Advanced Deep Learning

DATA.ML.230

Sequence processing

Motivation

- Classic neural networks do not persist information

- But what if we want to classify events that are happening at every point in a sequence (e.g., in a movie)? Information extracted from previous frames would certainly be helpful

Recurrent Neural Networks (RNNs)

RNNs have loops that persist information and can act as memory

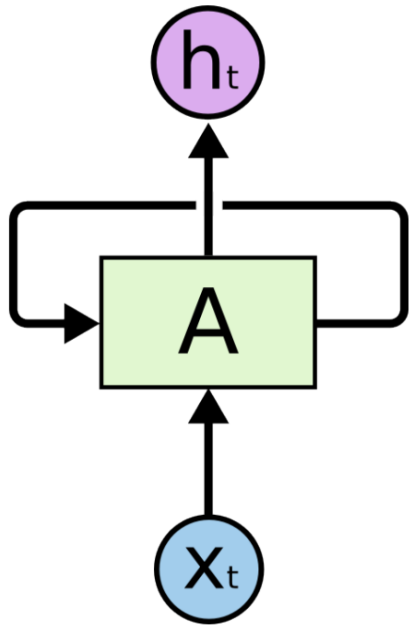


Figure: Recurrent neural network, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks (RNNs)

If we have t time-steps, RNNs can be viewed as multiple copies of the same network (with the same weights), each taking information corresponding to time step l as input, and passing some information to the network at next time-step

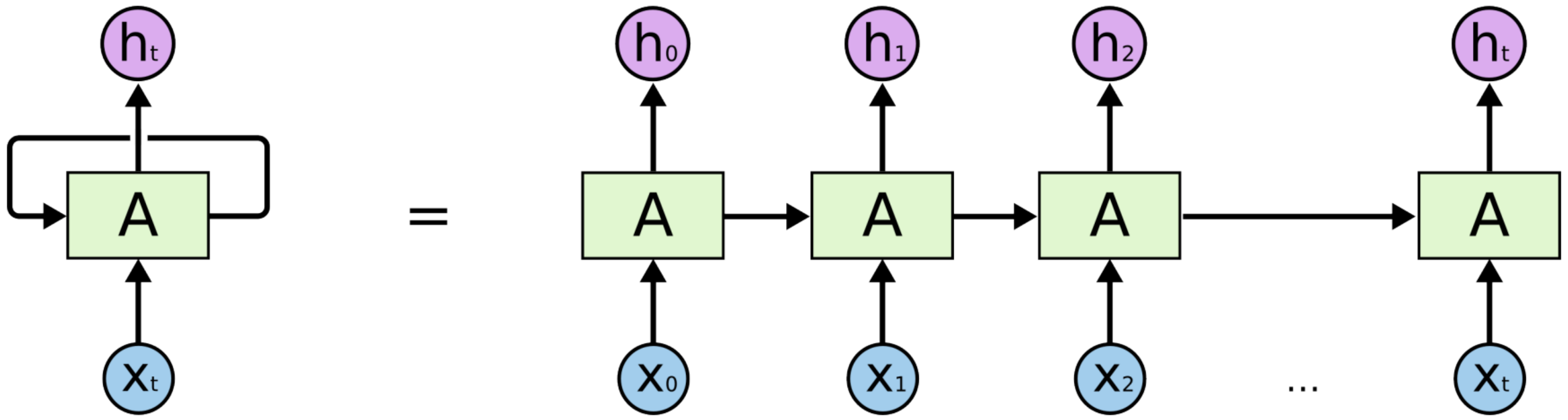


Figure: Unrolled RNN, image taken from:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks (RNNs)

RNNs have many applications, including:

- Processing of videos, e.g., action recognition
- Processing speech, e.g., speech to text, speech recognition
- Processing text, e.g., translation

Recurrent Neural Networks (RNNs)

Sometimes the network only needs to look at recent information to perform the task.

E.g., if we are trying to predict the next word in sentences, for the sentence, “the color of the sky is …”, the network can use recent information to output “blue”.

However, sometimes the network needs information from far back in time, e.g., to complete the sentence “I am from Spain. [a few sentences]. I speak fluent …”, the network needs the context from the first sentence to output “Spanish”.

Even though RNNs are theoretically capable of extracting long-term information, in practive, they do not learn to rely on it.

Long- Short-Term Memory Networks (LSTMs)

LSTM is a type of RNN that is capable of learning both sort-term and long-term dependencies

RNN modules consist of a single tanh layer

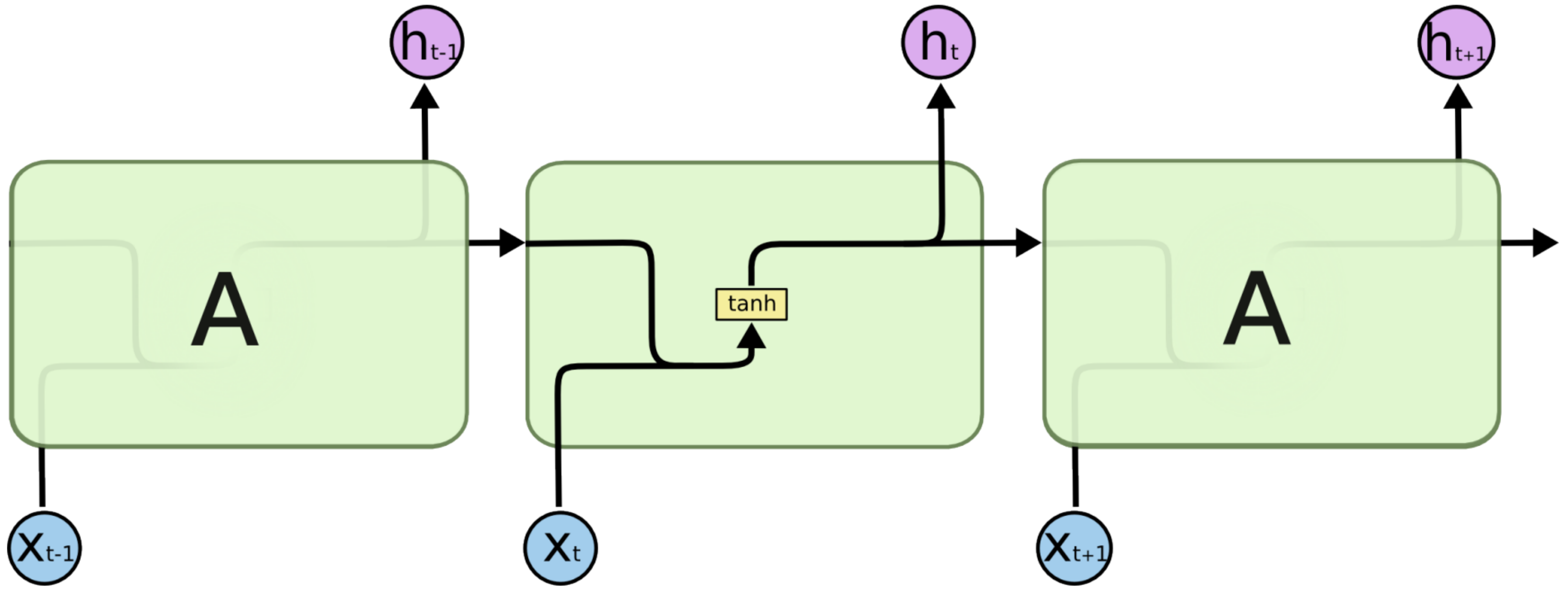


Figure: Intervals of RNN modules, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

LSTM modules consist of four layers that interact with each other

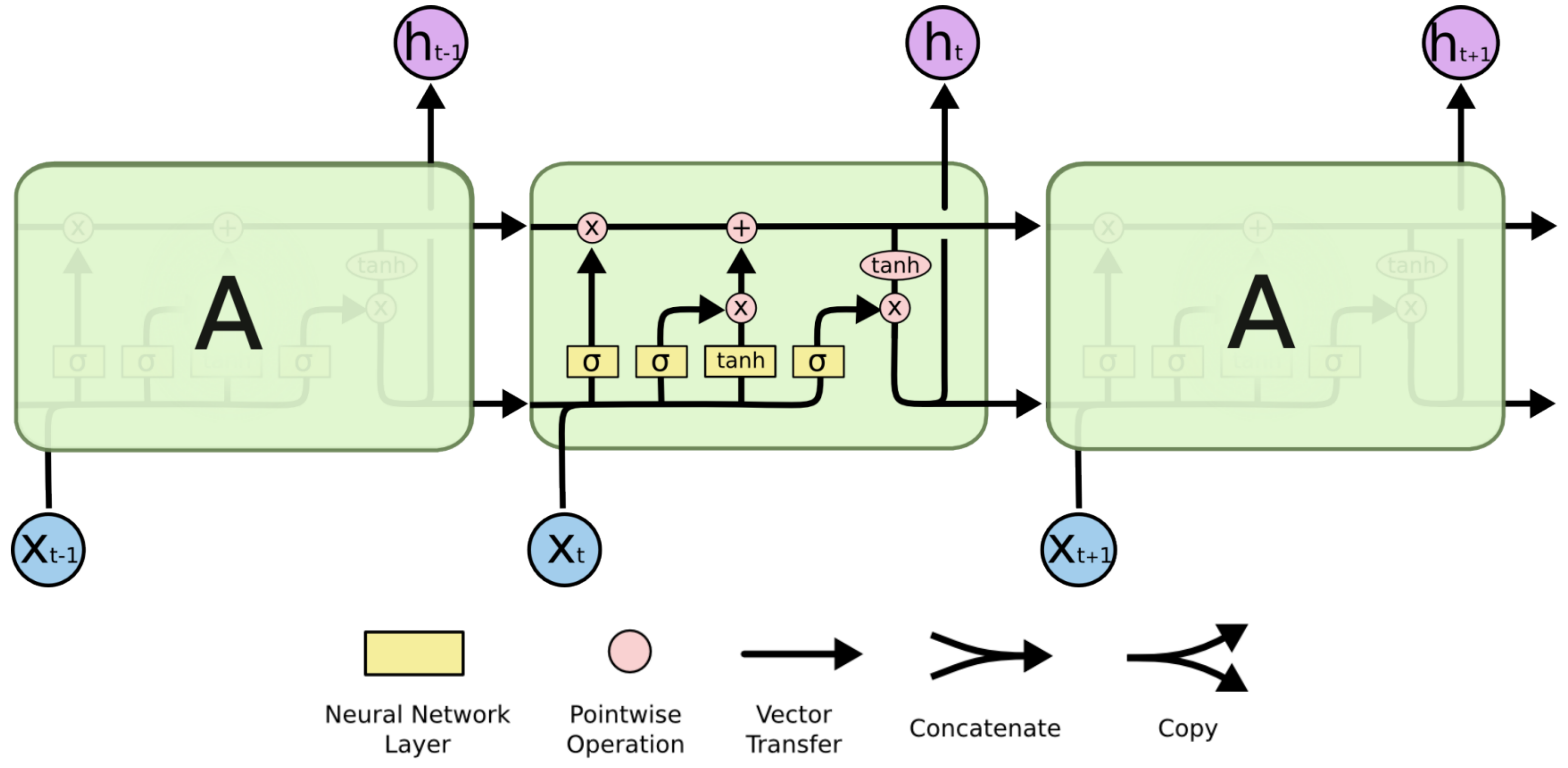


Figure: Intervals of LSTM modules, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

Top horizontal line is the *cell state*

It is very easy for information to flow through unchanged

LSTM has the ability to add or remove information from the cell state using *gates*

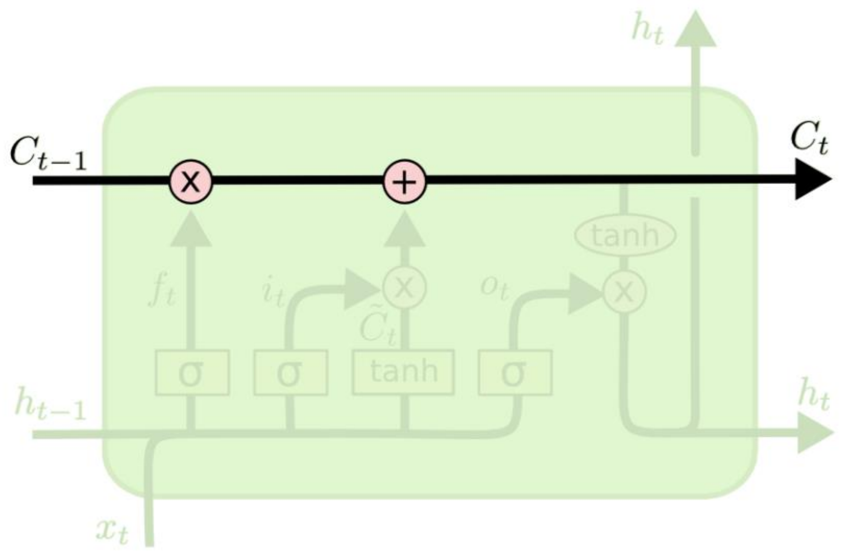


Figure: LSTM cell state, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

Gates are composed of a *sigmoid layer* and a pointwise multiplication operation.

The output of sigmoid is between 0 and 1, so it can control how much of its information passes through

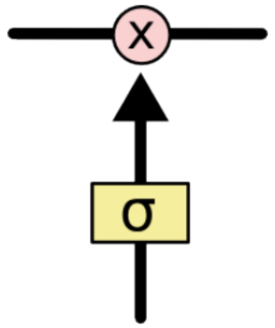
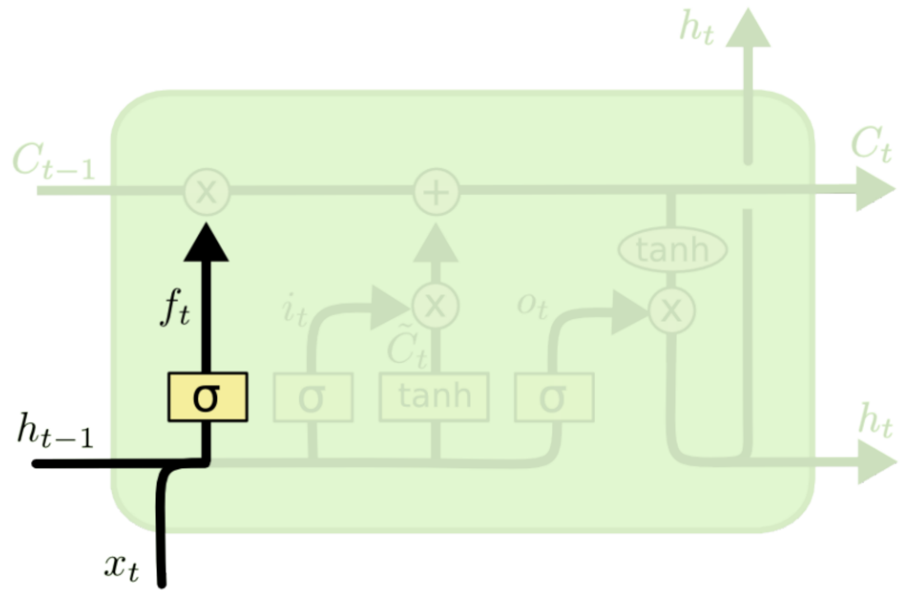


Figure: LSTM gate, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

The first layer in LSTM decides what information to throw away from cell state. The decision is made using a sigmoid gate called *forget gate* layer

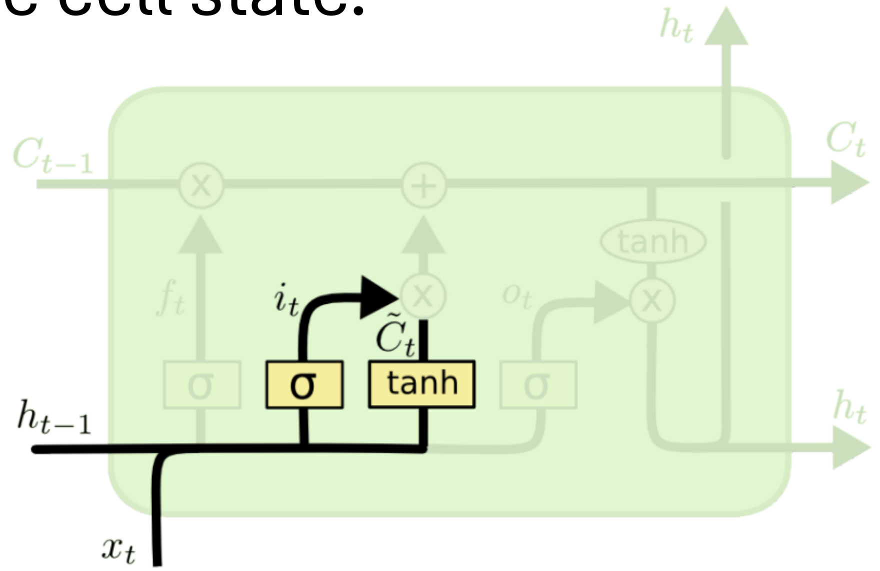


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure: LSTM forget gate layer, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

The second layer in LSTM decides what new information to store in the cell state. A sigmoid input gate layer decides which values to update, and a tanh layer creates a vector of new candidate values to be added to the cell state.

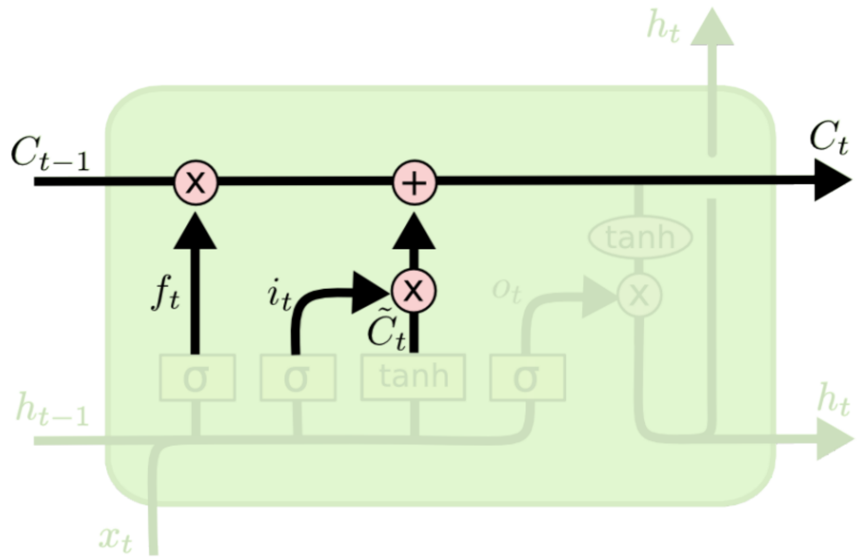


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure: LSTM input gate layer and candidate values, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

The cell state is updated based on the results of the aforementioned layers.



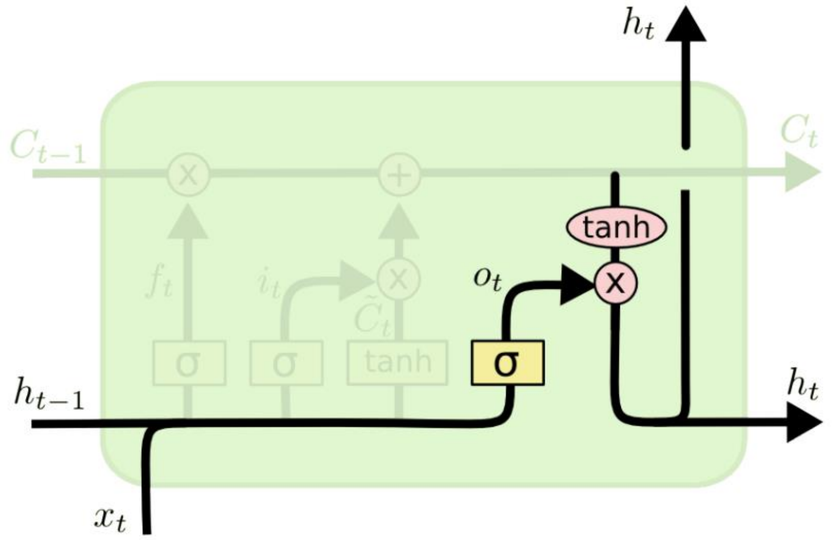
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure: LSTM cell state update, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long- Short-Term Memory Networks (LSTMs)

The output of the LSTM is determined based on the cell state.

First, a sigmoid layer decides which parts of the cell state are used for the output. Then, the cell state values go through a tanh layer and are multiplied by the output of the sigmoid gate.



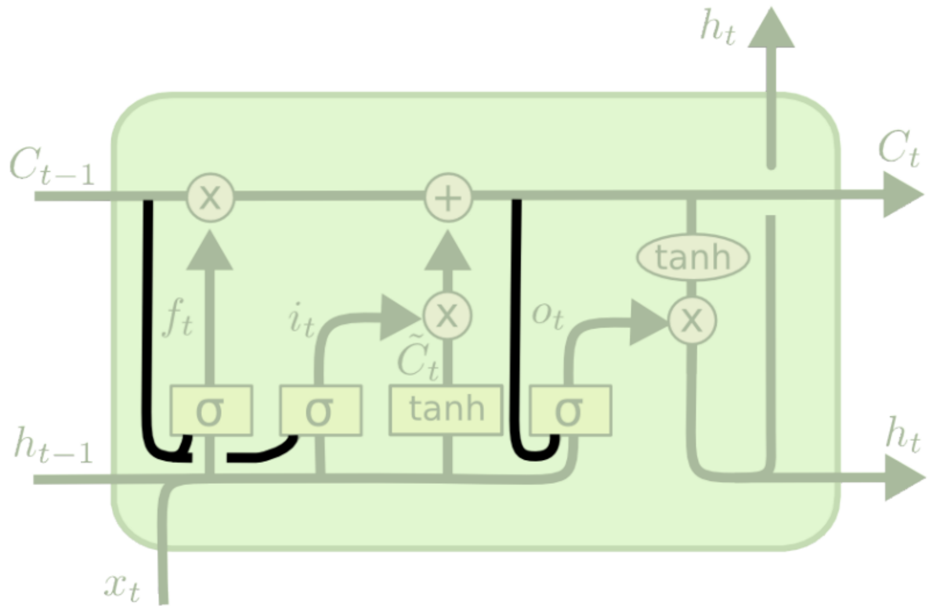
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure: LSTM output, image taken from:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs with Peephole Connections

One variant of LSTM adds peepholes to all gates, so that gates also take the cell state into account.

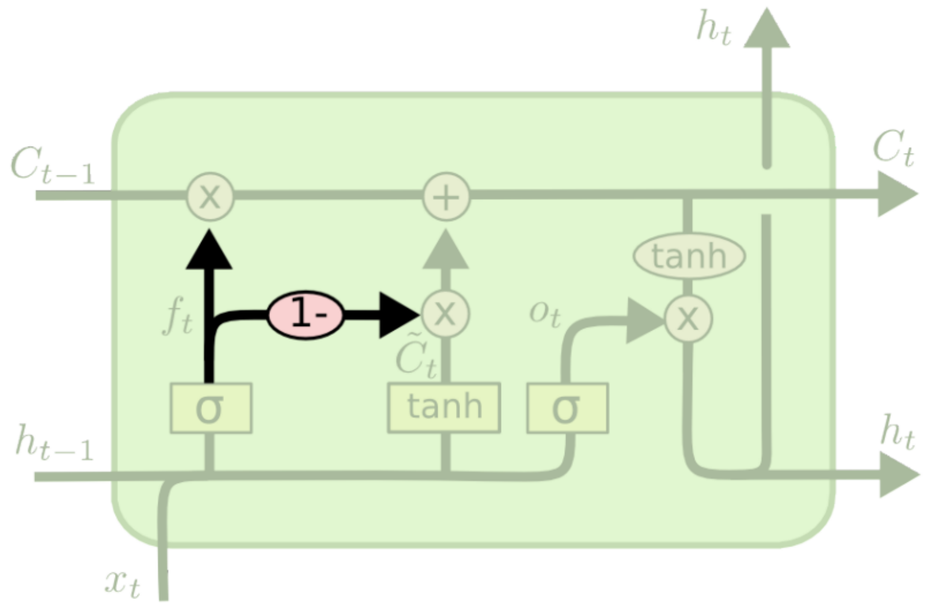


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Figure: LSTM with peephole connections, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs with Coupled Forget and Input Gates

Another variant of LSTM makes the decisions about what to forget and what to add to the cell state together instead of separately.

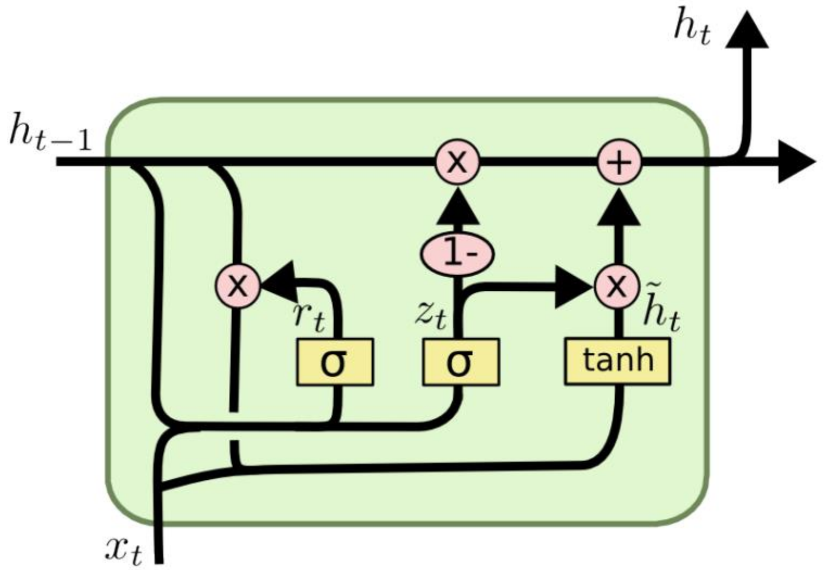


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Figure: LSTM with coupled forget and input gates, image taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

GRU combines the forget and input gates in LSTM into a single update gate. GRU also merges cell state and hidden state, resulting in a simpler model than LSTM.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

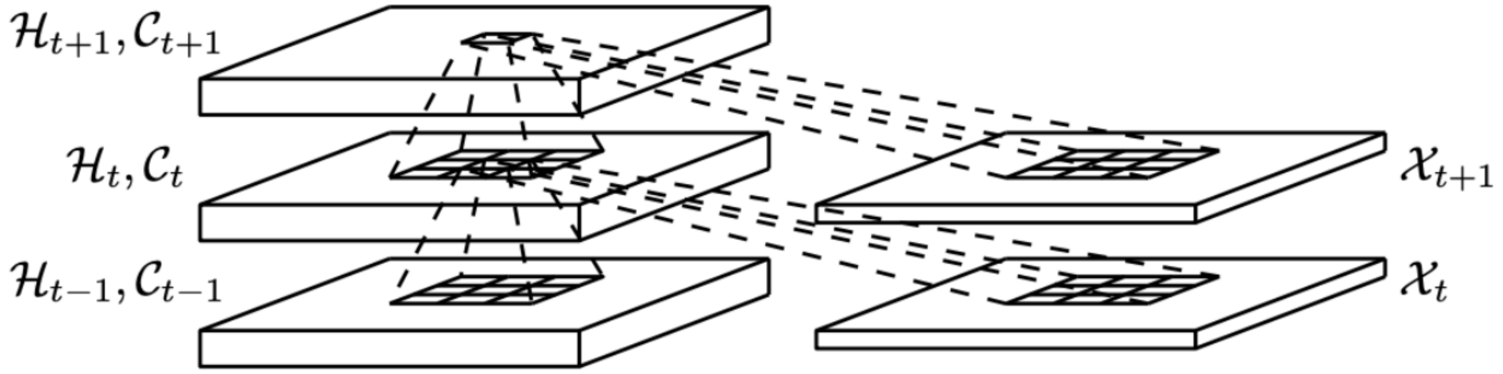
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure: GRU, image taken from:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Convolutional LSTM (ConvLSTM)

ConvLSTM replaces the matrix multiplications in LSTM with convolution operation. ConvLSTM is commonly used for processing spatio-temporal data such as videos.



$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_{t}, \mathcal{C}_{t}$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_{t}$

Figure: Image taken from: Shi, Xingjian, et al., "Convolutional LSTM network: A machine learning approach for precipitation nowcasting", NIPS 2015

Convolutional LSTM (ConvLSTM)

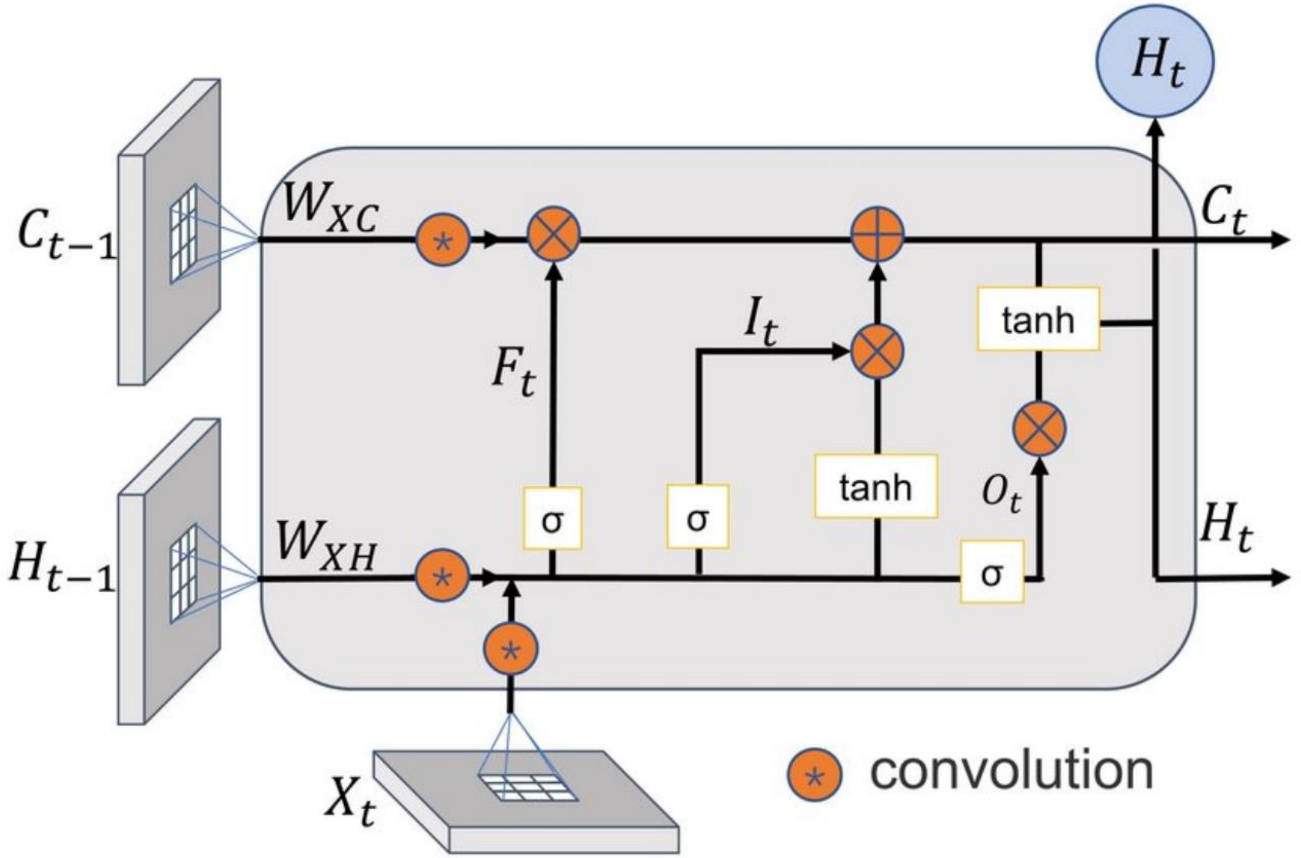
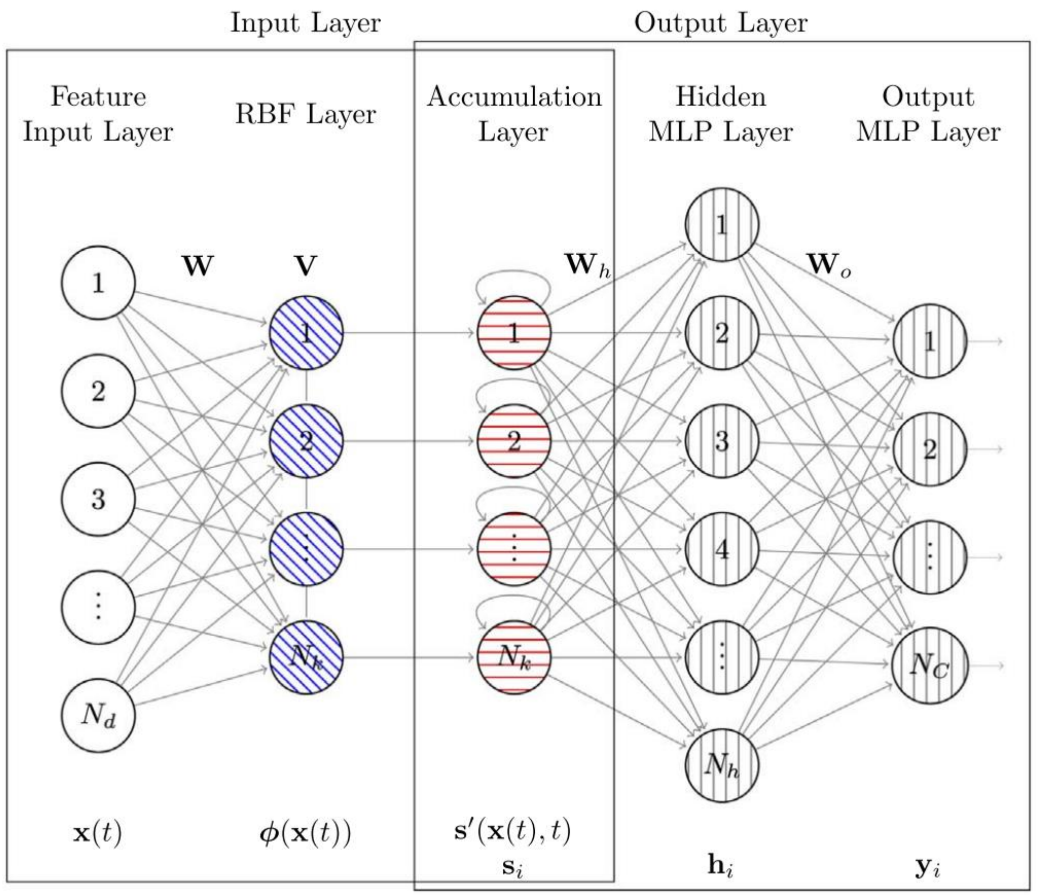


Figure: Image taken from: Shi et al., "Learning Multiscale Temporal-Spatial-Spectral Features via a Multipath Convolutional LSTM Neural Network for Change Detection with Hyperspectral Images", IEEE Trans. Geosc. And Rem. Sens, 2022

Neural Bag of Features (NBoFs)

NBoFs considers the sequence as a *set of items*:

- It is formed by three types of layers:
    - A quantization layer
    - An accumulation layer
    - MLP layers



Passalis, Tefas, "Neural Bag-of-Features Learning", Pattern Recognition, 2017

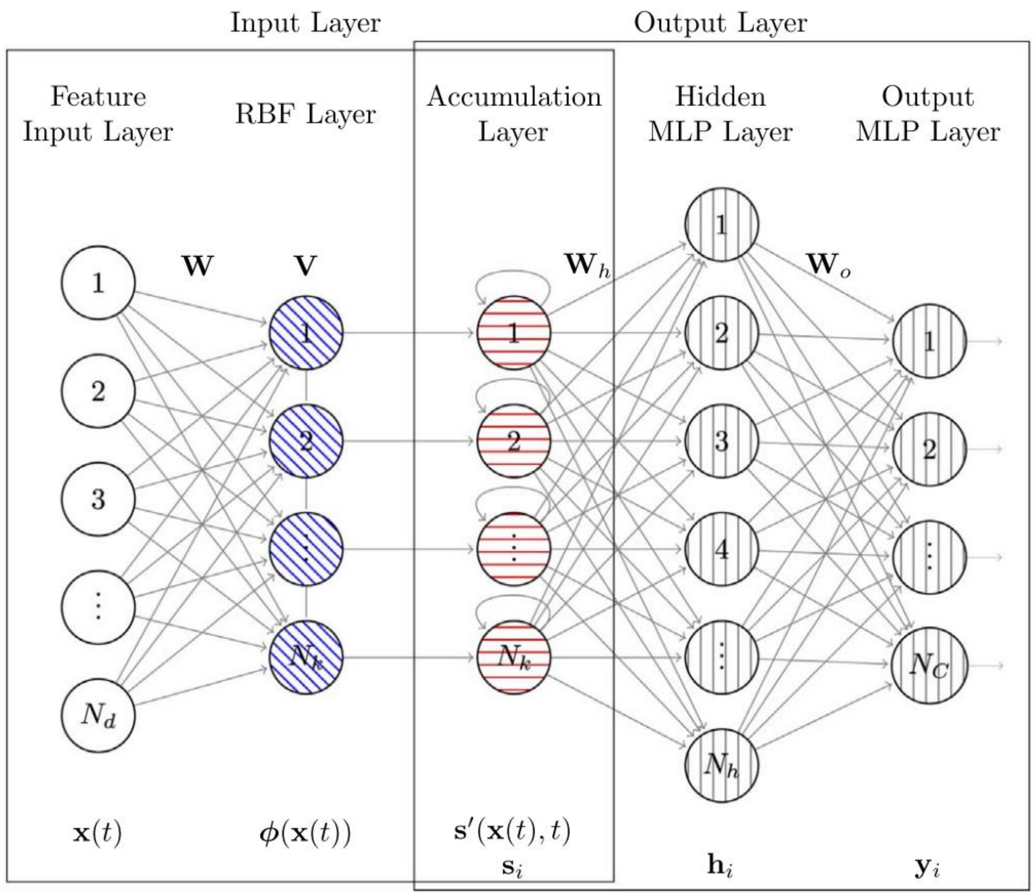
Neural Bag of Features (NBoFs)

Quantization layer:

- Input sequence $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$
- K neurons (codewords) in the RBF layer with $\mathbf{v}_k \in \mathbb{R}^D$ being the k_{th} codeword
- Output of the kth RBF neuron is:

$$\phi_{n,k} = \frac{\exp\big(-\|(\mathbf{x}_n - \mathbf{v_k}) \odot \mathbf{w}_k\|_2\big)}{\sum_{m=1}^{K} \exp\big(-\|(\mathbf{x}_n - \mathbf{v}_m) \odot \mathbf{w}_m\|_2\big)}$$

  with $\odot$ being the element-wise product
- $\mathbf{x}_n$ is mapped to the quantized vector $\boldsymbol{\phi}_n$

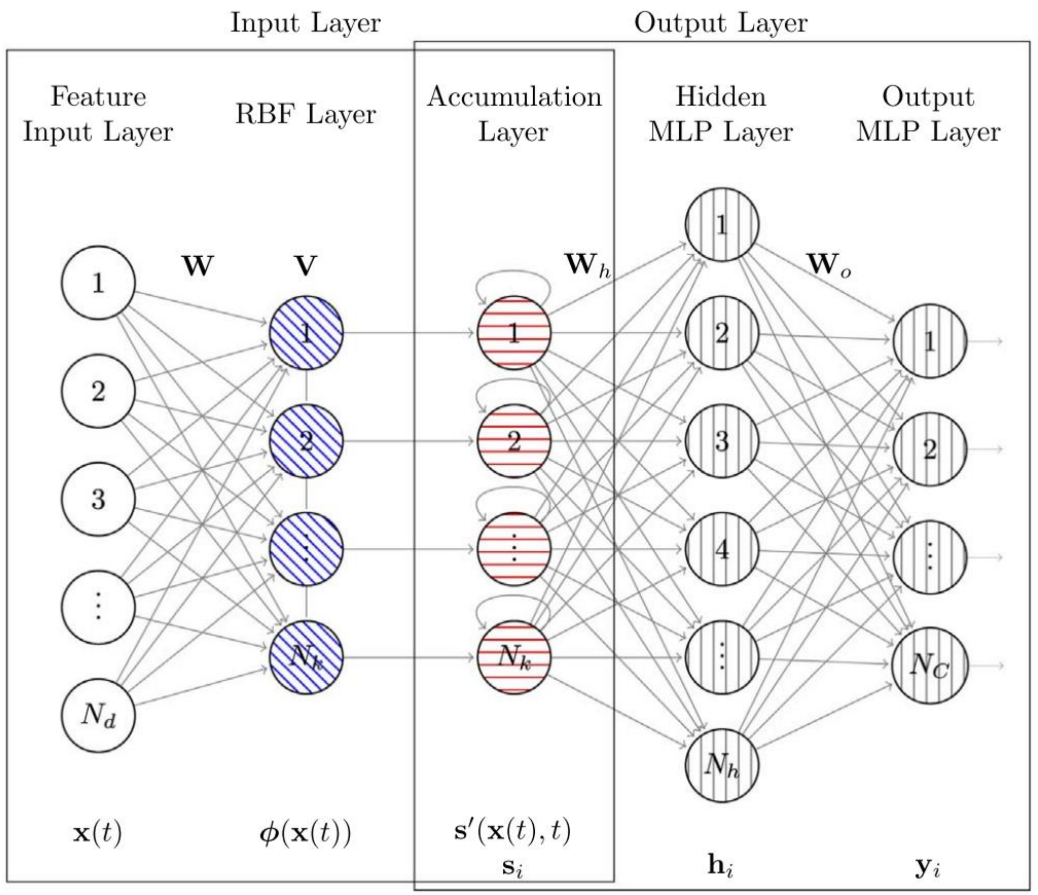


Neural Bag of Features (NBoFs)

Accumulation layer:

- Input $\mathbf{\Phi} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N] \in \mathbb{R}^{K \times N}$
- Aggregates the information in $\mathbf{\Phi}$ by:

$$\mathbf{y} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\phi}_n$$



Attention-based NBoFs

2D-Attenntion:

- If the sequence data is represented by a matrix $\mathbf{S} \in \mathbb{R}^{M \times N}$, 2DA returns a matrix $\tilde{\mathbf{S}} \in \mathbb{R}^{M \times N}$:

$$\tilde{\mathbf{S}} = \mathcal{F}_{2\text{DA}}(\mathbf{S})$$

- Selection or rejection of the columns of **S** is done by element-wise matrix multiplication:

$$\tilde{\mathbf{S}} = \tau(\mathbf{S} \odot \mathbf{A}) + (1 - \tau)\mathbf{S}$$

where the attention mask $\mathbf{A} \in \mathbb{R}^{M \times N}$ has values in the range [0,1]

Tran, Passalis, Tefas, Gabbouj, Iosifidis, “Attention-based Neural Bag-of-Features Learning for Sequence Data”, IEEE Access, 2022

Attention-based NBoFs

2D-Attenntion:

- If the sequence data is represented by a matrix $\mathbf{S} \in \mathbb{R}^{M \times N}$, 2DA returns a matrix $\tilde{\mathbf{S}} \in \mathbb{R}^{M \times N}$:

$$\tilde{\mathbf{S}} = \mathcal{F}_{2\mathrm{DA}}(\mathbf{S})$$

- Selection or rejection of the columns of **S** is done by element-wise matrix multiplication:

$$\tilde{\mathbf{S}} = \tau(\mathbf{S} \odot \mathbf{A}) + (1 - \tau)\mathbf{S}$$

Learnable parameters

where the attention mask $\mathbf{A} \in \mathbb{R}^{M \times N}$ has values in the range [0,1]

Tran, Passalis, Tefas, Gabbouj, Iosifidis, “Attention-based Neural Bag-of-Features Learning for Sequence Data”, IEEE Access, 2022

Attention-based NBoFs

2D-Attenntion:

- If the sequence data is represented by a matrix $\mathbf{S} \in \mathbb{R}^{M \times N}$, 2DA returns a matrix $\tilde{\mathbf{S}} \in \mathbb{R}^{M \times N}$:

$$\tilde{\mathbf{S}} = \mathcal{F}_{2\text{DA}}(\mathbf{S})$$

- Selection or rejection of the columns of **S** is done by element-wise matrix multiplication:

$$\tilde{\mathbf{S}} = \tau(\mathbf{S} \odot \mathbf{A}) + (1 - \tau)\mathbf{S}$$

  where the attention mask $\mathbf{A} \in \mathbb{R}^{M \times N}$ has values in the range [0,1]

- A takes the form: **A** = softmax(**Z**), **Z** = **SW**

W is a learnable matrix
Its diagonal elements are fixed to 1/N

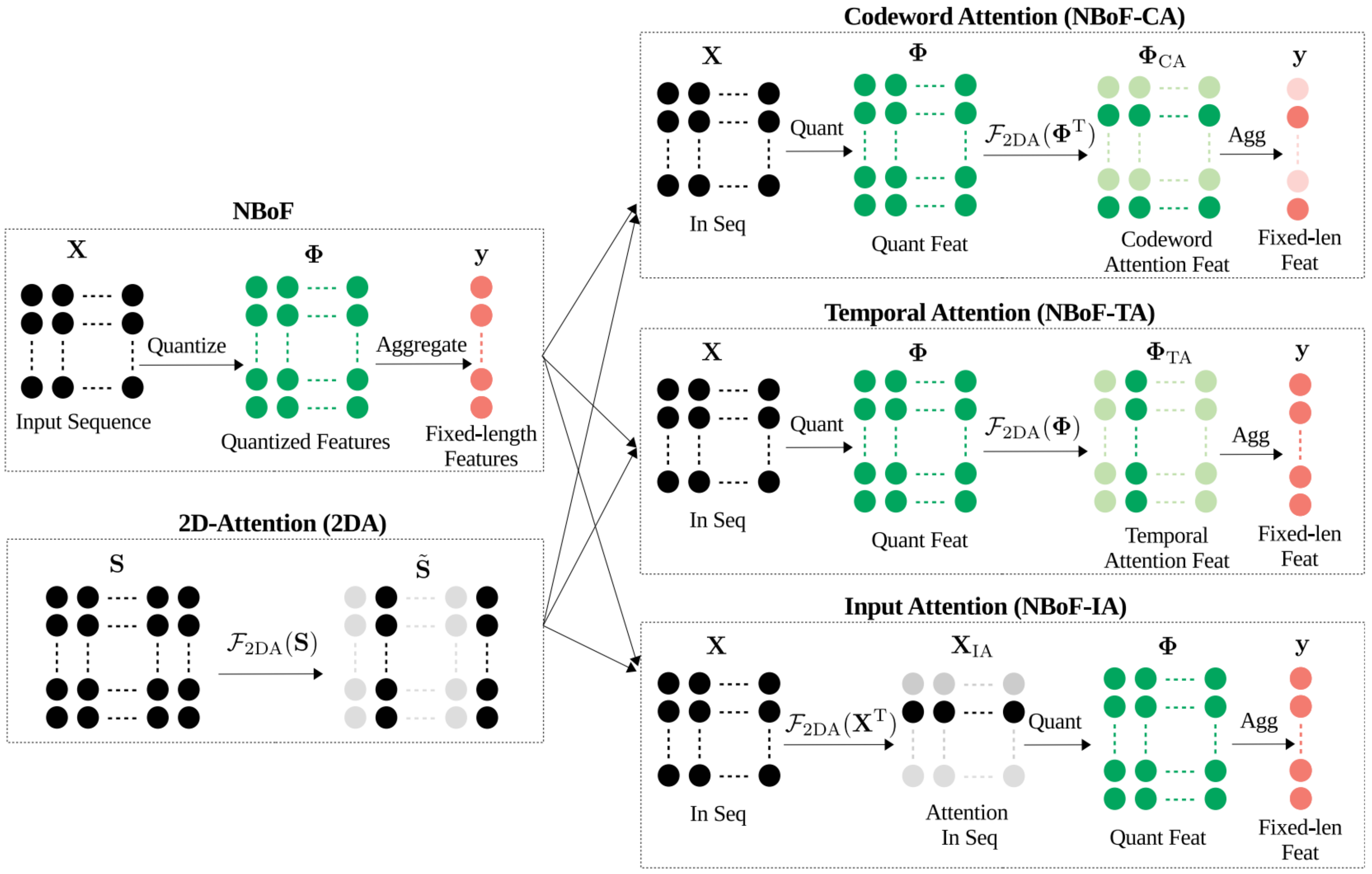
Attention-based NBoFs

Given the quantized features $\Phi \in \mathbb{R}^{K \times N}$:

- Codeword Attention: $\Phi_{\mathrm{CA}} = \mathcal{F}_{\mathrm{2DA}}(\Phi^{\mathrm{T}})$
- Temporal Attention: $\Phi_{\mathrm{TA}} = \mathcal{F}_{\mathrm{2DA}}(\Phi)$
- Input Attention: $\mathbf{X}_{\mathrm{IA}} = \mathcal{F}_{\mathrm{2DA}}(\mathbf{X}^{\mathrm{T}})$

Tran, Passalis, Tefas, Gabbouj, Iosifidis, "Attention-based Neural Bag-of-Features Learning for Sequence Data", IEEE Access, 2022

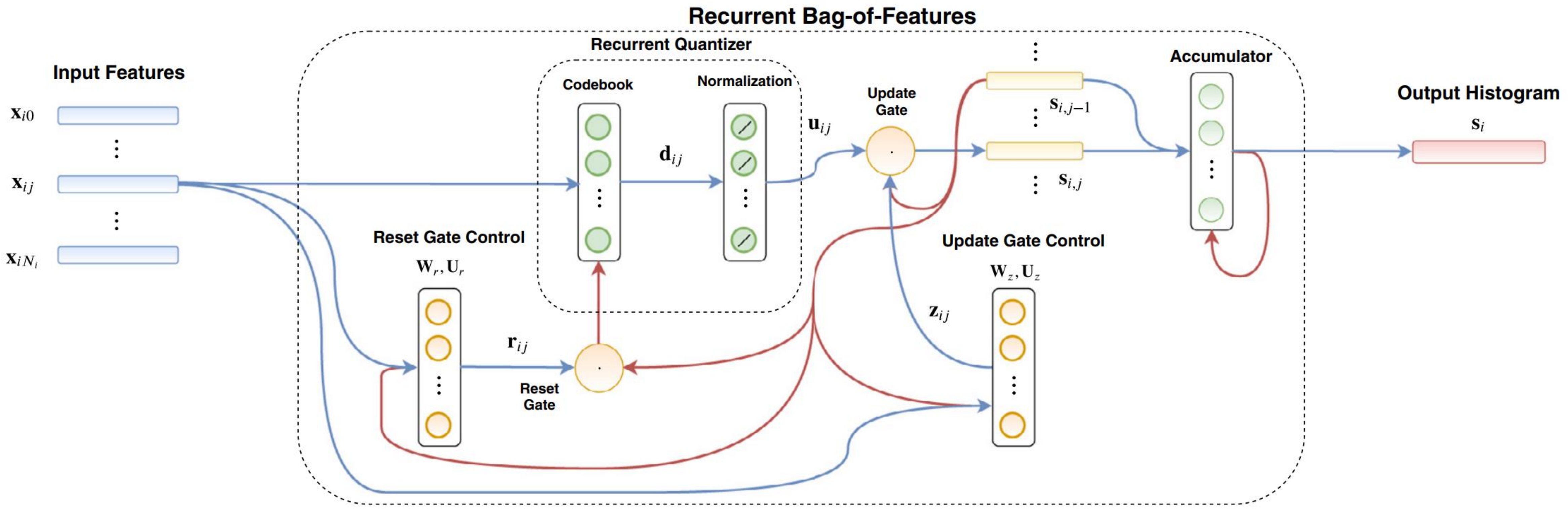
Attention-based NBoFs



Tran, Passalis, Tefas, Gabbouj, Iosifidis, “Attention-based Neural Bag-of-Features Learning for Sequence Data”, IEEE Access, 2022

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation



Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation:

- Recurrent quantizer: $\mathbf{d}_{ij} = \sigma\left(\mathbf{V}\mathbf{x}_{ij} + \mathbf{V}_h(\mathbf{r}_{ij} \odot \mathbf{s}_{i,j-1})\right) \in \mathbb{R}^{N_K}$

Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation:

- Recurrent quantizer: $\mathbf{d}_{ij} = \sigma\left(\mathbf{V}\mathbf{x}_{ij} + \mathbf{V}_h(\mathbf{r}_{ij} \odot \mathbf{s}_{i,j-1})\right) \in \mathbb{R}^{N_K}$
- Reset gate (inspired by GRU): $\mathbf{r}_{ij} = \sigma(\mathbf{W}_r\mathbf{x}_{ij} + \mathbf{U}_r\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$

Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation:

- Recurrent quantizer: $\mathbf{d}_{ij} = \sigma\left(\mathbf{V}\mathbf{x}_{ij} + \mathbf{V}_h(\mathbf{r}_{ij} \odot \mathbf{s}_{i,j-1})\right) \in \mathbb{R}^{N_K}$
- Reset gate (inspired by GRU): $\mathbf{r}_{ij} = \sigma(\mathbf{W}_r\mathbf{x}_{ij} + \mathbf{U}_r\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$
- Recurrent histogram calculation: $\mathbf{s}_{i,j} = (\mathbf{1} - \mathbf{z}_{ij}) \odot \mathbf{s}_{i,j-1} + \mathbf{z}_{ij} \odot \mathbf{u}_{ij} \in \mathbb{R}^{N_K}$

Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation:

- Recurrent quantizer: $\mathbf{d}_{ij} = \sigma\left(\mathbf{V}\mathbf{x}_{ij} + \mathbf{V}_h(\mathbf{r}_{ij} \odot \mathbf{s}_{i,j-1})\right) \in \mathbb{R}^{N_K}$
- Reset gate (inspired by GRU): $\mathbf{r}_{ij} = \sigma(\mathbf{W}_r\mathbf{x}_{ij} + \mathbf{U}_r\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$
- Recurrent histogram calculation: $\mathbf{s}_{i,j} = (\mathbf{1} - \mathbf{z}_{ij}) \odot \mathbf{s}_{i,j-1} + \mathbf{z}_{ij} \odot \mathbf{u}_{ij} \in \mathbb{R}^{N_K}$
- Update gate: $\mathbf{z}_{ij} = \sigma(\mathbf{W}_z\mathbf{x}_{ij} + \mathbf{U}_z\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$

Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Recurrent NBoFs

It incorporates a recurrent mechanism in NBoFs calculation:

- Recurrent quantizer: $\mathbf{d}_{ij} = \sigma\left(\mathbf{V}\mathbf{x}_{ij} + \mathbf{V}_h(\mathbf{r}_{ij} \odot \mathbf{s}_{i,j-1})\right) \in \mathbb{R}^{N_K}$
- Reset gate (inspired by GRU): $\mathbf{r}_{ij} = \sigma(\mathbf{W}_r\mathbf{x}_{ij} + \mathbf{U}_r\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$
- Recurrent histogram calculation: $\mathbf{s}_{i,j} = (\mathbf{1} - \mathbf{z}_{ij}) \odot \mathbf{s}_{i,j-1} + \mathbf{z}_{ij} \odot \mathbf{u}_{ij} \in \mathbb{R}^{N_K}$
- Update gate: $\mathbf{z}_{ij} = \sigma(\mathbf{W}_z\mathbf{x}_{ij} + \mathbf{U}_z\mathbf{s}_{i,j-1}) \in \mathbb{R}^{N_K}$
- Sequence-level histogram calculation: $\mathbf{s}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} \mathbf{s}_{i,j} \in \mathbb{R}^{N_K}$

Krestenitis, Passalis, Iosifidis, Gabbouj, Tefas, “Recurrent bag-of-features for visual information analysis”, Pattern Recognition 2020

Bilinear Networks

A sequence can be represented as a matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{T_l}] \in \mathbb{R}^{D \times T}$

Tran, Kanniainen, Gabbouj, Iosifidis, “Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis”, IEEE Transactions on Neural Networks and Learning Systems, 2019

Bilinear Networks

A sequence can be represented as a matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{T_l}] \in \mathbb{R}^{D \times T}$

A bilinear layer (BL) transforms the input of size D x T to a new matrix of size D’ x T’ by:

$$\mathbf{Y} = \phi(\underbrace{\mathbf{W}_1}_{\text{Feature transformation}} \mathbf{X} \underbrace{\mathbf{W}_2}_{\text{Temporal transformation}} + \underbrace{\mathbf{B}}_{\text{Bias term}})$$

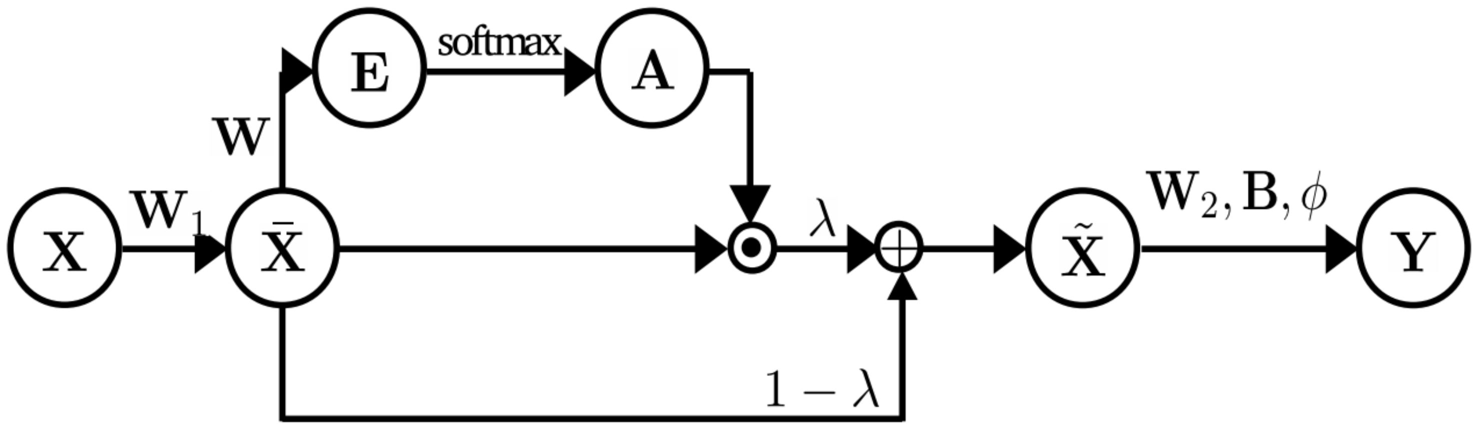
Tran, Kanniainen, Gabbouj, Iosifidis, “Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis”, IEEE Transactions on Neural Networks and Learning Systems, 2019

Temporal Attention-Augmented Bilinear Networks (TABL)

Add a parallel transformation of X which learns how different elements of the transformed time-series features should interact:

$$\begin{aligned}
\bar{\mathbf{X}} &= \mathbf{W}_1 \mathbf{X} \\
\mathbf{E} &= \bar{\mathbf{X}} \mathbf{W} \\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \\
\tilde{\mathbf{X}} &= \lambda(\bar{\mathbf{X}} \odot \mathbf{A}) + (1-\lambda)\bar{\mathbf{X}} \\
\mathbf{Y} &= \phi(\tilde{\mathbf{X}} \mathbf{W}_2 + \mathbf{B})
\end{aligned}$$

Diagonal elements of **W** are fixed to 1/T



Tran, Kanniainen, Gabbouj, Iosifidis, "Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis", IEEE Transactions on Neural Networks and Learning Systems, 2019