Bayesian Sparsification of Gated Recurrent Neural Networks

Ekaterina Lobacheva*[1] Nadezhda Chirkova*[1] Dmitry Vetrov[1,2]

[1]Higher School of Economics, Samsung-HSE Laboratory
[2]Samsung AI Center, Moscow, Russia

NATIONAL RESEARCH UNIVERSITY

SAMSUNG Research

p(B|A)yesgroup.ru

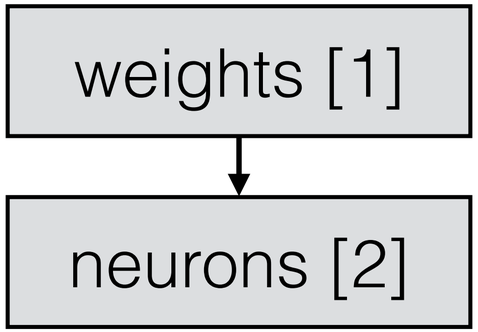
Problem setting

Goal: learn structurally sparse gated RNNs (LSTM, GRU…)

Problem setting

Goal: learn structurally sparse gated RNNs (LSTM, GRU…)

Which structure units to sparsify
in vanilla RNN?

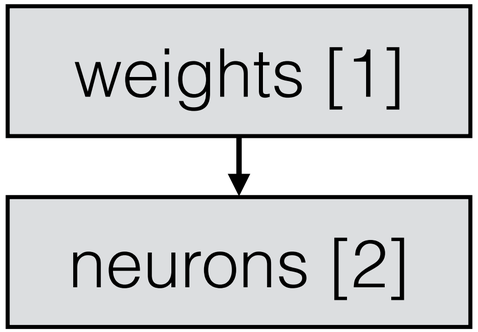


[1] Narang et al. Exploring sparsity in recurrent neural networks. ICLR 2017
[2] Wen, et al. Learning intrinsic sparse structures within long short-term memory. ICLR 2018

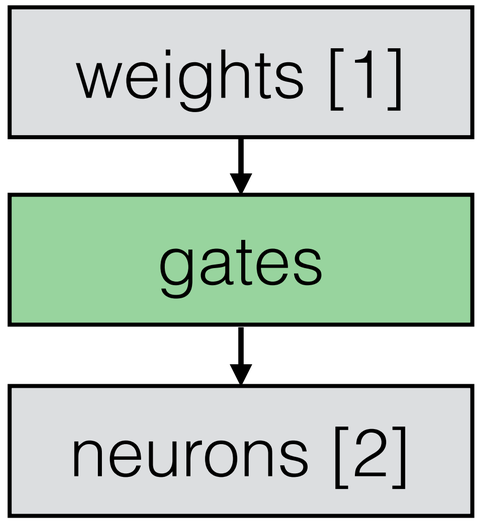
Problem setting

Goal: learn structurally sparse gated RNNs (LSTM, GRU…)

Which structure units to sparsify
in vanilla RNN?



in gated RNN?

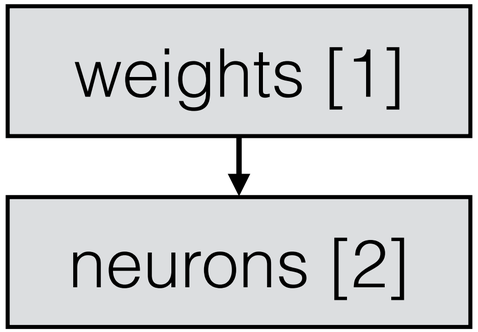


[1] Narang et al. Exploring sparsity in recurrent neural networks. ICLR 2017
[2] Wen, et al. Learning intrinsic sparse structures within long short-term memory. ICLR 2018

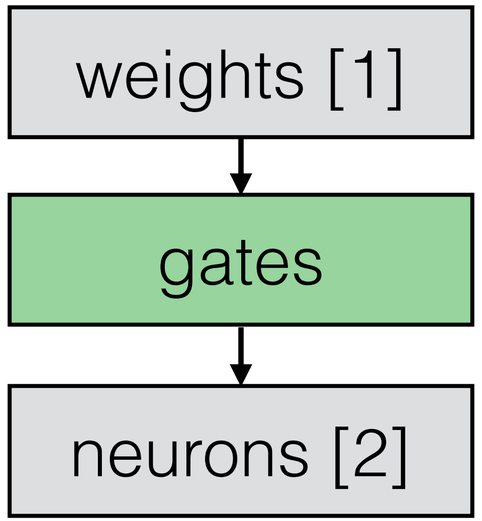
Problem setting

Goal: learn structurally sparse gated RNNs (LSTM, GRU…)

Which structure units to sparsify
in vanilla RNN?



in gated RNN?



Main idea: introduce intermediate level of sparsification — gates

[1] Narang et al. Exploring sparsity in recurrent neural networks. ICLR 2017
[2] Wen, et al. Learning intrinsic sparse structures within long short-term memory. ICLR 2018

Plan of the talk

1. **Idea**: introduce intermediate level of sparsification in RNN — gates

2. **Implementation** of the idea in Bayesian sparsification framework

3. **Results**: better compression and interpretable resulting gates structure

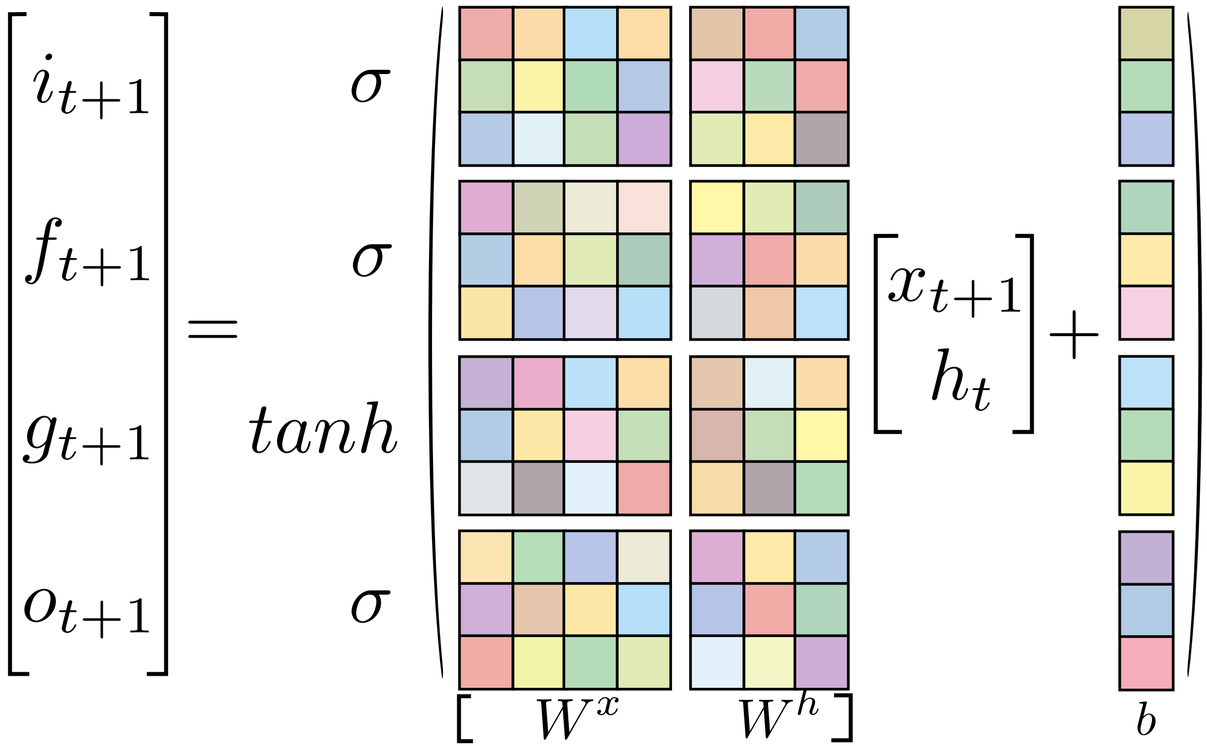
Plan of the talk

1. **Idea**: introduce intermediate level of sparsification in RNN — gates

2. **Implementation** of the idea in Bayesian sparsification framework

3. **Results**: better compression and interpretable resulting gates structure

LSTM

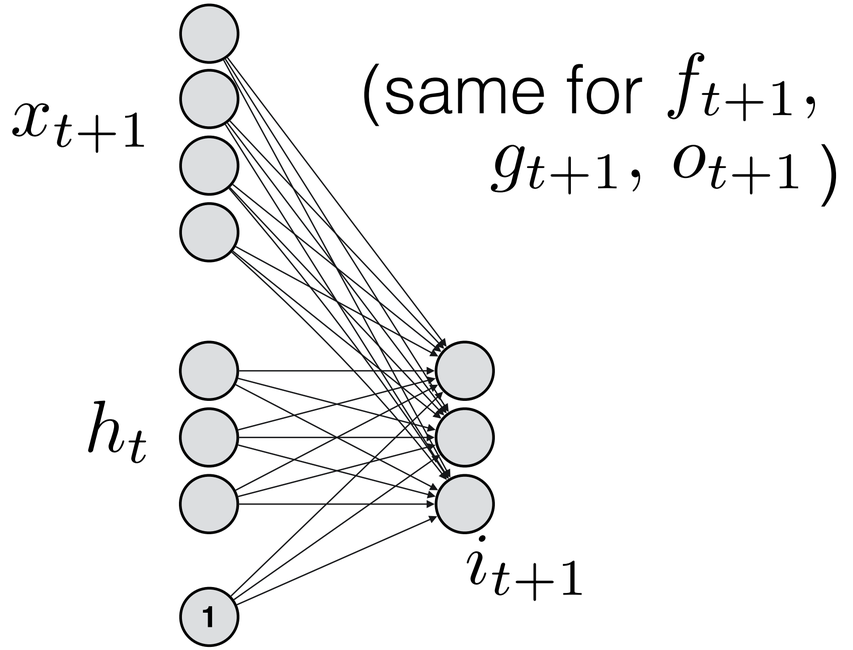


$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$

$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

LSTM

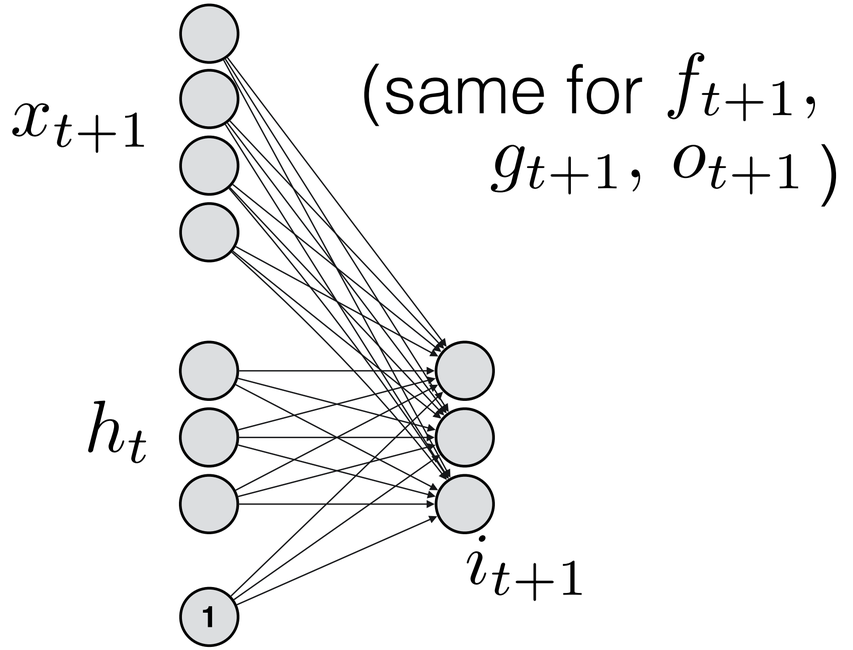
$$\begin{bmatrix} i_{t+1} \\ f_{t+1} \\ g_{t+1} \\ o_{t+1} \end{bmatrix} = \begin{matrix} \sigma \\ \sigma \\ tanh \\ \sigma \end{matrix} \left(\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & W^x & & W^h \end{bmatrix} \begin{bmatrix} x_{t+1} \\ h_t \end{bmatrix} + \begin{bmatrix} \\ \\ \\ \\ b \end{bmatrix} \right)$$



$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

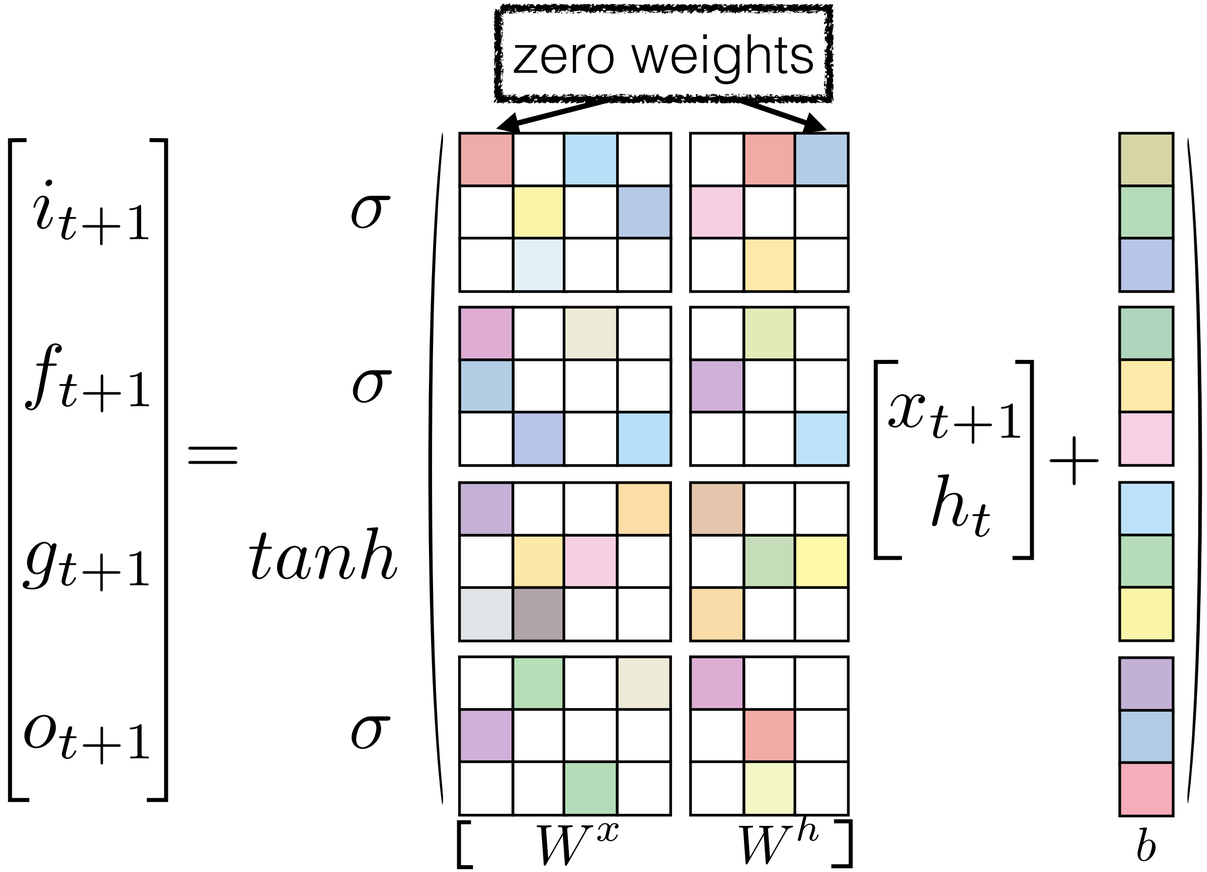
LSTM

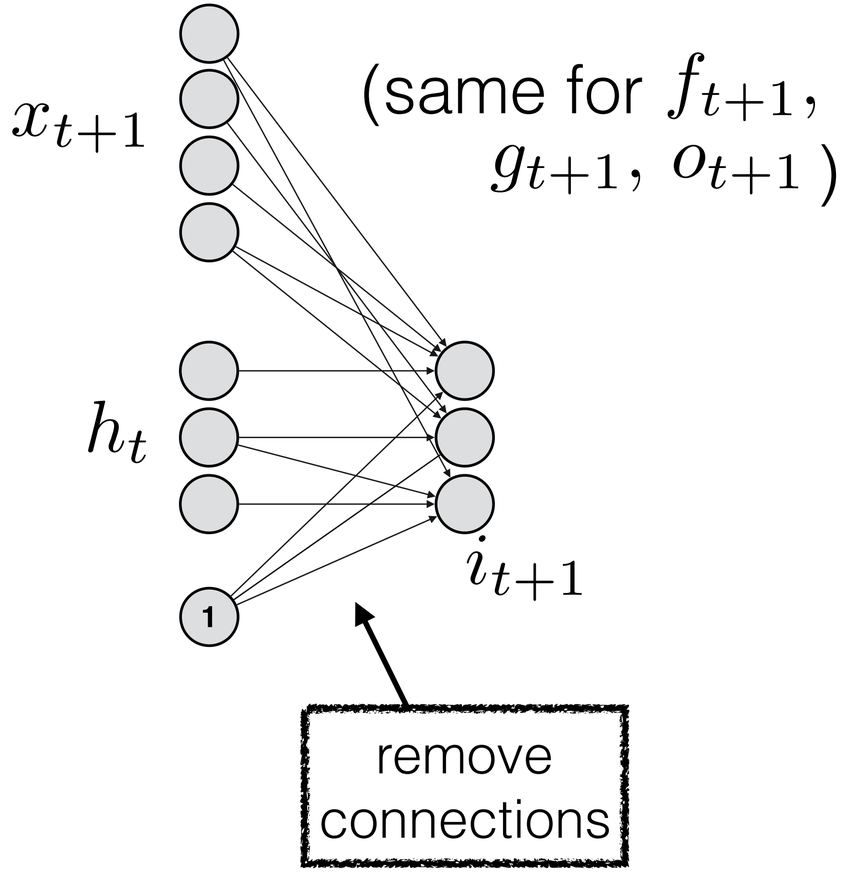
$$\begin{bmatrix} i_{t+1} \\ f_{t+1} \\ g_{t+1} \\ o_{t+1} \end{bmatrix} = \begin{matrix} \sigma \\ \sigma \\ tanh \\ \sigma \end{matrix} \left(\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & W^x & & W^h \end{bmatrix} \begin{bmatrix} x_{t+1} \\ h_t \end{bmatrix} + \begin{bmatrix} \\ \\ \\ \\ b \end{bmatrix} \right)$$



$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

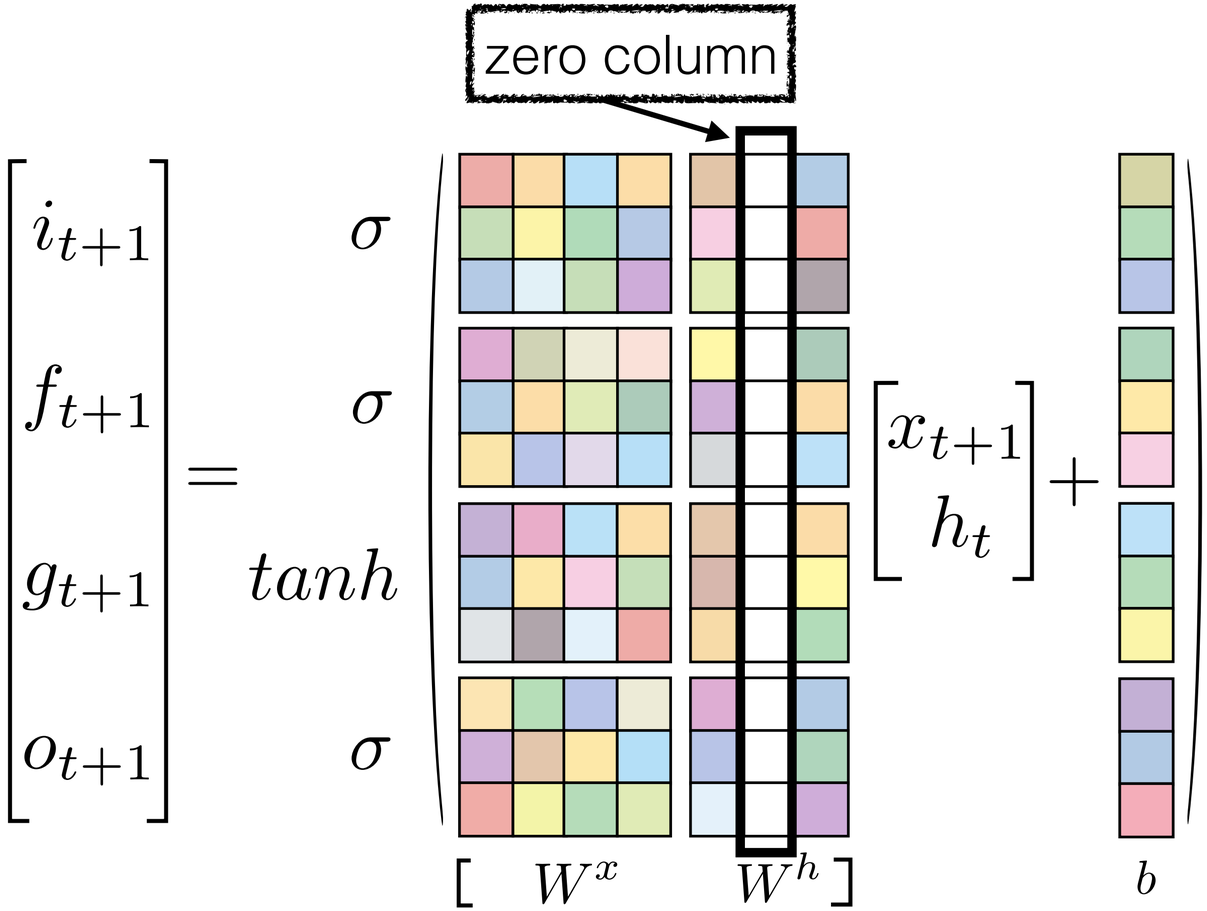
LSTM: remove individual connections



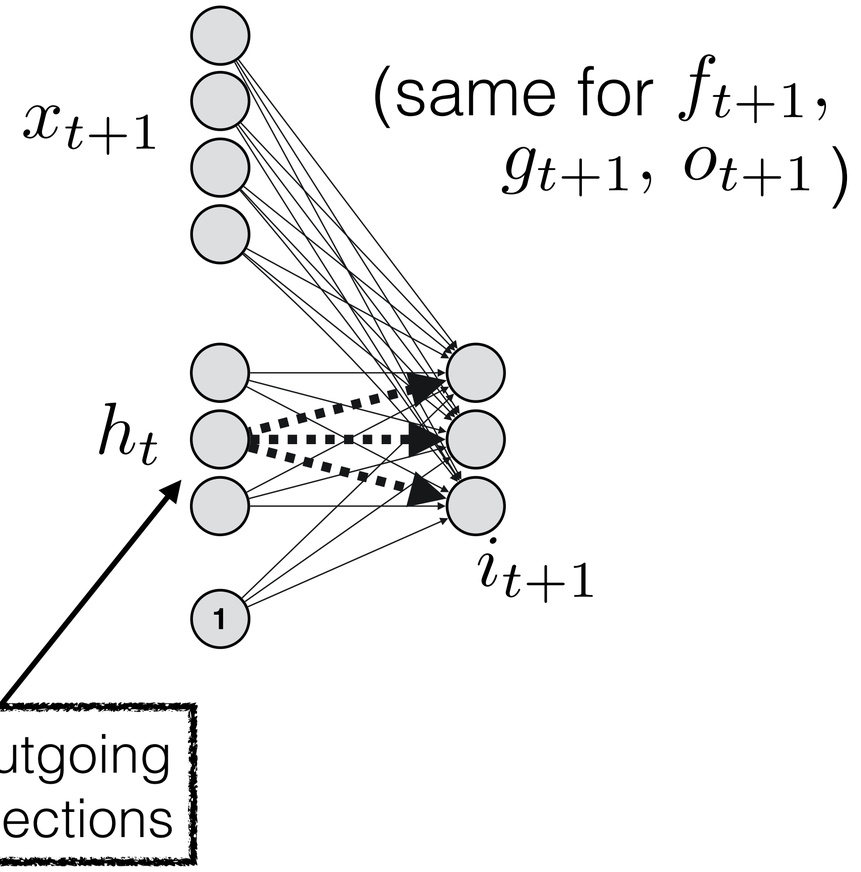


$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

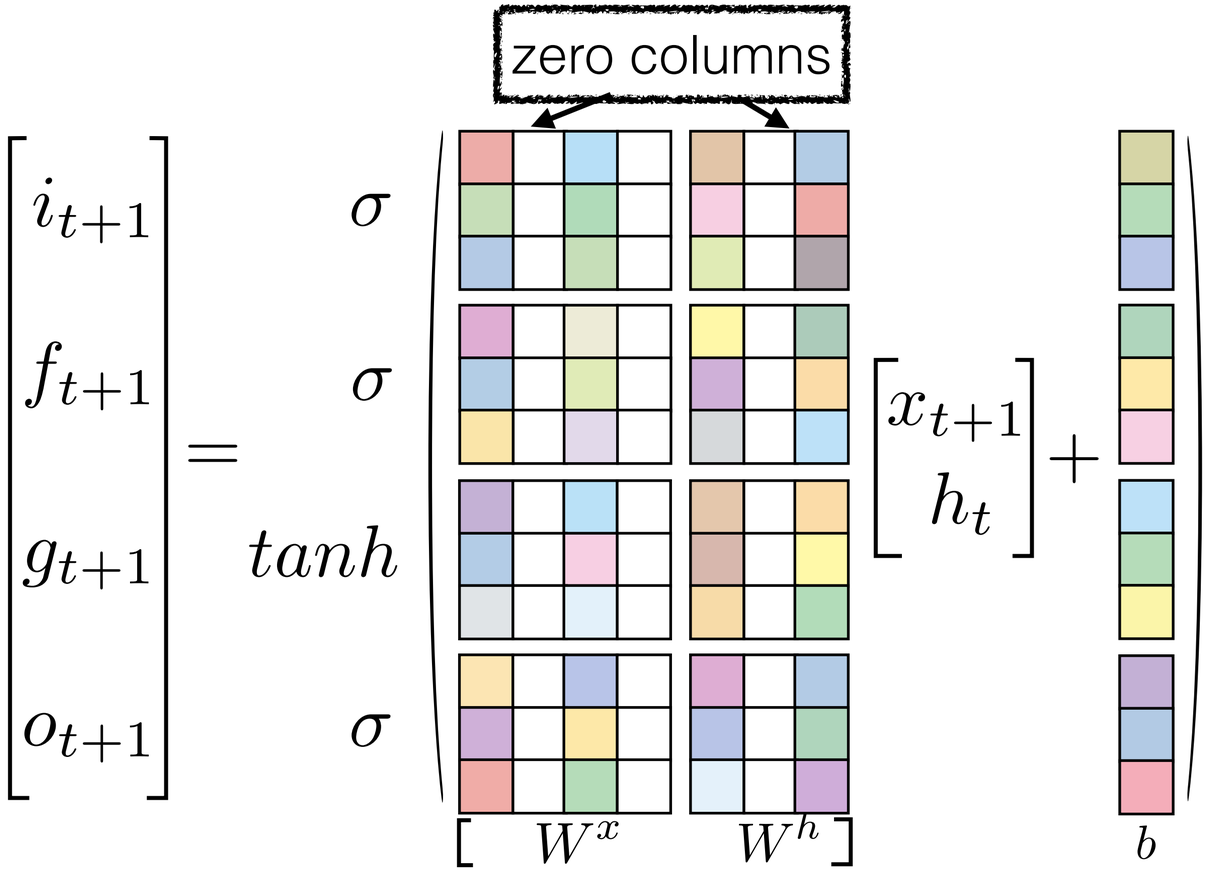
LSTM: remove neurons



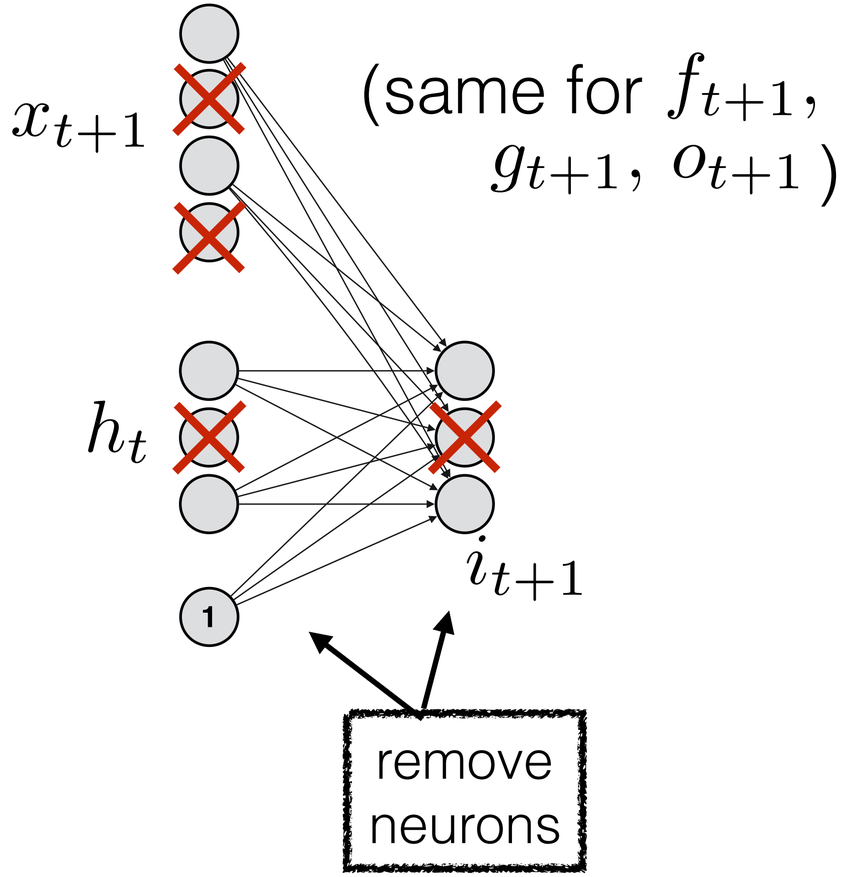
$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$



LSTM: remove neurons



$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

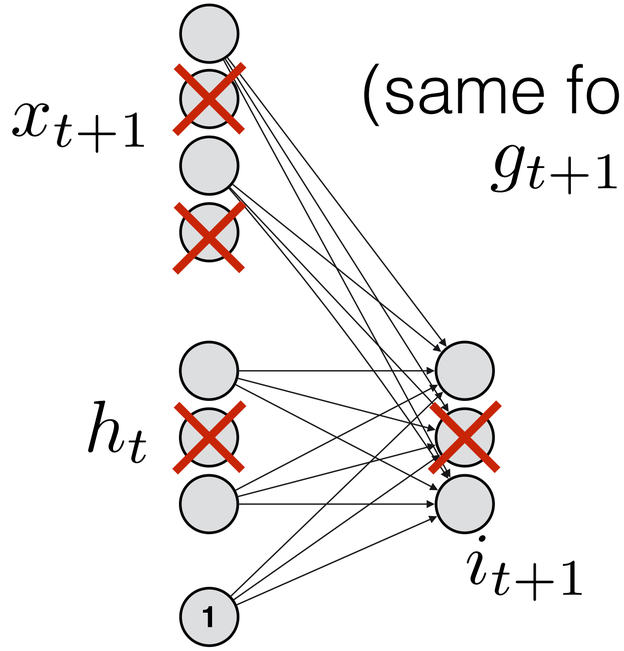


LSTM: remove neurons

$$\begin{bmatrix} i_{t+1} \\ f_{t+1} \\ g_{t+1} \\ o_{t+1} \end{bmatrix} = \begin{matrix} \sigma \\ \sigma \\ tanh \\ \sigma \end{matrix} \left(\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} x_{t+1} \\ h_t \end{bmatrix} + \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \right)$$

$$\begin{bmatrix} W^x & W^h \end{bmatrix} \qquad b$$

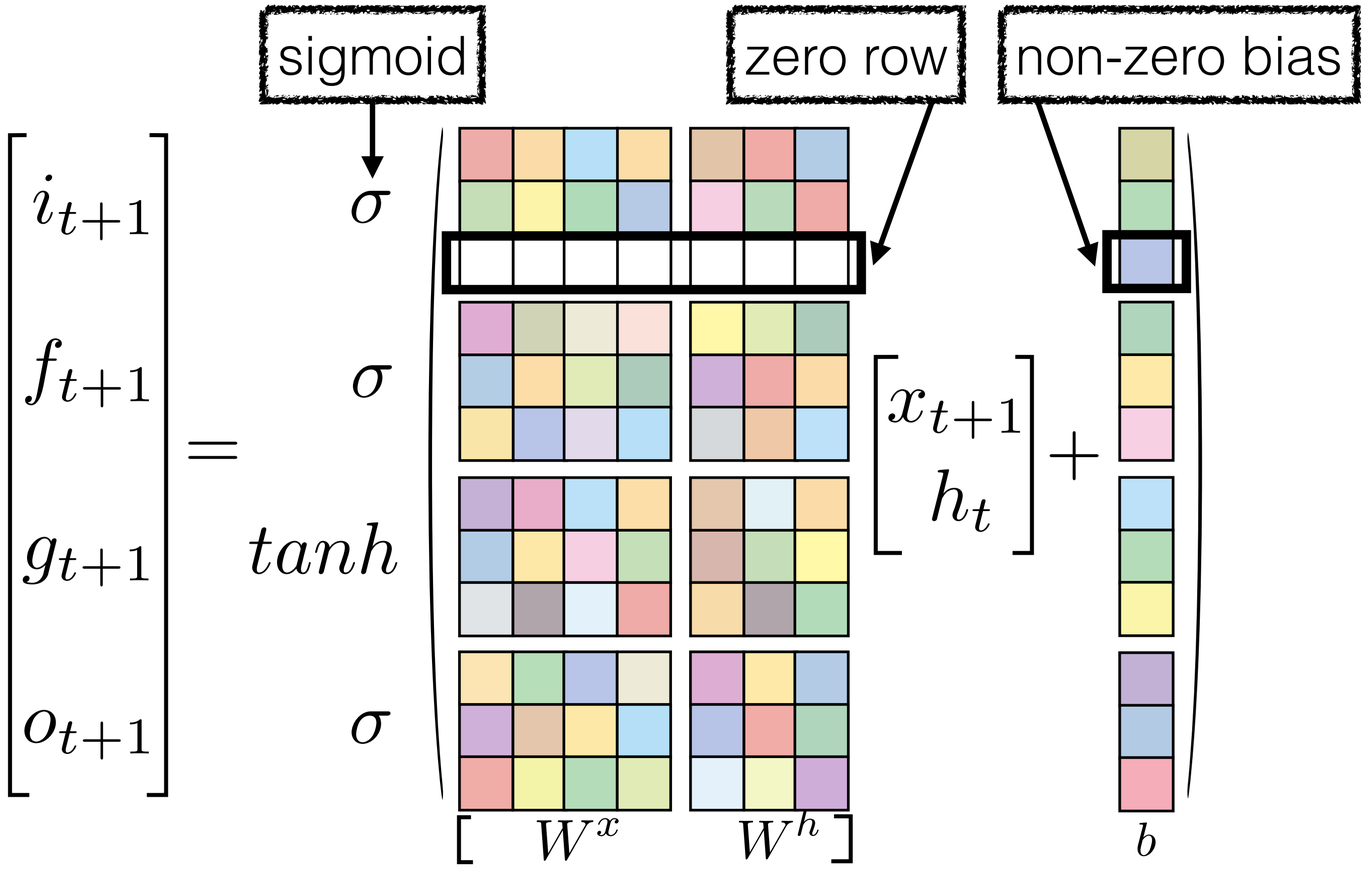
$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

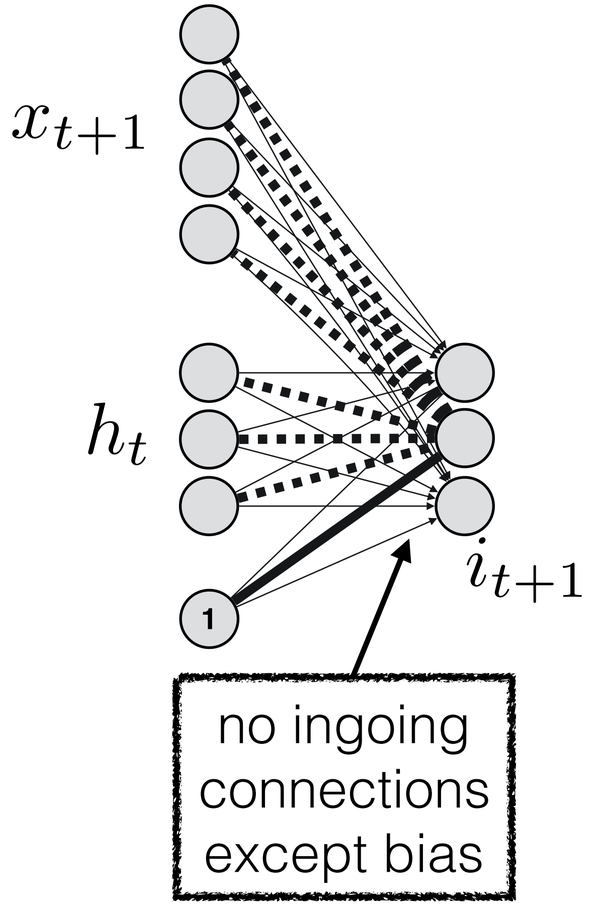


(same for f_{t+1}, g_{t+1}, o_{t+1})

there are other ways to group weights to remove neurons

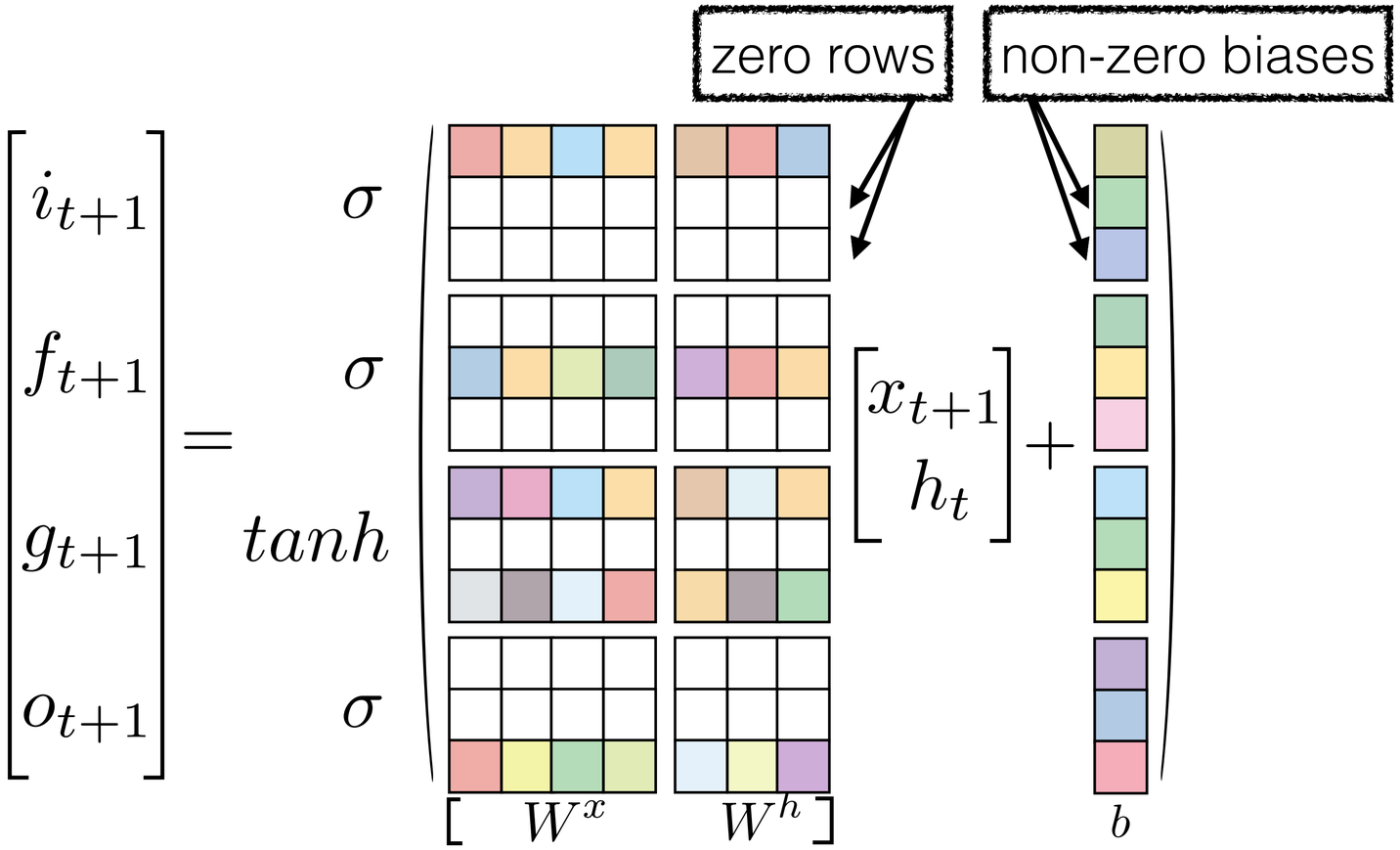
LSTM: make gates constant

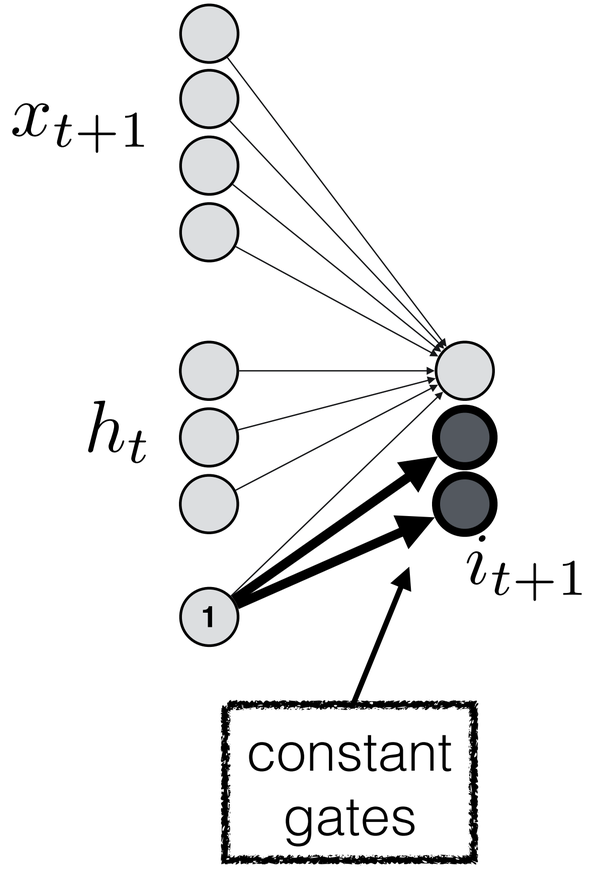




$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

LSTM: make gates constant

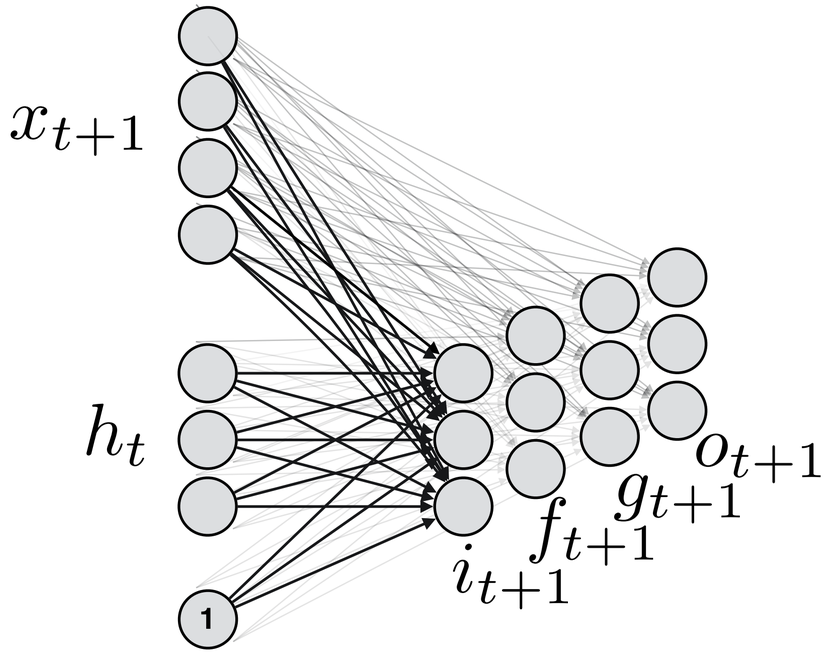




$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot g_{t+1}$$
$$h_{t+1} = o_{t+1} \odot g_{t+1}$$

Putting everything together

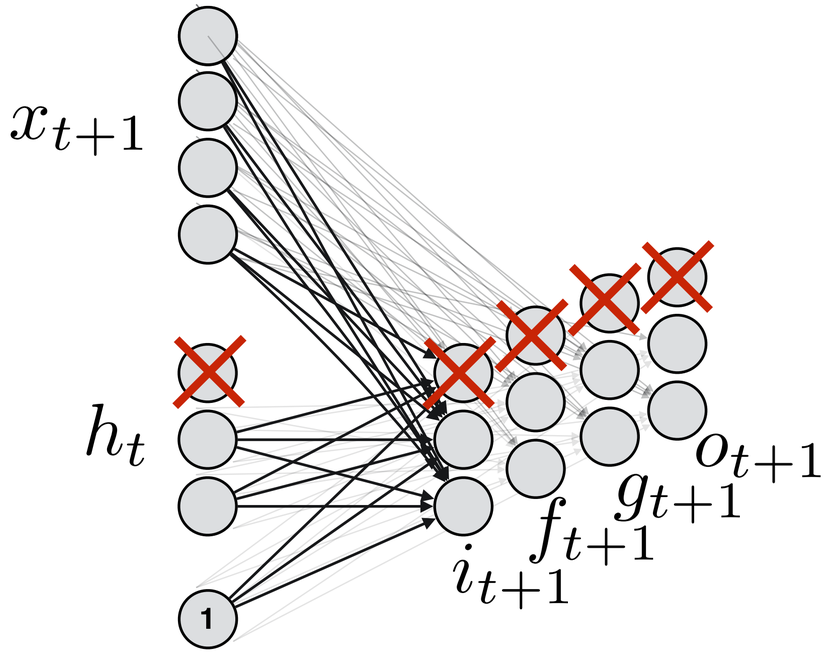
LSTM:



Levels of sparsification:

Putting everything together

LSTM:

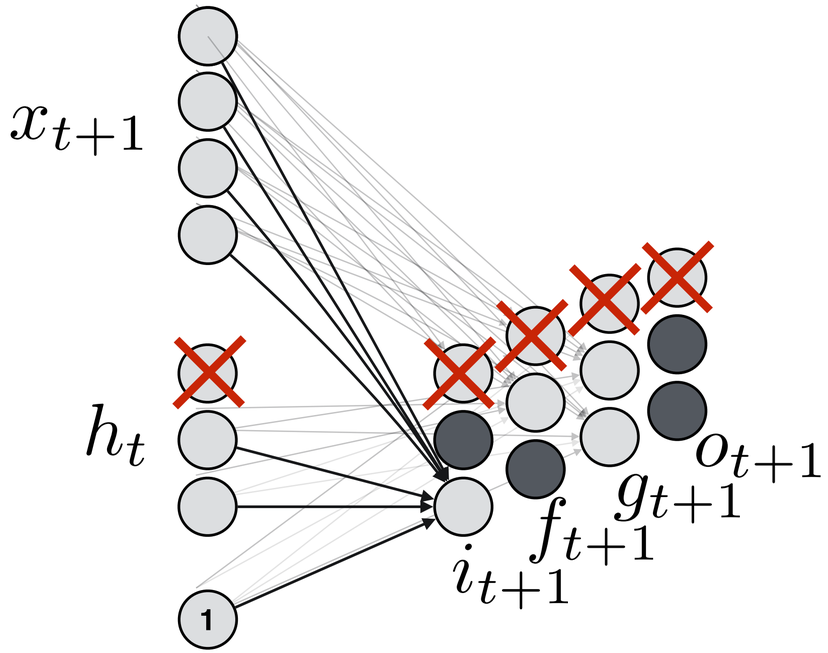


Levels of sparsification:

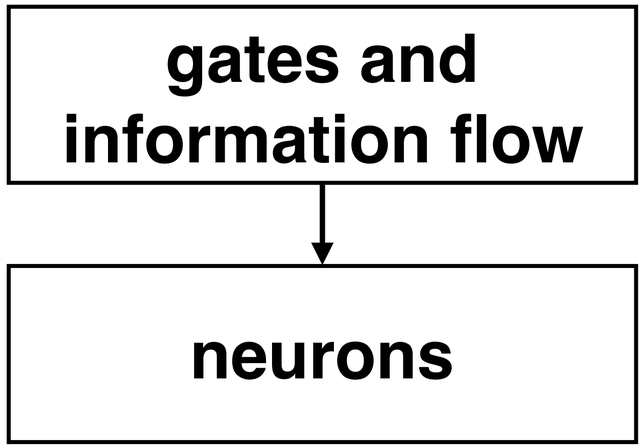
neurons

Putting everything together

LSTM:

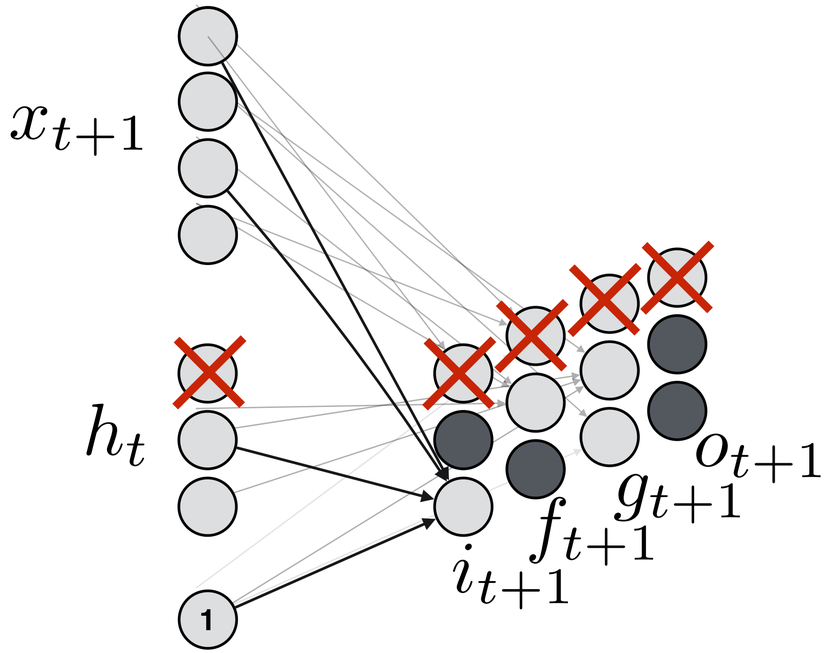


Levels of sparsification:

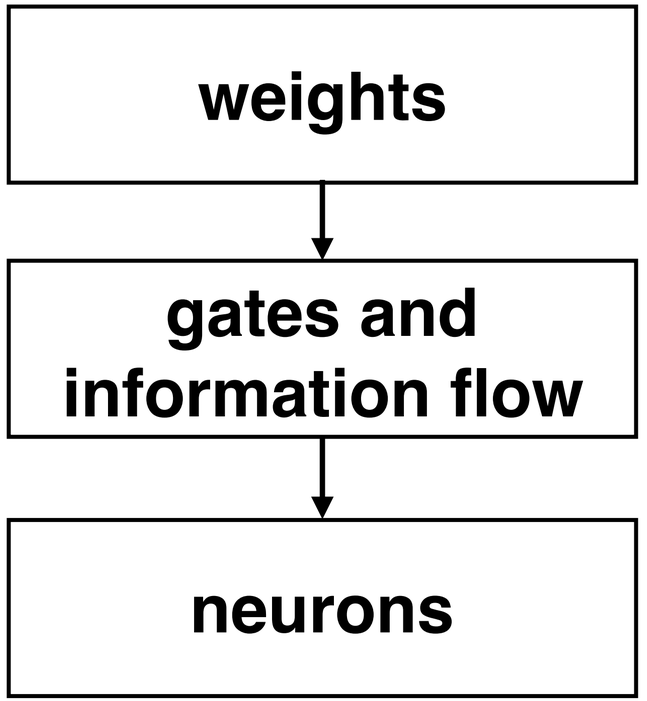


Putting everything together

LSTM:



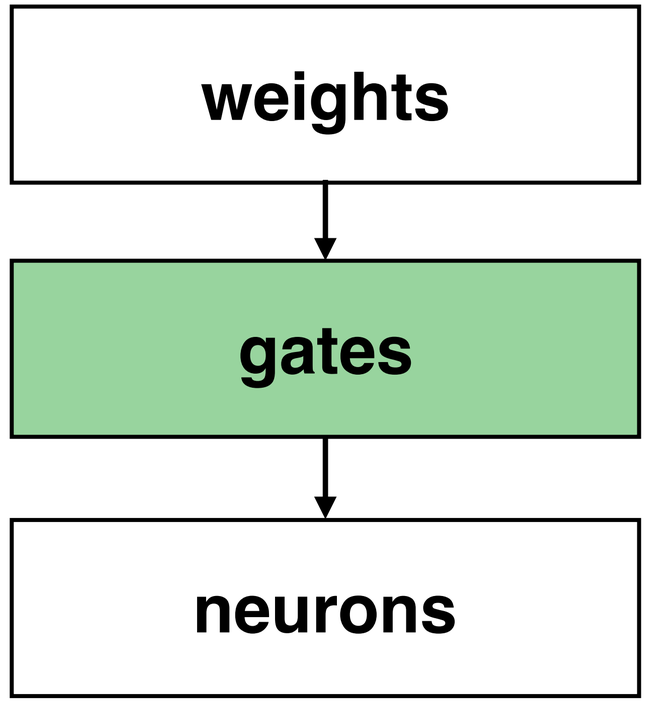
Levels of sparsification:



Putting everything together

- Less computations on testing stage
- Better overall compression on testing stage
- Show that LSTM works in different manner on different tasks

Levels of sparsification:



Plan of the talk

1. **Idea**: introduce intermediate level of sparsification in RNN — gates

2. **Implementation** of the idea in Bayesian sparsification framework

3. **Results**: better compression and interpretable resulting gates structure

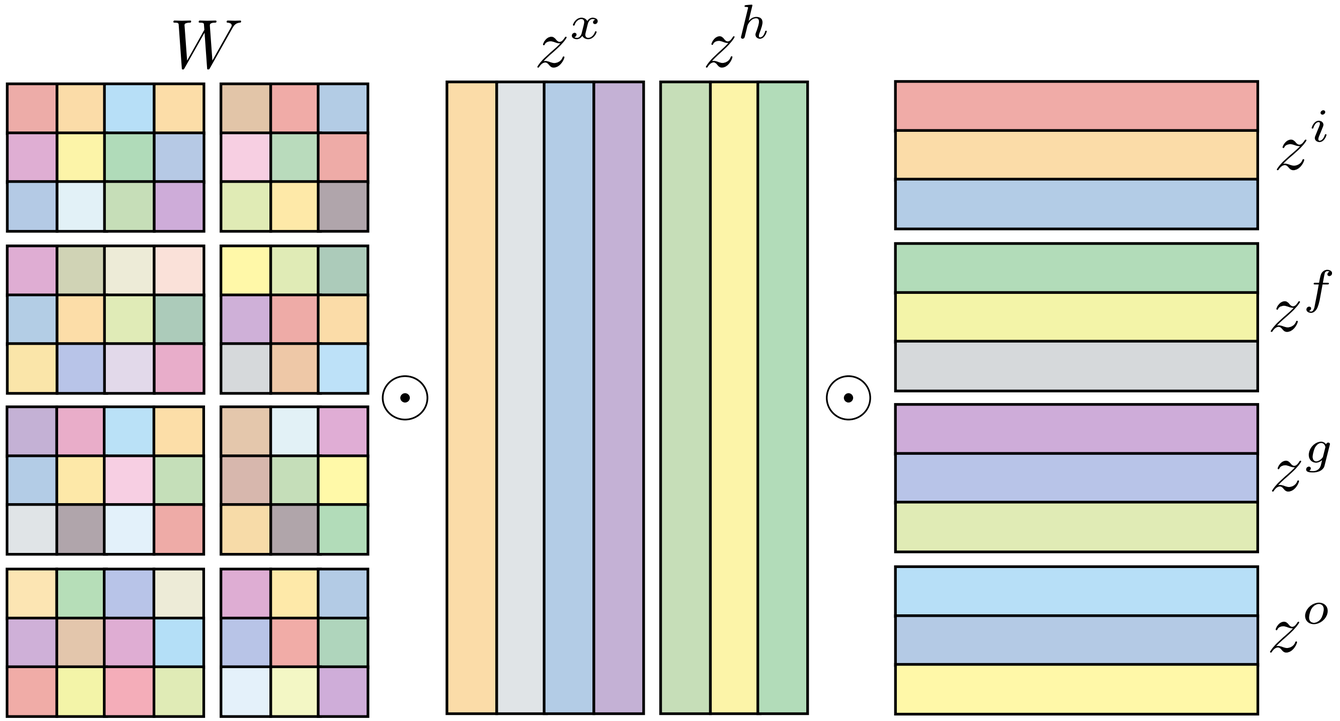
Implementation: introducing group weights

$$f_{t+1} = \sigma\left(W_f^x(x_{t+1} \odot z^x) + \left(W_f^h(h_t \odot z^h)\right) \odot z^f + b_f\right)$$

similarly for $i,\ g,\ o$

$$\tilde{W}^h_{f,ij} = W^h_{f,ij} \cdot z^h_i \cdot z^f_j$$

$$\tilde{W}^x_{f,ij} = W^x_{f,ij} \cdot z^x_i \cdot z^x_j$$



Implementation: introducing group weights

zero components → remove neuron

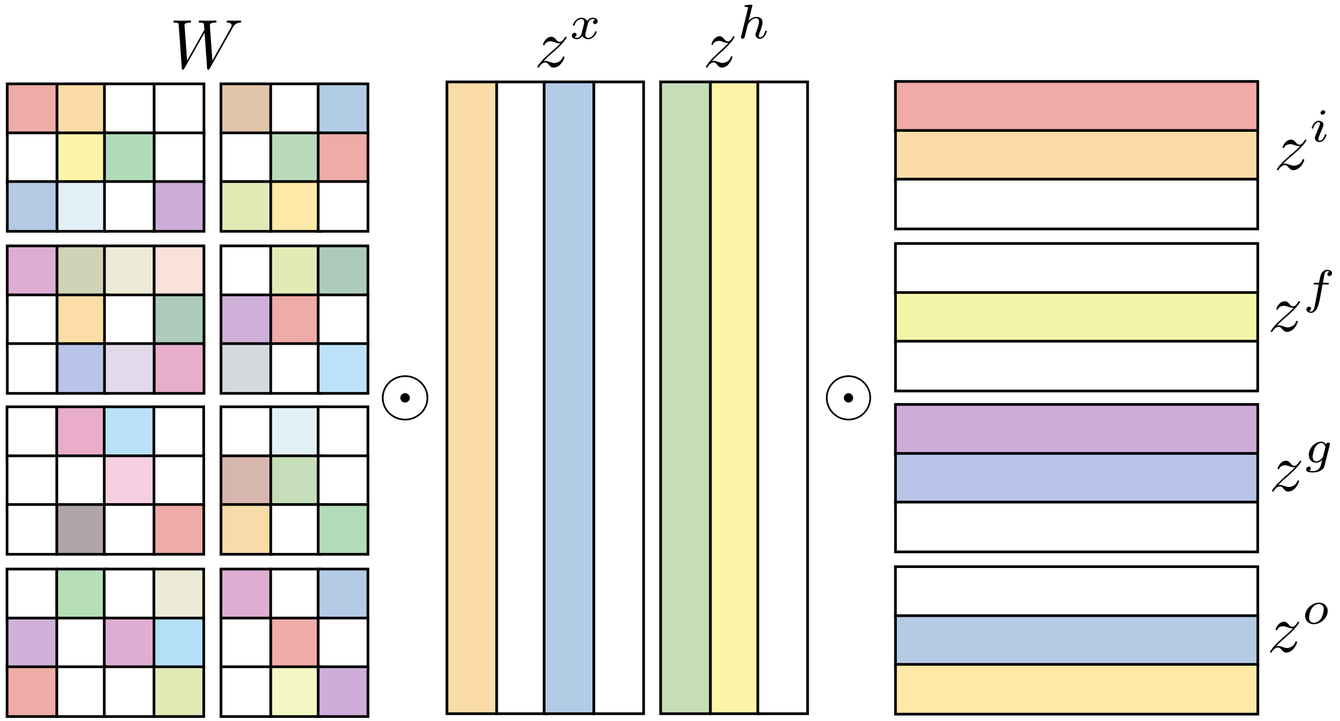
zero component → constant gate

$$f_{t+1} = \sigma\left(W_f^x (x_{t+1} \odot z^x) + \left(W_f^h (h_t \odot z^h)\right) \odot z^f + b_f \right)$$

similarly for $i,\ g,\ o$

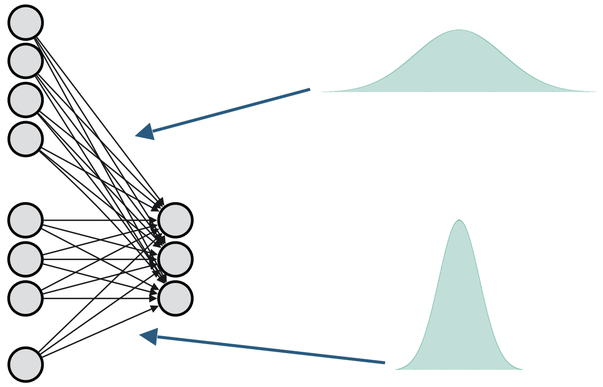
$$\tilde{W}^h_{f,ij} = W^h_{f,ij} \cdot z^h_i \cdot z^f_j$$

$$\tilde{W}^x_{f,ij} = W^x_{f,ij} \cdot z^x_i \cdot z^x_j$$

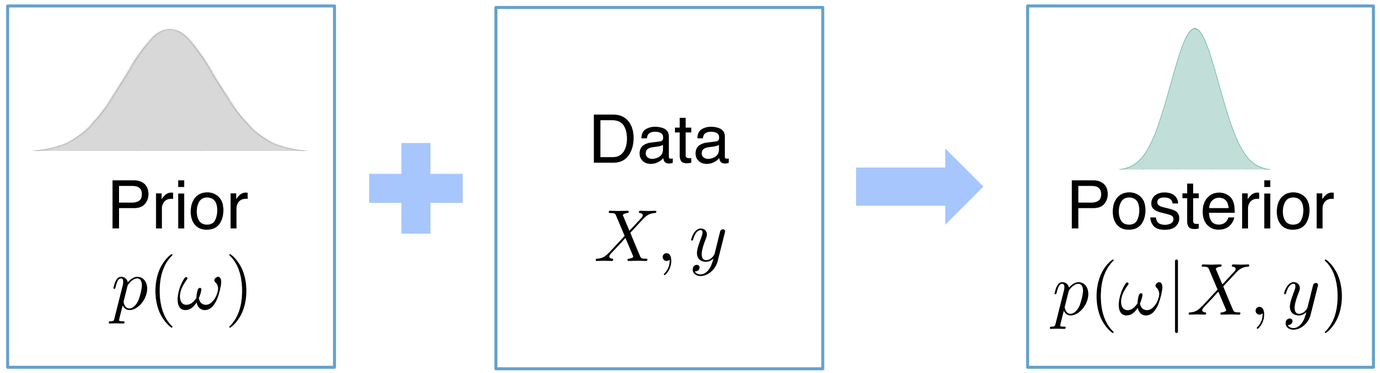


Bayesian neural networks

Stochastic weights:

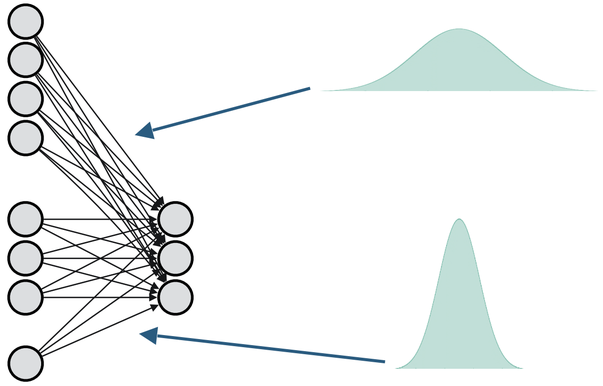


Bayesian Inference:

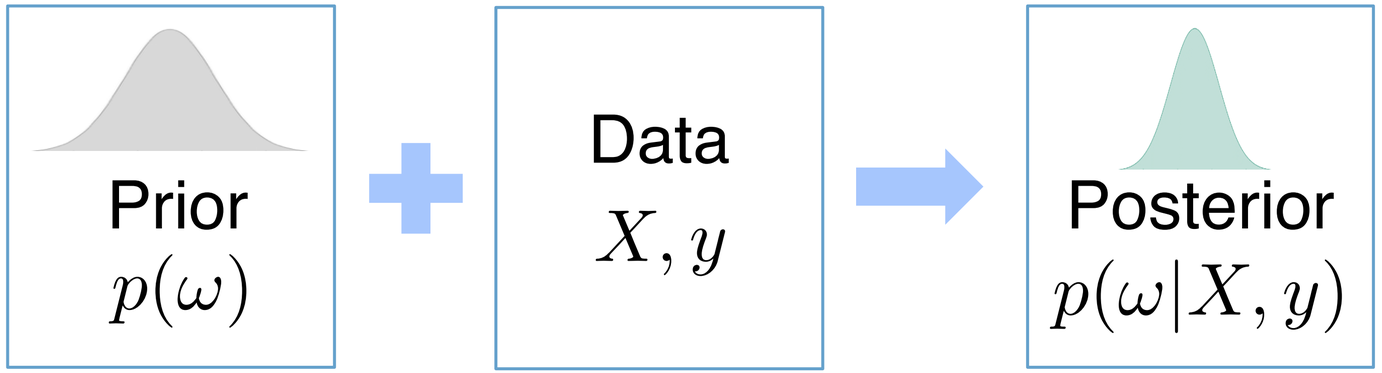


Bayesian neural networks

Stochastic weights:



Bayesian Inference:

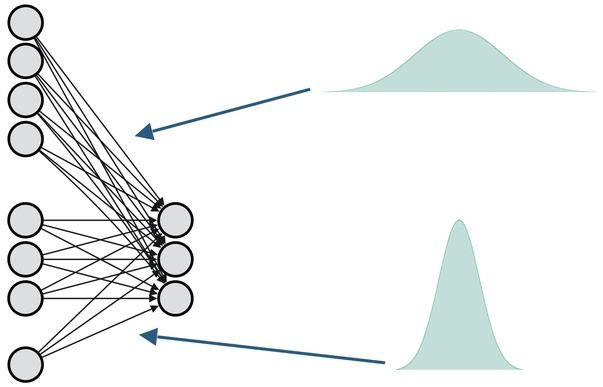


Posterior is intractable in neural networks → approximate it with $q(\omega|\lambda)$:

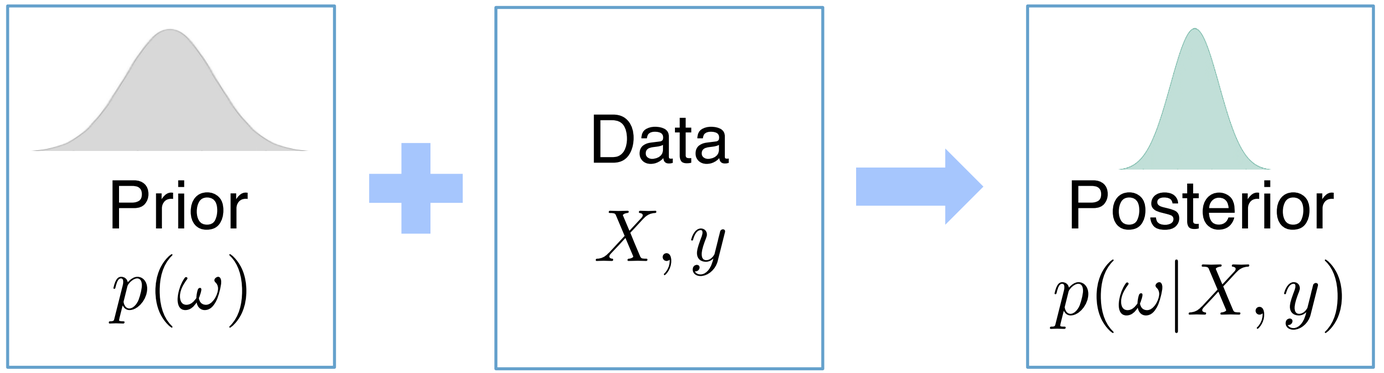
$$KL(q(\omega|\lambda)||p(\omega|X,y)) \rightarrow \min_{\lambda}$$

Bayesian neural networks

Stochastic weights:



Bayesian Inference:



Equivalently, we can optimize ELBO to find approximate posterior $q(\omega|\lambda)$:

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(\omega|\lambda)} \log p(y^i|x^i, \omega)}_{\text{Data term}} - \underbrace{KL(q(\omega|\lambda)||p(\omega))}_{\text{Regularizer}} \to \max_{\lambda}$$

Data term
Monte-Carlo sampling

Regularizer
Analytical approximation

LSTM with group weights

$$f_{t+1} = \sigma\left(W_f^x(x_{t+1} \odot z^x) + \left(W_f^h(h_t \odot z^h)\right) \odot z^f + b_f\right)$$

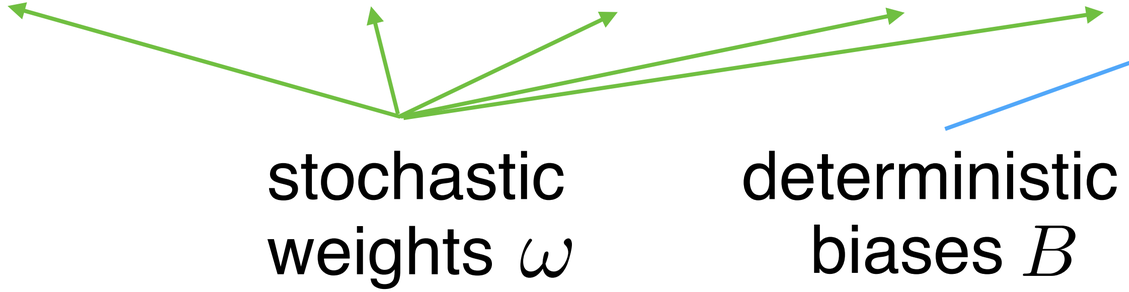
weights ω

$\omega = \{W, z\}$

biases B

Bayesian LSTM with group weights

$$f_{t+1} = \sigma\left(W_f^x(x_{t+1} \odot z^x) + \left(W_f^h(h_t \odot z^h)\right) \odot z^f + b_f\right)$$



Sparse Variational Dropout [1] for LSTM

$$f_{t+1} = \sigma\left(\boxed{W_f^x}(x_{t+1} \odot \boxed{z^x}) + \left(\boxed{W_f^h}(h_t \odot \boxed{z^h})\right) \odot \boxed{z^f} + \boxed{b_f}\right)$$

stochastic weights ω

deterministic biases B

Prior: $$p(\omega_k) \propto \frac{1}{|\omega_k|}$$

Approximate posterior: $$q(\omega_k|\theta_k, \sigma_k) = \mathcal{N}\left(\omega_k|\theta_k, \sigma_k^2\right)$$

$$k = 1, \ldots, |\omega|$$

[1] D. Molchanov, A. Ashukha, D. Vetrov. Variational dropout sparsifies deep neural networks. ICML 2017

Training Sparse Variational Dropout

Optimize ELBO to find θ, σ, B:

$$-\sum_{i=1}^{N} \int q(\omega|\theta,\sigma) \log p(y^i|x_0^i,\ldots,x_T^i,\omega,B) d\omega + \text{data term}$$

$$+\sum_{k=1}^{|\omega|} KL(q(\omega_k|\theta_k,\sigma_k)||p(\omega_k)) \rightarrow \min_{\theta,\sigma,B}$$

sparsity inducing regularizer

Training Sparse Variational Dropout

Optimize ELBO to find θ, σ, B:

$$-\sum_{i=1}^{N}\int q(\omega|\theta,\sigma)\log p(y^i|x_0^i,\dots,x_T^i,\omega,B)d\omega+$$

data term

$$+\sum_{k=1}^{|\omega|} KL(q(\omega_k|\theta_k,\sigma_k)||p(\omega_k)) \to \min_{\theta,\sigma,B}$$

sparsity inducing regularizer

$$\prod_{k=1}^{|\omega|}\mathcal{N}\left(\omega_k|\theta_k,\sigma_k^2\right)$$

- First term: Monte-Carlo estimate with 1 sample, reparametrization trick:

$$\hat{\omega}_k = \theta_k + \hat{\epsilon}_k\sigma_k, \quad \hat{\epsilon}_k \sim \mathcal{N}(\epsilon_k|0,1)$$

Training Sparse Variational Dropout

Optimize ELBO to find θ, σ, B:

$$-\sum_{i=1}^{N} \log p(y^i|x_0^i, \dots, x_T^i, \hat{\omega}, B) + \text{(data term)}$$

$$+\sum_{k=1}^{|\omega|} KL(q(\omega_k|\theta_k, \sigma_k)||p(\omega_k)) \to \min_{\theta,\sigma,B} \quad \text{(sparsity inducing regularizer)}$$

- First term: 1 sample, reparametrization trick:

$$\hat{\omega}_k = \theta_k + \hat{\epsilon}_k \sigma_k, \quad \hat{\epsilon}_k \sim \mathcal{N}(\epsilon_k|0, 1)$$

Training Sparse Variational Dropout

Optimize ELBO to find θ, σ, B:

$$-\sum_{i=1}^{N} \log p(y^i|x_0^i, \dots, x_T^i, \hat{\omega}, B) + \text{(data term)}$$

$$+\sum_{k=1}^{|\omega|} KL(q(\omega_k|\theta_k, \sigma_k)||p(\omega_k)) \rightarrow \min_{\theta,\sigma,B} \quad \text{(sparsity inducing regularizer)}$$

- First term: 1 sample, reparametrization trick:

$$\hat{\omega}_k = \theta_k + \hat{\epsilon}_k \sigma_k, \quad \hat{\epsilon}_k \sim \mathcal{N}(\epsilon_k|0, 1)$$

- Second term: analytical approximation with $f_{KL}(\sigma_k^2/\theta_k^2)$

Training Sparse Variational Dropout

Optimize ELBO to find θ, σ, B:

data term

$$-\sum_{i=1}^{N} \log p(y^i|x_0^i, \dots, x_T^i, \hat{\omega}, B) + \\ + \sum_{k=1}^{|\omega|} f_{KL}(\sigma_k^2/\theta_k^2) \to \min_{\theta,\sigma,B}$$

sparsity inducing regularizer

- First term: 1 sample, reparametrization trick:

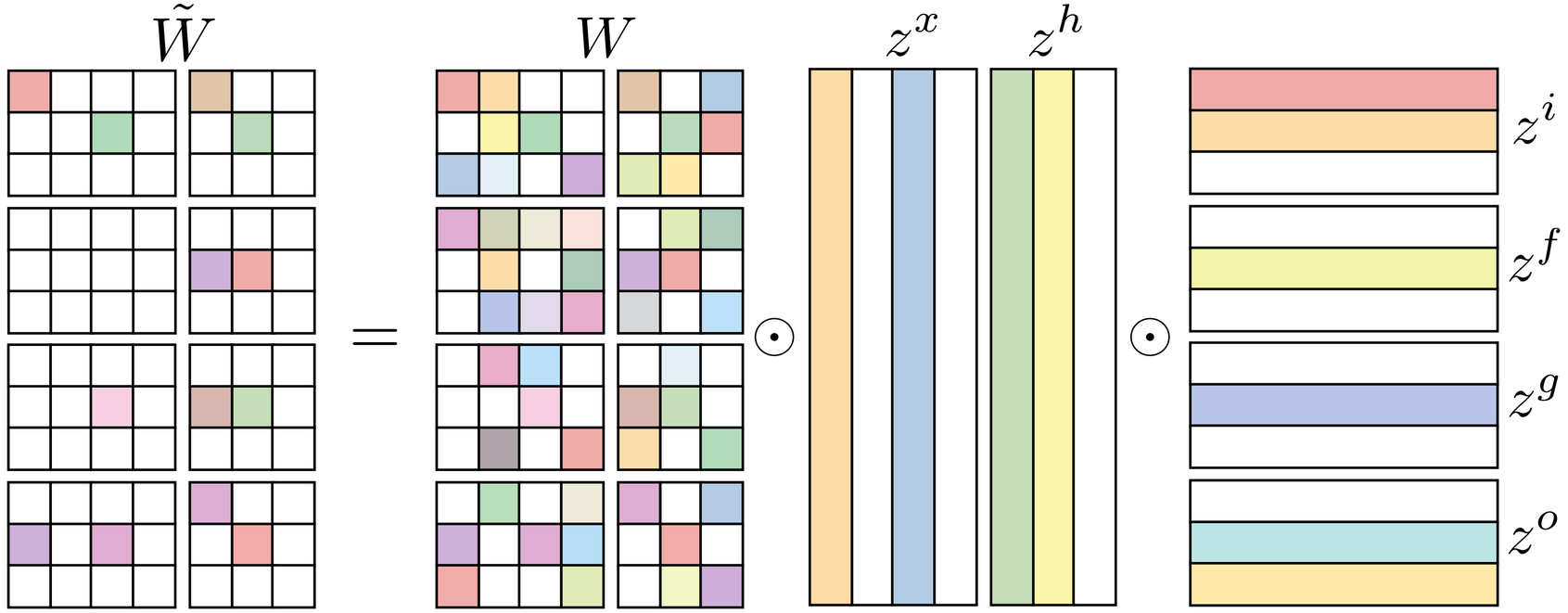
$$\hat{\omega}_k = \theta_k + \hat{\epsilon}_k \sigma_k, \quad \hat{\epsilon}_k \sim \mathcal{N}(\epsilon_k|0,1)$$

- Second term: analytical approximation with $f_{KL}(\sigma_k^2/\theta_k^2)$

Compression after training

$$\text{If } \theta_k^2/\sigma_k^2 < \text{threshold: } \theta_k = 0,\ \sigma_k = 0 \qquad k = 1, \ldots, |\omega| \qquad \omega = \{W, z\}$$

- On testing stage: use deterministic LSTM with $\omega = \theta$



Final algorithm

Training on a sequence $x = x_1 \dots x_T$ with label y :

1. Sample weights: $\hat{\omega} = \theta + \hat{\epsilon} \odot \sigma, \quad \hat{\epsilon} \sim \mathcal{N}(\epsilon|0, I)$
2. Forward pass: $y_{pred} = \mathrm{LSTM}(x, \hat{\omega}, B)$
3. Backward pass: compute stochastic gradients of ELBO:

$$\nabla_{\theta, \log \sigma, B} \left(N\mathrm{Loss}\big(y, y_{pred}\big) + \mathrm{SparseReg}(\theta, \sigma) \right)$$

Pruning after training:

$\mathrm{If}\ \theta_k^2/\sigma_k^2 < \mathrm{threshold}$:

$\theta_k = 0, \ \sigma_k = 0$

Prediction for a sequence $x = x_1 \dots x_T$:

Return $\mathrm{LSTM}(x, \theta, B)$

Plan of the talk

1. **Idea**: introduce intermediate level of sparsification in RNN — gates

2. **Implementation** of the idea in Bayesian sparsification framework

3. **Results**: better compression and interpretable resulting gates structure

Experiments: text classification

IMDb (25K texts, 2 classes), AGNews (120K texts, 4 classes)

Architecture: Embedding → LSTM → FC on h_T

Task	Method	Quality	Compression	Neurons x - h	Gates
IMDb Accuracy %	Original	84.1	1x	$300 - 128$	512
	SparseVD W	**84.47**	1135x	$7 - 9$	22
	SparseVD W+N	83.98	17874x	$1 - 5$	12
	SparseVD W+G+N	83.98	**19747x**	$\mathbf{1 - 4}$	**6**
AGNews Accuracy %	Original	**90.6**	1x	$300 - 512$	2048
	SparseVD W	89.01	350x	$193 - 65$	72
	SparseVD W+N	88.55	645x	$43 - 17$	62
	SparseVD W+G+N	88.41	**647x**	$\mathbf{43 - 14}$	**39**

W: weights N: neurons G: gates (ours)

Experiments: text generation

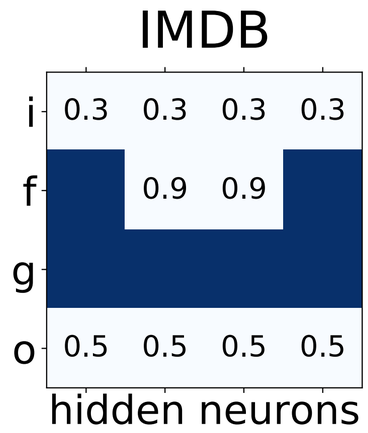
PTB Corpus: char-level (6M chars), word-level (0.9M words)

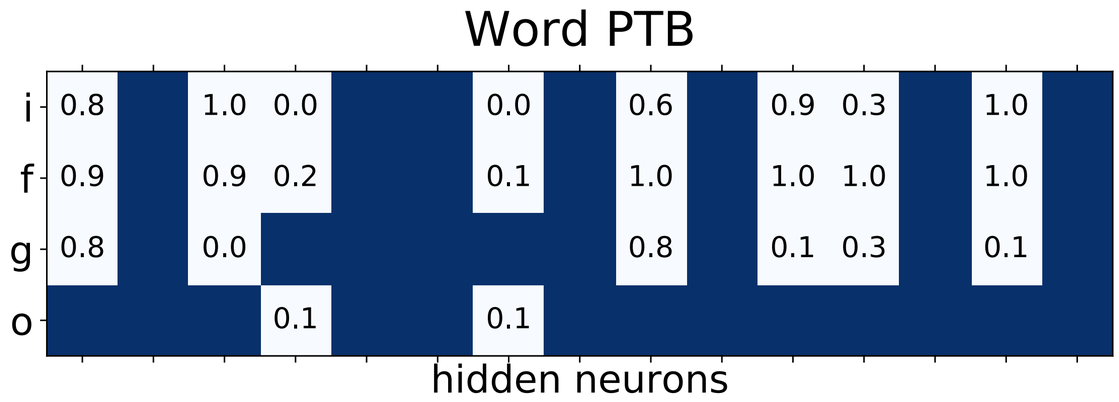
Architecture: LSTM → FC

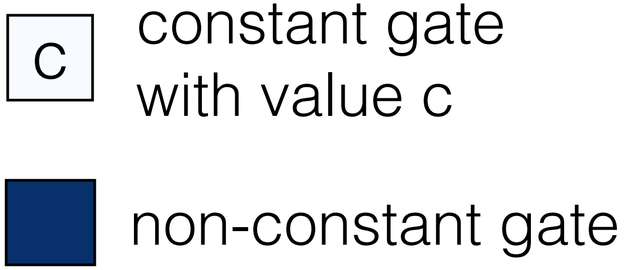
Task	Method	Quality	Compression	Neurons x - h	Gates
Char PTB Bits-per-char Valid-Test	Original	$1.498 - 1.455$	1x	$50 - 1000$	4000
	SparseVD W	$1.472 - 1.429$	7.9x	$50 - 431$	1718
	SparseVD W+N	$1.478 - 1.430$	**10.0x**	$\mathbf{50 - 390}$	**1560**
	SparseVD W+G+N	$\mathbf{1.467 - 1.425}$	9.8x	$50 - 404$	1563
Word PTB Perplexity Valid-Test	Original	$135.6 - 129.3$	1x	$10000 - 256$	1024
	SparseVD W	$\mathbf{115.0 - 109.0}$	22.0x	$9985 - 153$	281
	SparseVD W+N	$116.2 - 111.0$	23.4x	$9993 - 134$	335
	SparseVD W+G+N	$122.2 - 116.5$	**24.4x**	$\mathbf{9973 - 114}$	**220**

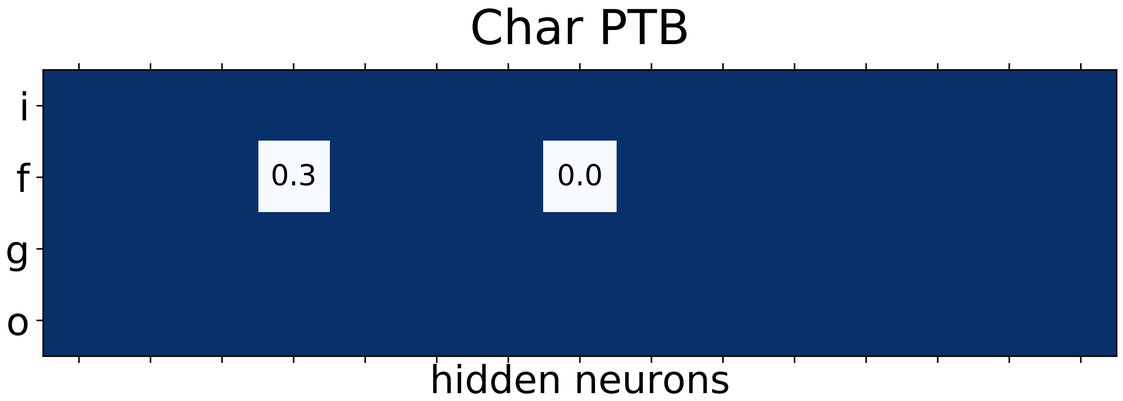
W: weights N: neurons G: gates (ours)

Experiments: resulting gates structure









Different gates are important for different tasks

Summary

- Three levels of sparsity for gated RNNs:
  (1) sparsify weights; (2) make gates constant; (3) remove neurons

- Can be easily implemented in any sparsification framework
  Our implementation: introduce group weights and use Bayesian sparsification

- Improve compression and reveal LSTM specifics on different tasks

Extended abstract

Source code

$p(B|A)$yesgroup.ru