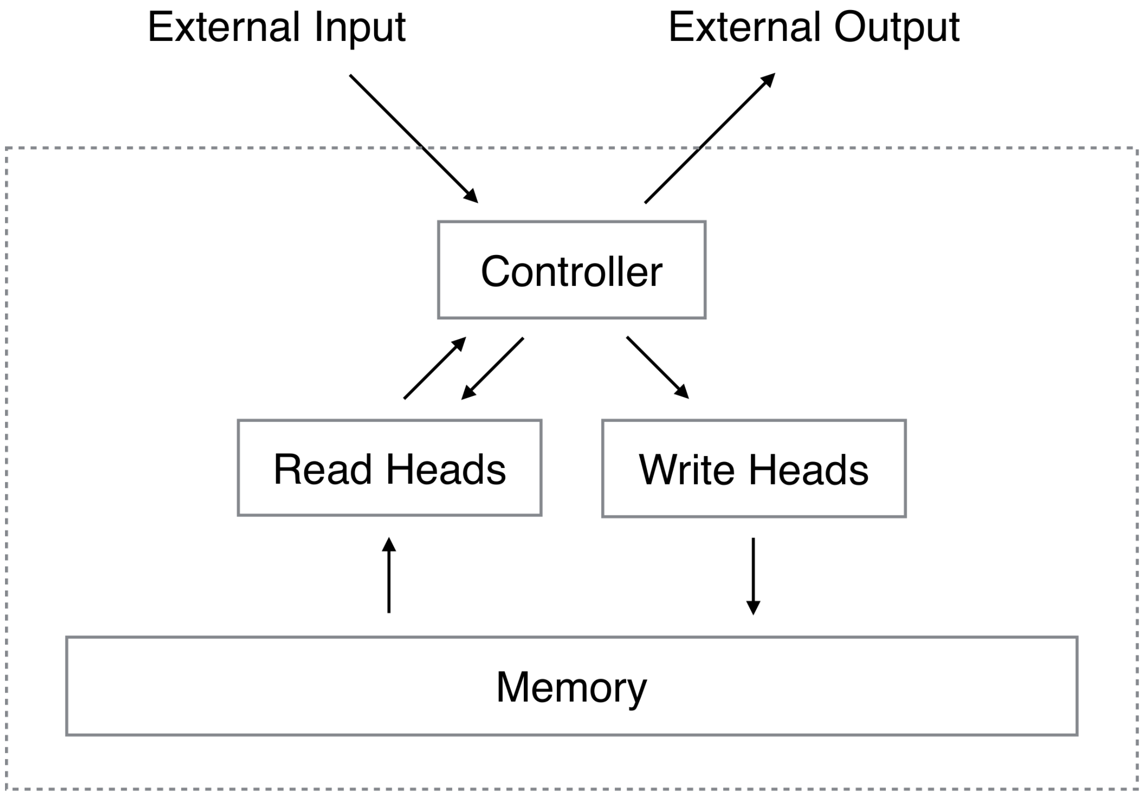


Figure 1 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

Reading

To read the memory in a differentiable way, the controller at time t emits a read distribution $\boldsymbol{w}_t$ over memory locations, and the returned read vector $\boldsymbol{r}_t$ is then

$$\boldsymbol{r}_t = \sum_i w_t(i) \cdot \boldsymbol{M}_t(i).$$

Writing

Writing is performed in two steps – an **erase** followed by an **add**: the controller at time t emits a write distribution $\boldsymbol{w}_t$ over memory locations, together with an *erase vector* $\boldsymbol{e}_t$ and an *add vector* $\boldsymbol{a}_t$. The memory is then updated as

$$\boldsymbol{M}_t(i) = \boldsymbol{M}_{t-1}(i)\big[1 - w_t(i)\boldsymbol{e}_t\big] + w_t(i)\boldsymbol{a}_t.$$

Neural Turing Machine

The addressing mechanism is designed to allow both

- content addressing, and
- location addressing.

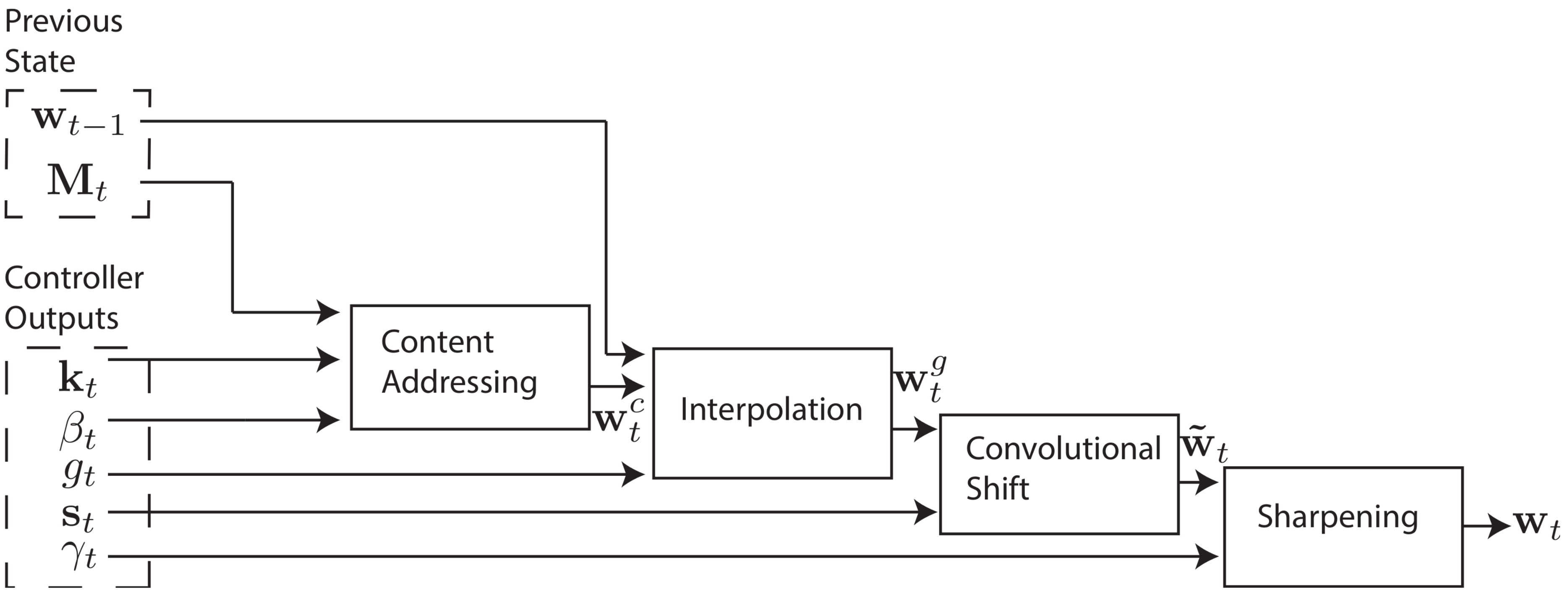


Figure 2 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

Content Addressing

Content addressing starts by the controller emitting the *key vector* $\boldsymbol{k}_t$, which is compared to all memory locations $\boldsymbol{M}_t(i)$, generating a distribution using a $\operatorname{softmax}$ with temperature β_t.

$$w_t^c(i) = \frac{\exp(\beta_t \cdot \operatorname{distance}(\boldsymbol{k}_t, \boldsymbol{M}_t(i))}{\sum_j \exp(\beta_t \cdot \operatorname{distance}(\boldsymbol{k}_t, \boldsymbol{M}_t(j))}$$

The $\operatorname{distance}$ measure is usually the cosine similarity

$$\operatorname{distance}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}^T \boldsymbol{b}}{||\boldsymbol{a}|| \cdot ||\boldsymbol{b}||}.$$

Location-Based Addressing

To allow iterative access to memory, the controller might decide to reuse the memory location from the previous timestep. Specifically, the controller emits an *interpolation gate* g_t and sets

$$\boldsymbol{w}_t^g = g_t \boldsymbol{w}_t^c + (1 - g_t)\boldsymbol{w}_{t-1}.$$

Then, the current weighting may be shifted, i.e., the controller might decide to "rotate" the weights by a small integer. For a given range (the simplest case are only shifts $\{-1, 0, 1\}$), the network emits a $\operatorname{softmax}$ distribution over the shifts, and the weights are then defined using a circular convolution

$$\tilde{w}_t(i) = \sum_j w_t^g(j) s_t(i - j).$$

Finally, not to lose precision over time, the controller emits a *sharpening factor* γ_t, and the final memory location weights are $w_t(i) = \tilde{w}_t(i)^{\gamma_t} / \sum_j \tilde{w}_t(j)^{\gamma_t}$.

Overall Execution

Even if not specified in the original paper, following the DNC paper, the LSTM controller can be implemented as a (potentially deep) LSTM. Assuming R read heads and one write head, the input is $\boldsymbol{x}_t$ and R read vectors $\boldsymbol{r}_{t-1}^1, \ldots, \boldsymbol{r}_{t-1}^R$ from the previous time step, the output of the controller are vectors $(\boldsymbol{\nu}_t, \boldsymbol{\xi}_t)$, and the final output is $\boldsymbol{y}_t = \boldsymbol{\nu}_t + \boldsymbol{W}_r\big[\boldsymbol{r}_t^1, \ldots, \boldsymbol{r}_t^R\big]$. The $\boldsymbol{\xi}_t$ is a concatenation of

$$\boldsymbol{k}_t^1, \beta_t^1, g_t^1, \boldsymbol{s}_t^1, \gamma_t^1, \boldsymbol{k}_t^2, \beta_t^2, g_t^2, \boldsymbol{s}_t^2, \gamma_t^2, \ldots, \boldsymbol{k}_t^w, \beta_t^w, g_t^w, \boldsymbol{s}_t^w, \gamma_t^w, \boldsymbol{e}_t^w, \boldsymbol{a}_t^w.$$

Copy Task

Repeat the same sequence as given on input. Trained with sequences of length up to 20.

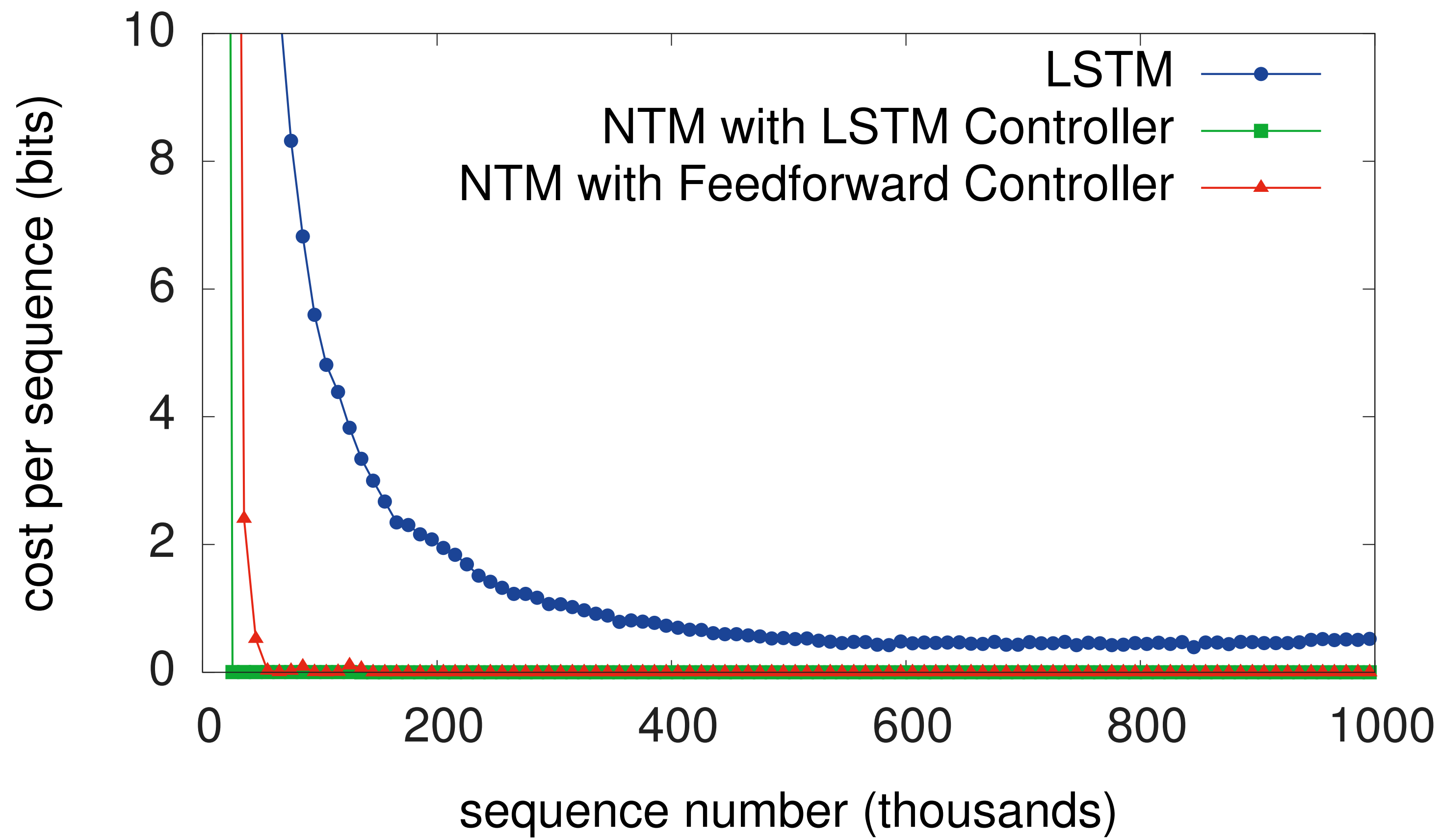


Figure 3 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

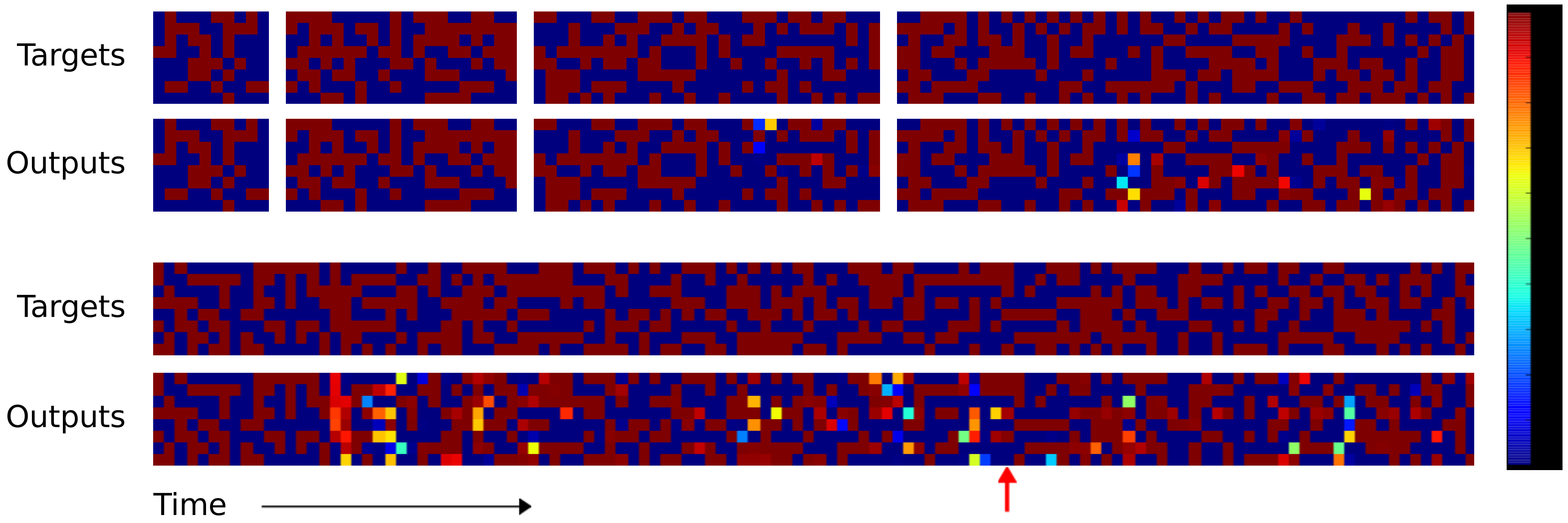


Figure 4: NTM Generalisation on the Copy Task. The four pairs of plots in the top row depict network outputs and corresponding copy targets for test sequences of length 10, 20, 30, and 50, respectively. The plots in the bottom row are for a length 120 sequence. The network was only trained on sequences 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 4 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

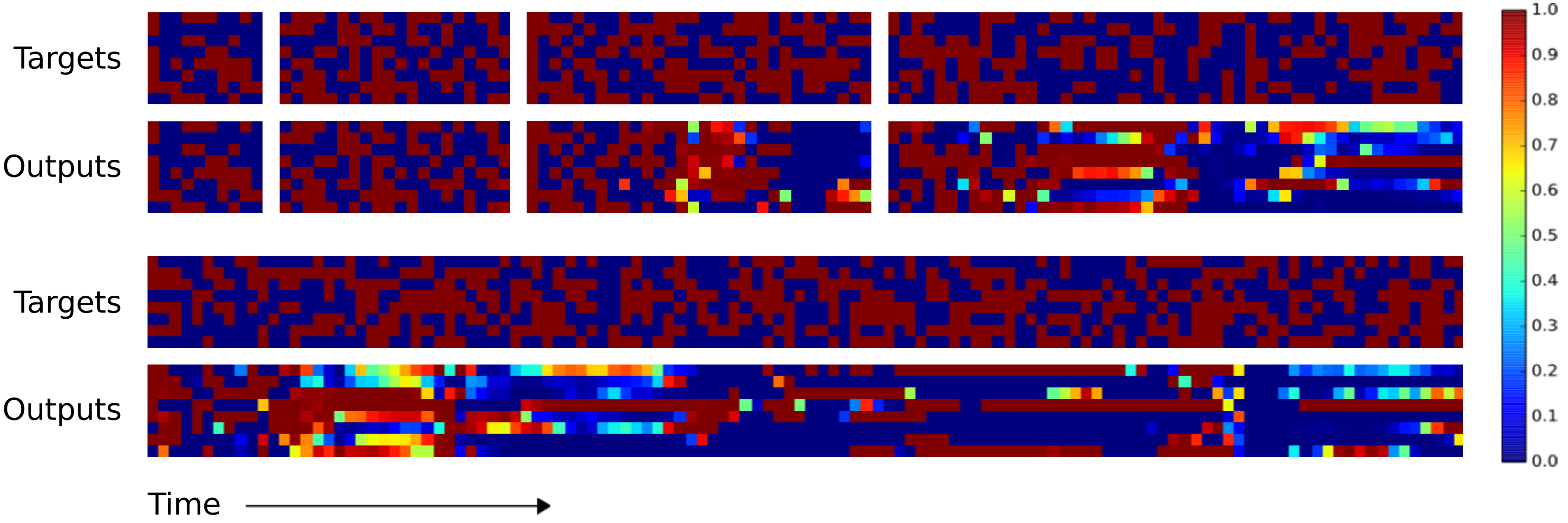


Figure 5: LSTM Generalisation on the Copy Task. The plots show inputs and outputs for the same sequence lengths as Figure 4. Like NTM, LSTM learns to reproduce sequences of up to length 20 almost perfectly. However it clearly fails to generalise to longer sequences. Also note that the length of the accurate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

Figure 5 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

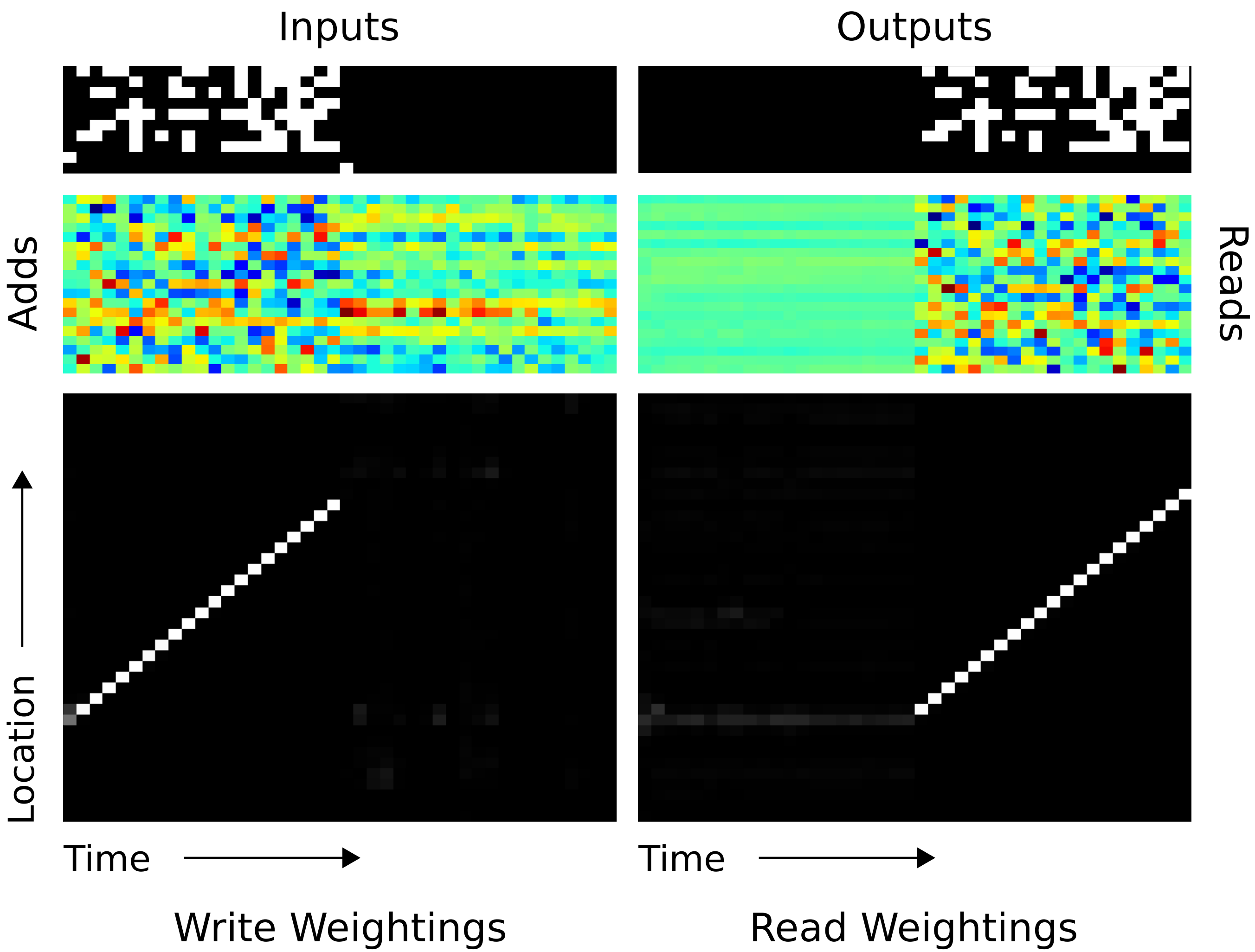


Figure 6 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

Associative Recall

In associative recall, a sequence is given on input, consisting of subsequences of length 3. Then a randomly chosen subsequence is presented on input and the goal is to produce the following subsequence.

Neural Turing Machines

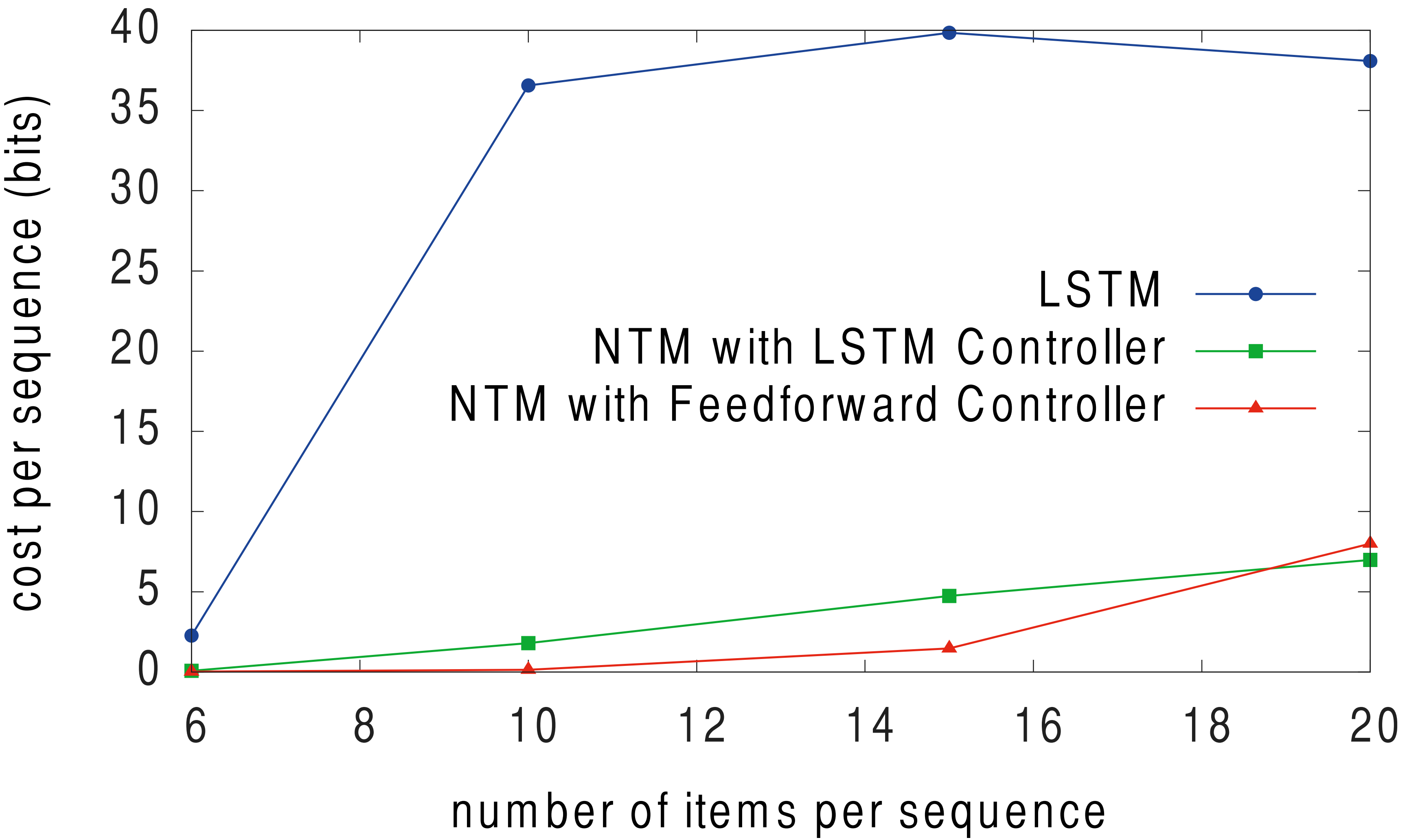


Figure 11 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

Neural Turing Machines

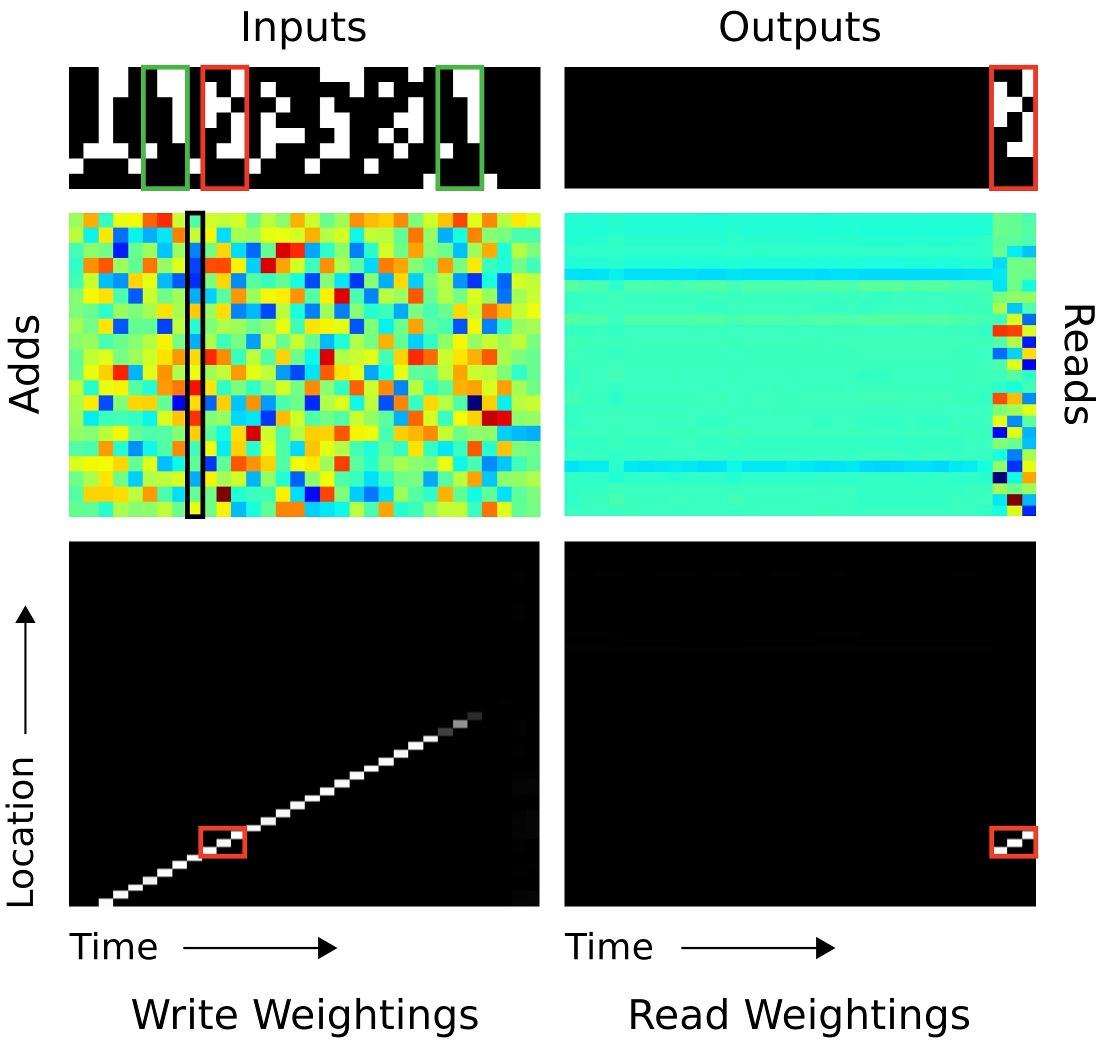


Figure 12 of "Neural Turing Machines", https://arxiv.org/abs/1410.5401

NTM was later extended to a Differentiable Neural Computer.

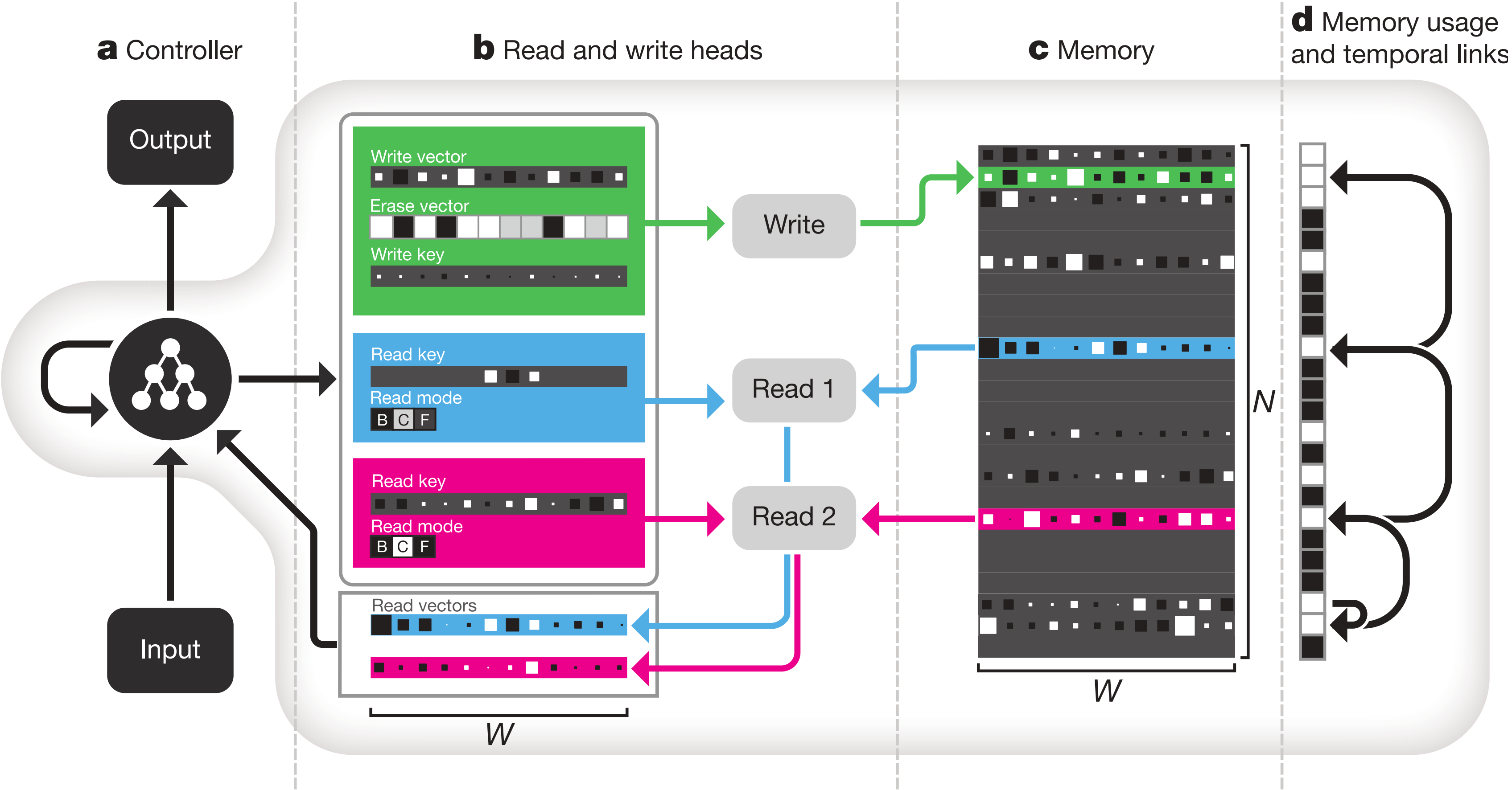


Figure 1 of "Hybrid computing using a neural network with dynamic external memory", https://www.nature.com/articles/nature20101

Differentiable Neural Computer

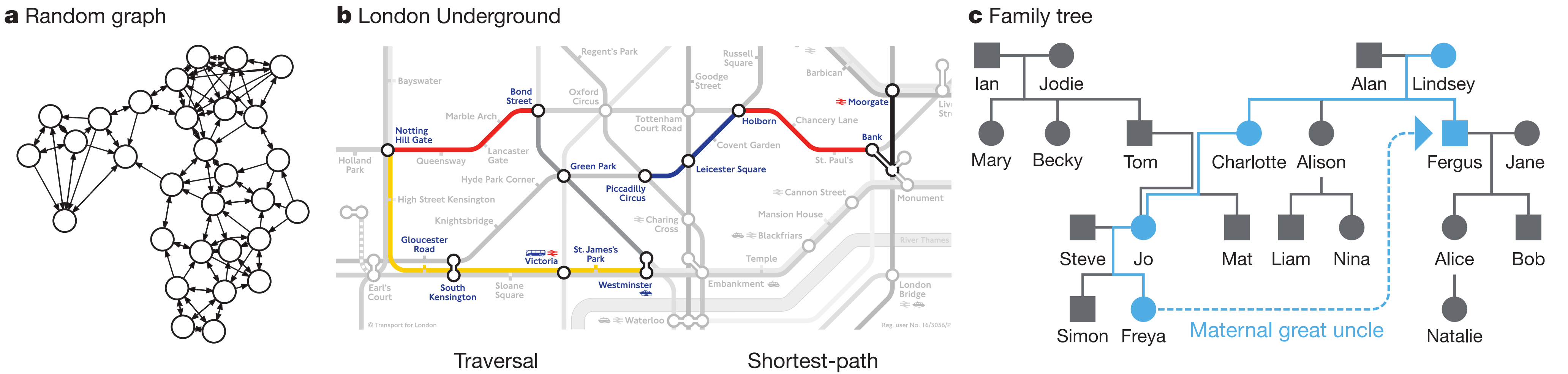
The DNC contains multiple read heads and one write head.

The controller is a deep LSTM network, with input at time t being the current input $\boldsymbol{x}_t$ and R read vectors $\boldsymbol{r}^1_{t-1}, \ldots, \boldsymbol{r}^R_{t-1}$ from previous time step. The output of the controller are vectors $(\boldsymbol{\nu}_t, \boldsymbol{\xi}_t)$, and the final output is $\boldsymbol{y}_t = \boldsymbol{\nu}_t + W_r\big[\boldsymbol{r}^1_t, \ldots, \boldsymbol{r}^R_t\big]$. The $\boldsymbol{\xi}_t$ is a concatenation of parameters for read and write heads (keys, gates, sharpening parameters, …).

In DNC, the usage of every memory location is tracked, which enables performing dynamic allocation – at each time step, a cell with least usage can be allocated.

Furthermore, for every memory location, we track which memory location was written to previously and subsequently, allowing to recover sequences in the order in which it was written, independently on the real indices used.

The write weighting is defined as a weighted combination of the allocation weighting and write content weighting, and read weighting is computed as a weighted combination of read content weighting, previous write weighting, and subsequent write weighting.



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:
(BondSt, _, Central),
(_, _, Circle), (_, _, Circle),
(_, _, Circle), (_, _, Circle),
(_, _, Jubilee), (_, _, Jubilee),

Answer:
(BondSt, NottingHillGate, Central)
(NottingHillGate, GloucesterRd, Circle)
⋮
(Westminster, GreenPark, Jubilee)
(GreenPark, BondSt, 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)
(Nina, Alison, Mother)
(Lindsey, Fergus, Son1)
⋮
(Bob, Jane, Mother)
(Natalie, Alice, Mother)
(Mary, Ian, Father)
(Jane, Alice, Daughter1)
(Mat, Charlotte, Mother)

54 edges in total

Inference question:
(Freya, _, MaternalGreatUncle)

Answer:
(Freya, Fergus, MaternalGreatUncle)

Figure 2 of "Hybrid computing using a neural network with dynamic external memory", https://www.nature.com/articles/nature20101

Differentiable Neural Computer

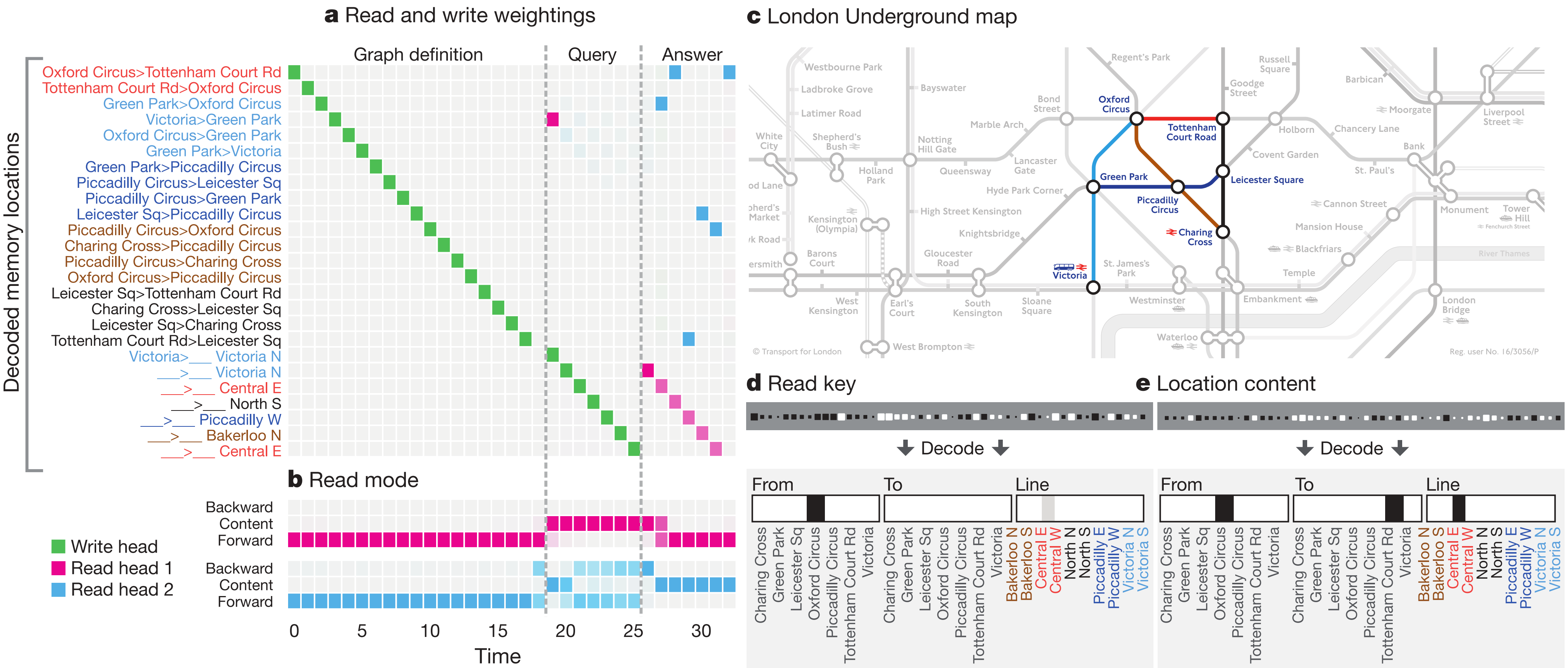
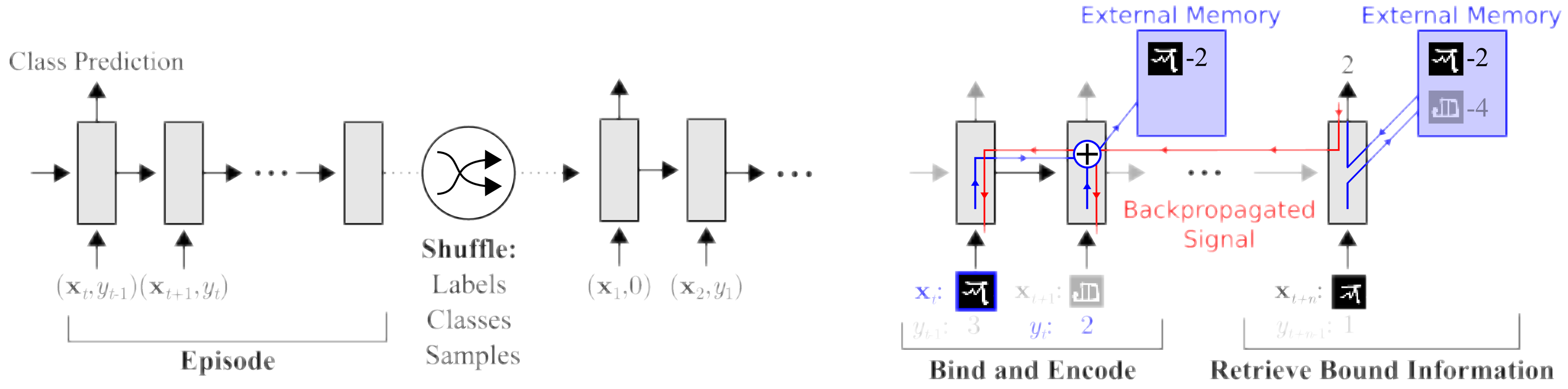


Figure 3 of "Hybrid computing using a neural network with dynamic external memory", https://www.nature.com/articles/nature20101

External memory can be also utilized for **learning to learn**. Consider a network, which should learn classification into a user-defined hierarchy by observing ideally a small number of samples.

Apart from finetuning the model and storing the information in the *weights*, an alternative is to store the samples in **external memory**. Therefore, the model learns how to store the data and access it efficiently, which allows it to learn without changing its weights.



(a) Task setup (b) Network strategy

Figure 1 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065

Memory-augmented NNs

$$K\big(\mathbf{k}_t, \mathbf{M}_t(i)\big) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\parallel \mathbf{k}_t \parallel \parallel \mathbf{M}_t(i) \parallel}, \tag{2}$$

which is used to produce a read-weight vector, $\mathbf{w}_t^r$, with elements computed according to a softmax:

$$w_t^r(i) \leftarrow \frac{\exp\big(K\big(\mathbf{k}_t, \mathbf{M}_t(i)\big)\big)}{\sum_j \exp\big(K\big(\mathbf{k}_t, \mathbf{M}_t(j)\big)\big)}. \tag{3}$$

A memory, $\mathbf{r}_t$, is retrieved using this weight vector:

$$\mathbf{r}_t \leftarrow \sum_i w_t^r(i)\mathbf{M}_t(i). \tag{4}$$

Page 3 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065

$$\mathbf{w}_t^u \leftarrow \gamma \mathbf{w}_{t-1}^u + \mathbf{w}_t^r + \mathbf{w}_t^w. \tag{5}$$

Here, γ is a decay parameter and $\mathbf{w}_t^r$ is computed as in (3). The *least-used* weights, $\mathbf{w}_t^{lu}$, for a given time-step can then be computed using $\mathbf{w}_t^u$. First, we introduce the notation $m(\mathbf{v}, n)$ to denote the n^{th} smallest element of the vector $\mathbf{v}$. Elements of $\mathbf{w}_t^{lu}$ are set accordingly:

$$w_t^{lu}(i) = \begin{cases} 0 & \text{if } w_t^u(i) > m(\mathbf{w}_t^u, n) \\ 1 & \text{if } w_t^u(i) \leq m(\mathbf{w}_t^u, n) \end{cases}, \tag{6}$$

where n is set to equal the number of reads to memory. To obtain the write weights $\mathbf{w}_t^w$, a learnable sigmoid gate parameter is used to compute a convex combination of the previous read weights and previous least-used weights:

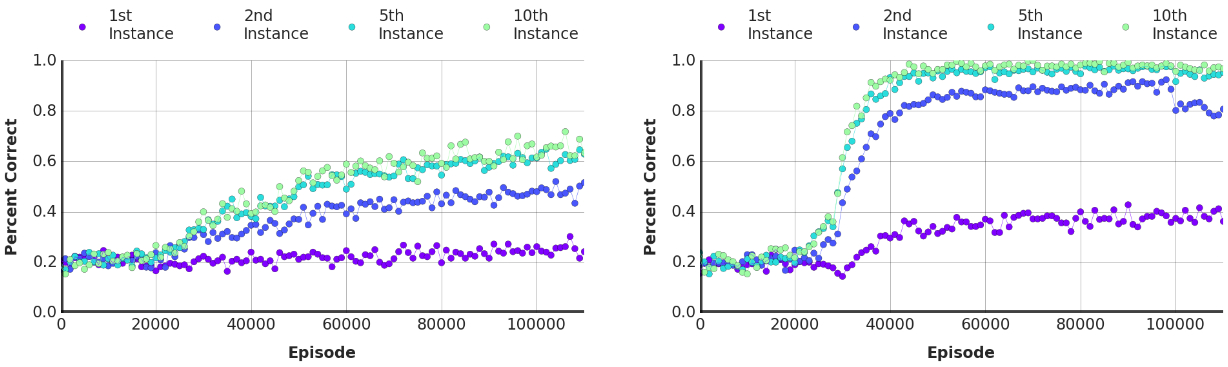
$$\mathbf{w}_t^w \leftarrow \sigma(\alpha)\mathbf{w}_{t-1}^r + (1 - \sigma(\alpha))\mathbf{w}_{t-1}^{lu}. \tag{7}$$

Here, $\sigma(\cdot)$ is a sigmoid function, $\frac{1}{1+e^{-x}}$, and α is a scalar gate parameter to interpolate between the weights. Prior to writing to memory, the least used memory location is computed from $\mathbf{w}_{t-1}^u$ and is set to zero. Writing to memory then occurs in accordance with the computed vector of write weights:

$$\mathbf{M}_t(i) \leftarrow \mathbf{M}_{t-1}(i) + w_t^w(i)\mathbf{k}_t, \forall i \tag{8}$$

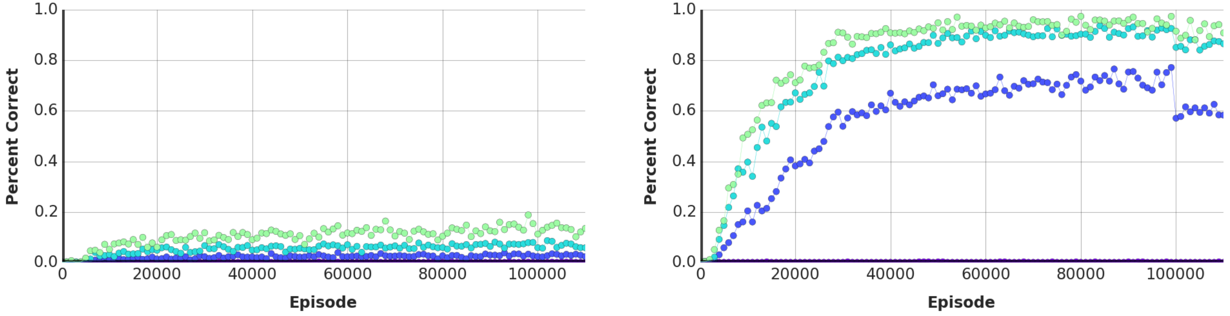
Page 4 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065

Memory-augmented NNs



(a) LSTM, five random classes/episode, one-hot vector labels

(b) MANN, five random classes/episode, one-hot vector labels



(c) LSTM, fifteen classes/episode, five-character string labels

(d) MANN, fifteen classes/episode, five-character string labels

Figure 2. Omniglot classification. The network was given either five (a-b) or up to fifteen (c-d) random classes per episode, which were of length 50 or 100 respectively. Labels were one-hot vectors in (a-b), and five-character strings in (c-d). In (b), first instance accuracy is above chance, indicating that the MANN is performing "educated guesses" for new classes based on the classes it has already seen and stored in memory. In (c-d), first instance accuracy is poor, as is expected, since it must make a guess from 3125 random strings. Second instance accuracy, however, approaches 80% during training for the MANN (d). At the 100,000 episode mark the network was tested, without further learning, on distinct classes withheld from the training set, and exhibited comparable performance.

Figure 2 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065

Table 1. Test-set classification accuracies for humans compared to machine algorithms trained on the Omniglot dataset, using one-hot encodings of labels and five classes presented per episode.

Model	Instance (% Correct) 1st	2nd	3rd	4th	5th	10th
Human	34.5	57.3	70.1	71.8	81.4	92.4
Feedforward	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	**36.4**	**82.8**	**91.0**	**92.6**	**94.9**	**98.1**

Table 1 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065

Table 2. Test-set classification accuracies for various architectures on the Omniglot dataset after 100000 episodes of training, using five-character-long strings as labels. See the supplemental information for an explanation of 1st instance accuracies for the kNN classifier.

Model	Controller	# of Classes	Instance (% Correct) 1st	2nd	3rd	4th	5th	10th
kNN (raw pixels)	–	15	0.5	18.7	23.3	26.5	29.1	37.0
kNN (deep features)	–	15	0.4	32.7	41.2	47.1	50.6	60.0
Feedforward	–	15	0.0	0.1	0.0	0.0	0.0	0.0
LSTM	–	15	0.0	2.2	2.9	4.3	5.6	12.7
MANN (LRUA)	Feedforward	15	0.1	12.8	22.3	28.8	32.2	43.4
MANN (LRUA)	LSTM	15	0.1	**62.6**	**79.3**	**86.6**	**88.7**	**95.3**
MANN (NTM)	LSTM	15	0.0	35.4	61.2	71.7	77.7	88.4

Table 2 of "One-shot learning with Memory-Augmented Neural Networks", https://arxiv.org/abs/1605.06065