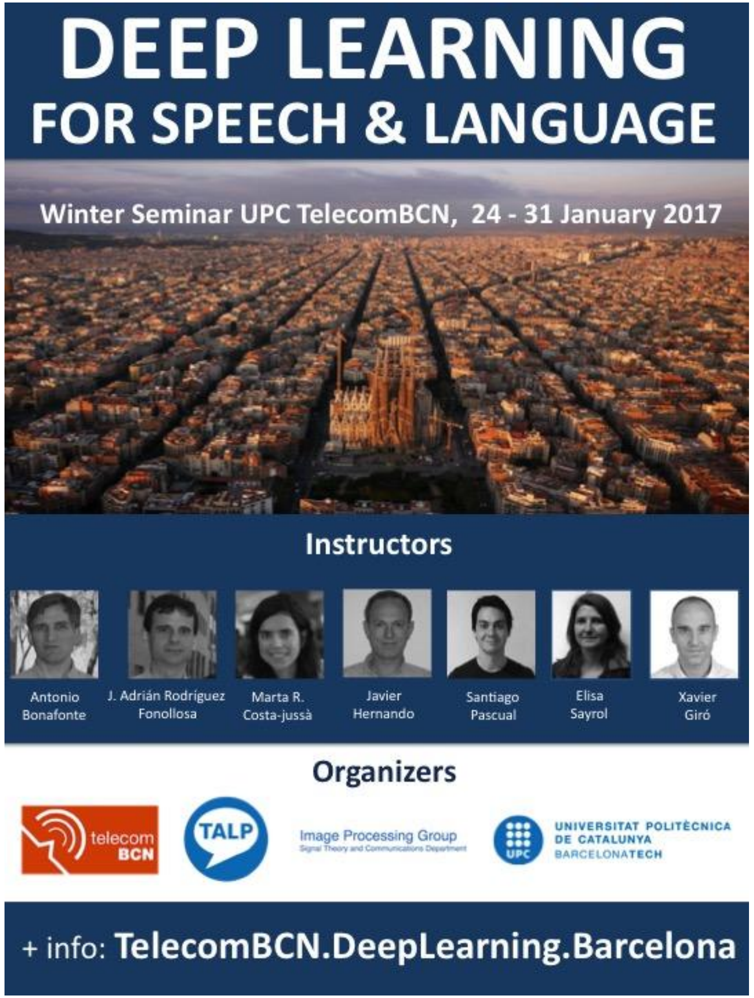

DEEP LEARNING
FOR SPEECH & LANGUAGE
Winter Seminar UPC TelecomBCN, 24 - 31 January 2017
Instructors
Antonio Bonafonte
J. Adrián Rodríguez Fonollosa
Marta R. Costa-jussà
Javier Hernando
Santiago Pascual
Elisa Sayrol
Xavier Giró
Organizers
telecom BCN
TALP
Image Processing Group
Signal Theory and Communications Department
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
+ info: TelecomBCN.DeepLearning.Barcelona

[course site]

Day 2 Lecture 3

Recurrent Neural Networks II

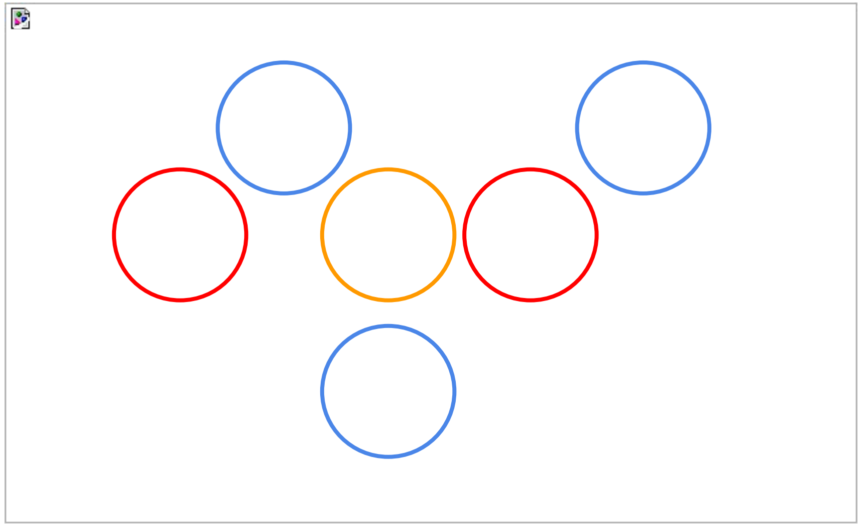
Santiago Pascual

Outline

Recap LSTM...

An LSTM cell is defined by two groups of neurons plus the cell state (memory unit):

1. Gates
2. Activation units
3. Cell state



$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\hat{\mathbf{C}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{C}_t = \mathbf{i}_t \odot \hat{\mathbf{C}}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1}$$

If we don't have any input, the cell is still applying a forget factor!

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

Phased LSTM

[Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences (Neil et al. 2016)]

Extend the LSTM cell by **adding a new time gate**, controlled by a **parametrized oscillation with a frequency range that produces updates of the memory cell only during small percentage of the cycle**.

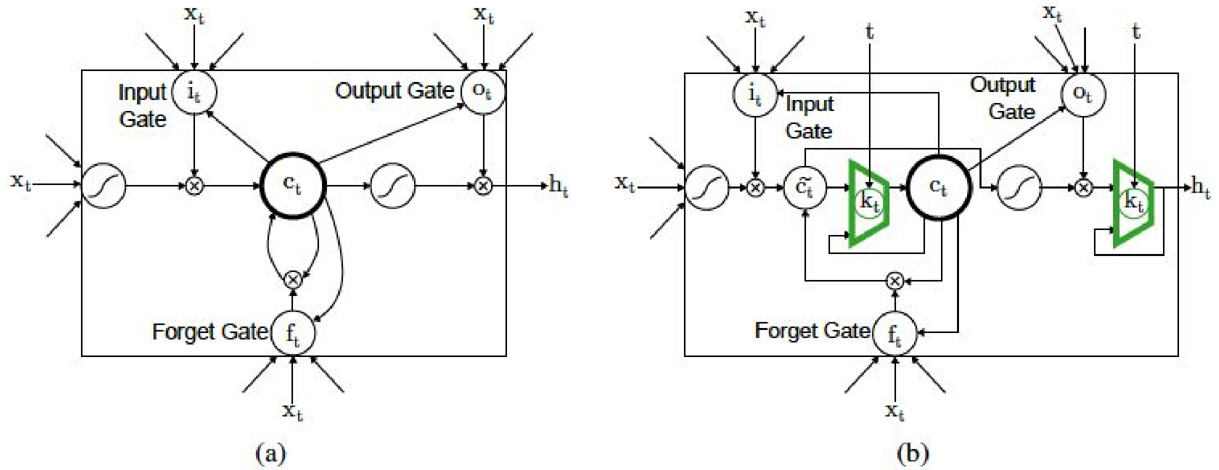
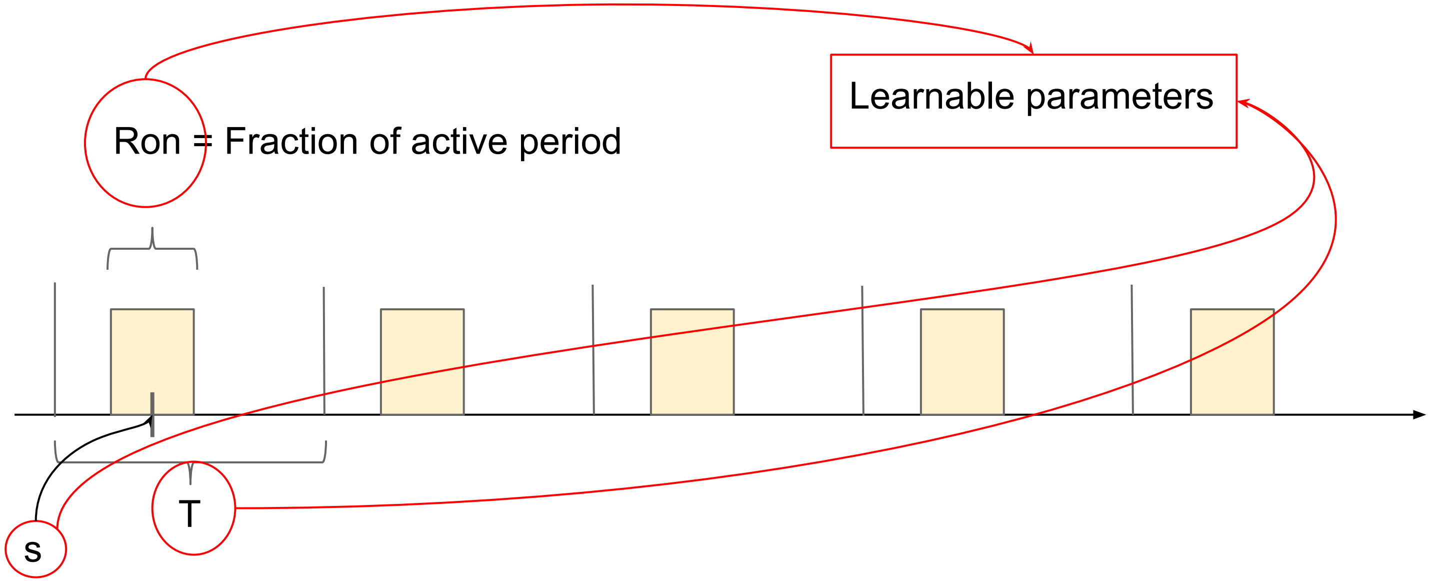


Figure 1: Model architecture. (a) Standard LSTM model. (b) Phased LSTM model, with time gate k_t controlled by timestamp t. In the Phased LSTM formulation, the cell value c_t and the hidden output h_t can only be updated during an "open" phase; otherwise, the previous values are maintained.

Phased LSTM

Extend the LSTM cell by **adding a new time gate**, controlled by a **parametrized oscillation with a frequency range that produces updates of the memory cell only during small percentage of the cycle**.



Phased LSTM

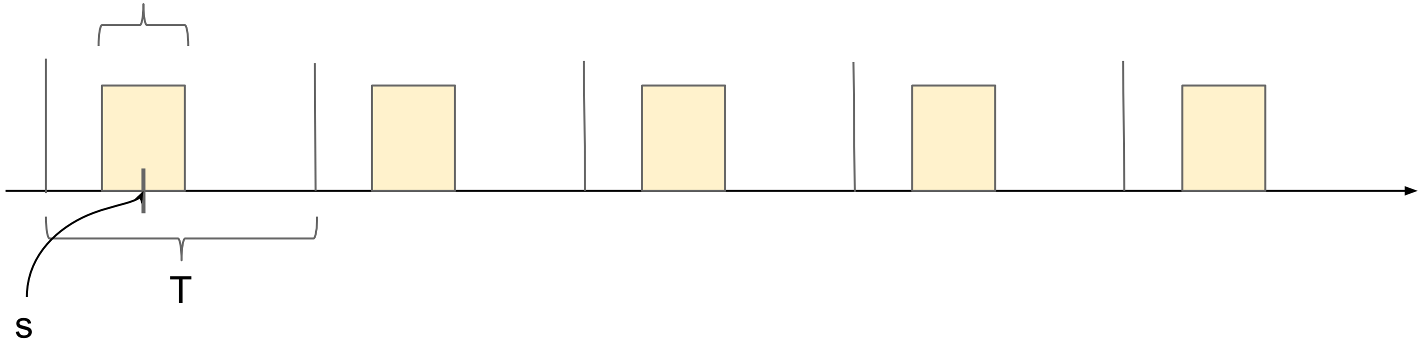
3 phases in kt gate: In first 2 phases the “openness” of the gate **rises from 0 to 1** (first phase) and **from 1 to 0** (second phase). During third phase, **gate closes and cell state is maintained.** α parameter is a leaky factor to allow gradient flowing during training, 0 during test.

$$\phi_t = \frac{(t-s) \bmod \tau}{\tau}, \qquad k_t = \begin{cases} \frac{2\phi_t}{r_{on}}, & \text{if } \phi_t < \frac{1}{2} r_{on} \\ 2 - \frac{2\phi_t}{r_{on}}, & \text{if } \frac{1}{2} r_{on} < \phi_t < r_{on} \\ \alpha\phi_t, & \text{otherwise} \end{cases}$$

$$\Rightarrow$$

$$\begin{aligned} \widetilde{c_j} &= f_j \odot c_{j-1} + i_j \odot \sigma_c(x_j W_{xc} + h_{j-1} W_{hc} + b_c) \\ c_j &= k_j \odot \widetilde{c_j} + (1-k_j) \odot c_{j-1} \\ \widetilde{h_j} &= o_j \odot \sigma_h(\widetilde{c_j}) \\ h_j &= k_j \odot \widetilde{h_j} + (1-k_j) \odot h_{j-1} \end{aligned}$$

Ron = Fraction of active period



Phased LSTM

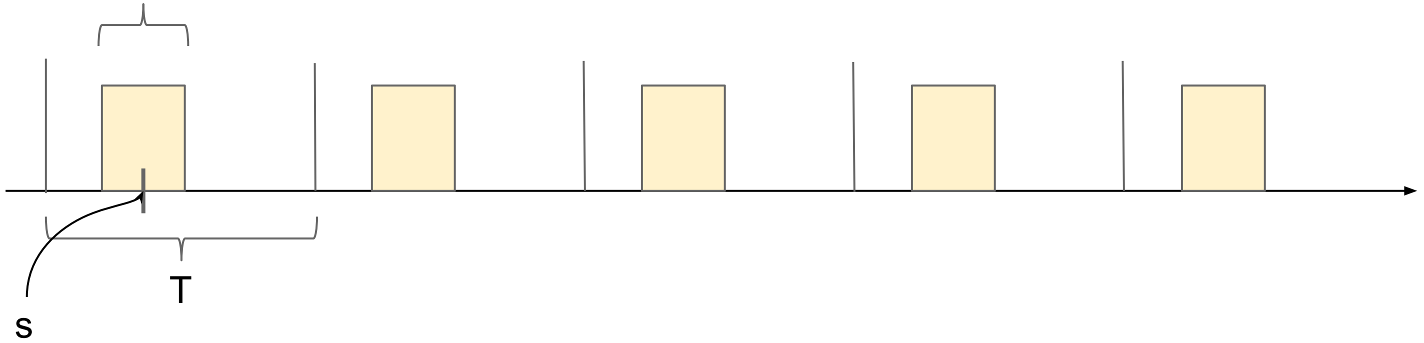
3 phases in kt gate: In first 2 phases the “openness” of the gate **rises from 0 to 1** (first phase) and **from 1 to 0** (second phase). During third phase, **gate closes and cell state is maintained.** α parameter is a leaky factor to allow gradient flowing during training, 0 during test.

$$\phi_t = \frac{(t-s) \bmod \tau}{\tau},$$

$$k_t = \begin{cases} \frac{2\phi_t}{r_{on}}, & \text{if } \phi_t < \frac{1}{2} r_{on} \\ 2 - \frac{2\phi_t}{r_{on}}, & \text{if } \frac{1}{2} r_{on} < \phi_t < r_{on} \\ \alpha\phi_t, & \text{otherwise} \end{cases}$$

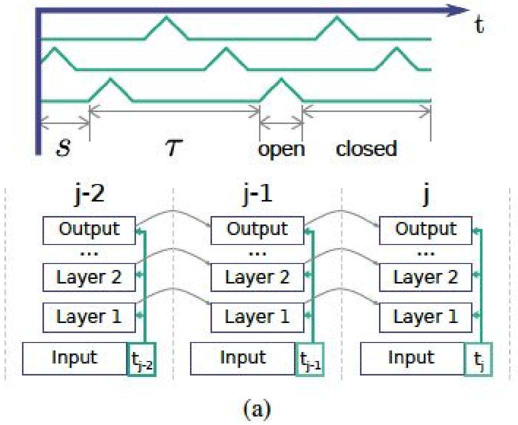
$$\begin{aligned} \widetilde{c_j} &= f_j \odot c_{j-1} + i_j \odot \sigma_c(x_j W_{xc} + h_{j-1} W_{hc} + b_c) \\ c_j &= k_j \odot \widetilde{c_j} + (1 - k_j) \odot c_{j-1} \\ \widetilde{h_j} &= o_j \odot \sigma_h(\widetilde{c_j}) \\ h_j &= k_j \odot \widetilde{h_j} + (1 - k_j) \odot h_{j-1} \end{aligned}$$

Ron = Fraction of active period



Phased LSTM

Updates happen when the cell gate is open, as depicted in the colored figure below, where the different cell states are updated when the clock signal is burst.



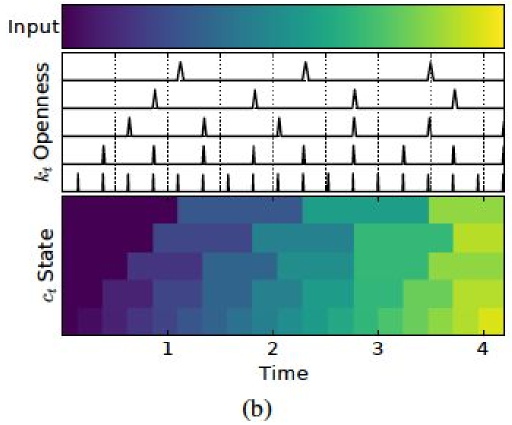


Figure 2: Diagram of Phased LSTM behaviour. (**a**) Top: The rhythmic oscillations to the time gates of 3 different neurons; the period τ and the phase shift s is shown for the lowest neuron. The parameter r_{on} is the ratio of the open period to the total period τ. Bottom: Note that in a multilayer scenario, the timestamp is distributed to all layers which are updated at the same time point. (**b**) Illustration of Phased LSTM operation. A simple linearly increasing function is used as an input. The time gate k_t of each neuron has a different τ, identical phase shift s, and an open ratio r_{on} of 0.05. Note that the input (top panel) flows through the time gate k_t (middle panel) to be held as the new cell state c_t (bottom panel) only when k_t is open.

Phased LSTM

Really **faster convergence than vanilla LSTM** architecture, as well as **more long-term memory retention**, as they preserve info about some time-steps far back in time when *kt* gate is closed, till it's open again, where current time-step receives fresh information about a possibly quite old state.

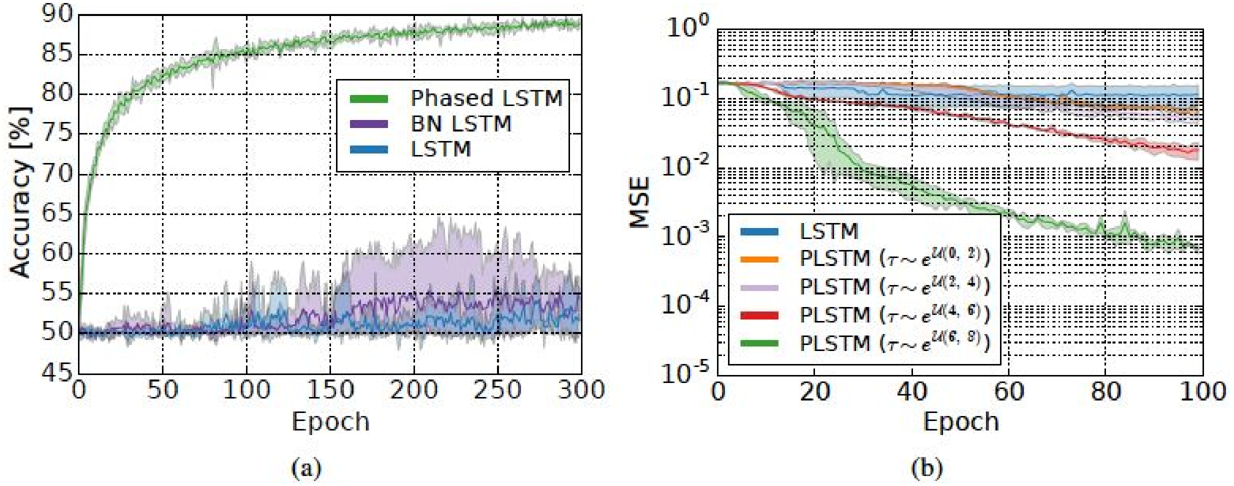
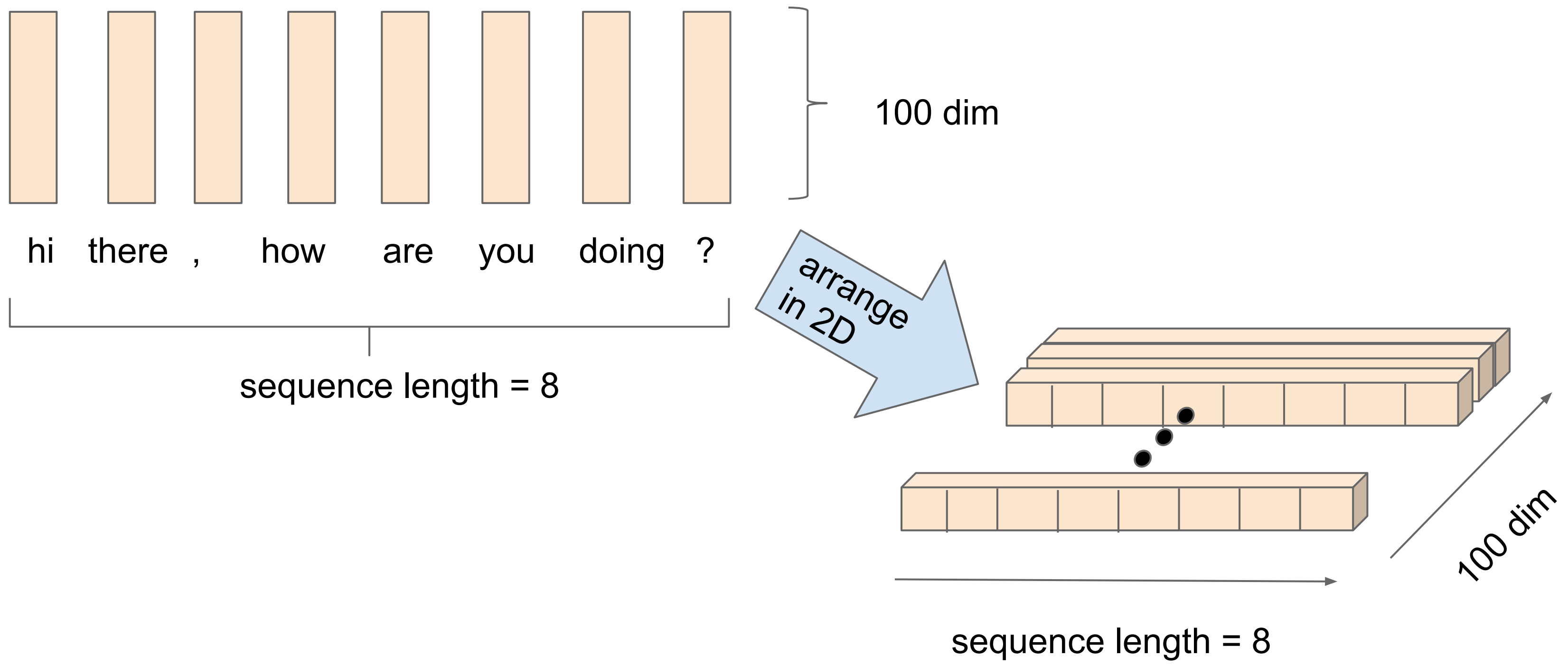


Figure 4: **(a)** Accuracy during training for the superimposed frequencies task. The Phased LSTM outperforms both LSTM and BN-LSTM while exhibiting lower variance. Shading shows maximum and minimum over 5 runs, while dark lines indicate the mean. **(b)** Mean-squared error over training on the addition task, with an input length of 500. Note that longer periods accelerate learning convergence.

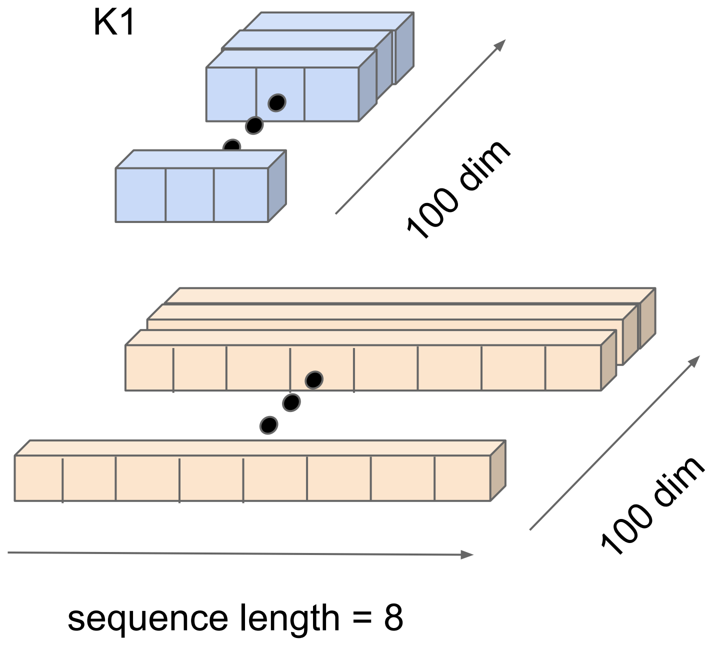
1D Convolutions

Let’s say we have a sequence of 100 dimensional vectors describing words.



1D Convolutions

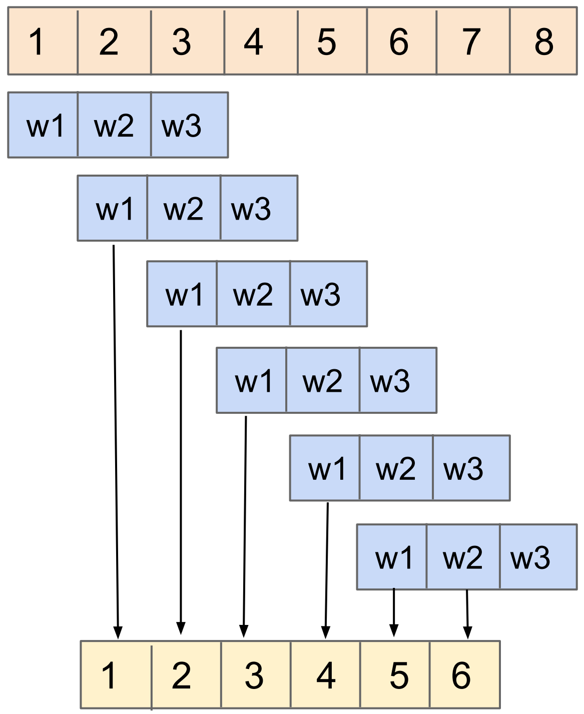
We can apply a 1D convolutional activation over the 2D matrix: for an arbitrari kernel of width=3



Each 1D convolutional kernel is a 2D matrix of size (3, 100)

1D Convolutions

Keep in mind we are working with 100 dimensions although here we depict just one for simplicity

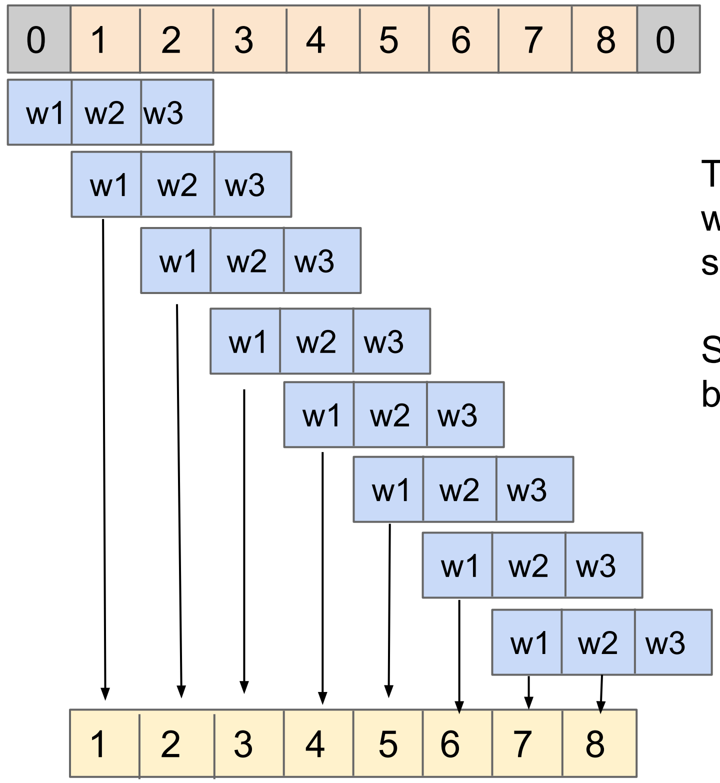


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 8 - 3 + 1 = 6

So the output matrix will be (6, 100) because there was no padding

1D Convolutions

When we add zero padding, we normally do so on both sides of the sequence (as in image padding)

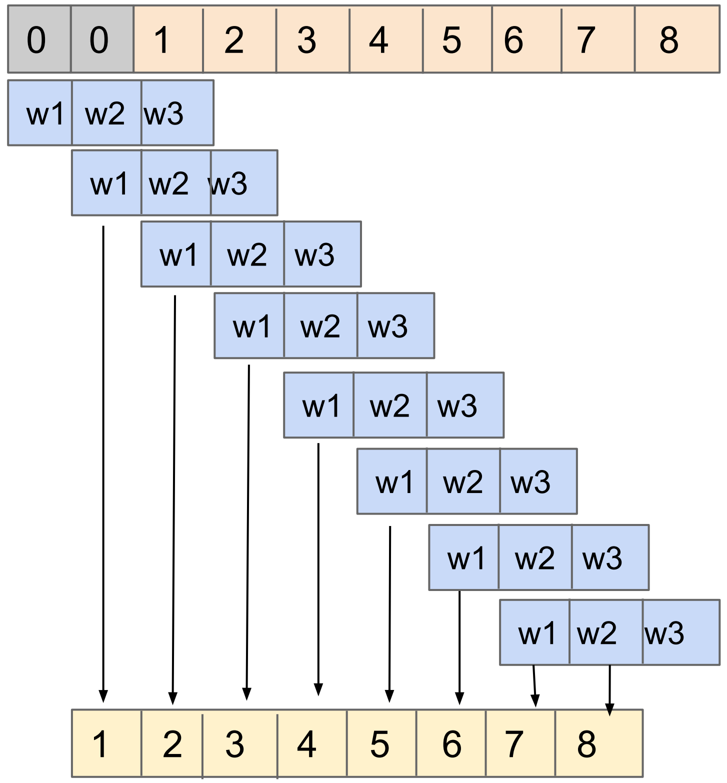


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

Causal 1D Convolutions

Add the zero padding just on the left side of the sequence, not symmetrically



The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

HOWEVER: now every time-step t depends on the two previous inputs as well as the current time-step → every output is causal

We make a causal convolution by padding left the sequence with (kwidth - 1) zeros

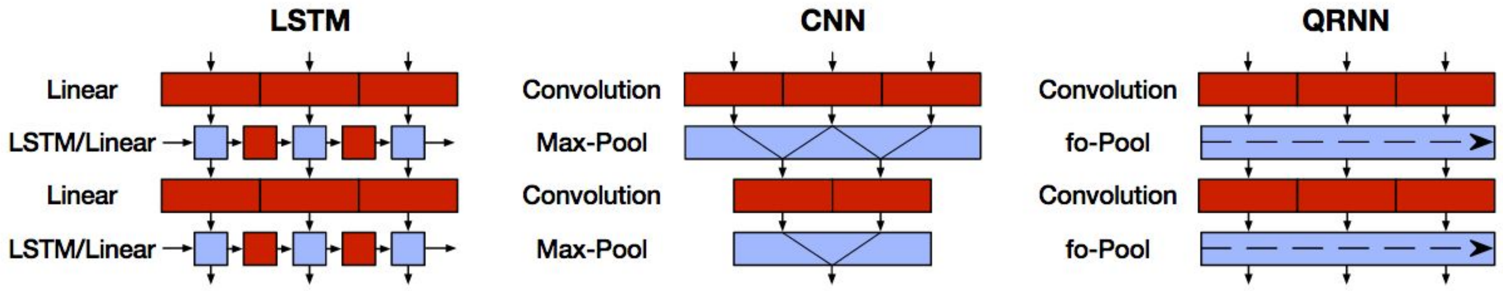
Quasi-RNN

[Quasi-Recurrent Neural Networks (Bradbury et al. 2016)]

Red blocks are parameterized projections (layer activations/dot products) and **blue blocks** are pooling operations:

- In the case of LSTM it is applying the gates transforms (i, f, o)
- In the case of CNN it is a parallel pooling regardless of the position (remember CNNs don't have a sense of order)

Advantage of CNN: We can compute all convolutions in parallel
Advantage of LSTM: It imposes the sense of order (appropriate for sequences)
QRNNs are up to x16 times faster than LSTMs!!

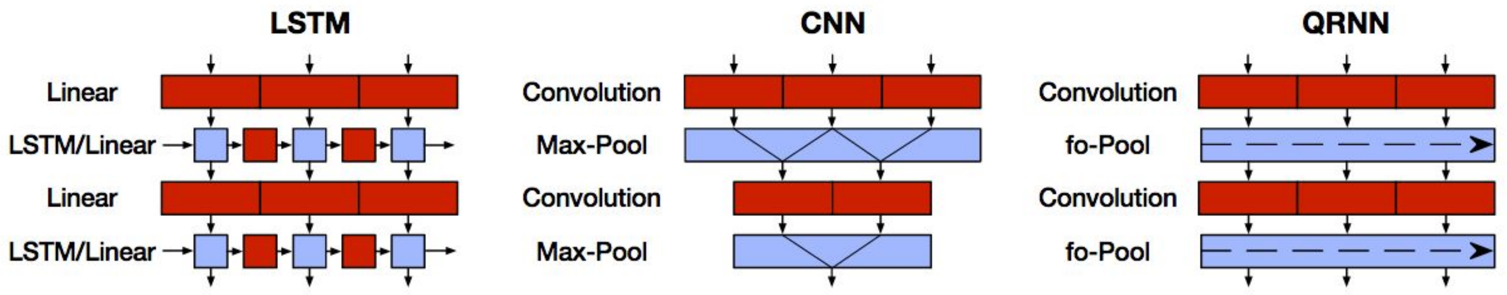


Quasi-RNN

QRNN: Merge the best of both worlds to speed up the operations for when we need to process large sequences.
A QRNN layer is composed of a **convolutional stage** and a **pooling stage**:

- Conv stage: **perform the different activations in parallel** (i.e. layer activations and gate activations) through time, such that **each red sub-block is independent.**
- Pooling stage: impose ordering information across time. It is a sequential process (as depicted by the arrow), yes, **BUT the heavy computation has already been performed in a single forward pass** in the convolutional stage!

Make all activations in parallel with convolutions and merge sequentially, with no recursive weight operations.



Breaking down Quasi-RNN...

Having a sequence of length T with n-dimensional vectors $\mathbf{X} \in \mathbb{R}^{T \times n}$ of T n-dimensional vectors $\mathbf{x}_1 \ldots \mathbf{x}_T$

Compute the layer input activations (like LSTM) with ***m*** conv 1D kernels of *Wz*: $\mathbf{Z} = \tanh(\mathbf{W}_z * \mathbf{X})$

We want to apply two learnable gates F (forget) and O (output): $\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$ $\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X})$

- Note the sigmoid functions when it comes to gate activations → because we want them to act like switches! (between 0 and 1)

Consider weights matrices are $\mathbf{W}_z, \mathbf{W}_f$, and $\mathbf{W}_o$, each in $\mathbb{R}^{k \times n \times m}$ thus containing ***m*** kernels, ***k*** kernel width each (amount of time-steps seen per convolution) and with ***n*** input dimensions (like previous slides example with 100 dim) → output Z is $\mathbf{Z} \in \mathbb{R}^{T \times m}$ of m-dimensional candidate vectors $\mathbf{z}_t$.

To exemplify: what happens when k=2? We process the current time-step and the right previous one
Looks similar to LSTM equations, huh?

$$\begin{aligned}
\mathbf{z}_t &= \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).
\end{aligned}$$

Breaking down Quasi-RNN...

Finally, **pooling stage is the recursive part, where order is imposed** (thus it becomes an RNN-like mechanism). Note this is just an element-wise product, so quite quicker than computing the gate activation at every time-step (like in LSTM).

Three different flavors depending on the gates we computed:

- f-pooling

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t,$$

- fo-pooling

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

- ifo-pooling

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

Main ideas to be kept regarding RNNs

- RNNs are very good to process sequences, so whenever order matters → think about including some recurrent module to keep track of temporal evolution
- Generally LSTM/GRU perform similarly, though it is interesting to try both by yourselves with any task!
    - Whenever you ask yourselves: which cell should I use? If you have lots of data you may want to go first with LSTMs (it has more parameters)
- If you have to process very long sequences (e.g. 2.000 time-steps) → maybe try PLSTM
- If you care about speed/efficiency during training → maybe try QRNN

Thanks ! Q&A ?

@Santty128