UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

# Chapter 3-1. Recurrent Neural Networks

Neural Networks

2023/2024

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

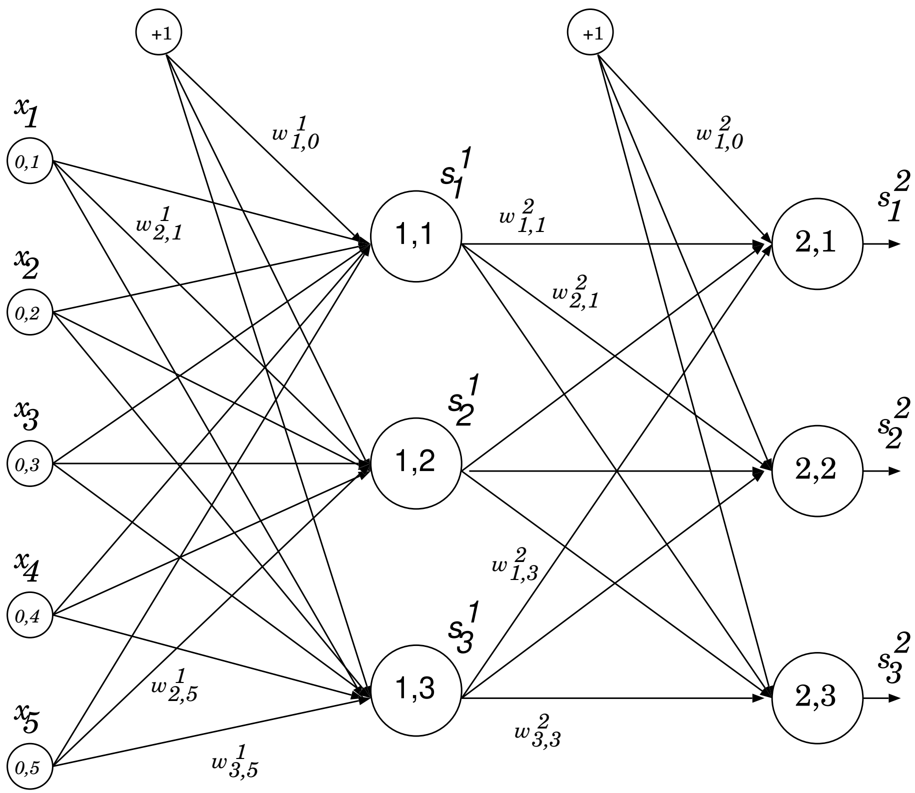
Departamento de Sistemas Informáticos y Computación

# Index

# Index

# Introduction

A feed-forward network




$$\mathbf{g} : \mathbb{R}^{M_0} \rightarrow \mathbb{R}^{M_2}$$

$$\begin{aligned} g_k(\mathbf{x};\theta) &\equiv s^2_k(\mathbf{x}) \\ &= f(\sum_{j=0}^{M_1} w^2_{k,j}\ s^1_j(\mathbf{x})) \\ &= f(\sum_{j=0}^{M_1} w^2_{k,j}\ f(\sum_{j'=0}^{M_0} w^1_{jj'}\ x_{j'})), \end{aligned}$$

$$1 \leq k \leq M_2$$

A compact notation: $\mathbf{g}(\mathbf{x};\theta) = \mathbf{f}(\mathbf{W}^2\ \mathbf{s}^1(\mathbf{x})) = \mathbf{f}(\mathbf{W}^2\ \mathbf{f}(\mathbf{W}^1\ \mathbf{x}))$

($\mathbf{f}$ is an extension of $f$ from scalars to vectors)

Problem: The representation of the objects should be of fixed size

# Introduction

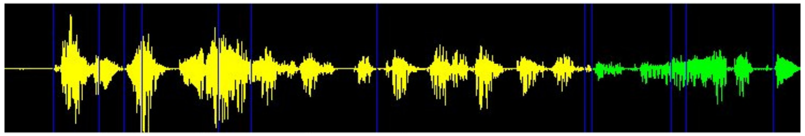
- Problem: The input must be numeric (e.g. from a vector space). What happens when the object are discrete? (for example words)

- Problem: The representation of the objects should be of fixed size. What happens when the object are of variable size? (for example sequence of subobjects)

# Index

# Sequence processing

- Speech recognition and understanding

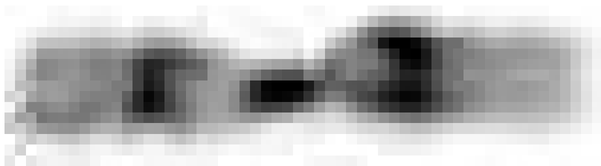


- Text translation

“¿ hay alguna habitación tranquila libre ?” / “is there a quiet room available” ?

- Handwritten text recognition

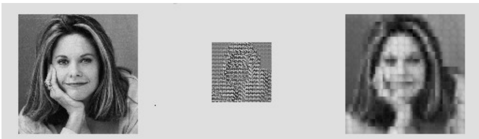

- Currency change
- Classification of chromosomes



- Weather forecast
- Protein sequences


- Video sequences (Video description, tracking, ...)

- Image processing (compression, description, VQA, ...)

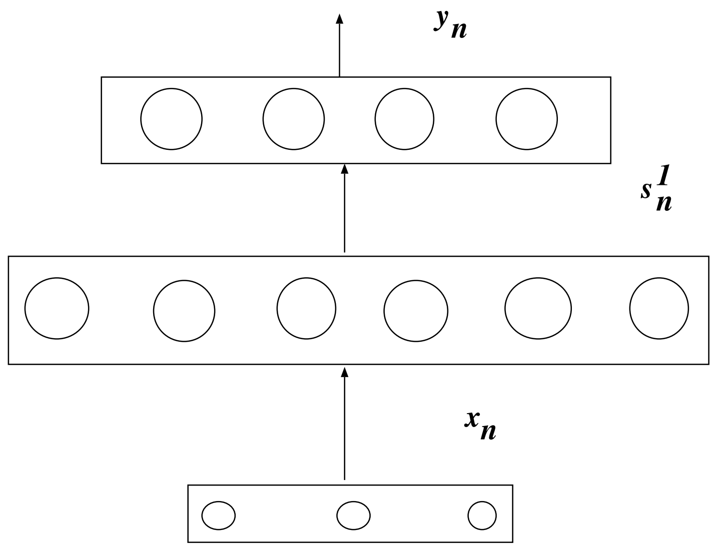


# Sequence processing

- Feed-forward networks cannot process directly sequences of variable length.
  - Naive solution:
    1. Define a maximum length (for example $N$ vectors in $\mathbb{R}^D$)
    2. Transform each sequence $\mathbf{x}$ into a vector in $\mathbb{R}^{N\,D}$ by linear or non-linear interpolation.
  - Alternative solutions:
    * Hybrid approaches: hidden Markov models and multilayer perceptrons.
    * Dynamic networks: bounded-memory networks and recurrent neural networks

# Sequence processing: dynamic networks

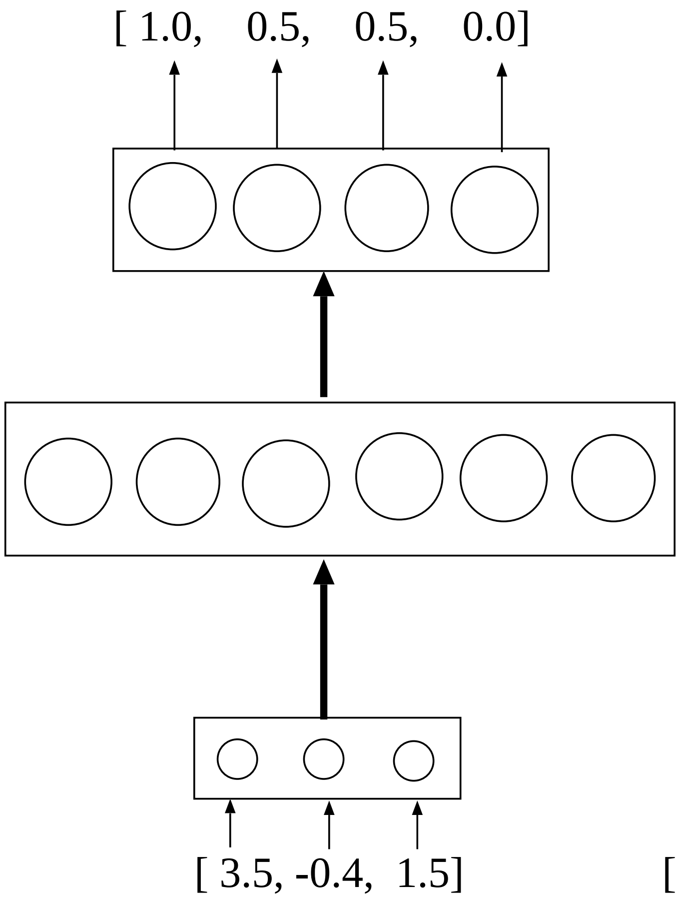
$$\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N \rightarrow \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N$$




$$1 \leq n \leq N$$

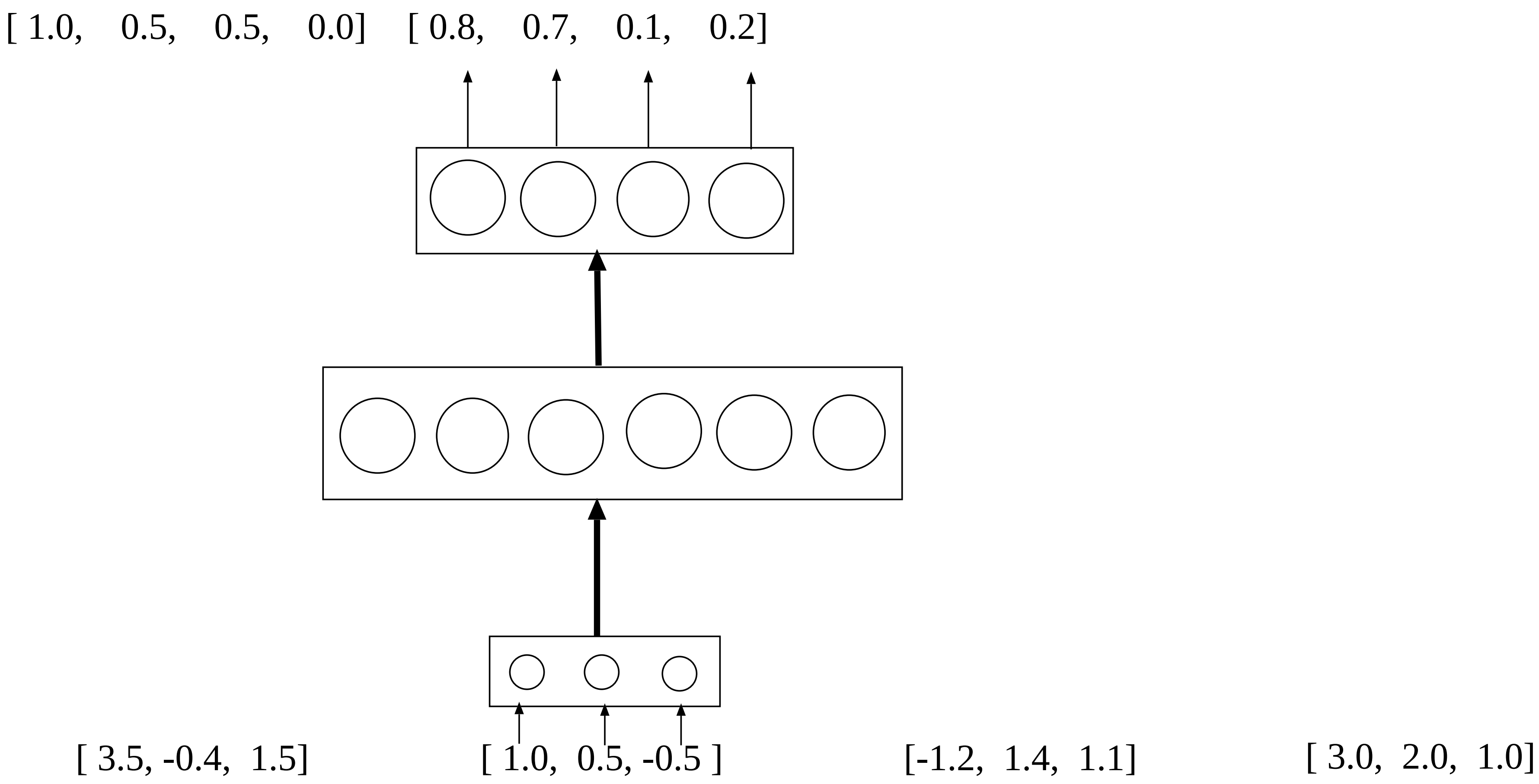
| $t$ | input layer | | hidden layer | | output layer |
|---|---|---|---|---|---|
| 1 | $\mathbf{x}_1$ | $\Rightarrow$ | $\mathbf{s}_1^1$ | $\Rightarrow$ | $\mathbf{y}_1$ |
| 2 | $\mathbf{x}_2$ | $\Rightarrow$ | $\mathbf{s}_2^1$ | $\Rightarrow$ | $\mathbf{y}_2$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ |
| $N$ | $\mathbf{x}_N$ | $\Rightarrow$ | $\mathbf{s}_N^1$ | $\Rightarrow$ | $\mathbf{y}_N$ |

# Sequence processing: dynamic networks

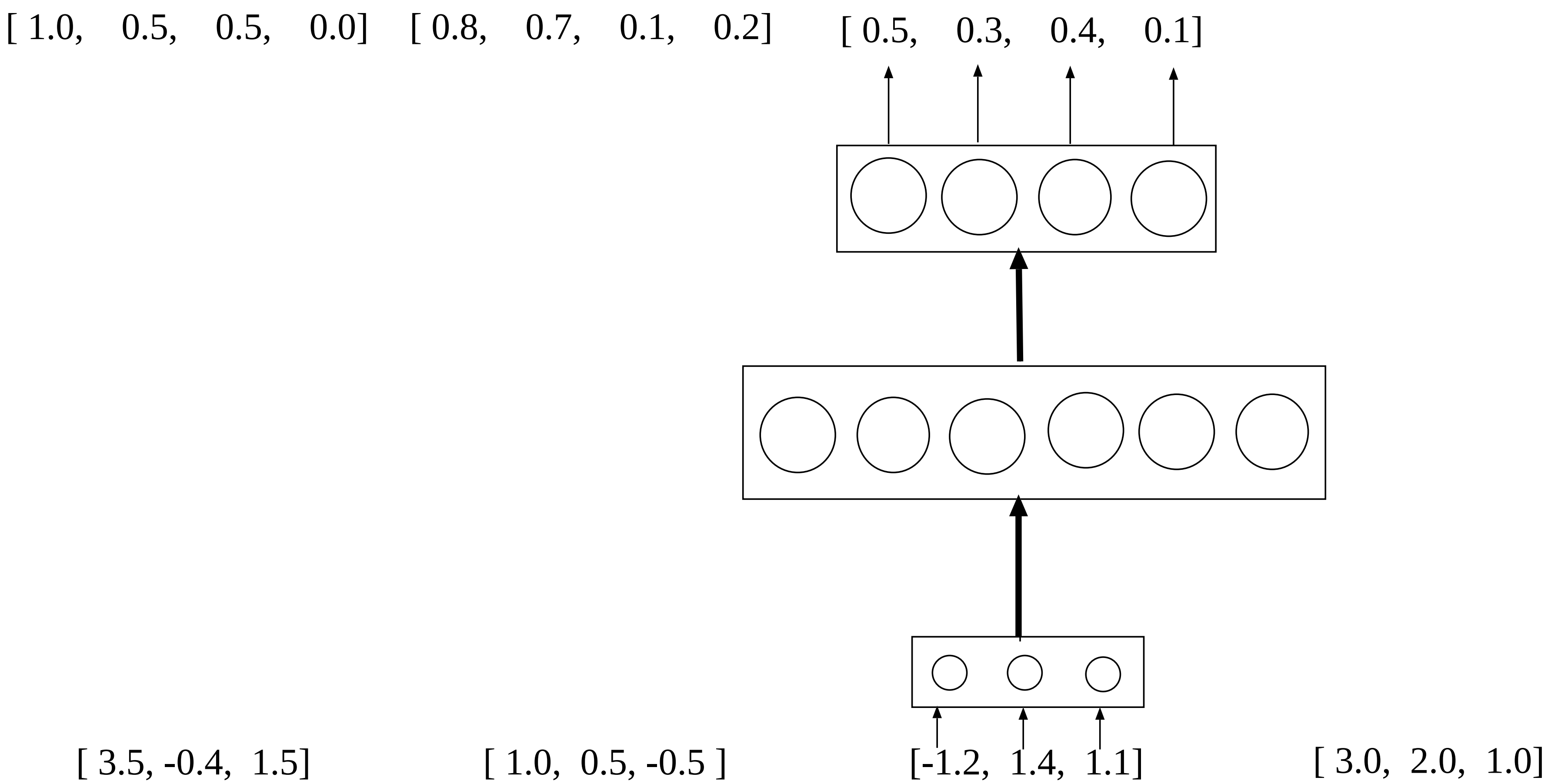



[ 1.0, 0.5, -0.5 ]  [-1.2, 1.4, 1.1]  [ 3.0, 2.0, 1.0]

# Sequence processing: dynamic networks

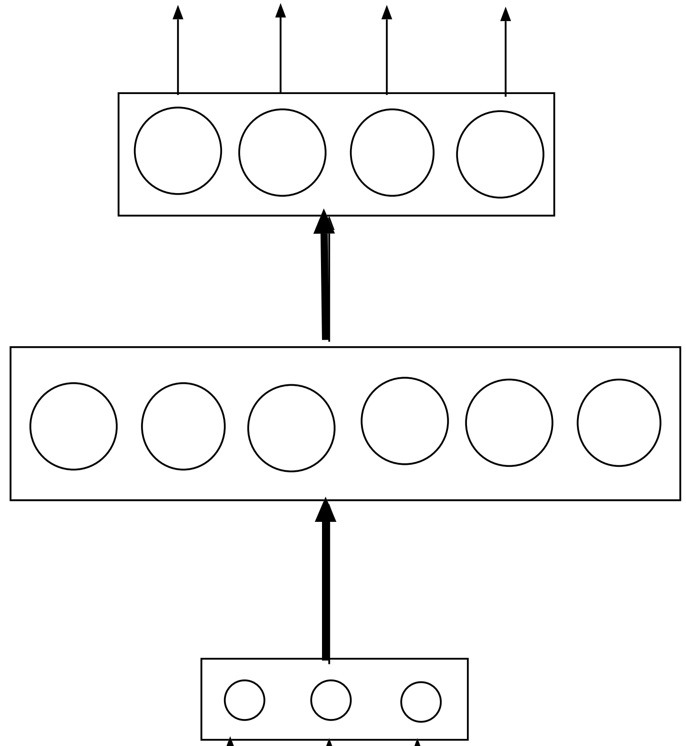


# Sequence processing: dynamic networks



# Sequence processing: dynamic networks

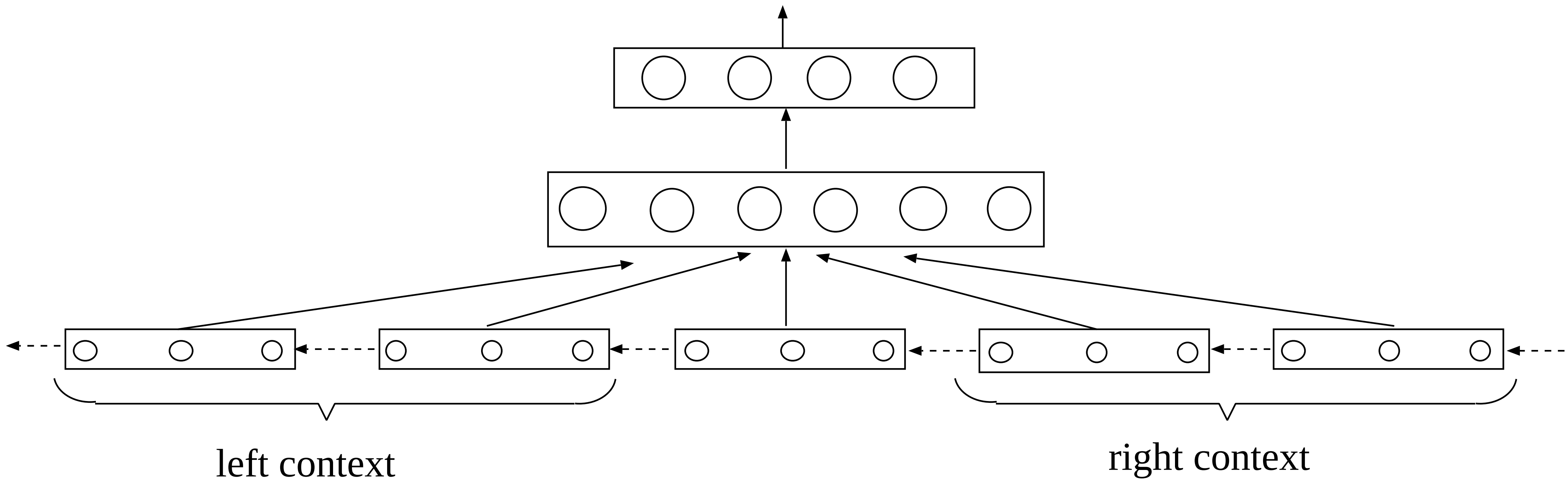
[ 1.0, 0.5, 0.5, 0.0] [ 0.8, 0.7, 0.1, 0.2] [ 0.5, 0.3, 0.4, 0.1] [ 0.1, 0.1, 0.7, 0.0]



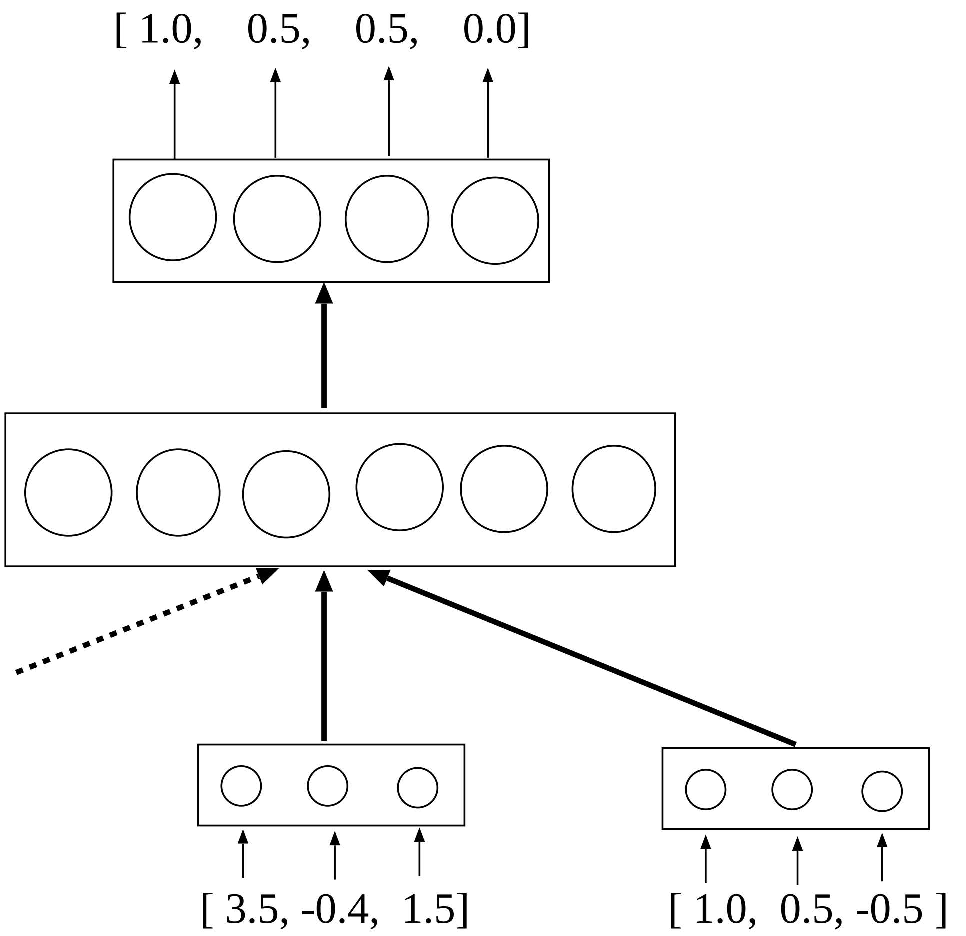
[ 3.5, -0.4, 1.5] [ 1.0, 0.5, -0.5 ] [-1.2, 1.4, 1.1] [ 3.0, 2.0, 1.0]

# Sequence processing: dynamic networks

**NETtalk** (a precedent of the modern convolutional neural networks -CNN-) that learns to pronounce written English text (text $\rightarrow$ phonetic transcription) (Sejnowski & Rosenberg. 1986)



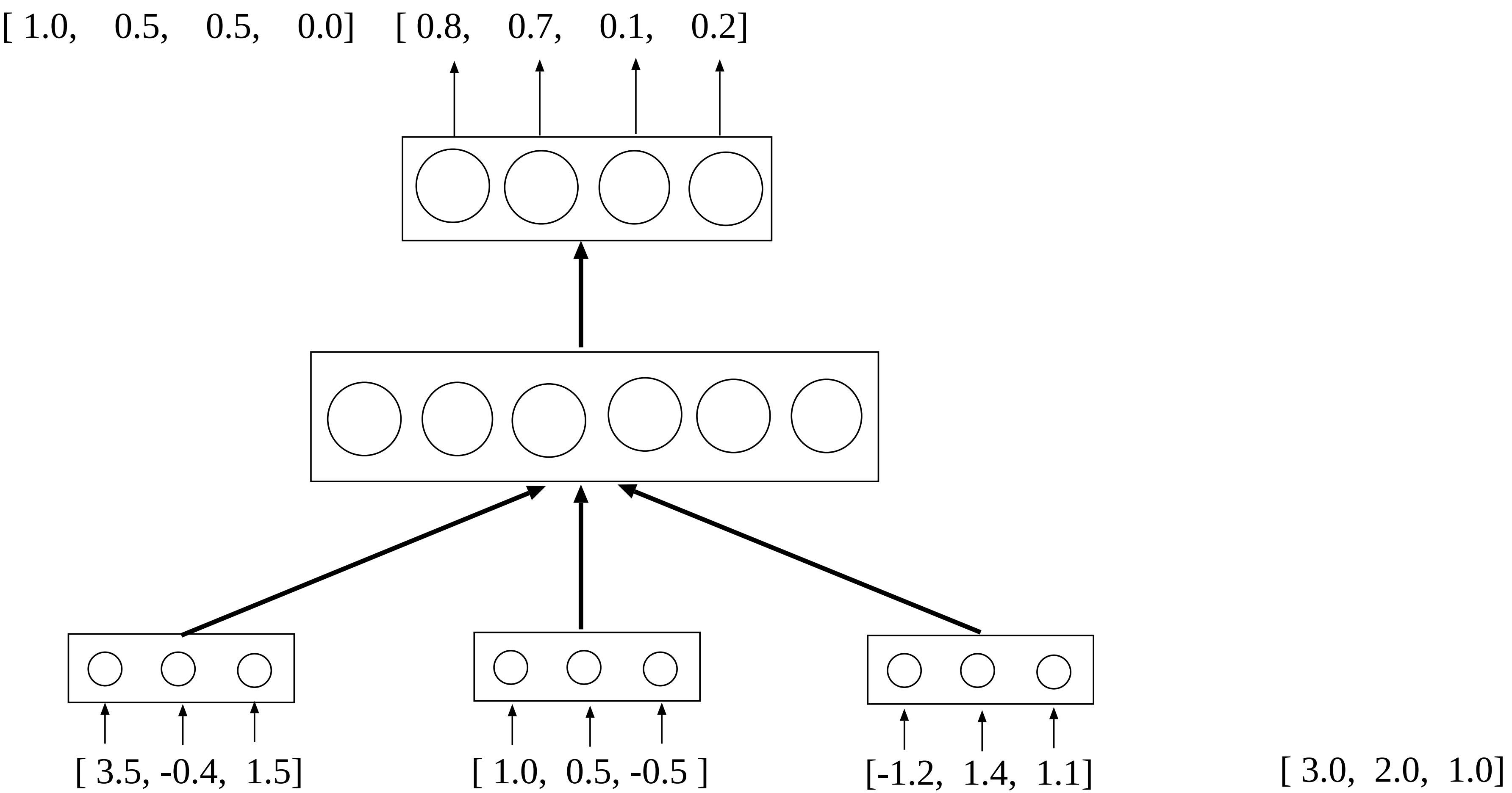
# Sequence processing: dynamic networks



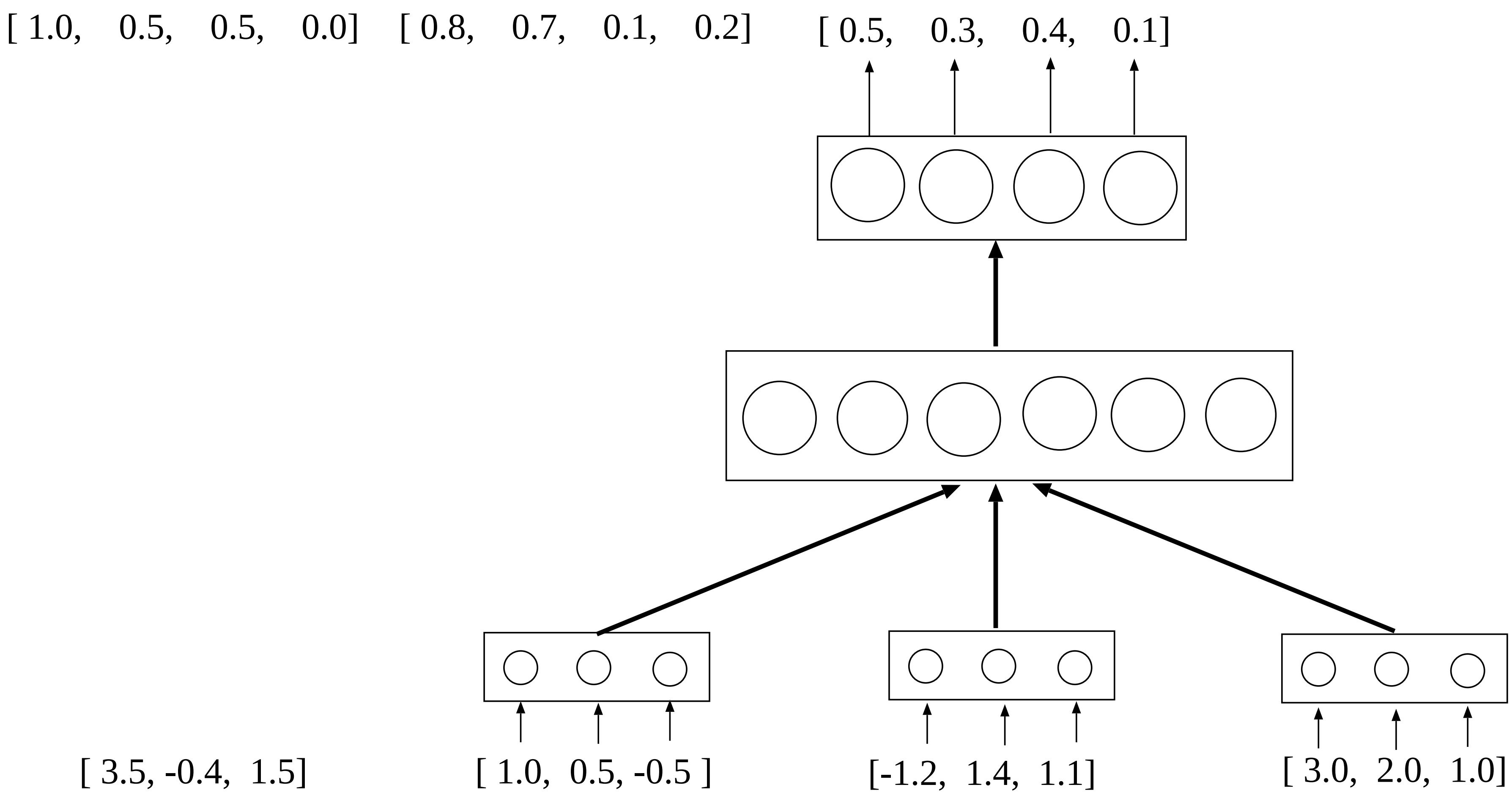

[-1.2, 1.4, 1.1]

[ 3.0, 2.0, 1.0]

# Sequence processing: dynamic networks

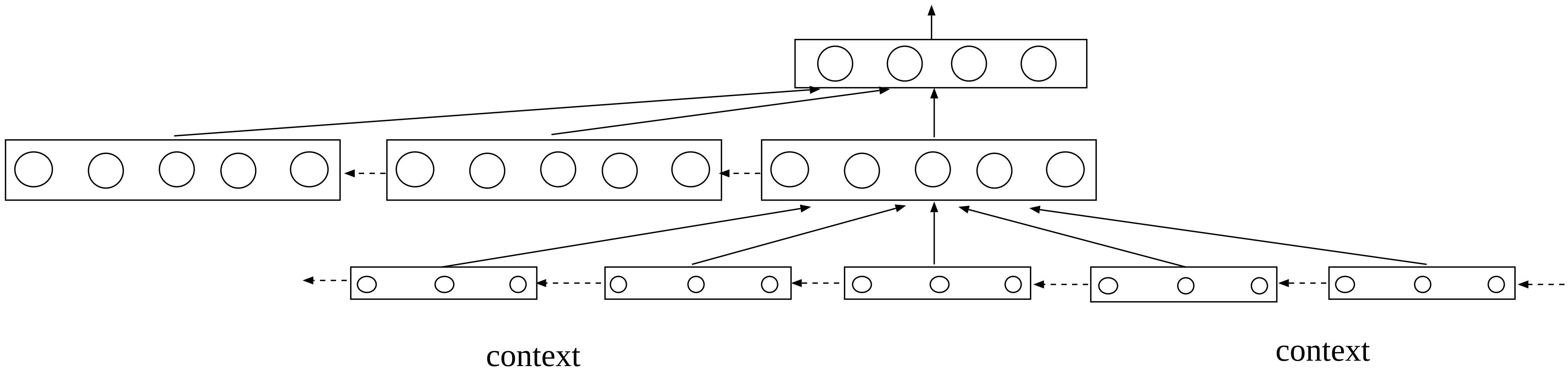


# Sequence processing: dynamic networks

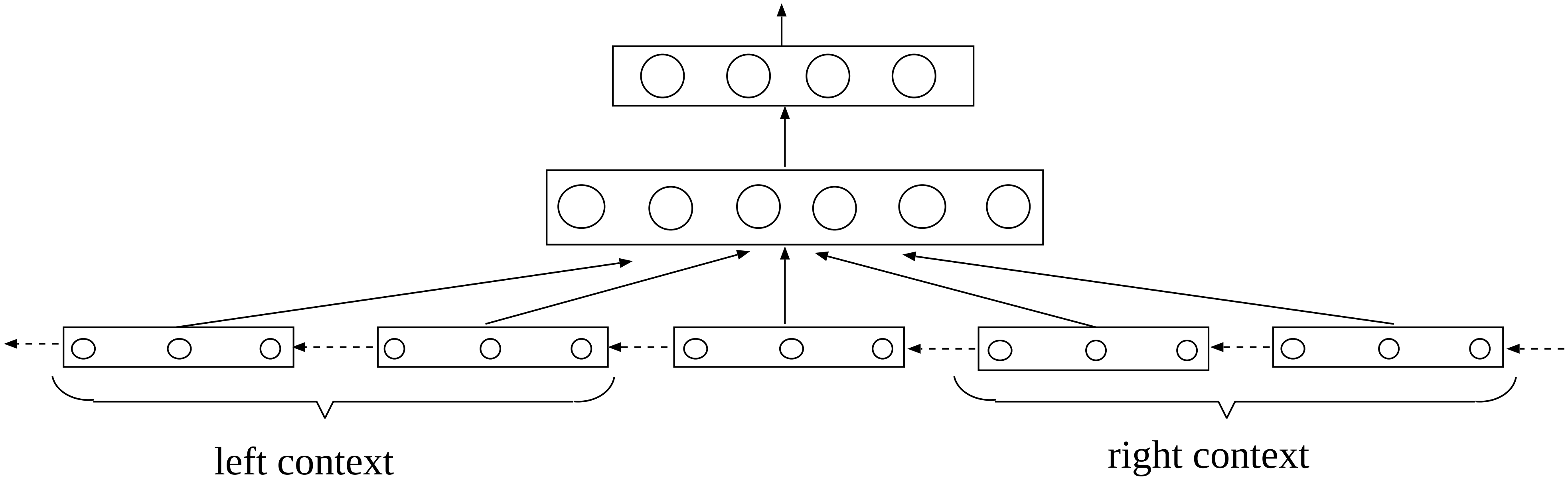


# Sequence processing: dynamic networks

**Time-delayed neural networks** (TDNN) that learn to classify pattern with shift-invariance for ASR, reading lip movement, HTR, video analysis, ... (Waibel et al. 1989)



# Training dynamic networks



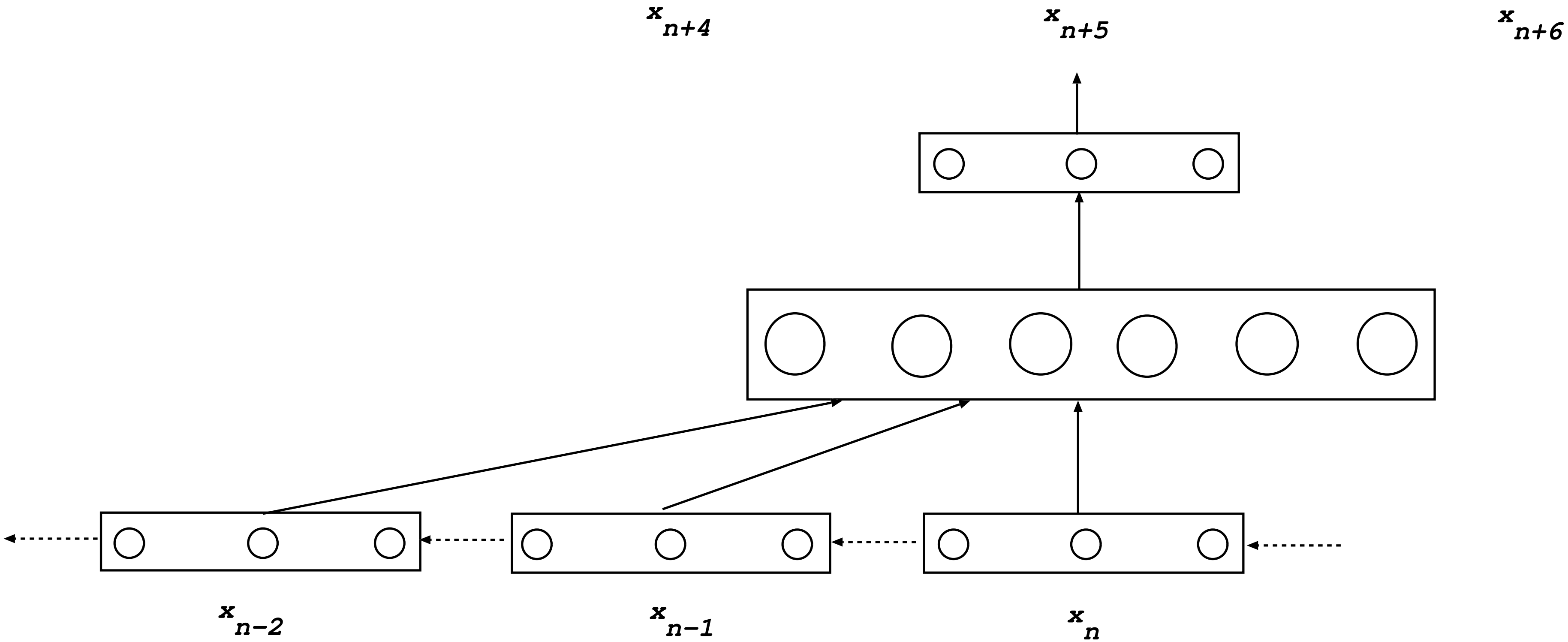

A training pair

$$(\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_N, \mathbf{t}_1\mathbf{t}_2\ldots\mathbf{t}_N) \in \mathbb{R}^{N_1 N} \times \mathbb{R}^{N_2 N}$$

is equivalent to a conventional training set

$$\{(\mathbf{0}\mathbf{0}\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3, \mathbf{t}_1), \ldots, (\mathbf{x}_{n-2}\mathbf{x}_{n-1}\mathbf{x}_n\mathbf{x}_{n+1}\mathbf{x}_{n+2}, \mathbf{t}_n), \ldots, (\mathbf{x}_{N-2}\mathbf{x}_{N-1}\mathbf{x}_N\mathbf{0}\mathbf{0}, \mathbf{t}_N)\}$$

# Sequence processing: prediction of temporal series




A training sequence is:

$$\{(\mathbf{000}, \mathbf{x}_5), (\mathbf{00x}_1, \mathbf{x}_6), (\mathbf{0x}_1\mathbf{x}_2, \mathbf{x}_7), (\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3, \mathbf{x}_8), \ldots (\mathbf{x}_{N-7}\mathbf{x}_{N-6}\mathbf{x}_{N-5}, \mathbf{x}_N))\}$$

Example

# A taxonomy of dynamic networks

1. Memory dependency
   - Networks with bounded-memory
     - NETtalk.
     - TDNN.
     - CNN.
   - Recurrent neural networks
     - Synchronous recurrent neural networks
     - Asynchronous recurrent neural networks: encoder-decoder architecture.
   - Transformer.
2. Recurrent networks
   - Time-synchronous recurrent neural networks
     - Regular recurrent neural networks.
   - Fixed-point recurrent networks
     - Hopfield networks, BAM.
     - Boltzmann machines.
3. Time representation
   - Continuous-time networks (models: differential equations)
   - Discrete-time networks (models: difference equations)

# Index

# Input/output coding of discrete variables

- **Local coding**: One input/output unit per symbol (maybe too many units).
- **Distributed coding**: Using compact codes
- **A learned word embedding***: by proyecting the word into $\mathbb{R}^D$ (Mikolov NIPS 2013)

| Symbol | local | | | | | | distributed | | | embedding | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| "a" | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.15 | 0.66 |
| "b" | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | -0.66 | 0.02 |
| "c" | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0.32 | -0.49 |
| "d" | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0.89 | 0.78 |
| "e" | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | -0.17 | -0.12 |
| "f" | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0.33 | 0.55 |

# Word embeddings

- Neural language model (Bengio 2003)
- Bag-of-words neural networks (Mikolov 2013)
- Continuous skip-grams (Mikolov 2013)
- Pre-trained neural netwotks: BERT (Devlin 2019)
- ...

- Extension to images: for example, patch embeddings [Dosovitskiy 2020]

# Word embeddings

- A word $x$ from a ordered vocabulary $V_X$ with index $i(x)$

- $\mathbf{W_E}(x) \equiv [\mathbf{W_E}]_{i(x)} = \mathbf{x} \in \mathbb{R}^{D_W}$: a row of $\mathbf{W_E}$ that is the word embedding of $x$

- A sentence $x_1^N$ is a finite-length sequence of words from $V_X$

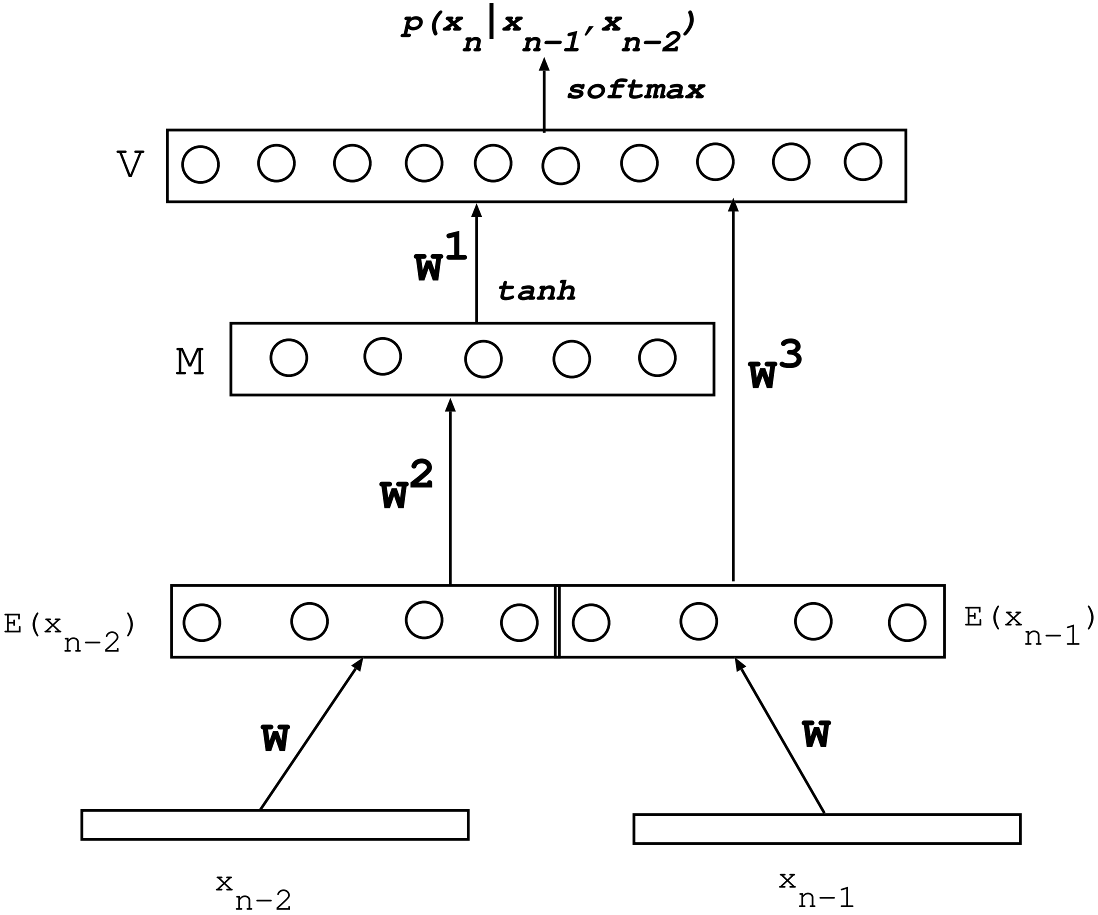
- $\mathbf{x}_1^N = \mathbf{W_E}(x_1) \ldots \mathbf{W_E}(x_N)$ is a finite-length sequence of word embeddings of the words in $x_1^N$

- $\mathbf{S_E}(x_1^N) \equiv \mathbf{X} \in \mathbb{R}^{D_S}$ is the sentence embedding (vector) of the sentence $x_1^N$

- word2vec is one of the most popular toolkit
  https://code.google.com/archive/p/word2vec/

- Curretly, pre-trained models as BERT and similars are also used.
  https://github.com/google-research/bert

# Word embeddings (Bengio et al. 2003)




Trigram language model:

$$p(X_n = x_n \mid x_{n-1}, x_{n-2}) \;=$$
$$[\mathbf{f}_{sm}(\mathbf{W}^3[\mathbf{W}(x_{n-1})\mathbf{W}(x_{n-2})]+$$
$$\mathbf{W}^1\mathbf{f}_{th}(\mathbf{W}^2[\mathbf{W}(x_{n-1})\mathbf{W}(x_{n-2})])]_{i(x_n)}$$

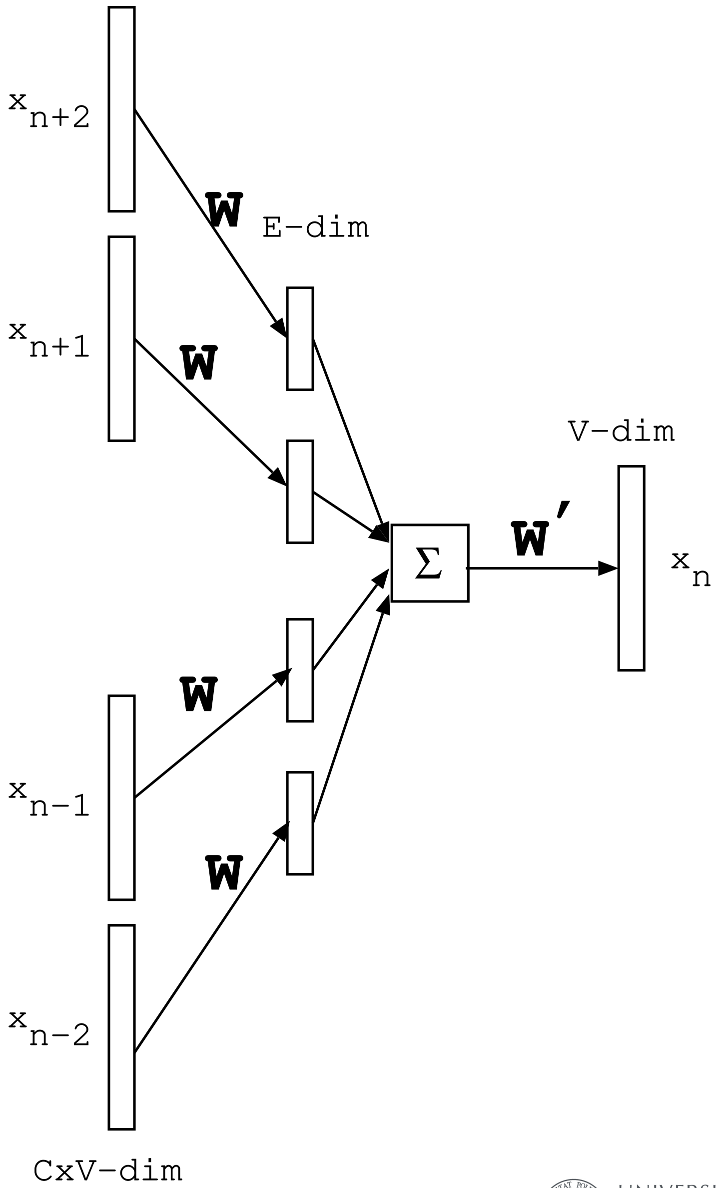
Goal: given a sequence of words $x_1^N$,

$$\mathbf{W_E} =$$
$$\operatorname*{argmax}_{\mathbf{W}} \max_{\mathbf{W}^1,\mathbf{W}^2,\mathbf{W}^3} (\sum_{n=1}^{N} \log p(x_n \mid x_{n-2}x_{n-1})+$$
$$R(\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3, \mathbf{W}))$$

Solution: back error propagation

# Encoder-decoders for word embeddings (Mikolov et al, 2013)

Bag-of-words neural networks



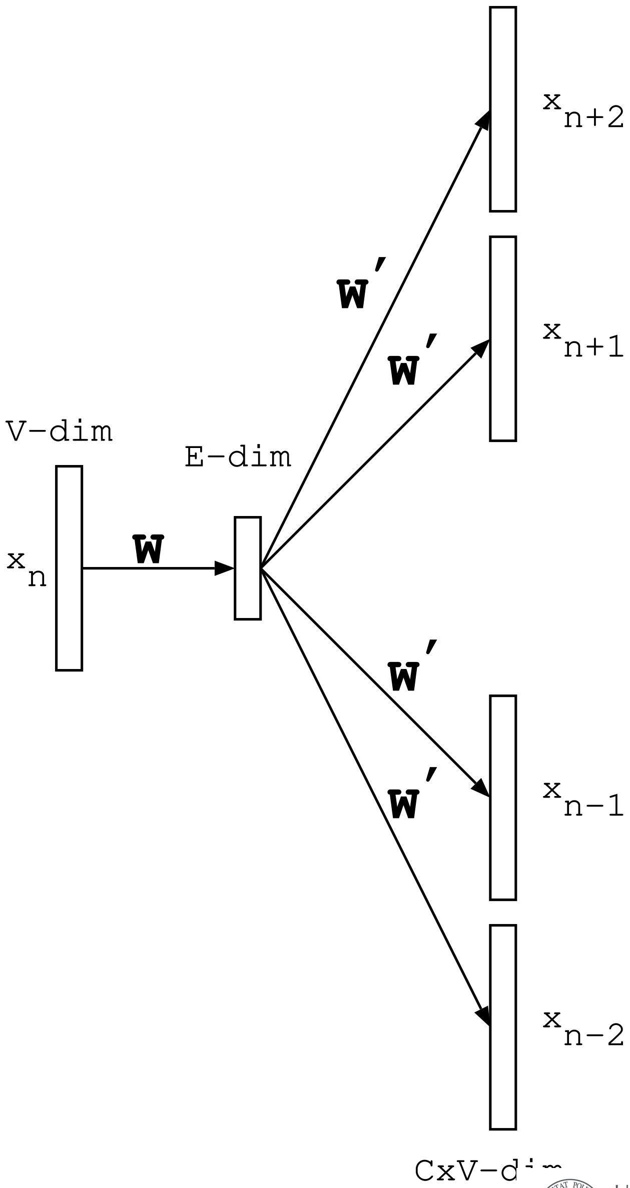

- $C(x_n) = \{x_{n-c}, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{n+c}\}$
- $p(x_n \mid C(x_n)) = [\mathbf{f}_{sm}(\mathbf{W}' \sum_{x \in C(x_n)} \mathbf{W}(x))]_{i(x_n)}$
- Goal: given a sequence of words $x_1^N$,

$$\mathbf{W_E} = \operatorname*{argmax}_{\mathbf{W}} \max_{\mathbf{W}'} \sum_{n=1}^{N} \log p(x_n \mid C(x_n))$$

- Solution: stochastic gradient ascent
- Toolkit: https://code.google.com/archive/p/word2vec/

# Encoder-decoders for word embeddings (Mikolov et al, 2013)

Continuous skip-gram neural networks




- $C(x_n) = \{x_{n-c}, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{n+c}\}$
- $p(C(x_n) \mid x_n) = \prod_{x \in C(x_n)} [\mathbf{f}_{sm}(\mathbf{W}' \; \mathbf{W}(x_n)]_{i(x)}$
- Goal: given a sequence of words $x_1^N$,

$$\mathbf{W_E} = \operatorname*{argmax}_{\mathbf{W}} \max_{\mathbf{W}'} \sum_{n=1}^{N} \log p(C(x_n) \mid x_n)$$

- Solution: stochastic gradient ascent
- Toolkit: https://code.google.com/archive/p/word2vec/

# Continuous representation of sentences

Given a sequence of words $x_1^N$,

- Sum: $\mathbf{S_E}(x_1^N) = \sum_{n=1}^{N} \mathbf{W_E}(x_n)$ or $\mathbf{S_E}(x_1^N) = \frac{\sum_{n=1}^{N} \mathbf{W_E}(x_N)}{N}$

- Product (element-wise): $\mathbf{S_E}(x_1^N) = \prod_{n=1}^{N} \mathbf{W_E}(x_n)$ or $\mathbf{S_E}(x_1^N) = \sqrt[N]{\prod_{n=1}^{N} \mathbf{W_E}(x_n)}$

- doc2vec (`https://github.com/piskvorky/gensim/`): A variation of the skip-gram neural network or the n-gram based approach. For each new sentence, a new row is added to the matrix of sentence embedding and trained.

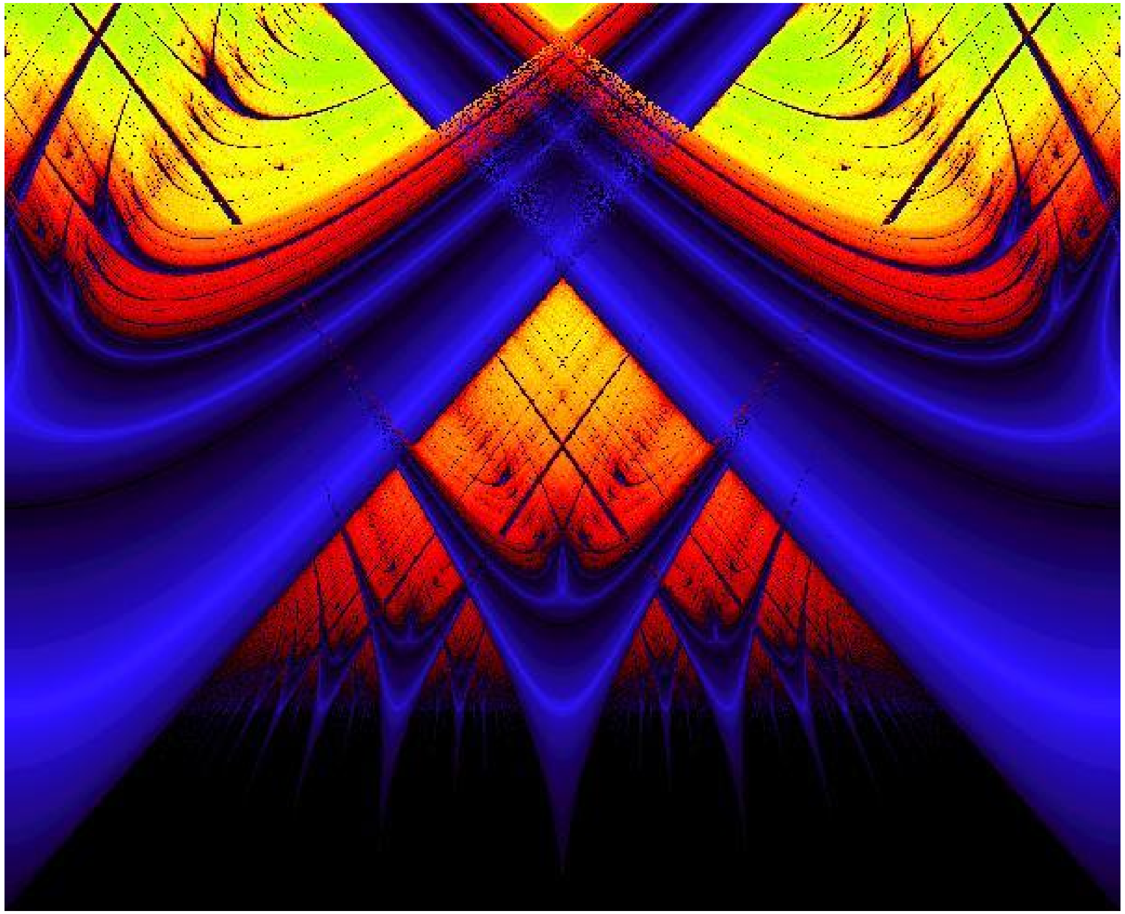
- From the state of a recurrent neural network at the end of processing a sentence.

- From pre-trained models as BERT or SentenceBERT (siameses network), Universal Sentence Encoder, ...

# Index

# Simple recurrent networks

“Image of the week” SDSC: A National Laboratory for Computational Science & Engineering
“Simple recurrent neural networks” Michael Casey, (SDSC). 1997.



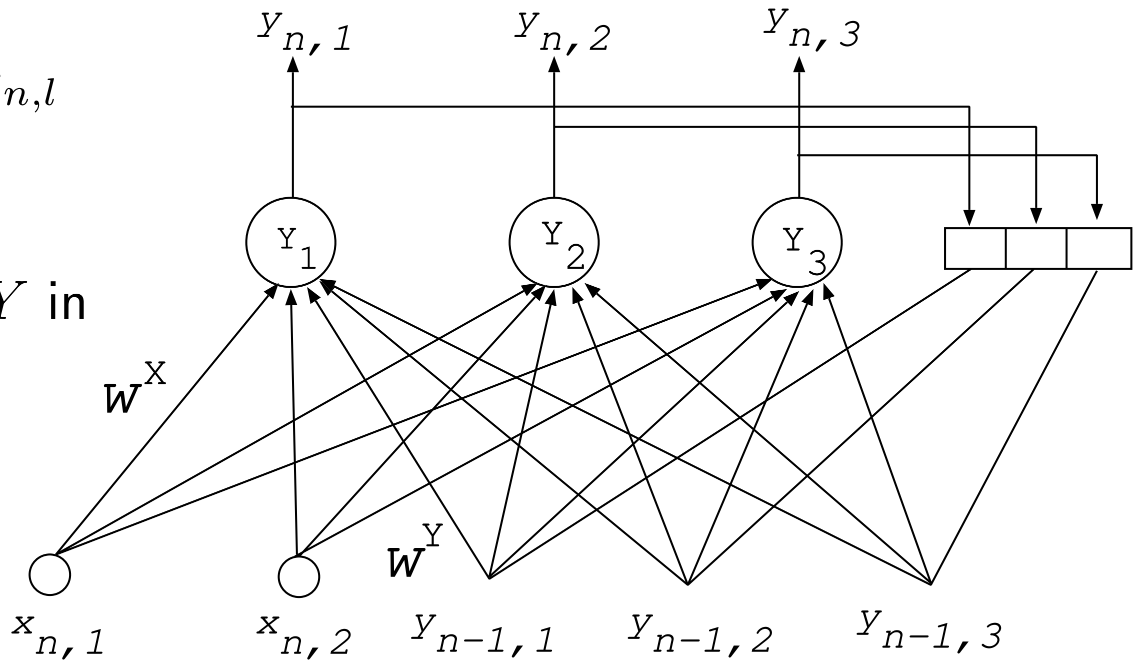
# Simple recurrent networks

- A set $Y$ of $d_Y$ units and a set $X$ of $d_X$ inputs.

- Input: $\mathbf{x}_n \in \mathbb{R}^{d_X}$ and output: $\mathbf{y}_n \in \mathbb{R}^{d_Y}$, $n = 1, 2, \ldots$

- **Total input** of the unit $k \in Y$ in $n$:

$$h_{n,k} = \sum_{l \in Y} \omega^Y_{k,l}\; y_{n-1,l} + \sum_{l \in X} \omega^X_{k,l}\; x_{n,l}$$



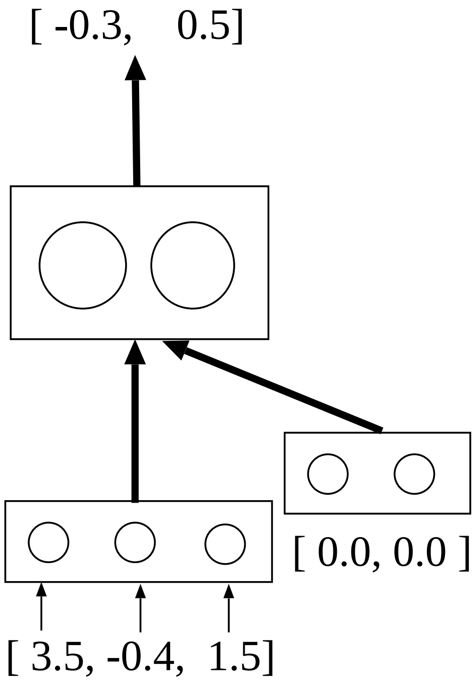

- The **state** (and the output) of $k \in Y$ in $n$:

$$y_{n,k} = \begin{cases} f(h_{n,k}) & n \geq 1 \\ 0 & n = 0 \end{cases}$$

$f$ is an activation function

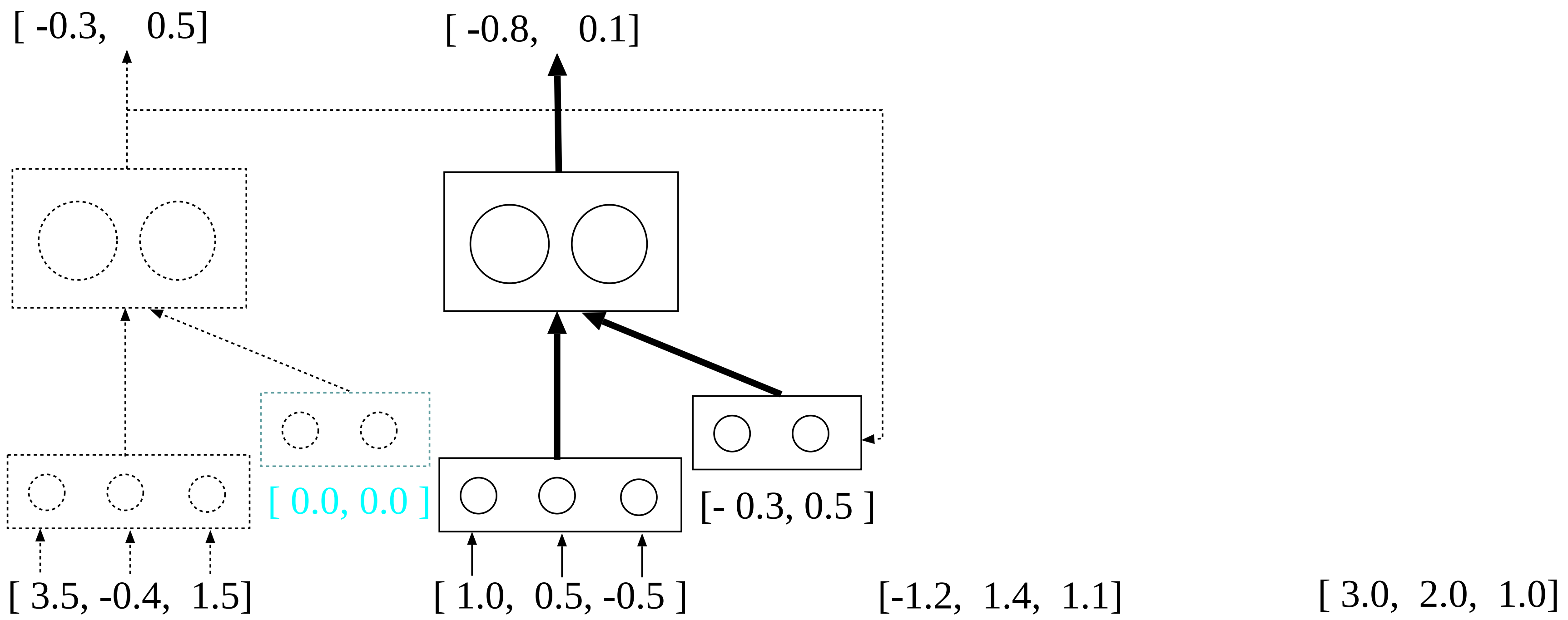
- In a compact notation: $\mathbf{y}_n = \mathbf{f}(W^Y \mathbf{y}_{n-1} + W^X \mathbf{x}_n)$

# Sequence processing: simple recurrent networks

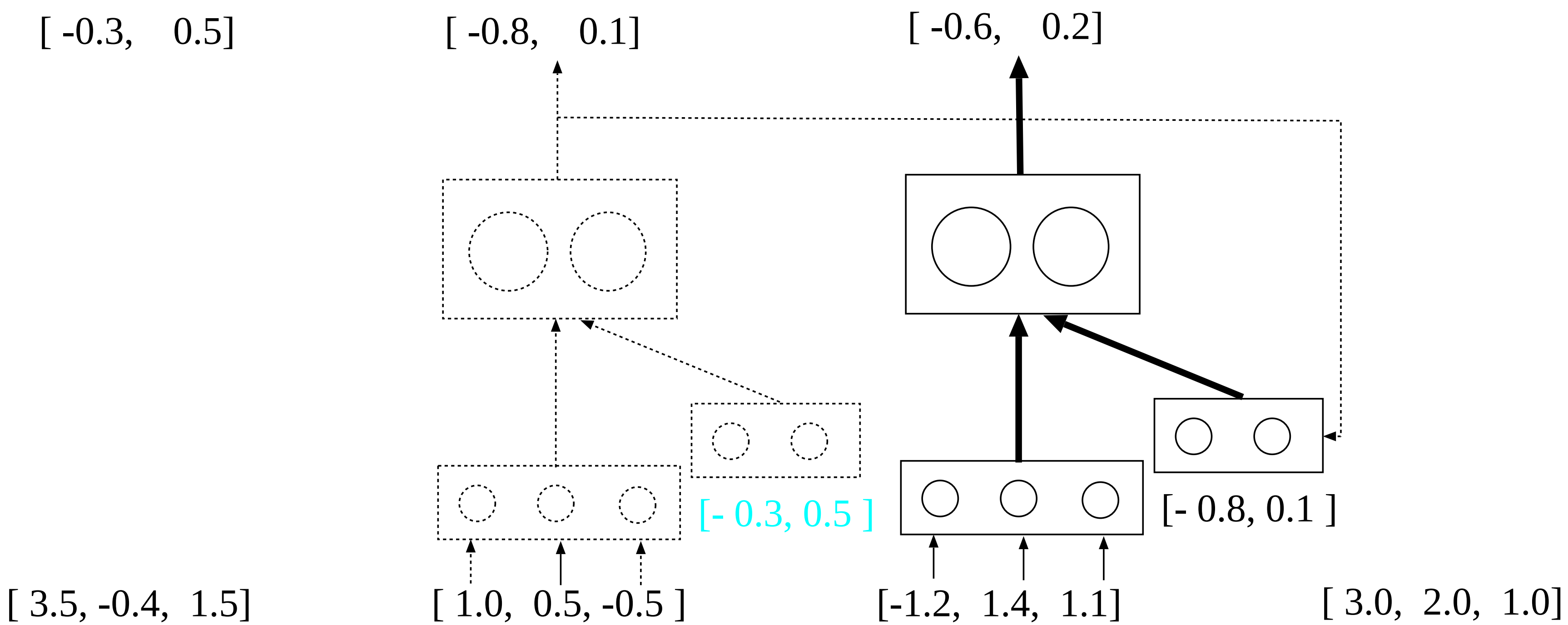



[ 1.0, 0.5, -0.5 ]    [-1.2, 1.4, 1.1]    [ 3.0, 2.0, 1.0]

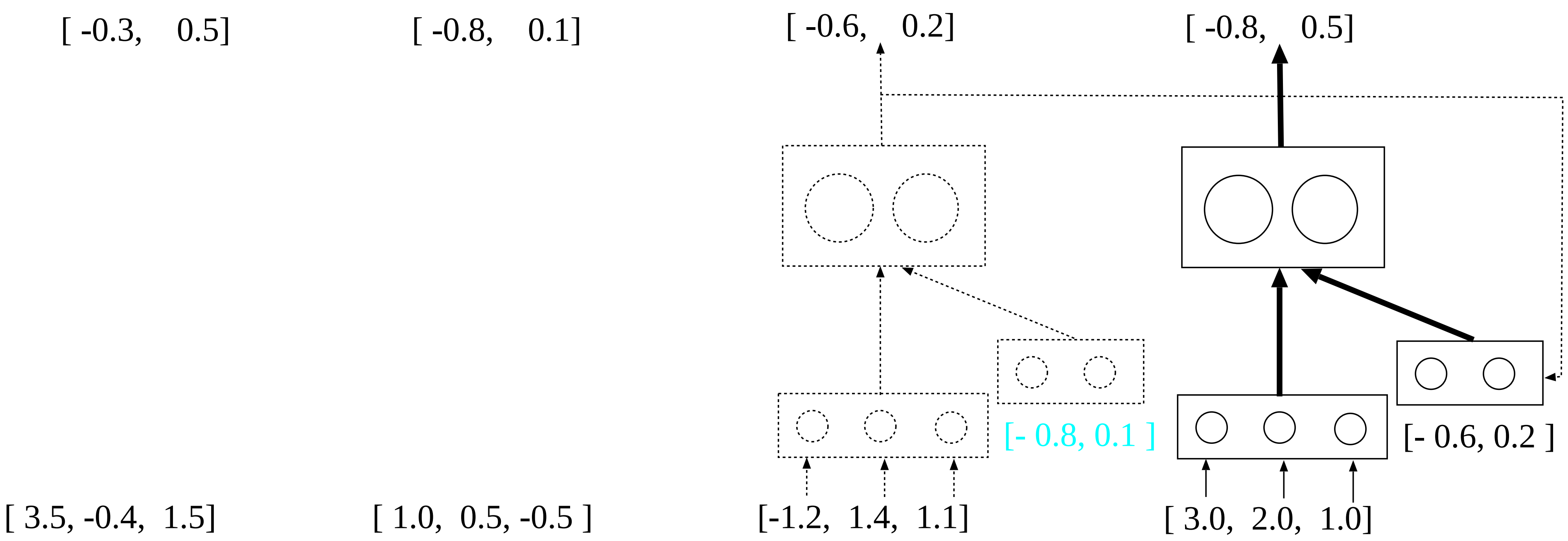
# Sequence processing: simple recurrent networks



# Sequence processing: simple recurrent networks



# Sequence processing: simple recurrent networks



# Training simple recurrent networks (regression)

- A **training sequence** $A = (\mathbf{x}_n, \mathbf{t}_n)_{n=1,..,N} : \mathbf{x}_n \in \mathbb{R}^{d_X}$ and $\mathbf{t}_n \in \mathbb{R}^{d_Y}$

- The **error in unit $k$ in $n$** is: $e_{n,k} = t_{n,k} - y_{n,k}(\mathbf{w})$ for $1 \leq n \leq N$ and $k \in Y$

- The **total error** (objective function) is: $\mathcal{E}_A(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k \in Y} (e_{n,k})^2$

- Computing a local minimum of $\mathcal{E}_A$ with respect to $\omega_{ij}$: GRADIENT DESCENT

$$\Delta\omega_{i,j} = -\rho \, \frac{\partial \mathcal{E}_A(\mathbf{w})}{\partial \omega_{i,j}} \qquad i \in Y, \quad j \in Y \cup X$$

- Approaches:
  - Forward gradient algorithm.
  - Back-propagation through time algorithm.

# Training simple recurrent networks (classification)

- A **training sequence** $A = (\mathbf{x}_n, \mathbf{t}_n)_{n=1,..,N} : \mathbf{x}_n \in \mathbb{R}^{d_X}$ and $\mathbf{t}_n \in \{0,1\}^{d_Y}$

- The objective function (cross-entropy): In this case the activation function to produce $y_{n,k}(\mathbf{w})$ is a softmax function:

$$C_A(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{N_2} t_{n,k} \, \log y_{n,k}(\mathbf{w})$$

- Computing a local minimum of $C_A$ with respect to $\omega_{ij}$: GRADIENT DESCENT

$$\Delta\omega_{i,j} = -\rho \, \frac{\partial C_A(\mathbf{w})}{\partial \omega_{i,j}} \quad i \in Y, \;\; j \in Y \cup X$$

- Approaches:
  - Forward gradient algorithm
  - Back-propagation through time algorithm

# Forward gradient algorithm

[Williams & Zipser, 1995]

# Forward gradient algorithm: Recurrent connections

$$\mathcal{E}_A = \frac{1}{2}\sum_{n=1}^{N}\sum_{k\in Y}(e_{n,k})^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{k\in Y}(t_{n,k} - y_{n,k})^2$$

$$\Delta\omega_{i,j}^{Y} = -\rho\,\frac{\partial\mathcal{E}_A}{\partial\omega_{i,j}^{Y}} = \rho\,\sum_{n=1}^{N}\sum_{k\in Y} e_{n,k}\,\frac{\partial y_{n,k}}{\partial\omega_{i,j}^{Y}}$$

$$y_{n,k} = f(h_{n,k}) = f\left(\sum_{l\in Y}\omega_{k,l}^{Y}\,y_{n-1,l} + \sum_{l\in X}\omega_{k,l}^{X}\,x_{n,l}\right)$$

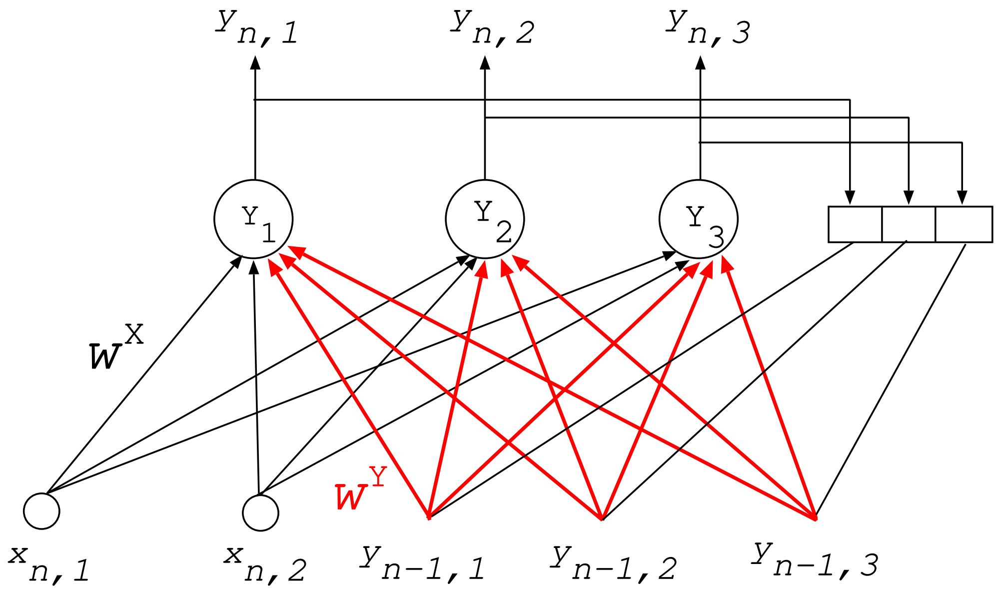
$$\begin{aligned}
\frac{\partial y_{n,k}}{\partial\omega_{i,j}^{Y}} = f'(h_{n,k})\,\frac{\partial h_{n,k}}{\partial\omega_{i,j}^{Y}} \quad &= \quad f'(h_{n,k})\sum_{l\in Y}\left(\delta_{k,i}\,\delta_{l,j}\,y_{n-1,l} + \omega_{kl}^{Y}\,\frac{\partial y_{n-1,l}}{\partial\omega_{i,j}^{Y}}\right)\\
&= \quad f'(h_{n,k})\left(\delta_{k,i}\,y_{n-1,j} + \sum_{l\in Y}\omega_{k,l}^{Y}\,\frac{\partial y_{n-1,l}}{\partial\omega_{i,j}^{Y}}\right)
\end{aligned}$$

# Forward gradient algorithm: Recurrent connections

$$\Delta\omega_{ij}^{Y} = \rho \sum_{n=1}^{N} \sum_{k \in Y} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega_{i,j}^{Y}}$$

$$\frac{\partial y_{n,k}}{\partial \omega_{i,j}^{Y}} = f'(h_{n,k}) \left( \delta_{k,i}\, y_{n-1,j} + \sum_{l \in Y} \omega_{k,l}^{Y} \frac{\partial y_{n-1,l}}{\partial \omega_{i,j}^{Y}} \right)$$

$$\frac{\partial y_{0,k}}{\partial \omega_{i,j}^{Y}} = 0$$



# Forward gradient algorithm: Input connections

$$\mathcal{E}_A = \frac{1}{2} \sum_{n=1}^{N} \sum_{k \in Y} (e_{n,k})^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{k \in Y} (t_{n,k} - y_{n,k})^2$$

$$\Delta \omega_{i,j}^{X} = -\rho \, \frac{\partial \mathcal{E}_A}{\partial \omega_{i,j}^{X}} = \rho \, \sum_{n=1}^{N} \sum_{k \in Y} e_{n,k} \, \frac{\partial y_{n,k}}{\partial \omega_{i,j}^{X}}$$

$$y_{n,k} = f(h_{n,k}) = f\left( \sum_{l \in Y} \omega_{k,l}^{Y} \, y_{n-1,l} + \sum_{l \in X} \omega_{k,l}^{X} \, x_{n,l} \right)$$

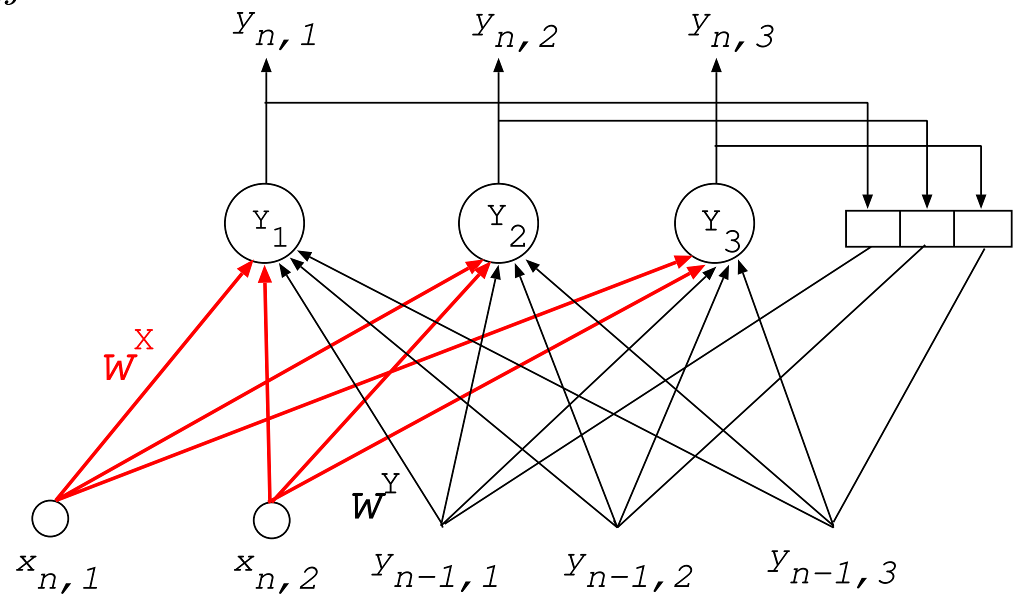
$$\begin{aligned}
\frac{\partial y_{n,k}}{\partial \omega_{i,j}^{X}} = f'(h_{n,k}) \, \frac{\partial h_{n,k}}{\partial \omega_{i,j}^{X}} \quad &= \quad f'(h_{n,k}) \left( \sum_{l \in Y} \omega_{k,l}^{Y} \, \frac{\partial y_{n-1,l}}{\partial \omega_{i,j}^{X}} + \sum_{l \in X} \delta_{i,k} \, \delta_{j,l} \, x_{n,l} \right) \\
&= \quad f'(h_{n,k}) \left( \sum_{l \in Y} \omega_{k,l}^{Y} \, \frac{\partial y_{n-1,l}}{\partial \omega_{i,j}^{X}} + \delta_{i,k} \, x_{n,j} \right)
\end{aligned}$$

# Forward gradient algorithm: Input connections

$$\Delta\omega_{i,j}^{X} = \rho \sum_{n=1}^{N} \sum_{k \in Y} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega_{i,j}^{X}}$$

$$\frac{\partial y_{n,k}}{\partial \omega_{i,j}^{X}} = f'(h_{n,k}) \left( \delta_{i,k}\, x_{n,j} + \sum_{l \in Y} \omega_{k,l}^{Y} \frac{\partial y_{n-1,l}}{\partial \omega_{i,j}^{X}} \right)$$

$$\frac{\partial y_{0,k}}{\partial \omega_{i,j}^{X}} = 0$$



# Forward gradient algorithm

- **Initialization:** $\dfrac{\partial y_{0,k}}{\partial \omega^Y_{i,j}} = 0;\quad \dfrac{\partial y_{0,k}}{\partial \omega^X_{i,j}} = 0;\quad y_{0,k} = 0;\quad \Delta\omega^Y_{i,j} = 0;\quad \Delta\omega^X_{i,j} = 0$
- **Iterate until convergence:**
  - For $n = 1, \ldots, N$, $k \in Y$, $i \in Y$ and $j \in X \cup Y$

$$\begin{aligned}
h_{n,k} &= \sum_{l\in Y} \omega^Y_{k,l}\, y_{n-1,l} + \sum_{l\in X} \omega^X_{k,l}\, x_{n,l} \\
y_{n,k} &= f(h_{n,k}); \qquad e_{n,k} = t_{n,k} - y_{n,k} \\
\frac{\partial y_{n,k}}{\partial \omega^Y_{i,j}} &= f'(h_{n,k}) \left( \delta_{k,i}\, y_{n-1,j} + \sum_{l\in Y} \omega^Y_{k,l} \frac{\partial y_{n-1,l}}{\partial \omega^Y_{i,j}} \right) \\
\frac{\partial y_{n,k}}{\partial \omega^X_{i,j}} &= f'(h_{n,k}) \left( \delta_{i,k}\, x_{n,j} + \sum_{l\in Y} \omega^Y_{k,l} \frac{\partial y_{n-1,l}}{\partial \omega^X_{i,j}} \right) \\
\Delta\omega^Y_{i,j} &+= \sum_{k\in D_n} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega^Y_{ij}}; \quad \Delta\omega^X_{i,j} += \sum_{k\in D_n} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega^X_{i,j}}
\end{aligned}$$

  - **Updating weights with:** $\omega^Y_{i,j} += \rho\, \Delta\omega^Y_{i,j} \quad i,j \in Y; \quad \omega^X_{i,j} += \rho\, \Delta\omega^X_{i,j} \quad i \in Y, j \in X$

# Forward gradient algorithm: Computational costs

$C$ = number of connections ($C = C_Y + C_X$),

$C_Y$ = number of recurrent connections, $C_X$ = number of input connections,

$d_Y$ = number of units and $N$ = the size of the training sample

Temporal cost: $\left\{ \begin{array}{ll} \text{Recurrent connections} & O(d_Y \; N \; C_Y \; d_Y) \\ \text{Input connections} & O(d_Y \; N \; C_X \; d_Y) \end{array} \right\}$ Total: $O(C \; {d_Y}^2 \; N)$

Spatial cost: $\left\{ \begin{array}{ll} \text{Recurrent connections} & O(d_Y \; C_Y) \\ \text{Input connections} & O(d_Y \; C_X) \end{array} \right\}$ Total: $O(C \; d_Y)$

# Truncate-gradient algorithm

- **Initialization:** $\dfrac{\partial y_{0,k}}{\partial \omega^Y_{i,j}} = 0; \quad \dfrac{\partial y_{0,k}}{\partial \omega^X_{i,j}} = 0; \quad y_{0,k} = 0; \quad \Delta\omega^Y_{i,j} = 0; \quad \Delta\omega^X_{i,j} = 0$
- **Iterate until convergence:**
  - For $n = 1, \ldots, N$, $k \in Y$, $i \in Y$ and $j \in X \cup Y$

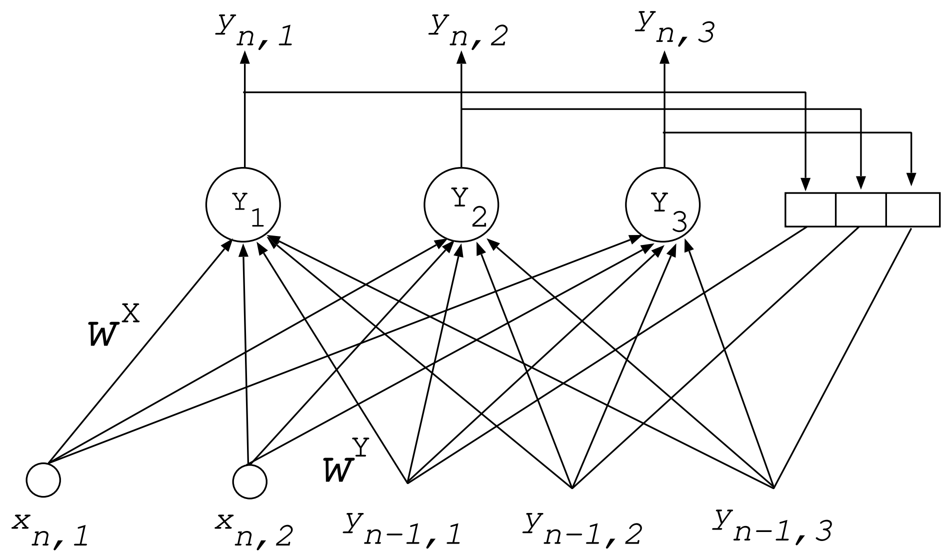
$$
\begin{aligned}
h_{n,k} &= \sum_{l \in Y} \omega^Y_{k,l}\, y_{n-1,l} + \sum_{l \in X} \omega^X_{k,l}\, x_{n,l} \\
y_{n,k} &= f(h_{n,k}); \qquad e_{n,k} = t_{n,k} - y_{n,k} \\
\frac{\partial y_{n,k}}{\partial \omega^Y_{i,j}} &= f'(h_{n,k})\, \delta_{k,i}\, y_{n-1,j} \\
\frac{\partial y_{n,k}}{\partial \omega^X_{i,j}} &= f'(h_{n,k})\, \delta_{i,k}\, x_{n,j} \\
\Delta\omega^Y_{i,j} &\mathrel{+}= \sum_{k \in D_n} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega^Y_{ij}}; \quad \Delta\omega^X_{i,j} \mathrel{+}= \sum_{k \in D_n} e_{n,k} \frac{\partial y_{n,k}}{\partial \omega^X_{i,j}}
\end{aligned}
$$

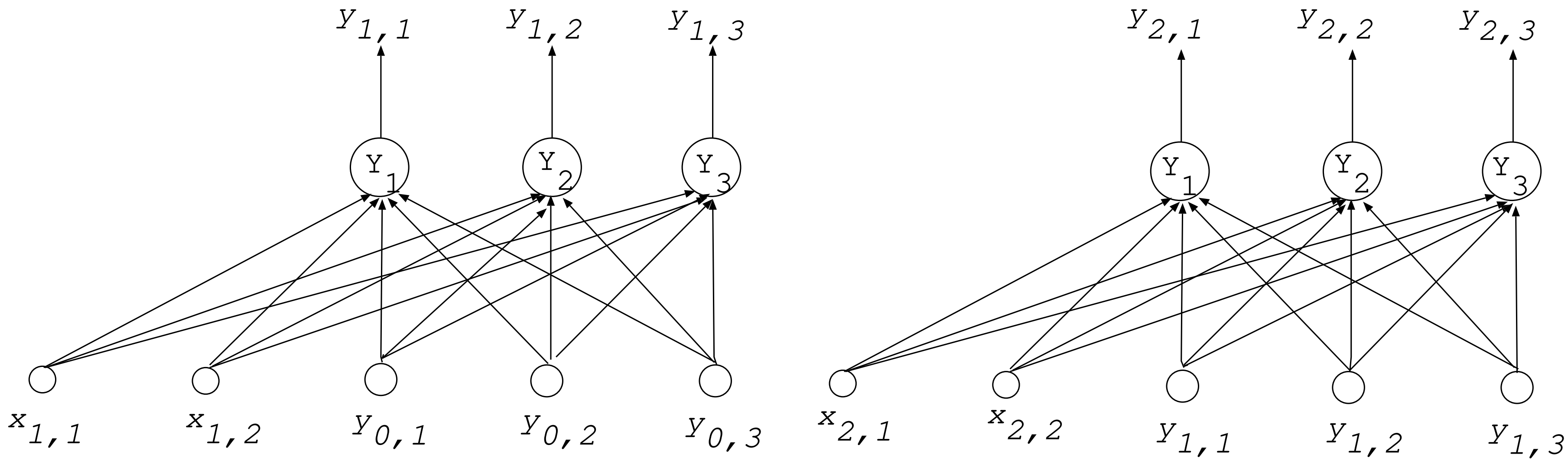
  - **Updating weights with:** $\omega^Y_{i,j} \mathrel{+}= \rho\, \Delta\omega^Y_{i,j} \quad i, j \in Y; \quad \omega^X_{i,j} \mathrel{+}= \rho\, \Delta\omega^X_{i,j} \quad i \in Y, j \in X$

Temporal cost: $O(C\ d_Y\ N)$; Spatial cost: $O(C\ d_Y)$

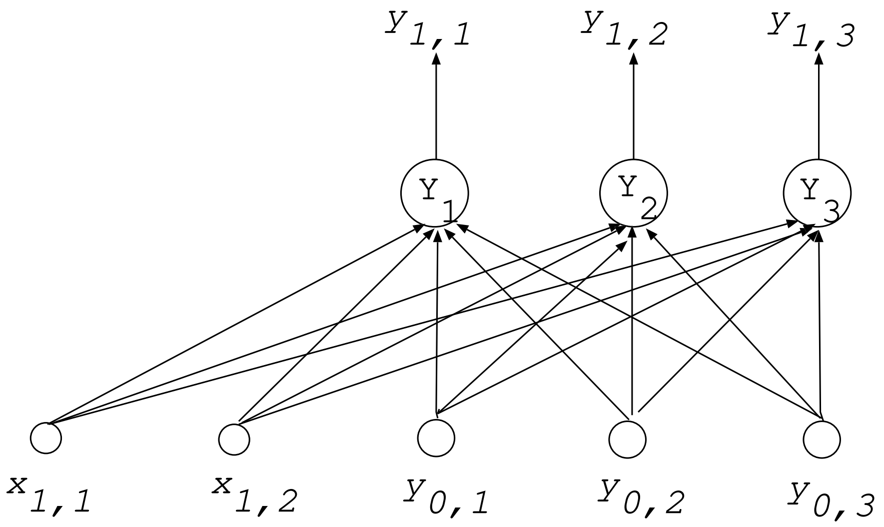
# Back-propagation through time algorithm

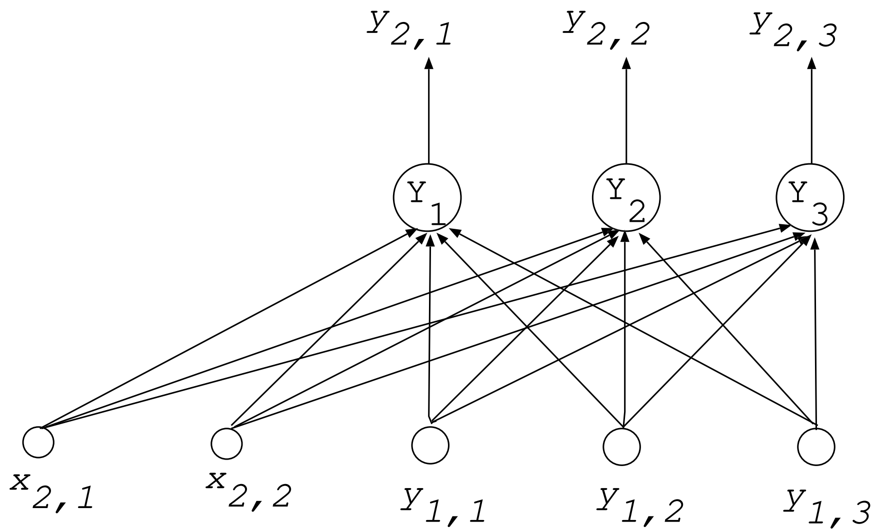
# Back-propagation through time algorithm (BPTT)

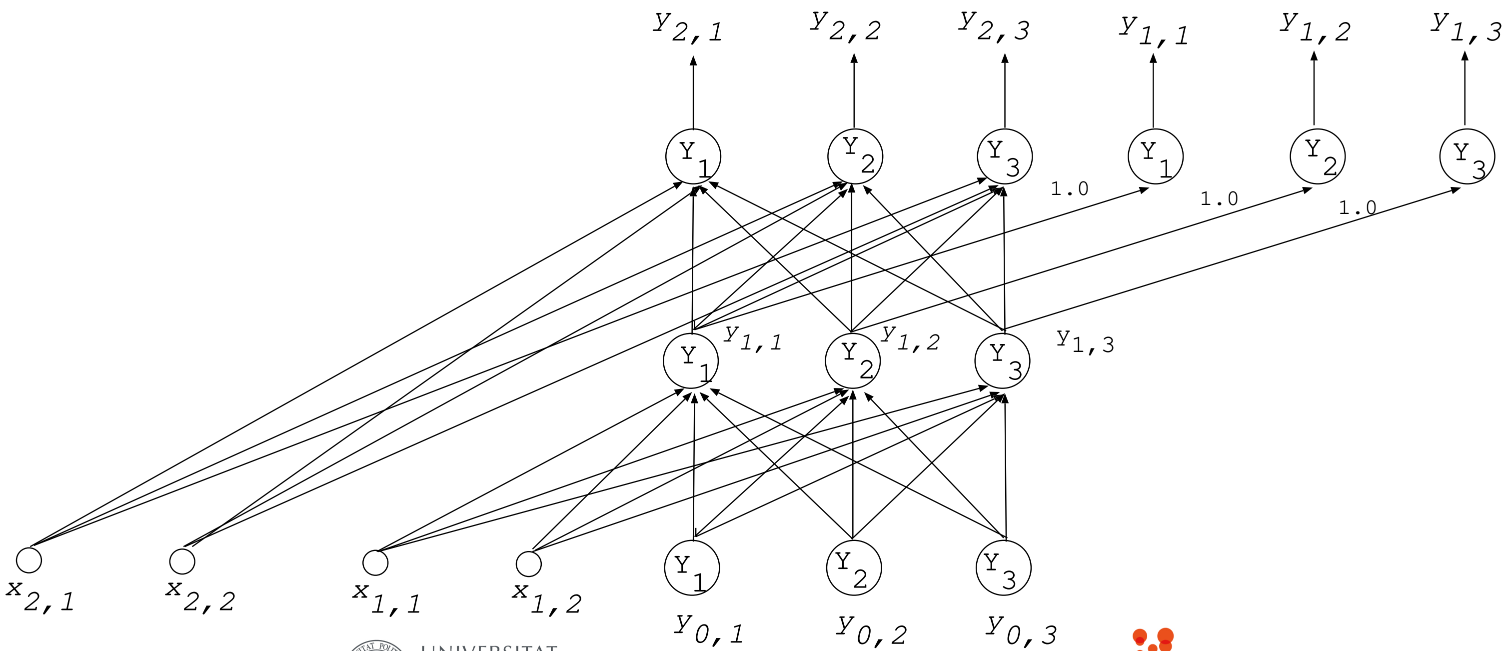


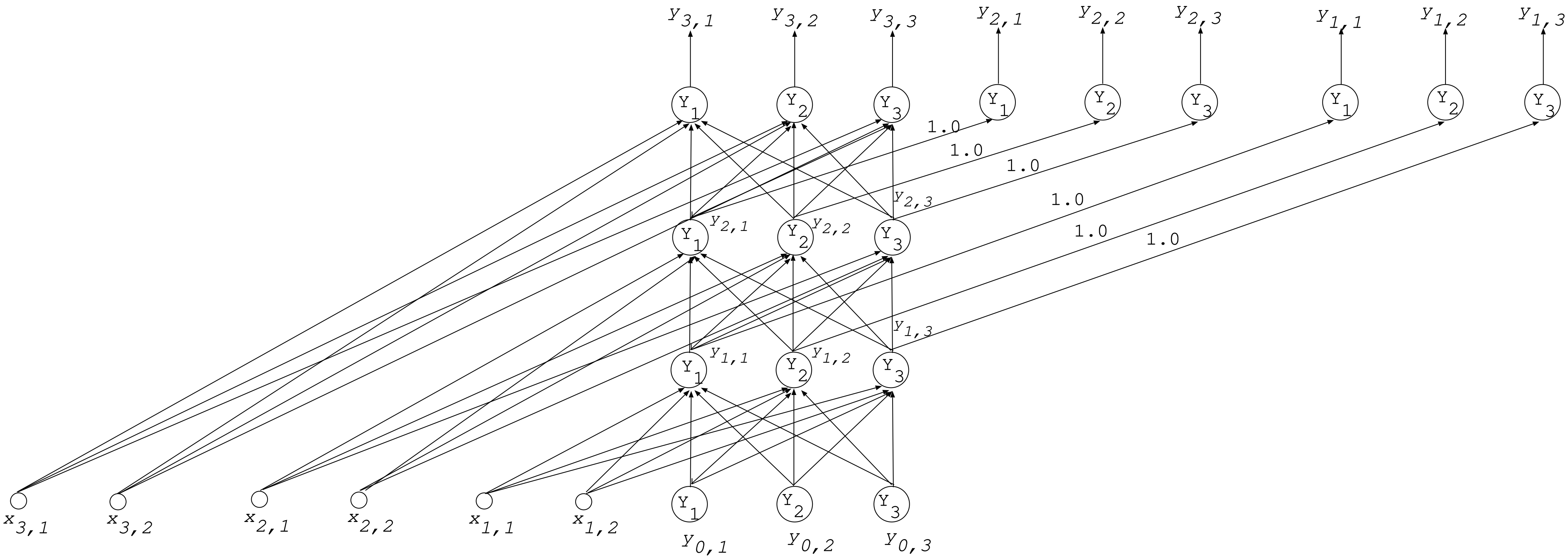
# Back-propagation through time algorithm (BPTT)



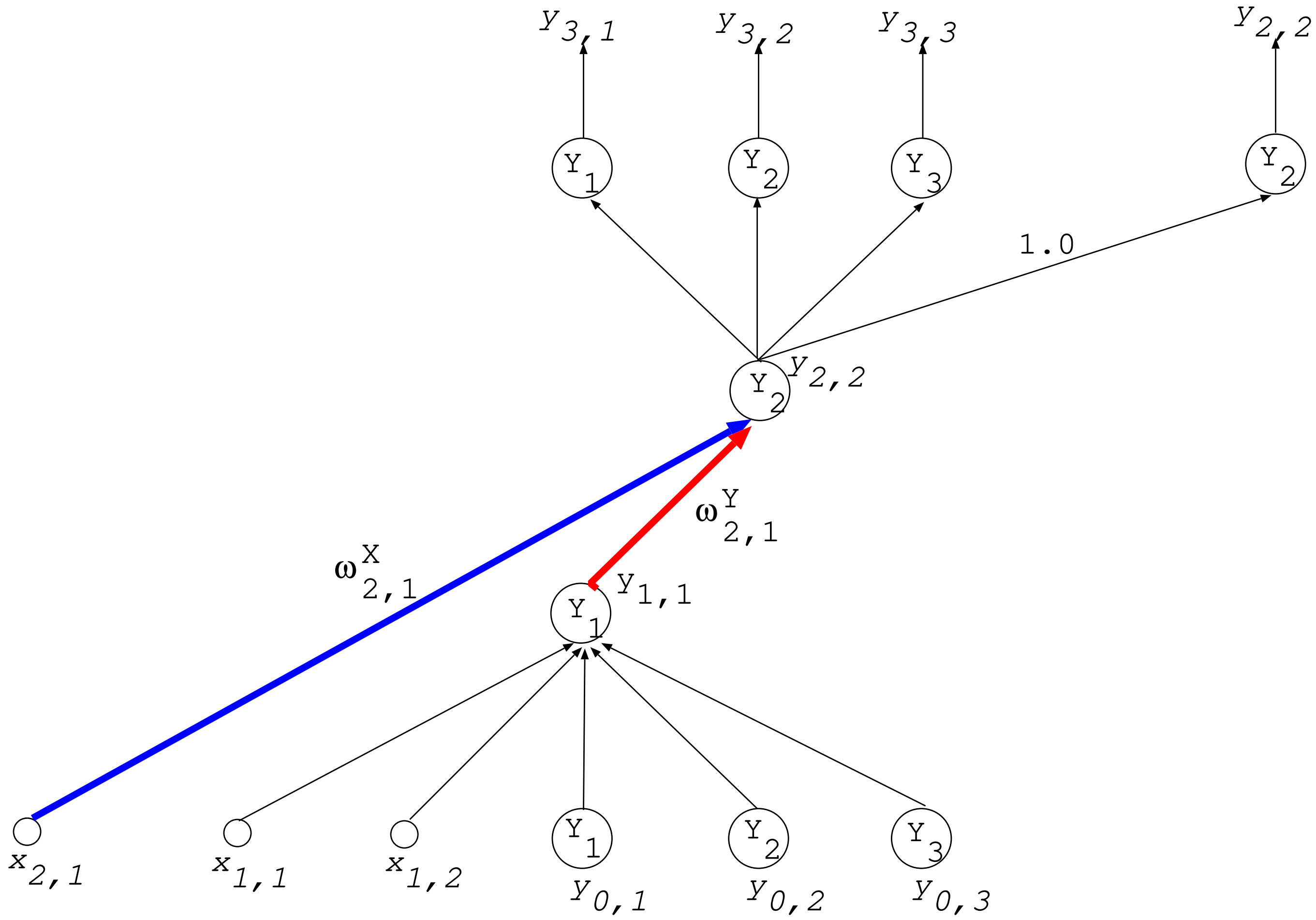



# Back-propagation through time algorithm (BPTT)

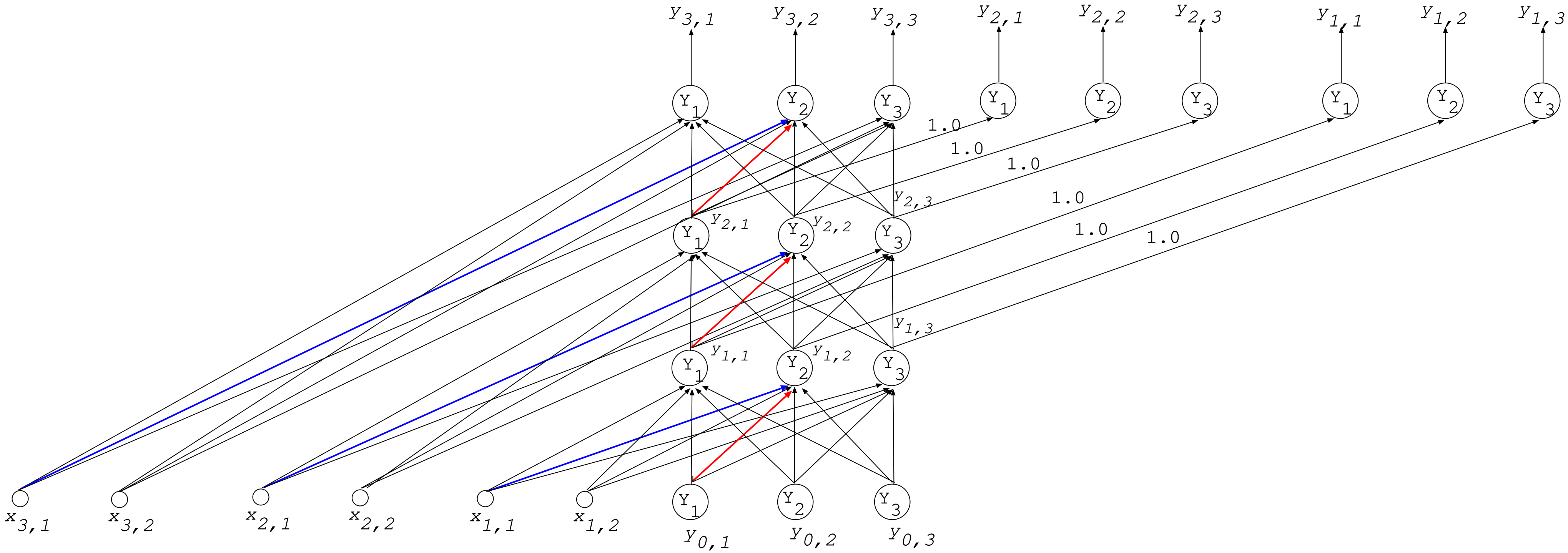


# Back-propagation through time algorithm (BPTT)




$$\Delta_2\,\omega^X_{2,1} = \rho\ e^b_{2,2}\ x_{2,1} \qquad \Delta_2\,\omega^Y_{2,1} = \rho\ e^b_{2,2}\ y_{1,1}$$

# Back-propagation through time algorithm (BPTT)




$$\Delta\omega_{2,1}^{X} = \Delta_1\,\omega_{2,1}^{X} + \Delta_2\,\omega_{2,1}^{X} + \Delta_3\,\omega_{2,1}^{X}$$

$$\Delta\omega_{21}^{Y} = \Delta_1\,\omega_{2,1}^{Y} + \Delta_2\,\omega_{2,1}^{Y} + \Delta_3\,\omega_{2,1}^{Y}$$

# Back-propagation through time algorithm (BPTT)

- From a **training sequence of vector pairs** $(\mathbf{x}_n, \mathbf{t}_n)_{n=1,..,N}$ : $\mathbf{x}_n \in \mathbb{R}^{d_X}$ and $\mathbf{t}_n \in \mathbb{R}^{d_Y}$ to a **training pair**: $([\mathbf{x}_1; \ldots; \mathbf{x}_N][\mathbf{t}_1; \ldots; \mathbf{t}_N])$

- **Feed-forward neural networks with tied weights** with

  – For $n = 1, \ldots, N$ and $k \in Y$:

    * $y_{n,k} = f(\sum_{l \in Y} \omega^Y_{k,l} \; y_{n-1,l} + \sum_{l \in X} \omega^X_{k,l} \; x_{n,l})$
    * $e_{n,k} \;=\; t_{n,k} - y_{n,k}$

# Back-propagation through time algorithm (BPTT)

- **Weights between two units**: $\Delta\,\omega^{Y}_{i,j} = \sum_{n=1}^{N} \Delta_n\,\omega^{Y}_{i,j} = \sum_{n=1}^{N} \rho\; e^{b}_{n,i}\; y_{n-1,j}$

- **Weights from inputs to units**: $\Delta\,\omega^{X}_{i,j} = \sum_{n=1}^{N} \Delta\,\omega^{X}_{i,j}(n) = \sum_{n=1}^{N} \rho\; e^{b}_{n,i}\; x_{n,j}$

- **Error propagation**

$$e^{b}_{n,i} \;=\; \begin{cases} \left[\left(\sum_{l} e^{b}_{n+1,l}\;\omega^{Y}_{l,i}\right) + e_{n,i}\right]\; f'(h_{n,i}) & \text{for } 1 \le n < N \\ e_{N,i}\; f'(h_{N,i}) & \text{for } n = N \end{cases}$$

- **Computational cost**: Temporal: $O(C\;N)$ and spatial: $O(N\;d_Y)$

# Other algorithms

- **Batch training or epochwise**: weights are updated at the end of the training sequence.
- **Truncated gradient algorithm**: heuristic simplification of the exact gradient algorithm.
- **Truncated algorithm or minibatch-based**: weights are updated at the end of a fixed number of time steps.
- **Schimidhuber algorithm**: combination of the truncated BPTT and exact gradient algorithm.
- **Incremental (On-line) training** or **real-time recurrent learning**: weights are updated at each $n$.
- **Momentum-based learning**.
- **Backward-forward algorithm**.

# Other algorithms (learning rate control)

- SDG (stochastic gradient descent).
- SGD with momentum
- Adagrad (Adaptive Gradient)
- Adadelta (an extension of Adagrad)
- Adam (Adaptive Moment Estimation)
- NAG, RMSProp, AdaMax, Nadam, ...

- Implementation through computational graphs in TensorFlow, PyTorch, ... (Koehn 2020)

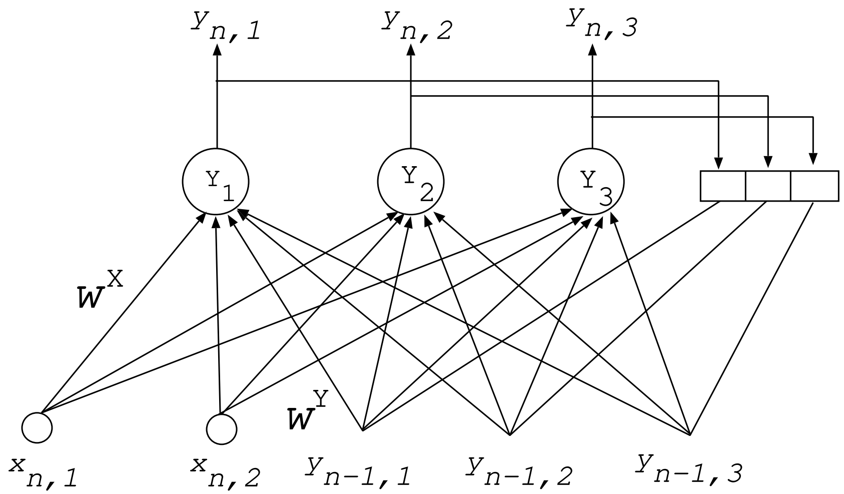
# Other recurrent neural networks

- Second-order recurrent neural networks
- Nonlinear AutoRegressive models with eXogenous inputs (NARX)
- Long Short-Term Memory (LSTM)
- Gated Recurrent Units (GRU)

# Second-order recurrent neural networks

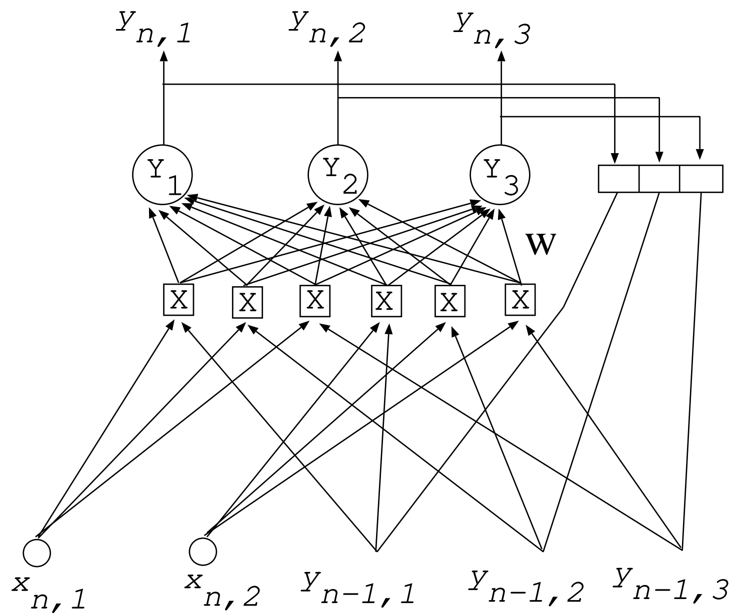
$$y_{0,k} = 0$$

$$y_{n,k} = f\left(\sum_{l \in Y} \omega^{Y}_{k,l}\; y_{n-1,l} + \sum_{l \in X} \omega^{X}_{k,l}\; x_{n,l}\right)$$

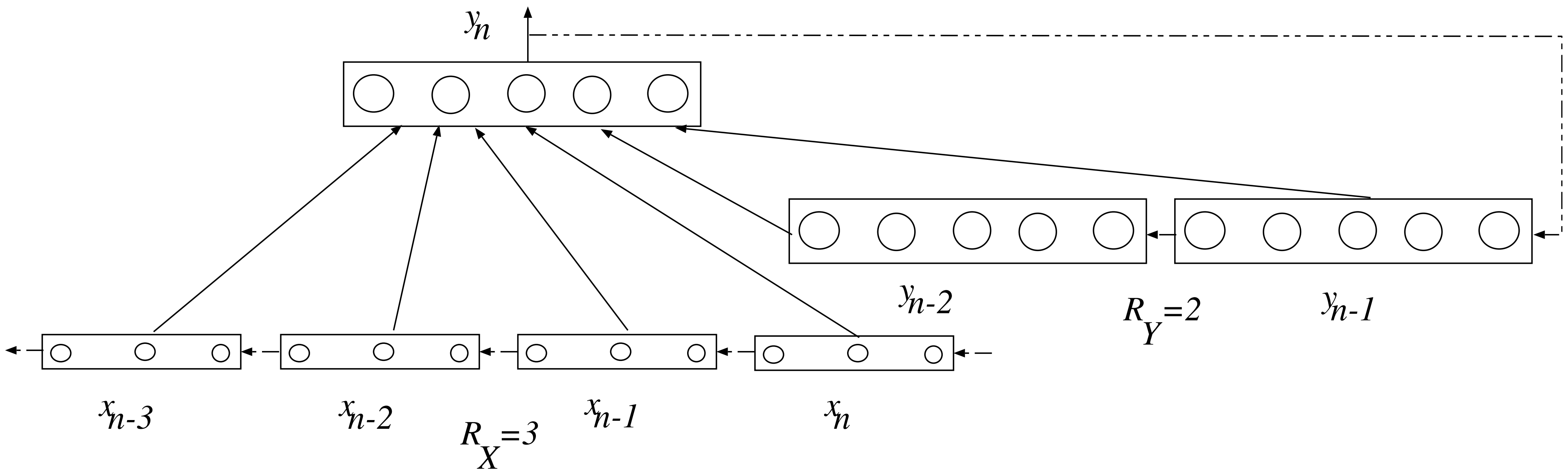



$$y_{0,k} = 0$$

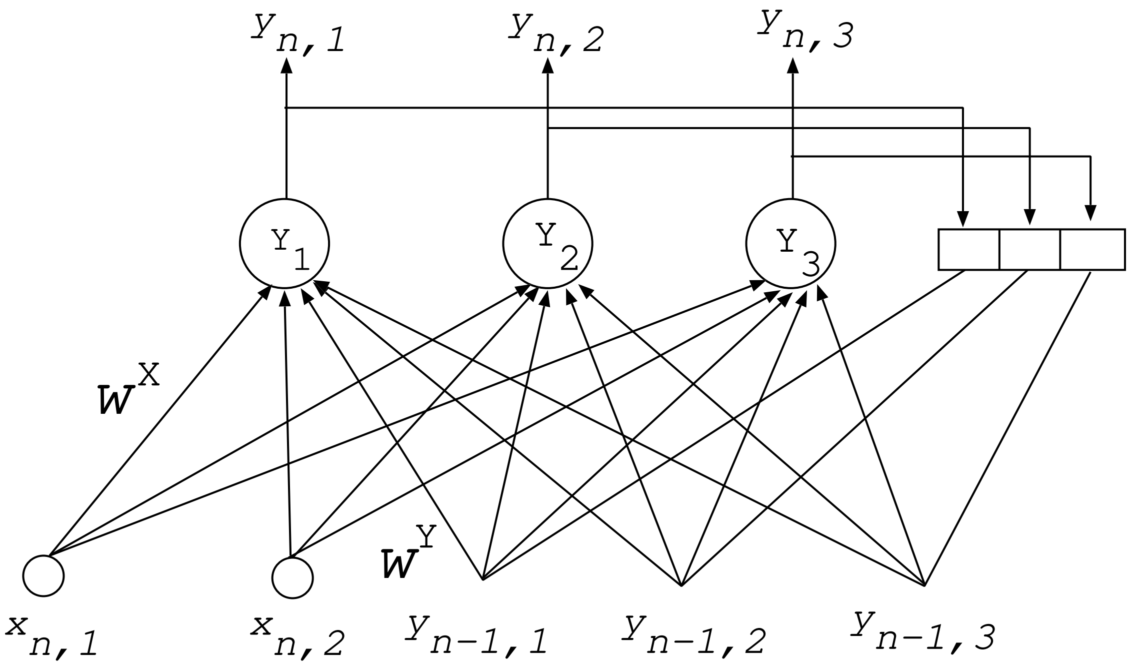
$$y_{n,k} = f\left(\sum_{l \in X} \sum_{l \in Y} \omega_{k,l,j}\; y_{n-1,l}\; x_{n,j}\right)$$

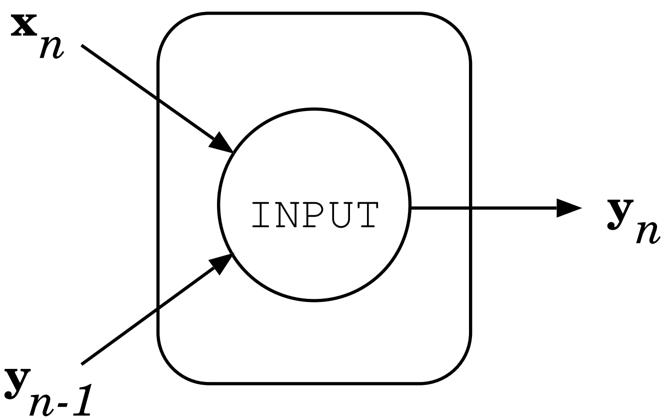


# Other simple recurrent network: NARX



# Simple units



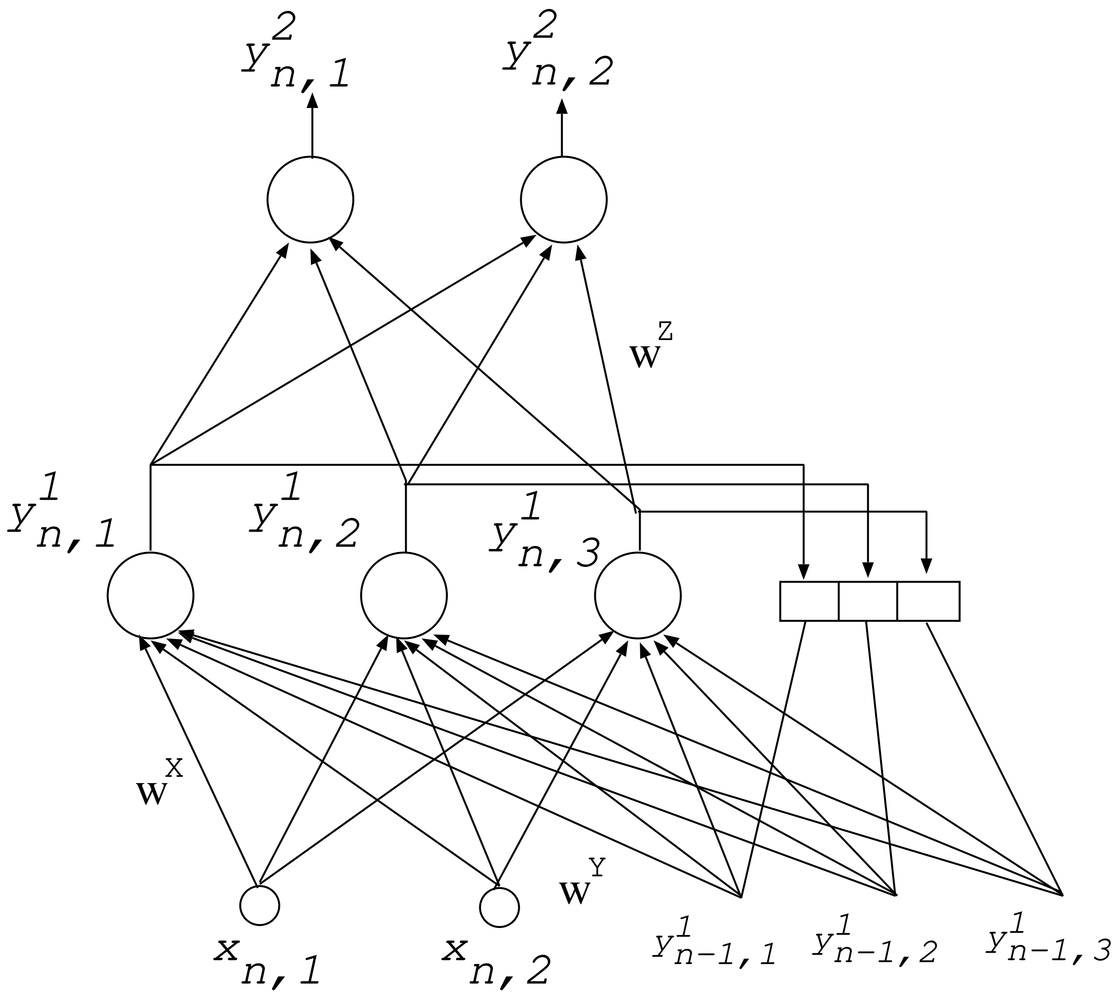

- $\mathbf{y}_n = \mathbf{f}(\mathbf{W}_X \ \mathbf{x}_n + \mathbf{W}_Y \ \mathbf{y}_{n-1})$

# Index

# Augmented recurrent neural networks

An augmented recurrent-neural network is a composition of a simple recurrent-neural network and a feed-forward neural network.

An Elman network: the feed-forward neural network has only one layer: $d_X$ input units, $d_{Y^1}$ hidden units and $d_{Y^2}$ output units.



# Augmented recurrent neural networks

- For the **hidden layer** $(1 \leq k \leq d_{Y^1})$

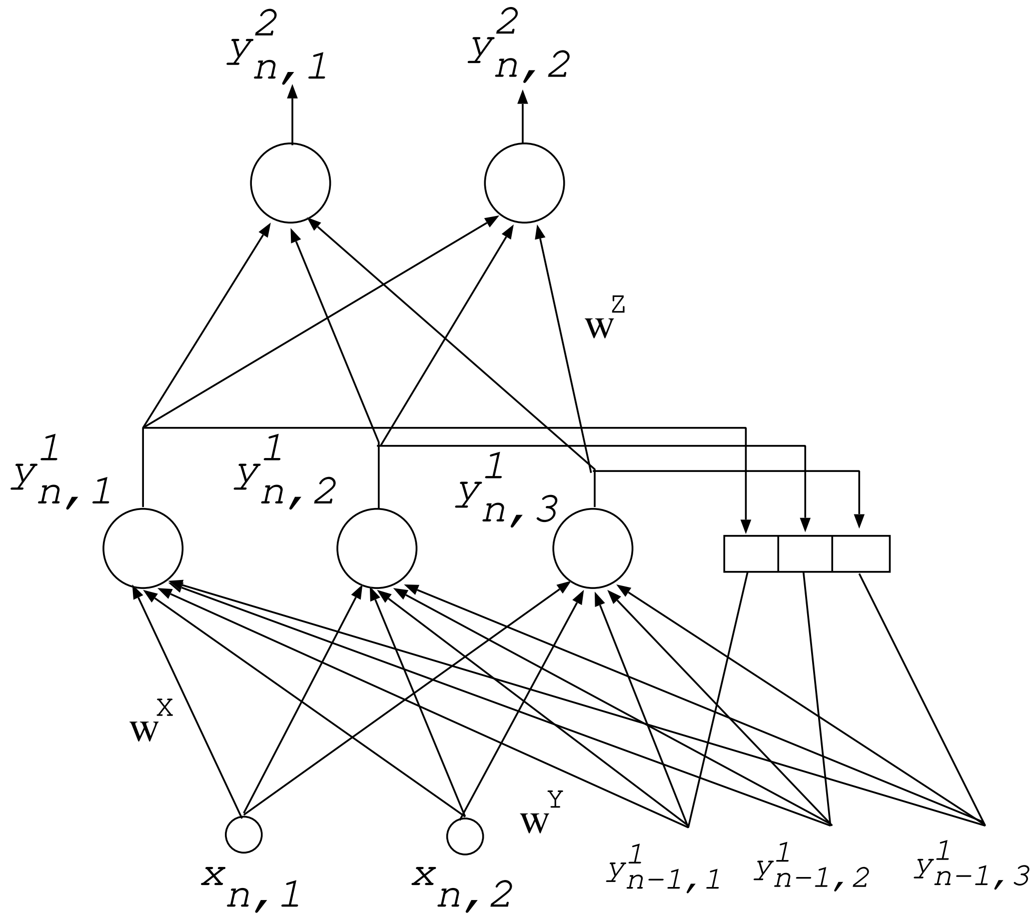
$$y^1_{0,k} = 0$$

$$y^1_{n,k} = f(h^1_{n,k}) =$$

$$f\left(\sum_{l=1}^{d_X} \omega^X_{k,l}\; x_{n,l} + \sum_{l=1}^{d_{Y^1}} \omega^Y_{k,l}\; y^1_{n-1,l}\right)$$

- For the **output layer** $(1 \leq p \leq d_{Y^2})$

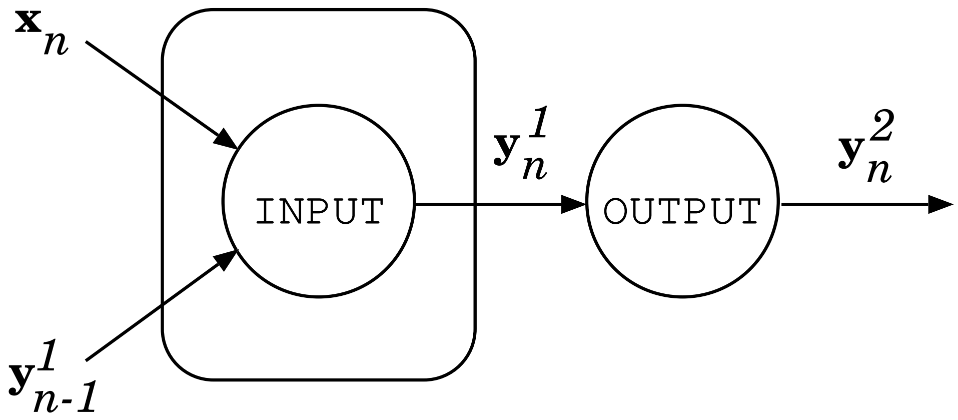
$$y^2_{n,p} = f(h^2_{n,p}) = f\left(\sum_{l=1}^{d_{Y^1}} \omega^Z_{p,l}\; y^1_{n,l}\right)$$



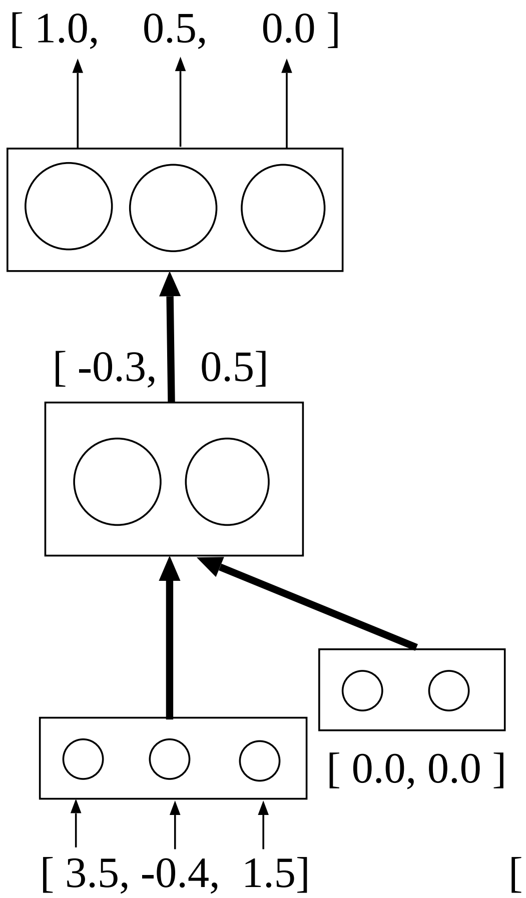
# Augmented recurrent neural networks

In a compact notation:

$$\mathbf{y}_n^2 = \mathbf{f}(\mathbf{W}_Z\ \mathbf{y}_n^1) = \mathbf{f}(\mathbf{W}_Z\ \mathbf{f}(\mathbf{W}_X\ \mathbf{x}_n + \mathbf{W}_Y\ \mathbf{y}_{n-1}^1))$$

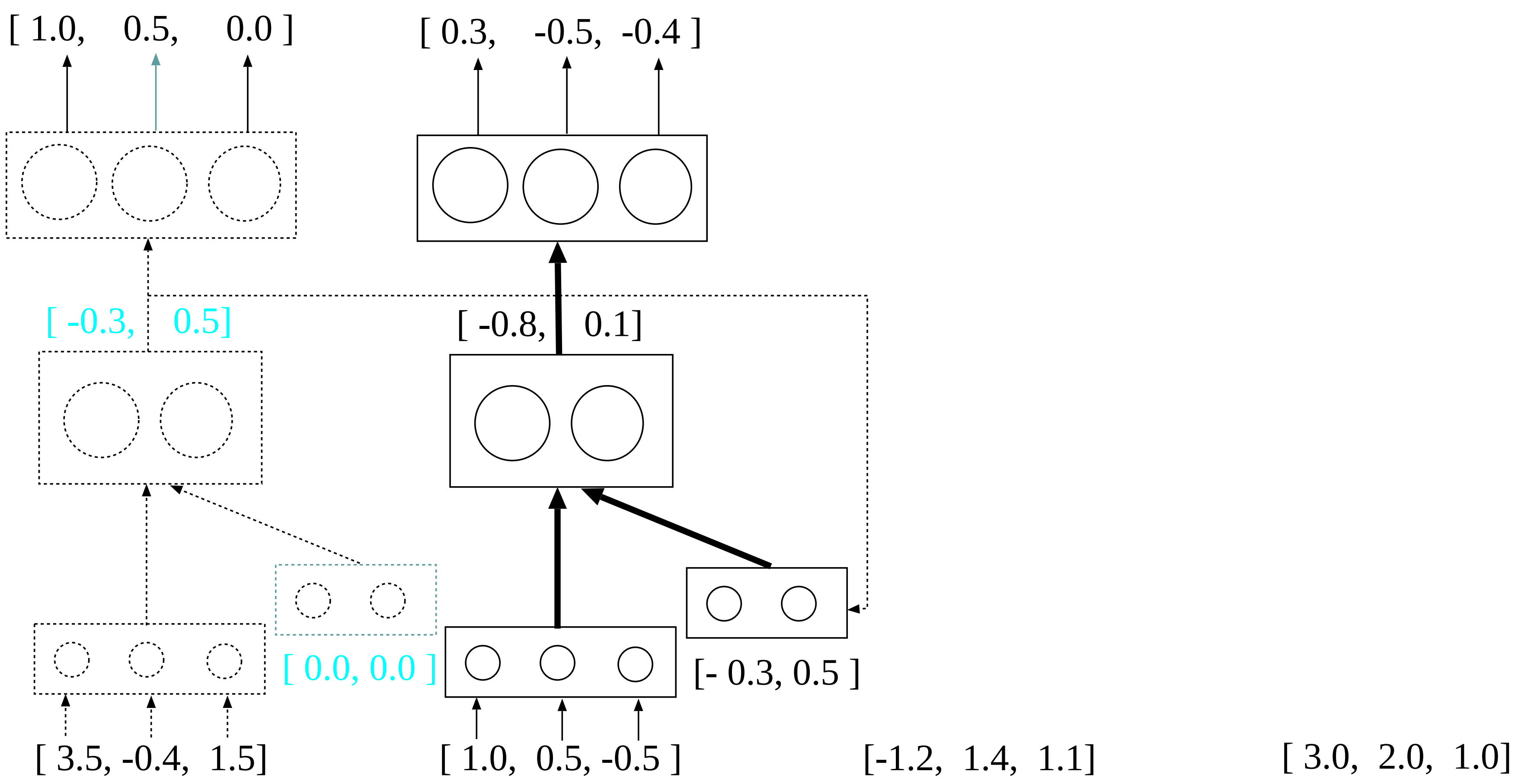


# Sequence processing: augmented recurrent neural networks



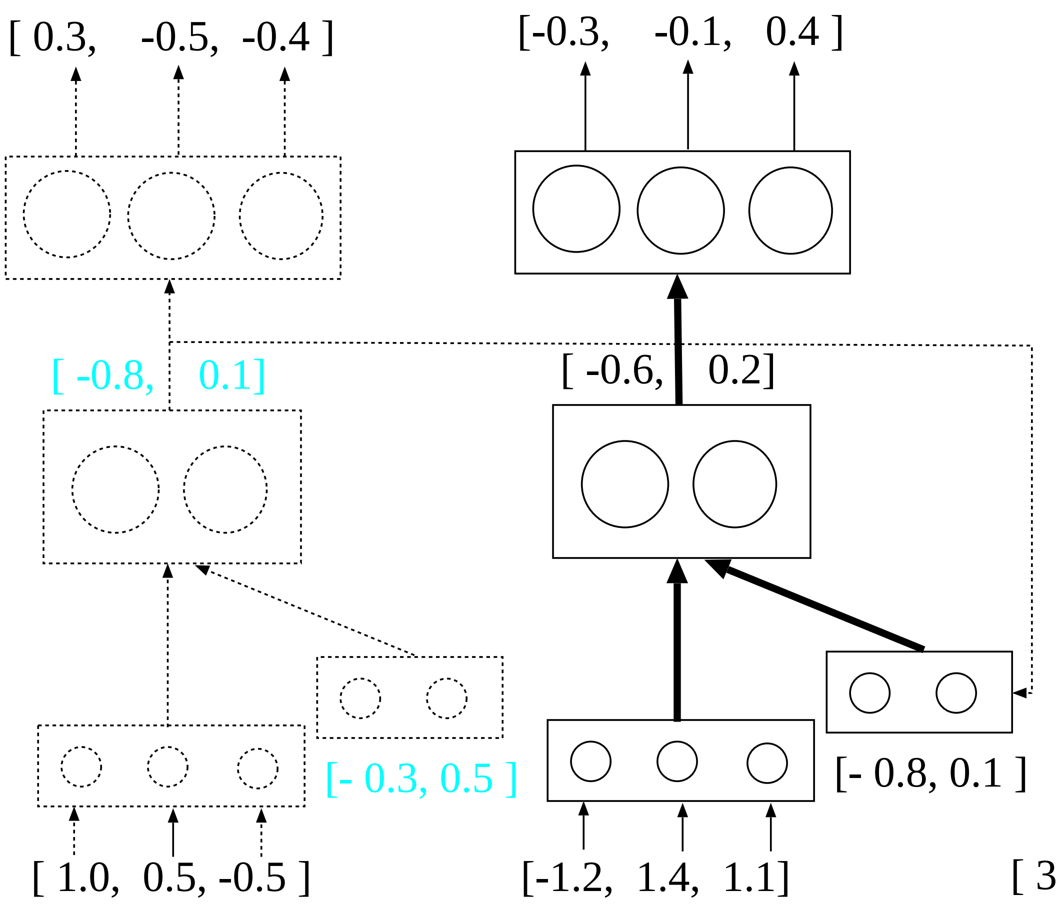

[ 1.0, 0.5, -0.5 ] [-1.2, 1.4, 1.1] [ 3.0, 2.0, 1.0]

# Sequence processing: augmented recurrent neural networks



# Sequence processing: augmented recurrent neural networks

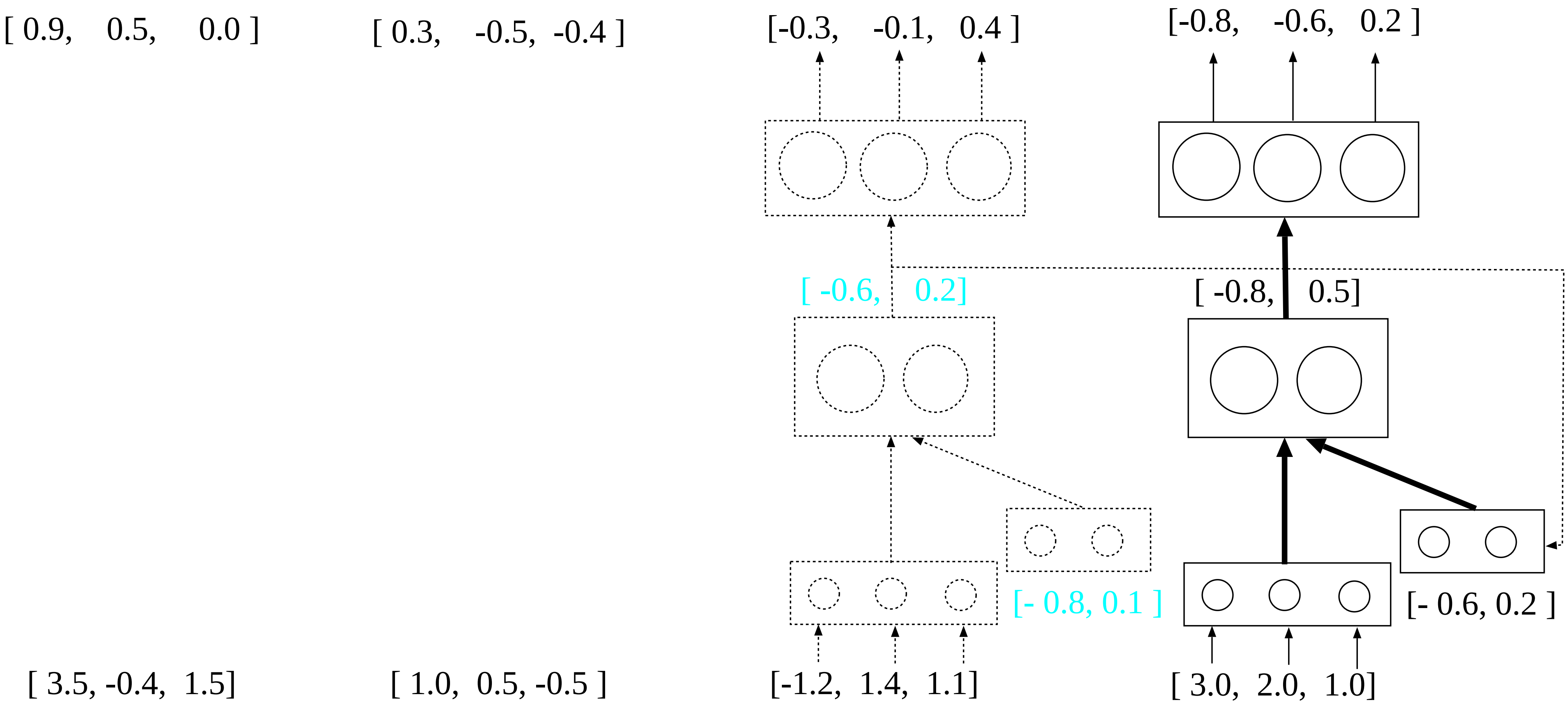
[ 0.9, 0.5, 0.0 ]




[ 3.5, -0.4, 1.5]

[ 3.0, 2.0, 1.0]

# Sequence processing: augmented recurrent neural networks



# Training augmented recurrent neural networks (regression)

- **A training sequence** $A = (\mathbf{x}_n, \mathbf{t}_n)_{n=1,..,N} : \mathbf{x}_n \in \mathbb{R}^{d_X}$ y $\mathbf{t}_n \in \mathbb{R}^{d_{Y^2}}$

- The **error in unit** $k$ **in** $n$ is: $e^2_{n,k} = t_{n,k} - y^2_{n,k}$ for $1 \leq n \leq N$ and $k \in Y^2$

- The **error between** $1$ **and** $N$ is

$$\mathcal{E}_A(\mathbf{w}) \; = \; \sum_{n=1}^{N} \frac{1}{2} \sum_{k \in Y} (e^2_{n,k})^2$$

- Search for a (local) minimum of $\mathcal{E}_A$: GRADIENT DESCENT

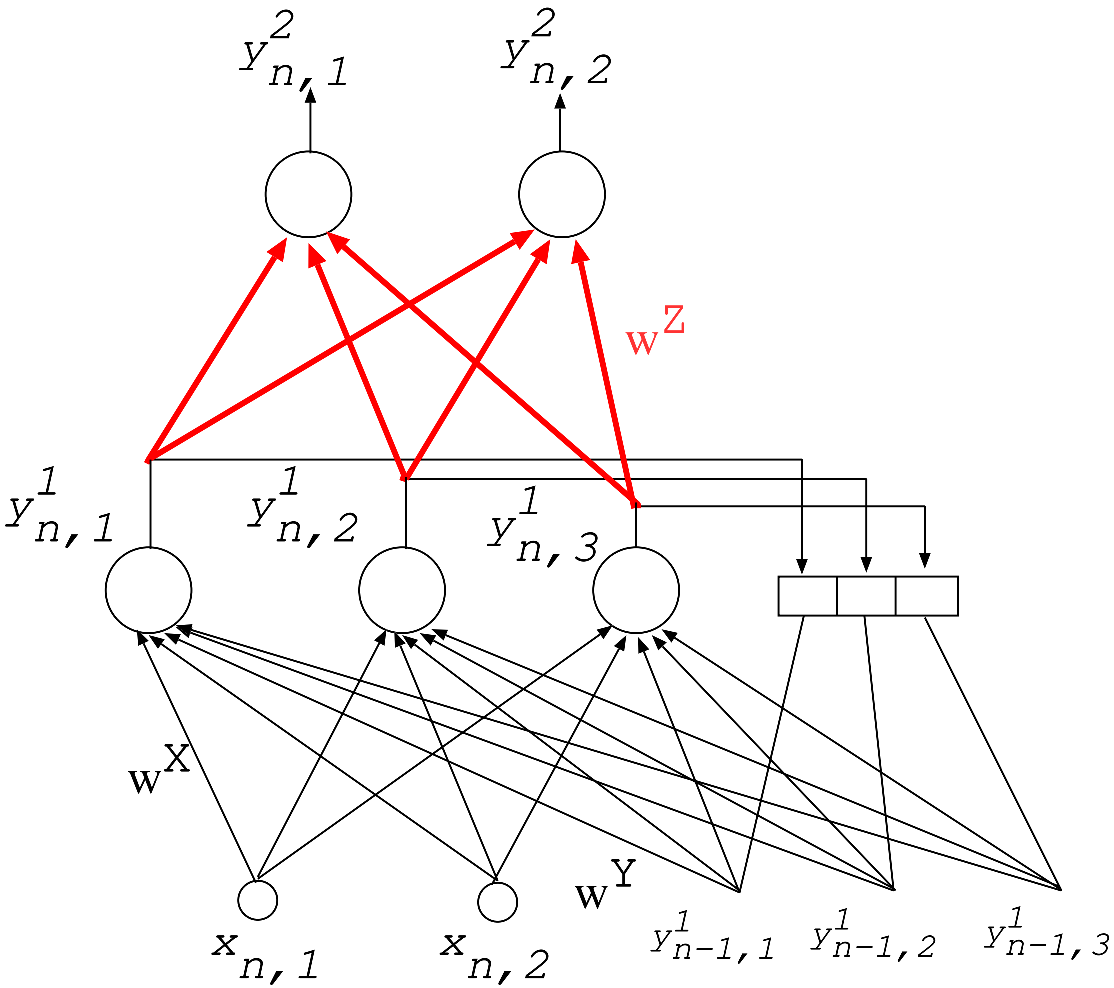
$$\Delta\omega_{i,j} = -\rho \, \frac{\partial \mathcal{E}_A(\mathbf{w})}{\partial \omega_{i,j}} \quad i \in Y^1, \; j \in X \cup Y^1 \;\; \text{and} \;\; i \in Y^2, \; j \in Y^1$$

# Forward-gradient based algorithm

- **Weights for the output layer**
  $1 \leq p \leq d_{Y^2} \wedge 1 \leq l \leq d_{Y^1}$

$$\Delta\omega^Z_{p,l} = \rho \sum_{n=1}^{N} e^2_{n,p}\, y^1_{n,l}$$

$$e^2_{n,p} = (t_{n,p} - y^2_{n,p})\; f'(h^2_{n,p})$$



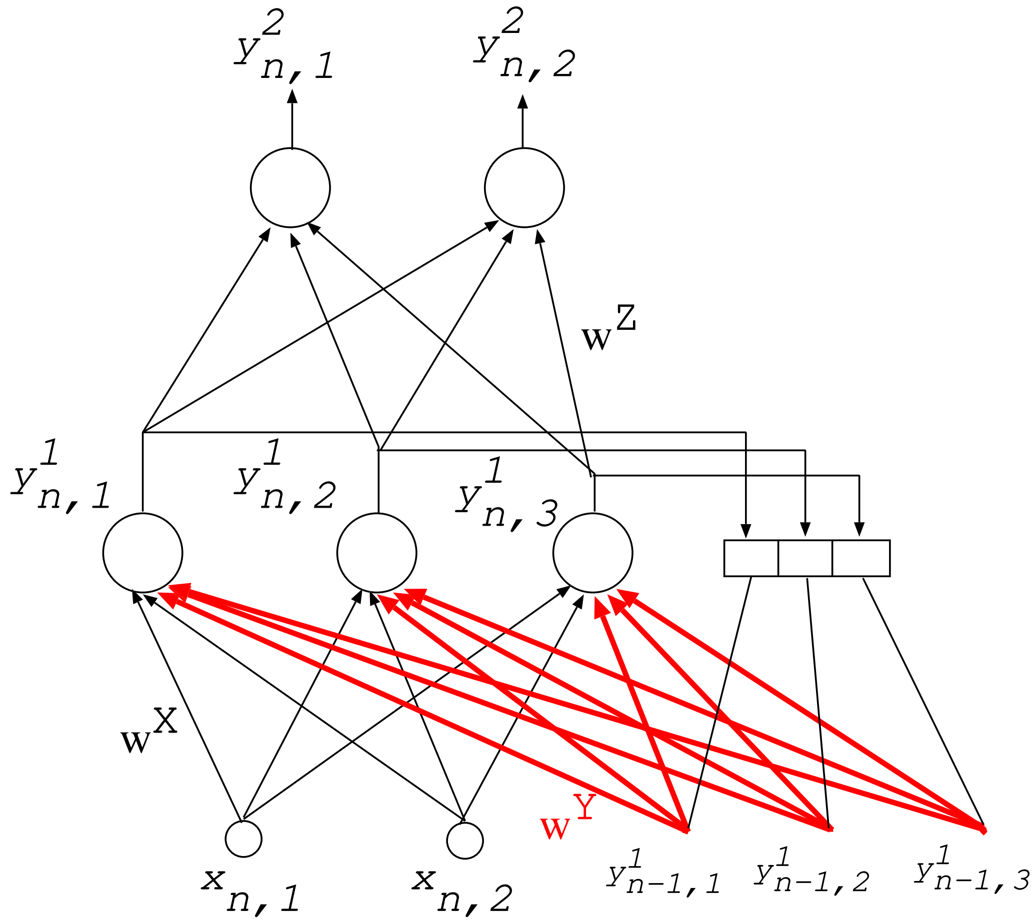
# Forward-gradient based algorithm

- **Weights of recurrent connections**
  $1 \leq k, l \leq d_{Y^1}$

$$\Delta\omega^Y_{k,l} \quad = \quad \rho \sum_{n=1}^{N} \sum_{q=1}^{d_{Y^1}} e^1_{n,q} \, \frac{\partial y^1_{n,q}}{\partial \omega^Y_{k,l}}$$

$$e^1_{n,q} \quad = \quad \sum_p e^2_{n,p} \; \omega^Z_{p,q}$$

$$\frac{\partial y^1_{0,q}}{\partial \omega^Y_{k,l}} = 0$$




$$\frac{\partial y^1_{n,q}}{\partial \omega^Y_{k,l}} \quad = \quad f'(h^1_{n,q}) \; \left( \delta_{k,q} \; y^1_{n-1,l} + \sum_{r=1}^{d_{Y^1}} \omega^Y_{q,r} \frac{\partial y^1_{n-1,r}}{\partial \omega^Y_{k,l}} \right)$$

# Forward-gradient based algorithm

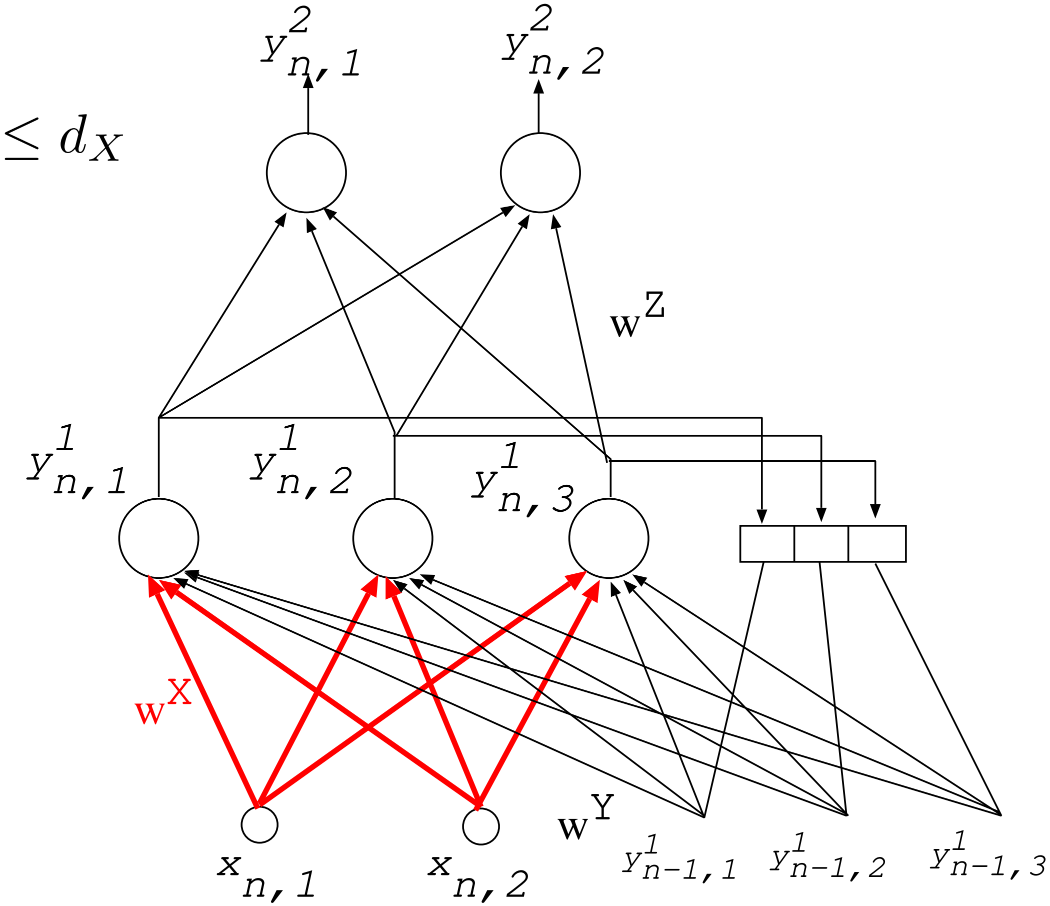
- **Weights for the connections from the input layer.** $1 \le k \le d_{Y^1} \wedge 1 \le l \le d_X$

$$\Delta\omega^X_{k,l} \quad = \quad \rho \sum_{n=1}^{N} \sum_{r=1}^{d_{Y^1}} e^1_{n,r} \, \frac{\partial y^1_{n,r}}{\partial \omega^X_{k,l}}$$

$$e^1_{n,r} \quad = \quad \sum_{p} e^2_{n,p} \; \omega^Z_{p,r}$$

$$\frac{\partial y^1_{0,r}}{\partial \omega^X_{k,l}} = 0$$

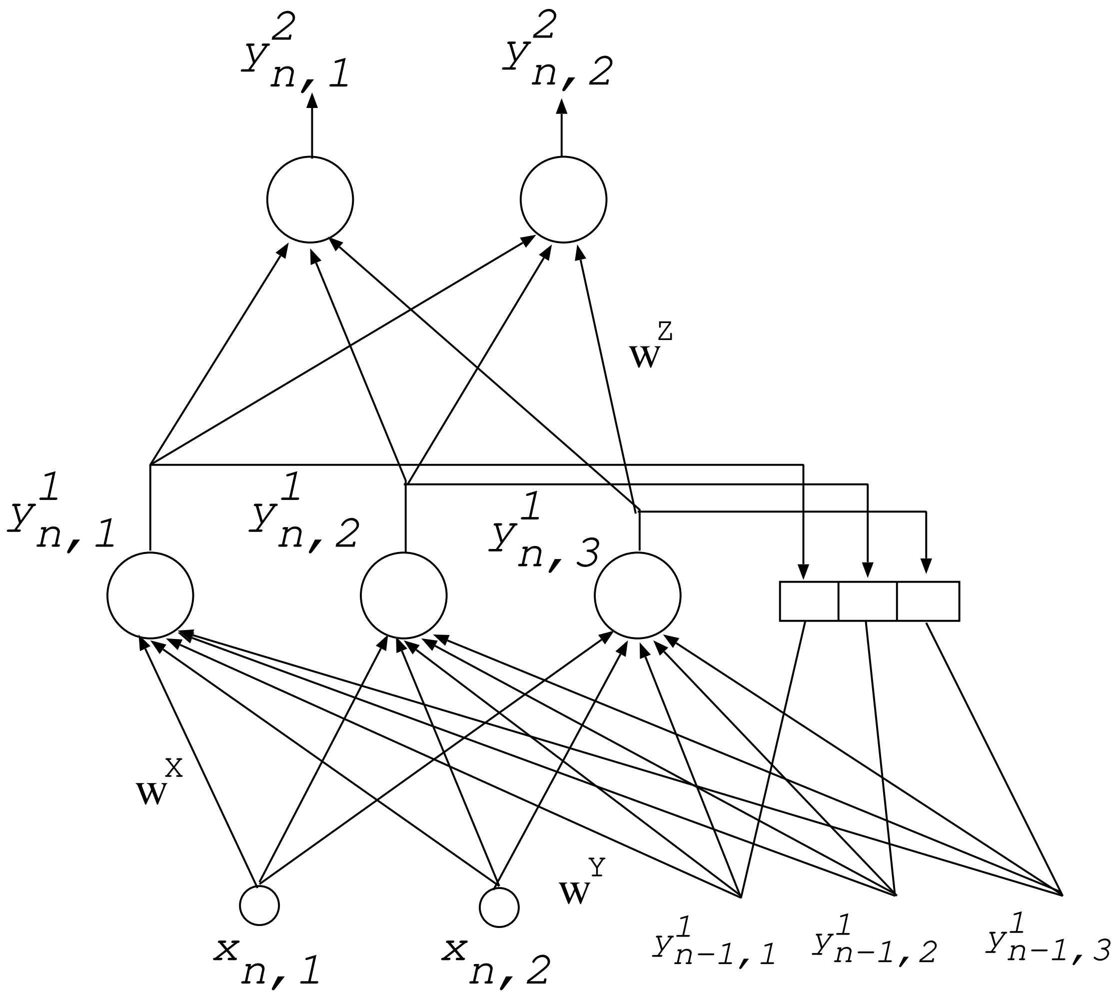
$$\frac{\partial y^1_{n,r}}{\partial \omega^X_{k,l}} \quad = \quad f'(h^1_{n,r}) \left( \delta_{k,r} \; x_{n,l} + \sum_{q=1}^{d_{Y^1}} \omega^X_{q,r} \frac{\partial y^1_{n-1,q}}{\partial \omega^X_{k,l}} \right)$$



# Forward-gradient based algorithm

## Temporal computacional cost:

$$O(((C_{Y^1} + C_X)\ d_H^2 + C_{Y^2})\ L)$$

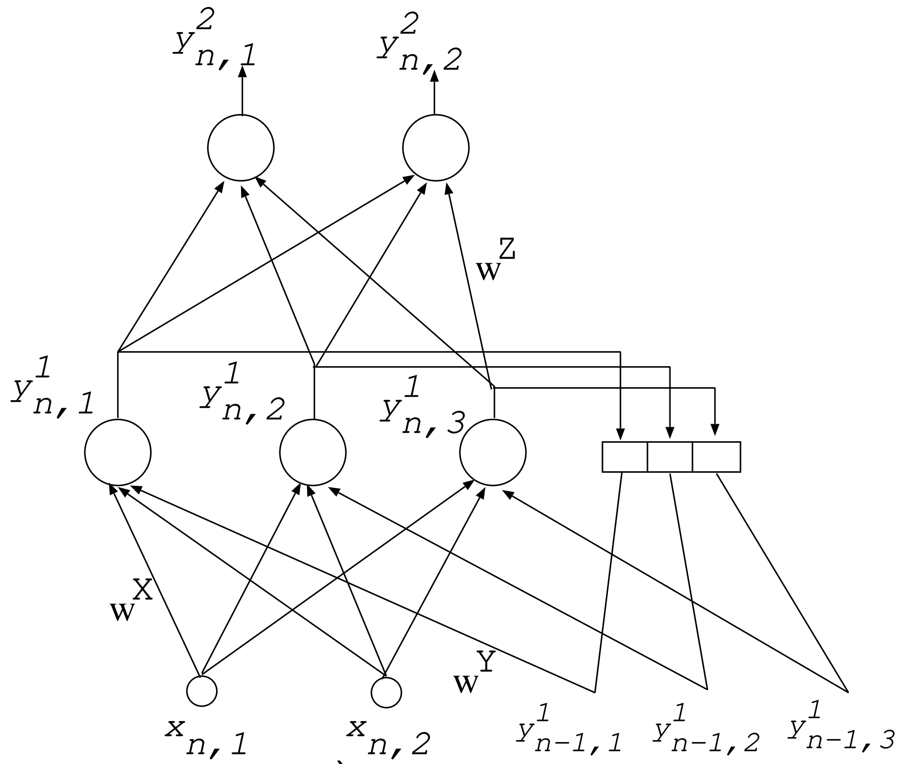


# Other algorithms

- Truncate-gradient algorithm
- Back-propagation through time
- Incremental training
- ...

# Simplified augmented recurrent neural networks

An augmented recurrent network such that each unit has a self-recurrent connections.



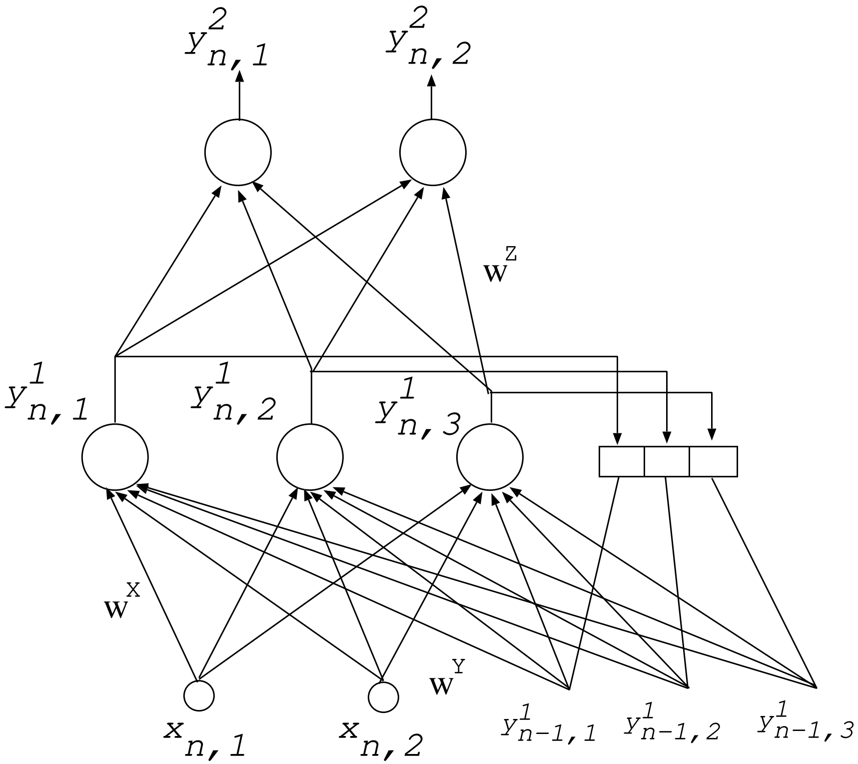

- The **hidden layer**

$$y^1_{n,k} = f(h^1_{n,k}) = f\left(\sum_{l=1}^{d_X} \omega^X_{k,l}\; x_{n,l} + \omega^Y_{k,k}\; y^1_{n-1,k}\right) \text{ with } 1 \le k \le d_{Y^1}$$

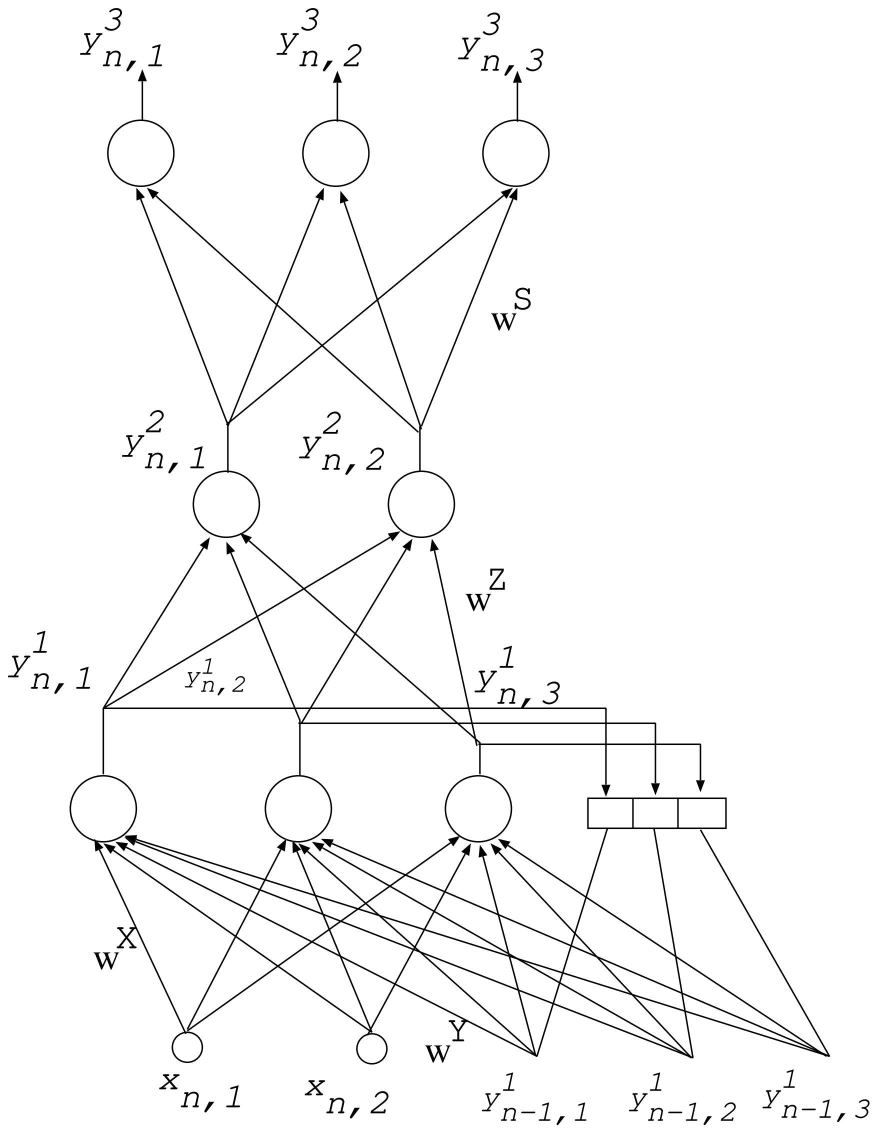
- The **output layer**

$$y^2_{n,p} \;=\; f(h^2_{n,p}) = f\left(\sum_{l=1}^{d_{Y1}} \omega^Z_{p,l}\; y^1_{n,l}\right) \text{ with } 1 \le p \le d_{Y^2}$$

# Generalization

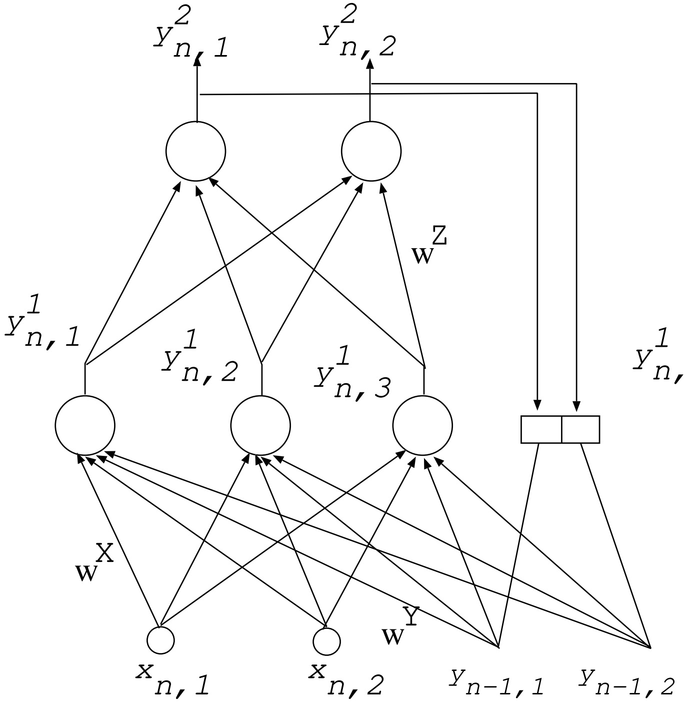



Two-layer recurrent neural network (Elman network)

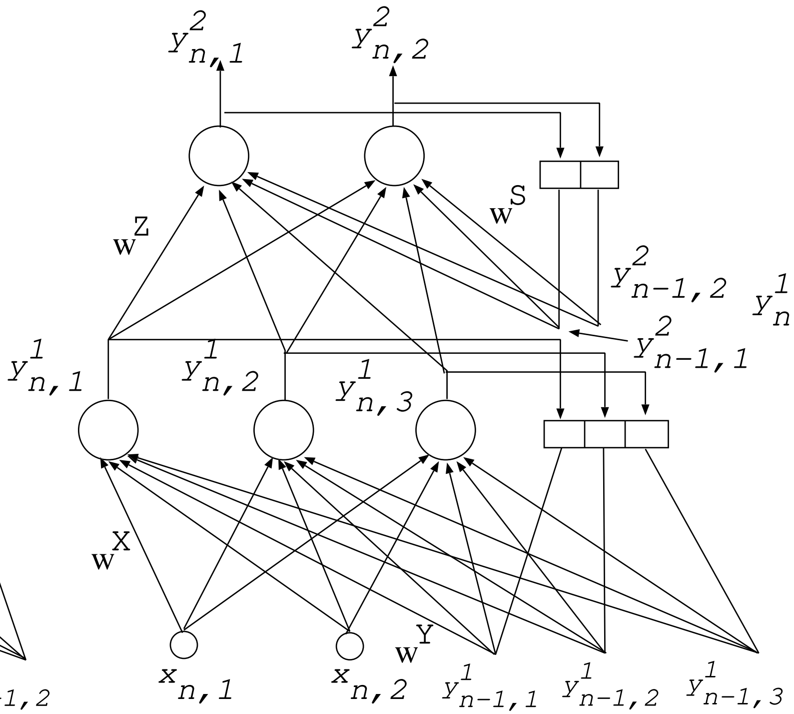



Three-layer recurrent neural network

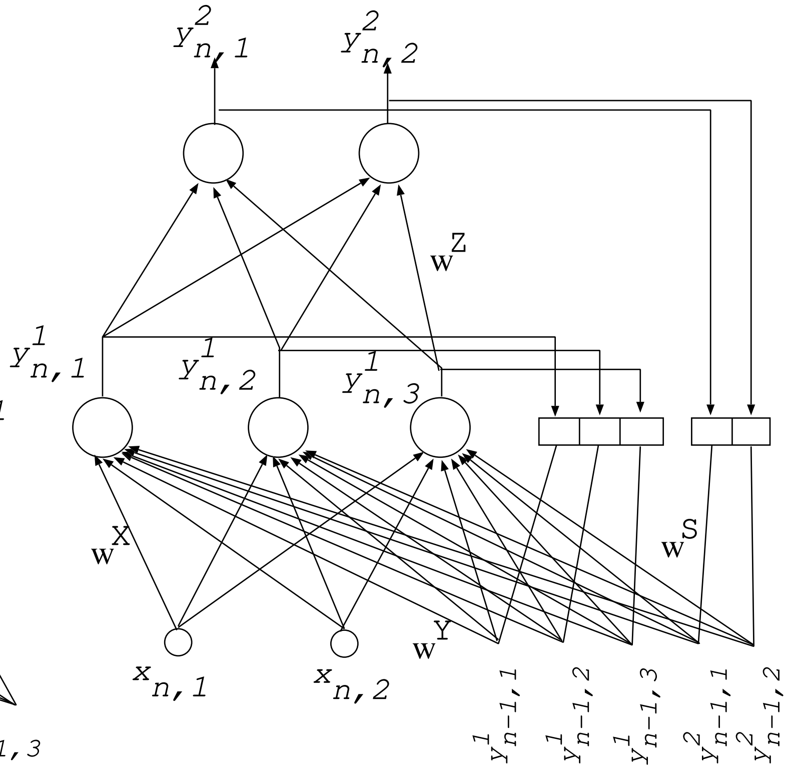
# Generalization




Jordan network




Hybrid network




Hybrid network

# Some computational results

- A first-order simple recurrent neural network cannot implement any finite-state machine (Goudreau et al., 1995).

- Any finite-state machine can be simulated using an Elman network.

- Every Turing machine can be simulated by a second-order simple recurrent neural network.

# Index

# Long Short-Term Memory (LSTM) [Hochreiter 1997]

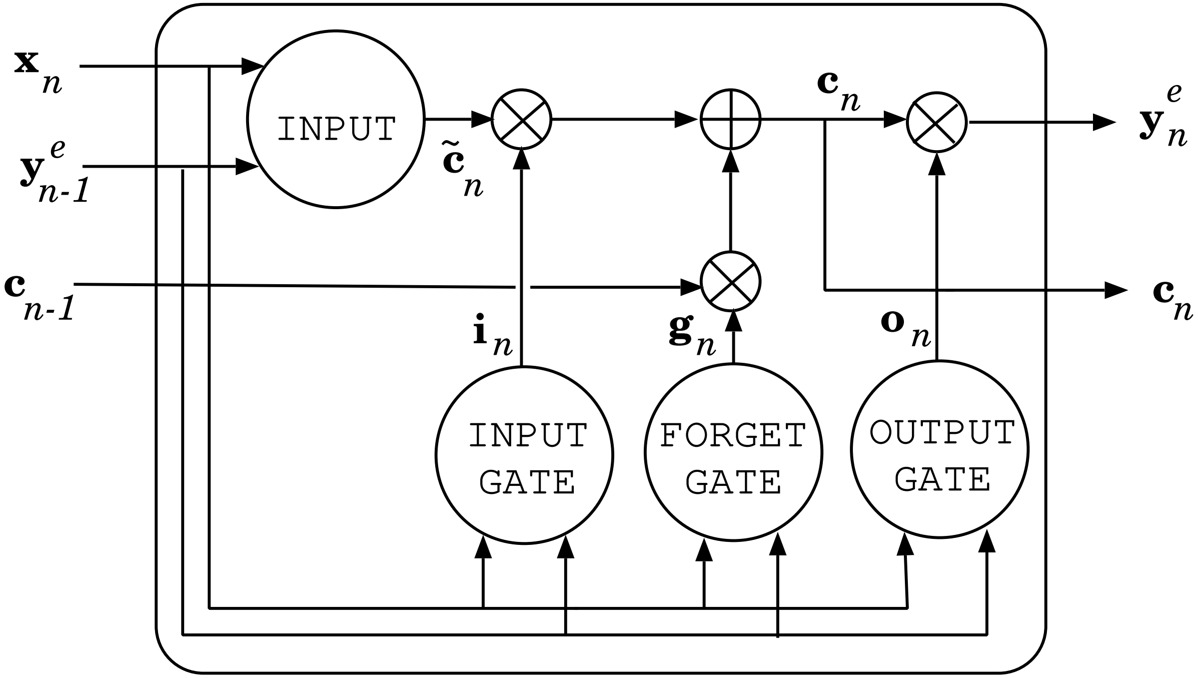
Problems with recurrent neural networks:

- The output depends on the complete past information and sometimes only recent information is needed and sometimes more far context is needed

- In back-propagation through time algorithm and exact gradient or real-time recurrent learning algorithms the errors that propagated backwards in time tend to vanish or to oscillate.

A solution: Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU)

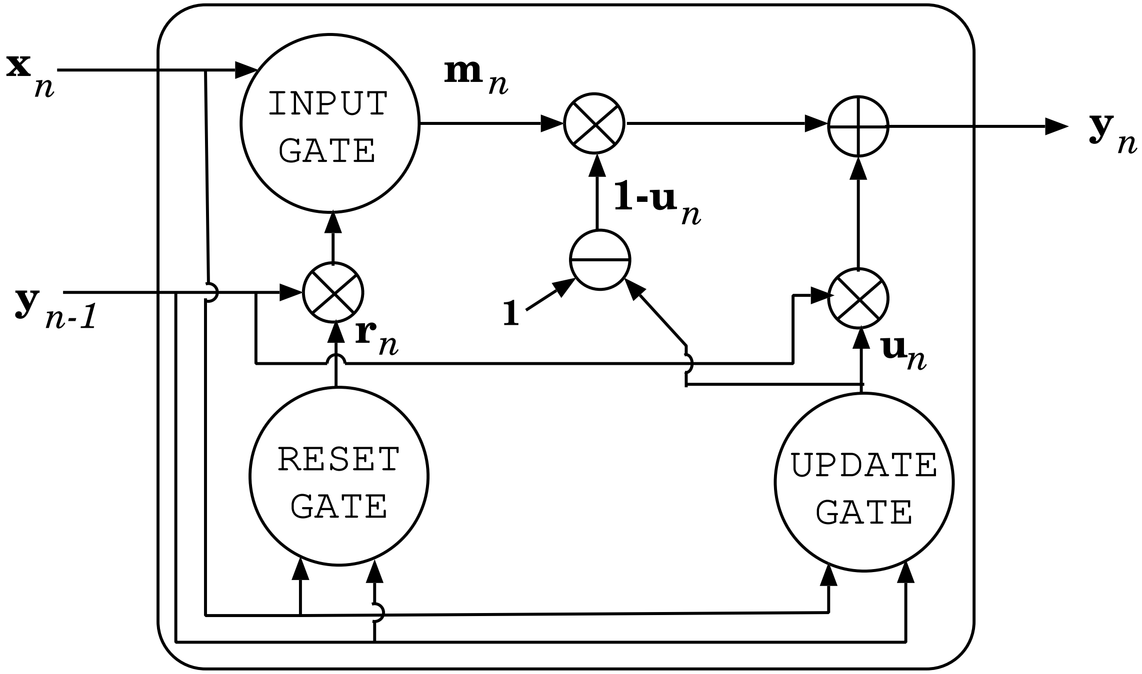
# Long Short-Term Memory (LSTM)




- $\mathbf{i}_n = \mathbf{f}_s(\mathbf{W}^I_Y \mathbf{y}_{n-1} + \mathbf{W}^I_X \mathbf{x}_n)$
- $\mathbf{g}_n = \mathbf{f}_s(\mathbf{W}^F_Y \mathbf{y}_{n-1} + \mathbf{W}^F_X \mathbf{x}_n)$
- $\mathbf{o}_n = \mathbf{f}_s(\mathbf{W}^O_Y \mathbf{y}_{n-1} + \mathbf{W}^O_X \mathbf{x}_n)$
- $\tilde{\mathbf{c}}_n = \mathbf{f}_{th}(\mathbf{W}^C_Y \mathbf{y}_{n-1} + \mathbf{W}^C_X \mathbf{x}_n)$
- $\mathbf{c}_n = \mathbf{g}_n \times \mathbf{c}_{n-1} + \mathbf{i}_n \times \tilde{\mathbf{c}}_n$
- $\mathbf{y}_n = \mathbf{o}_n \times \mathbf{f}_{th}(\mathbf{c}_n)$

$$\mathbf{y}_n \;=\; \mathbf{F}(\mathbf{x}_n, \mathbf{y}_{n-1})$$

# Gated Recurrent Units (GRU) (Cho et al. 2014)




- $\mathbf{r}_n = \mathbf{f}_s(\mathbf{W}_X^R\ \mathbf{x}_n + \mathbf{W}_H^R\ \mathbf{y}_{n-1})$
- $\mathbf{u}_n = \mathbf{f}_s(\mathbf{W}_X^U\ \mathbf{x}_n + \mathbf{W}_H^U\ \mathbf{y}_{n-1})$
- $\mathbf{m}_n = \mathbf{f}_{th}(\mathbf{W}_X^M\ \mathbf{x}_n + \mathbf{W}_H^M\ (\mathbf{r}_n \times \mathbf{y}_{n-1}))$
- $\mathbf{y}_n = \mathbf{u}_n \times \mathbf{y}_{n-1} + (1 - \mathbf{u}_n) \times \mathbf{m}_n = \mathbf{F}(\mathbf{x}_n, \mathbf{y}_{n-1})$

# Index

# Connectionist Temporal Classification (Graves 2006)

- A simple recurrent neural network with a set $U$ of $d_Y + 1$ units (each output unit $k$ has associated a label $l(k) \in \Sigma$ and the $d_Y + 1$ unit is called *no label* or *blank*) and a set $X$ of $d_X$ inputs.
- Input: $\mathbf{x}_n \in \mathbb{R}^{d_X}$ and output: $\mathbf{y}_n \in \mathbb{R}^{d_Y}$ for $1 \leq n \leq N$
- **Total input** of the unit $k \in U$ in "time" $n$:

$$h_{n,k} = \sum_{l \in U} \omega^Y_{k,l} \; y_{n-1,l} + \sum_{l \in I} \omega^X_{k,l} \; x_{n,l}$$

- The **state** of $k \in U$ in $n$:

$$y_{n,k} = \begin{cases} f(h_{n,k}) & n \geq 1 \\ 0 & n = 0 \end{cases}$$

$f$ is the softmax activation function

# Connectionist Temporal Classification (Graves 2006)

- Let $\Sigma'^N = (\Sigma \cup \{blank\})^N$ be the set of $N$-length sequences of labels, and $l'^N_1 \in \Sigma'^N$

- The state of $k \in U$ in $n$ is interpreted as the probability that the label of $\mathbf{x}_n$ is $l'(k) \in \Sigma'^N$: $p(l'(k) \mid \mathbf{x}_n) \equiv y_{n,k}$

- $p(l'^N_1 \mid \mathbf{x}^N_1) \;=\; \prod_{n=1}^{N} p(l'_n \mid l'^{n-1}_1, \mathbf{x}_n) \;\approx\; \prod_{n=1}^{N} y_{n,\pi_n}$

- Let $\mathcal{G} : \Sigma'^N \rightarrow \Sigma^M$ with $M \leq N$ for removing the no-labels or the blanks. Therefore, for $l^M_1 \in \Sigma^M$

$$p(l^M_1 \mid \mathbf{x}^N_1) \;=\; \sum_{l'^N_1 : \mathcal{G}(l'^N_1) = l^M_1} p(l'^N_1 \mid \mathbf{x}^N_1) \approx \max_{l'^N_1 : \mathcal{G}(l'^N_1) = l^M_1} p(l'^N_1 \mid \mathbf{x}^N_1)$$

# Connectionist Temporal Classification (Graves 2006)

- Decoding: Given a input sequence $\mathbf{x}_1^N$, search for a sequence of labels such that:

$$\widehat{l_1^{\widehat{M}}} = \operatorname*{argmax}_{M, l_1^M, M \leq N} p(l_1^M \mid \mathbf{x}_1^N)$$

  Approximations

  – Viterbi decoding: $\approx \mathcal{G}(\operatorname*{argmax}_{l'^N_1} p(l'^N_1 \mid \mathbf{x}_1^N))$

  – Prefix search decoding: adaptation of the forward-backward algorithm

- Training: Given a **training sequence** $A = (\mathbf{x}_1^N, t_1^M)$ , the goal function is the log-likelihood (equivalent to cross-entropy in this case):

$$\mathcal{L}_A(\mathbf{w}) = -\log(p(t_1^M \mid \mathbf{x}_1^N))$$

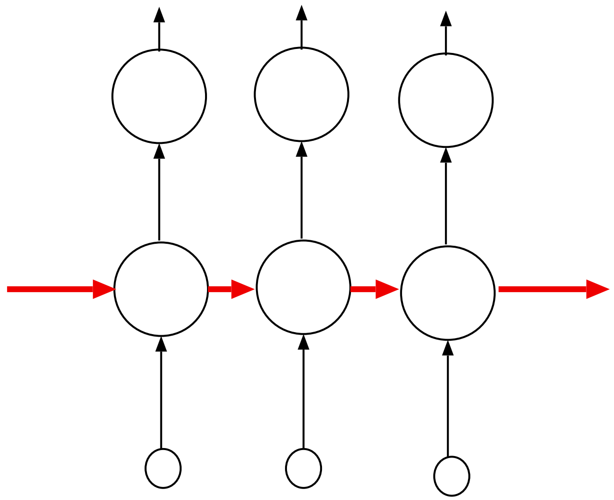
  – Forward-Backward algorithm

# Index

# Motivation (Schuster 1997)

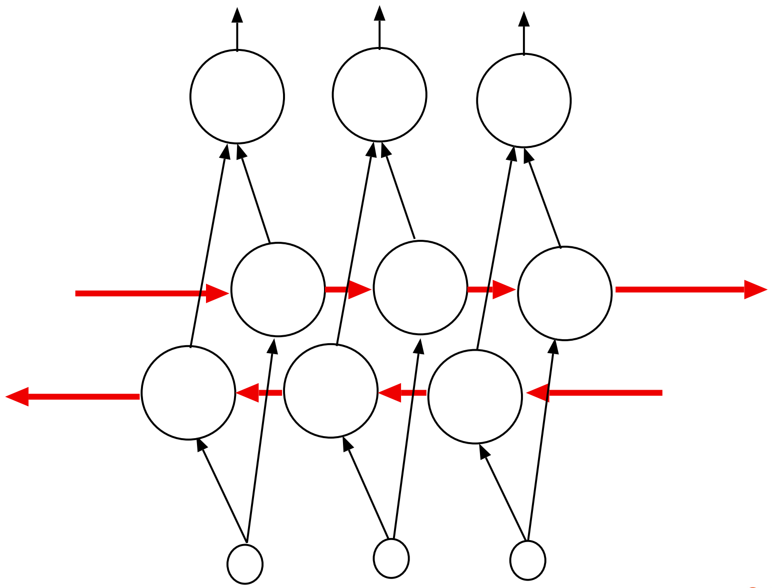
- Networks with bounded-memory: dependencies with the last $T$ inputs.
- Synchronous recurrent neural networks: dependencies with all past inputs.
- Dependencies with the future? i.e. off-line handwritting text recognition.

# Bidirectional recurrent neural networks



Standard dynamics

A recurrent neural network



Bidirectional dynamics

# Bidirectional recurrent neural networks

- **Forward** $(1 \le k \le d_{Y^1})$

$$y^{1f}_{0,k} \;=\; 0; \qquad y^{1f}_{n,k} \;=\; f\left(\sum_{l=1}^{d_X} \omega^X_{k,l}\, x_{n,l} + \sum_{l=1}^{d_{Y^1}} \omega^Y_{k,l}\, y^{1f}_{n-1,l}\right) \quad n = 1, \ldots, N$$

- **Backward** $(1 \le k \le d_{Y^1})$

$$y^{1b}_{N+1,k} \;=\; 0; \qquad y^{1b}_{n,k} \;=\; f\left(\sum_{l=1}^{d_X} \omega^X_{k,l}\, x_{n,l} + \sum_{l=1}^{d_{Y^1}} \omega^Y_{k,l}\, y^{1b}_{n+1,l}\right) \quad n = N, \ldots, 1$$

- The **output** $(1 \le p \le d_{Y^2})$

$$y^2_{n,p} \;=\; f\left(\sum_{l=1}^{d_{Y^1}} \omega^Z_{p,l}\, (y^{1f}_{n,l} + y^{1b}_{n,l})\right) \quad n = 1, \ldots, N$$

- The **output** $(1 \le p \le d_{Y^2})$ (an alternative)

$$y^2_{n,p} \;=\; f\left(\sum_{l=1}^{d_{Y^1}} \omega^Z_{p,l}\, [y^{1f}_{n,l}, y^{1b}_{n,l}]\right) \quad n = 1, \ldots, N$$

# Bidirectional recurrent neural networks (Schuster 1997)

Training BRNN

- BPTT

**Training forward pass:** Run all input data through the BRNN and determine all predicted outputs.

1. Do forward pass just for forward states $(n = 1, \ldots, N)$ and backward states $(m = N, \ldots, 1)$
2. Do forward pass for output.

**Training backward pass:** Calculate the part of the objective function derivative for the time slice used in the forward pass.

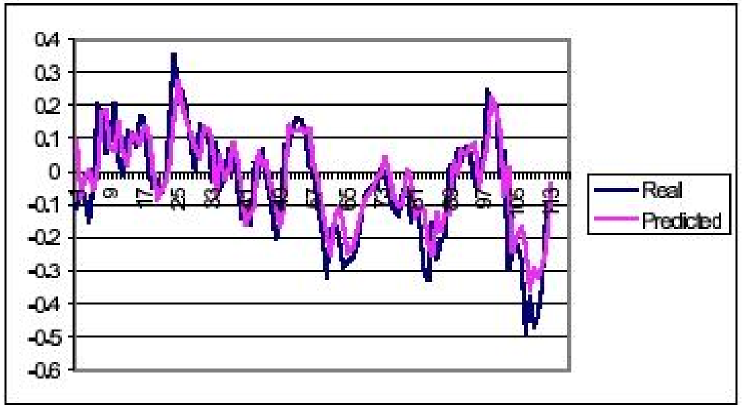
1. Do backward pass for output neurons.
2. Do backward pass just for forward states $(n = N, \ldots, 1)$ and backward states $(n = 1, \ldots, N)$

**Update weights**

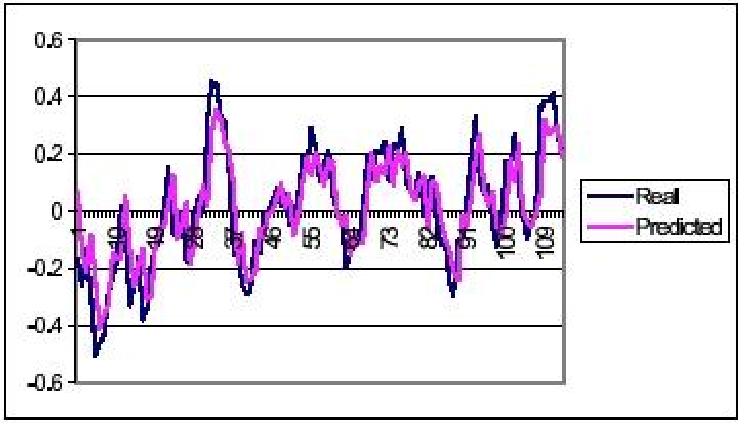
# Index

# Prediction of change of currency (Kondratenko & Kuperin,2003)

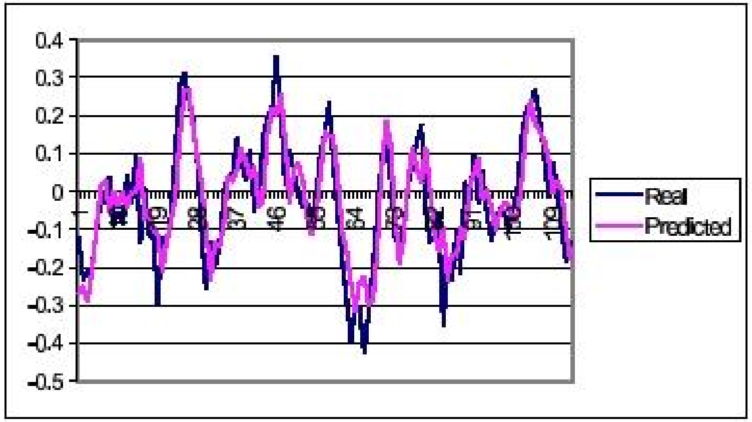
(Euro, dollar, swiss franc, yen, pound (from 18/04/2001 to 1/10/2001)
Hybrid Elman-Jordan network (100 hidden units, windows of five days, prediction to one day, 1,000 training samples)



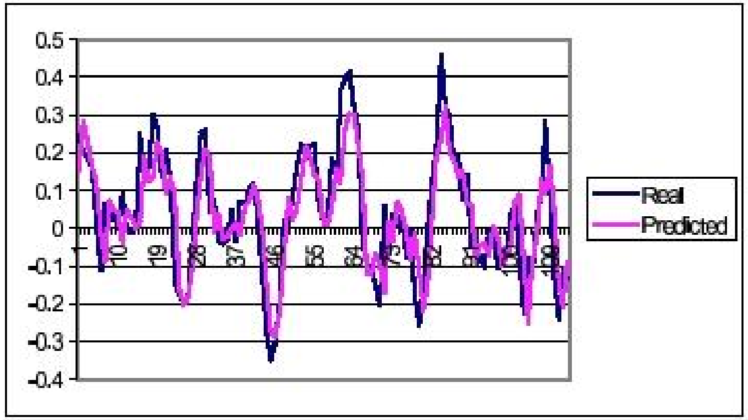

Swiss franc




Euro

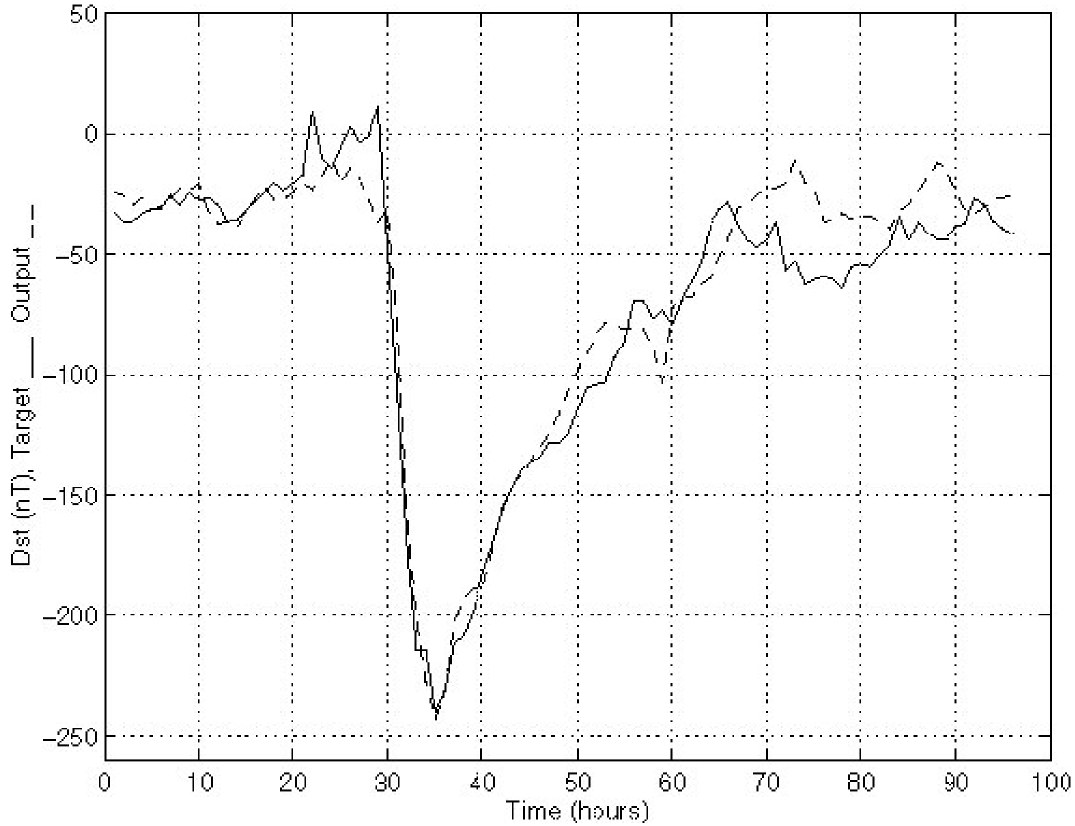



Pound




Yen

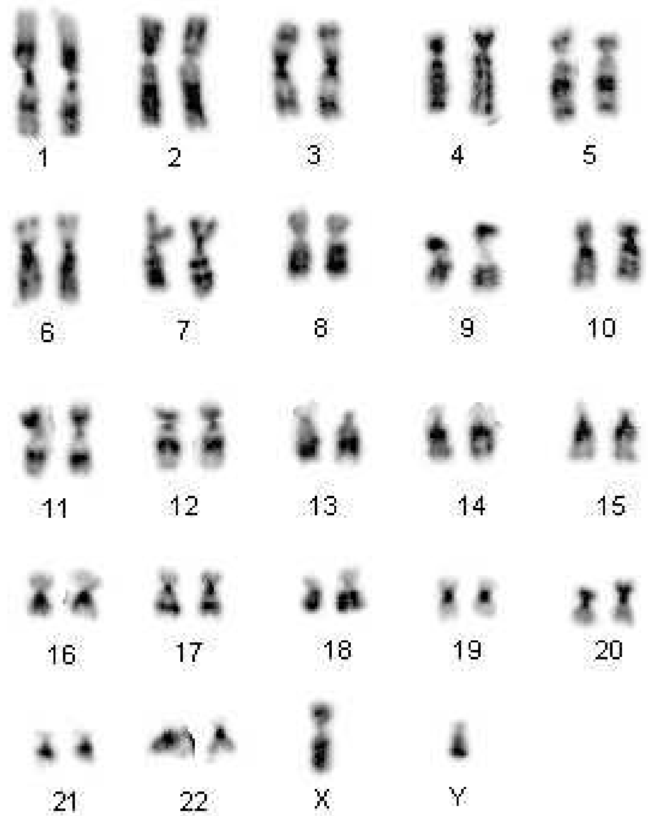
# Prediction of magnetic storms [Lundsteed 2001]




2h ahead based on solar wind data

DST= index that monitorizes worldwide magnetic storm level

# Classification of Chromosomes (Martínez 2007)



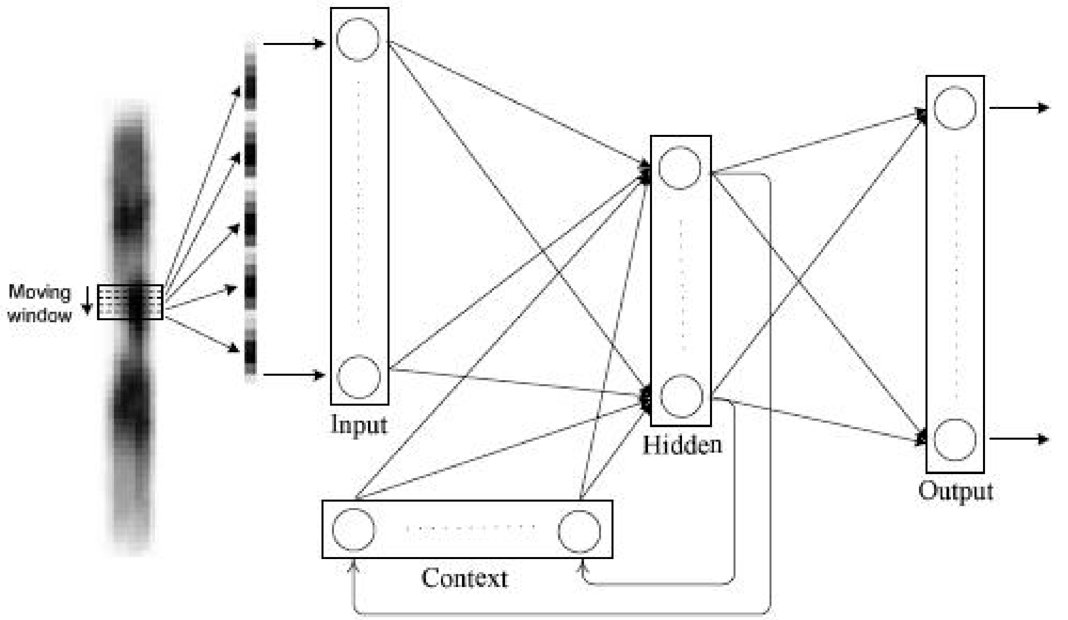




*Figure 4.* Illustration of the use of an Elman network for chromosome classification (chromosome image obtained from the Copenhagen data set).

# Classification of Chromosomes (Martínez 2007)

Copenhahen corpus: 2,804 cells with 48 chromosomes

1. Classification of each frame in one class.

2. Classification of each chromosome by voting

| System | Test-set error % |
|---|---|
| Multilayer perceptron | 6.5 |
| Hierarchical neural networks | 5.6 |
| Continuous HMM (cell dependent) | 4.6 |
| Elman network (cell dependent) | 3.9 |

# Text understanding using Elman networks (Castaño 1995)

- **Problem**: Translate an orthographic representation of a number (between $0$ and $10^6 - 1$) to an arithmetic representation:

/docemilseis/ ⇒ (+2+10)x1000+(+6)

- **Inputs**: a local representation

| | | | | |
|---|---|---|---|---|
| a | e | i | o | u |
| c | r | d | s | v |
| t | q | n | m | l |
| z | h | y | | |

- **Outputs**: a local representation

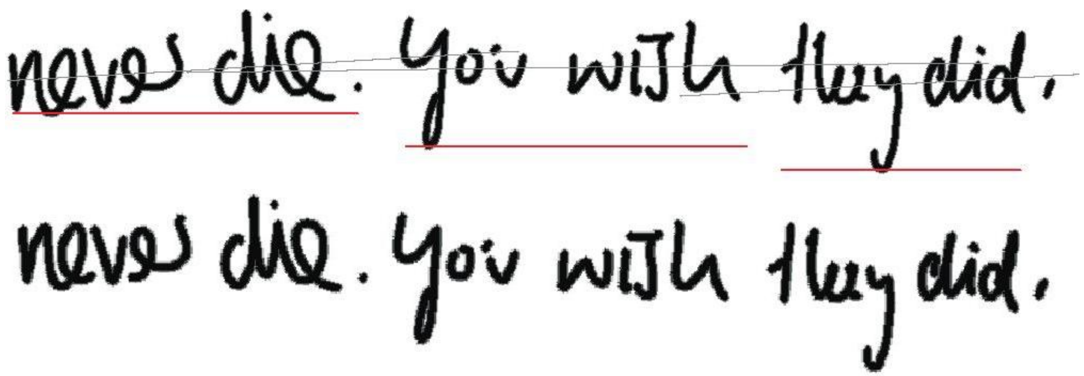
| | | | | | |
|---|---|---|---|---|---|
| +0 | +1 | +2 | +3 | +4 | +5 |
| +6 | +7 | +8 | +9 | +10 | x10 |
| +100 | x100 | +1000 | )x1000+( | | |

# Text understanding using Elman networks

- **Topology**
  - Elman network
  - 19 input units, 16 output units and 40 hidden units
  - windows of one input vector (without distortion) and windows of five input vectors (with distortion)
- **Training**
  - Truncate gradient with momentum and pattern-based training
  - Training corpus: 5000 pairs organized into 5 blocks
  - Convergency: after the presentation of 10 random blocks
- **Test**
  - Test corpus: 2000 (from another 5000) pairs.
  - Percentage of right sentences: 99%
  - Percentage of distorted sentences (10%) right recognized (distorted training): 42%

# Online handwritting text recognition (Liwicki 2007)

IAM-OnDB corpus: 86,272 word instances from a 11,050 word dictionary


| System | Test-set error % |
|---|---|
| Continuous HMM | 34.6 |
| Bidirectional RNN (LSTM) | 26.0 |

# Index

# Bibliography (I)


- Aggarwal. Neural Networks and Deep Learning. Chap. 7. Springer. 2018.
- Alaycrac et al. Flamingo: a Visual Language Model for Few-Shot Learning. NeuroNIPS. 2022.
- Bengio et al. A Neural Probabilistic Language Model. JMLR. 2003.
- Bolaños, A. Peris, F. Casacuberta, S. Soler, P. Radeva. Egocentric Video Description based on Temporally-Linked Sequences. Journal of Visual Communication and Image Representation, 2018.
- Castaño, F. Casacuberta. A Connectionist Approach to Machine Translation. Eurospeech. 1997.
- Cho et. al On the properties of neural machine translation: Encoder-decoder approaches. arXiv. 2014.
- Chowdhery et al. PaLM: Scaling Language Modeling with Pathways. arXiv. 2022.

# Bibliography (II)


- Conneau & Lample. Cross-lingual Language Model Pretraining. NeurNIPS
- Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL. 2019.
- Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ICLR 2020.
- Elliott et. al. Multilingual Image Description with Neural Sequence Models. ICLR. 2016.
- Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Springer. 2012.
- A. Graves, S. Fernández, F. Gómez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. ICML 2006.
- Hawthorne et al. General-purpose, long-context autoregressive modeling with Perceiver AR. ICML 2022.
- Hochreiter & Schmidhuber. "Long short-term memory. Neural Computation. 1997.
- Hoffman et. al. Training Compute-Optimal Large Language Models. arXiv. 2022.

# Bibliography (III)


- Kondratenko & Kuperin. Using Recurrent Neural Networks To Forecasting of Forex. arXiv. 2003.
- Katzer et. al. Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks. IEEE ICRA. 2020.
- Lewis et al. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. ACL. 2020.
- Li & Jia. Visual Question Answering with Question Representation Update. NeurNIPS. 2016.
- Lin et. al. Pre-training Multilingual Neural Machine Translation by Leveraging Alignment Information. arXiv. 2021.
- Liu et al. Multilingual Denoising Pre-training for Neural Machine Translation. TACL. 2020.

# Bibliography (IV)


- Liwicki et al.A Novel Approach to On-Line Handwriting Recognition Based on Bidirectional Long Short-Term Memory Networks. ICDAR 2007.
- Lundsteed. A prototype real-time forecast service of space weather and effects using knowledge-based. Neurocomputing, 2001.
- Luong et al. Effective Approaches to Attention-based Neural Machine Translation. EMNLP. 2015.
- C. Martínez, A. Juan, F. Casacuberta. Iterative Contextual Recurrent Classification of Chromosomes. Neural Processing Letters. 2007.
- Mandic & Chambers. Recurrent Neural Networks for Prediction. John Wiley & Sons 2001.
- Medsker & Jain. Recurrent Neural Networks. CRC Press. 2001.
- Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013.

# Bibliography (V)


- Mishra & Viradiya. Survey of Sentence Embedding Methods. JASC. 2019.
- Peters et. al. Deep contextualized word representations. NAACL. 2018.
- Qiu et. al. Pre-trained Models for Natural Language Processing: A Survey. SCTS. 2020.
- Radford et. al. Improving Language Understanding by Generative Pre-Training. OpenAI. 2018.
- Radford et. al. Learning Transferable Visual Models From Natural Language Supervision. ICML. 2021.
- Rae et. al. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. DeepMind. 2021
- Raffel et. al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. JMLR. 2020.
- Reed at al. A Generalist Agent. MLR. 2022.

# Bibliography (VI)


- Reimers & Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP 2019.
- Romero. Ultra-Large AI Models Are Over. https://thealgorithmicbridge.substack.com/p/ultra-large-ai-models-are-over. 2022.
- Schuster & Paliwal. Bidirectional Recurrent Neural Networks. IEEETSP. 1997.
- Sejnowski & Rosenberg. NETtalk: a parallel network that learns to read aloud. The Johns Hopkins University electrical engineering and computer science technical report. 1986.
- Schuster & Paliwal. Bidirectional recurrent neural networks. IEEE TSP. 1995.
- Sinha et. al. Extractive Text Summarization using Neural Networks. arXiv. 2018.
- Sundermeyer. Improvements in Language and Translation Modeling. Ph.D. RWTH Aachen. 2016.
- Tunstall et al. Efficient Few-Shot Learning Without Prompts. arXiv 2022.
- A. Vaswani et al. Attention Is All You Need. NIPS 2017.

# Bibliography (VII)


- Venugopalan et al. Sequence to Sequence - Video to Text. arXiv. 2015.
- Vinyals et al. A Neural Image Caption Generator (NIC). arXiv. 2015
- Wang et al. End-to-End Transformer Based Model for Image Captioning. AAAI 2022.
- Waibel et. al. Phoneme Recognition Using Time-Delay Neural Networks. IEEE TASSP 1989.
- Williams & Zipser. Gradient-based learning algorithms for recurrent networks and Their Computational Complexity. In "Back-propagation: Theory, Architectures and Applications". 1995.
- Xiong et al. On Layer Normalization in the Transformer Architecture. arXiv. 2020.
- Yang wt. al. XLNet: Generalized Autoregressive Pretraining for Language Understanding. NIPS. 2020.
- Yao & Wan. Multimodal Transformer for Multimodal Machine Translation. ACL. 2020.