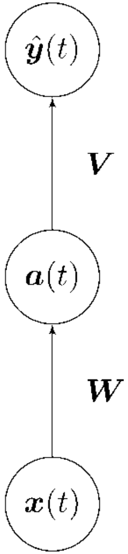
Chapter 1
Recurrent Neural Networks

Recurrent Neural Networks

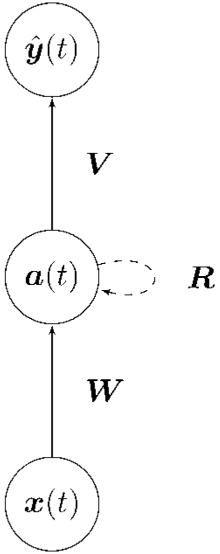
JOHANNES KEPLER UNIVERSITY LINZ

Feedforward Network



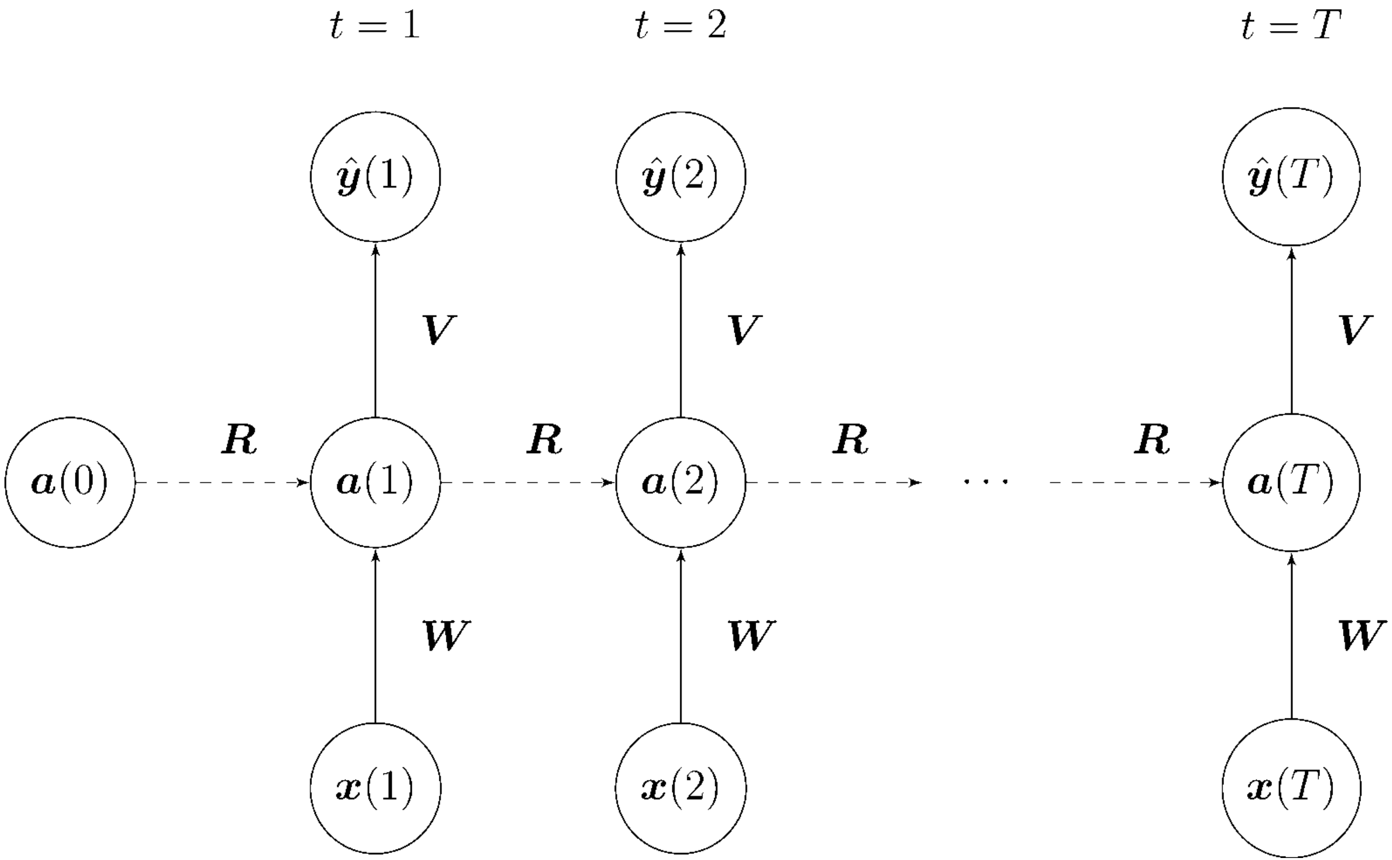
No loops

Recurrent Network



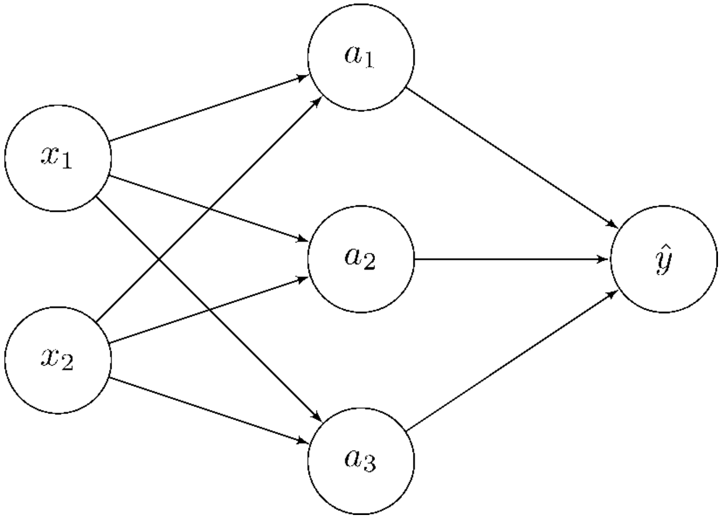
Loops

Recurrent Neural Networks



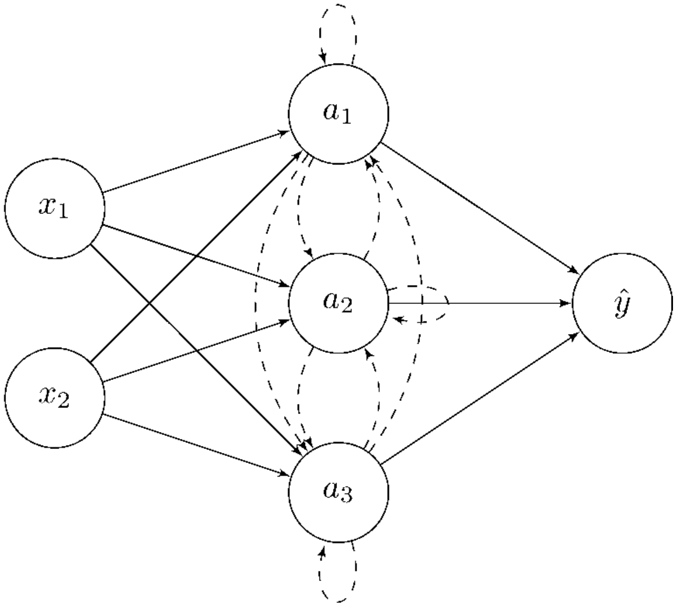
Recurrent Neural Networks

Feedforward Network



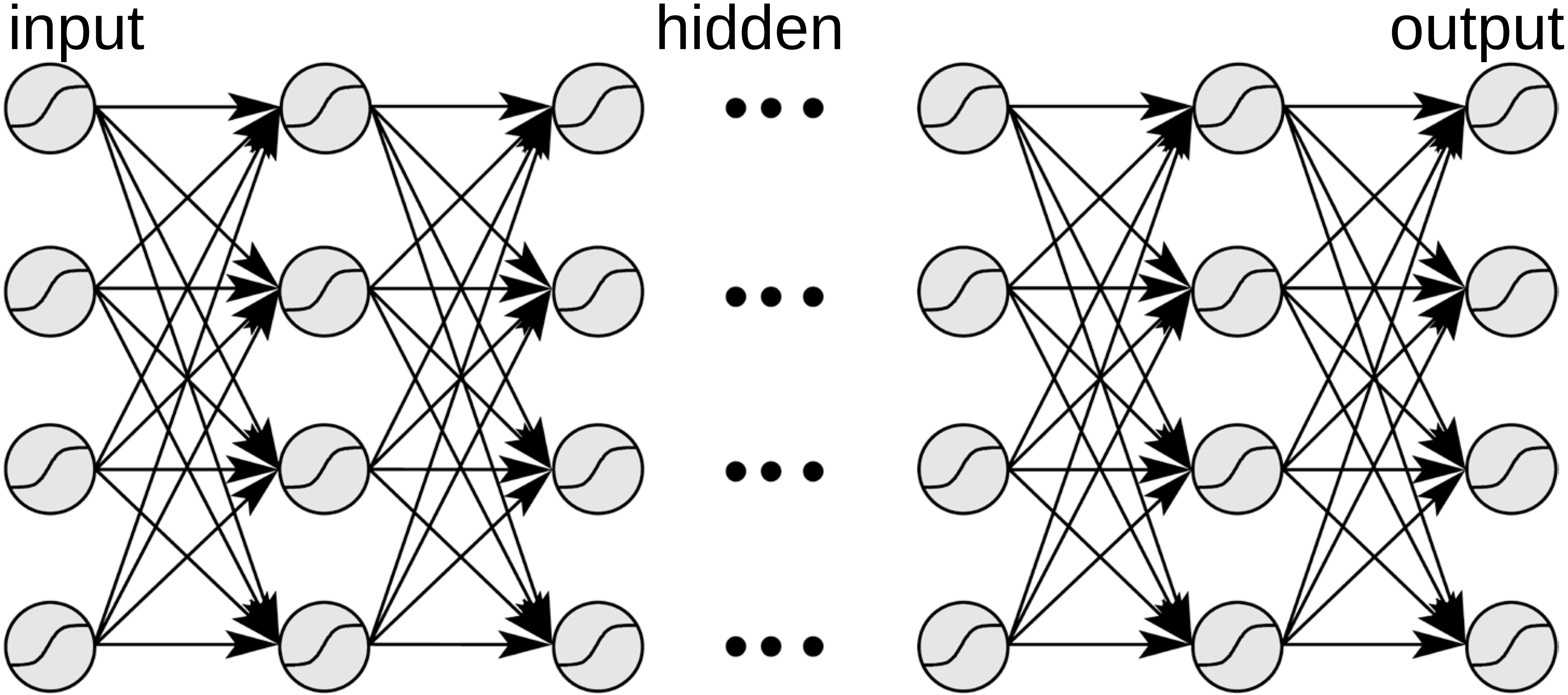
No loops

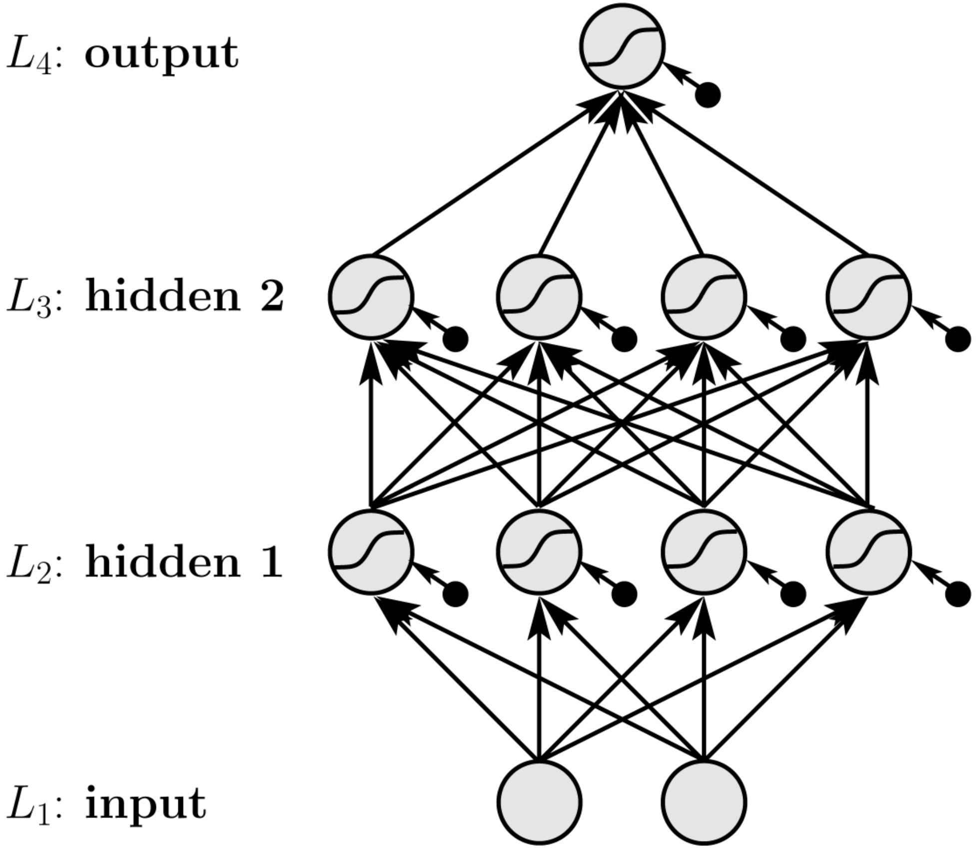
Recurrent Network



Loops

Deep Neural Networks



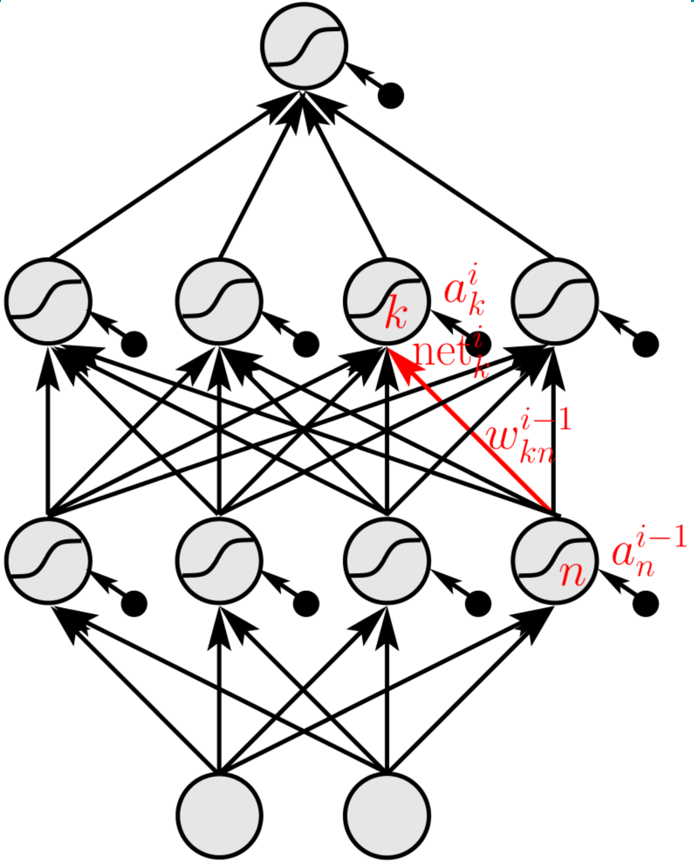

L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**

activity of the kth unit in layer i: a_k^i

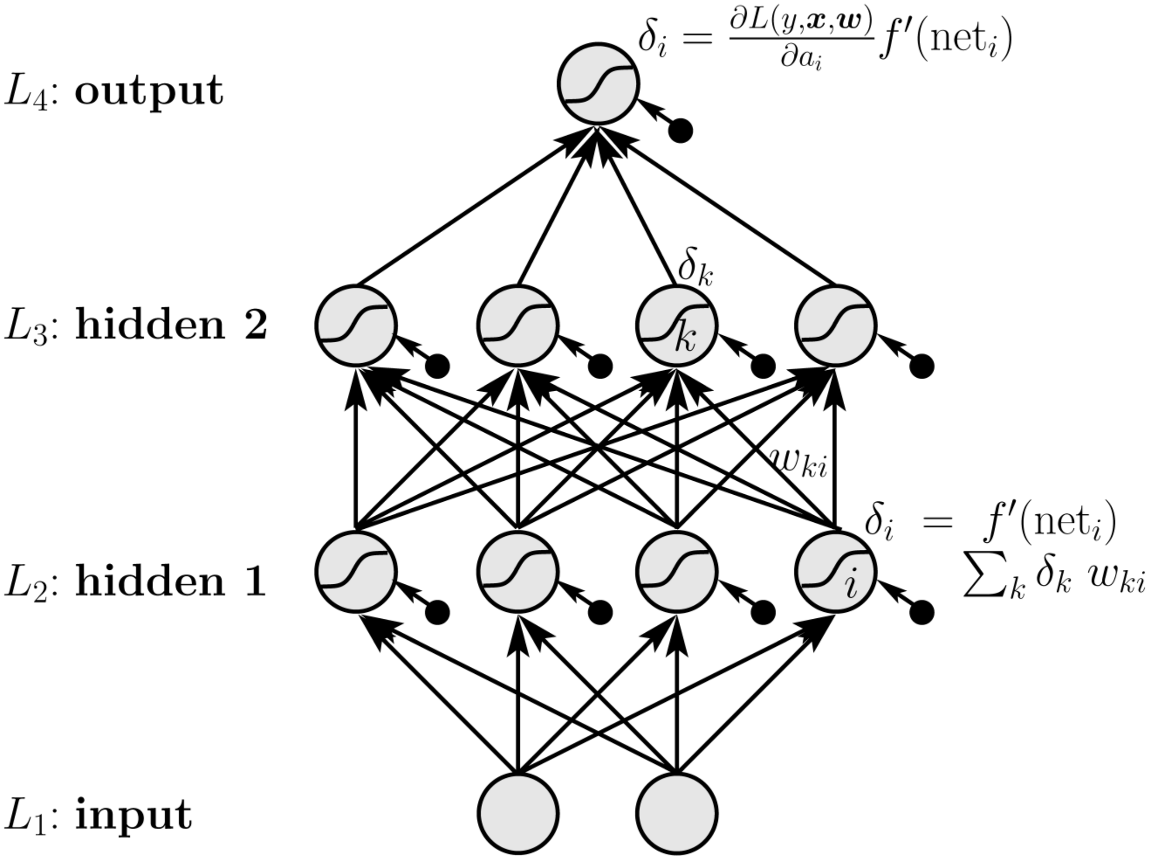
weight from unit n in layer i–1 to unit k in layer i: w_{kn}^{i-1}

network input to the kth unit in layer i: net_k^i

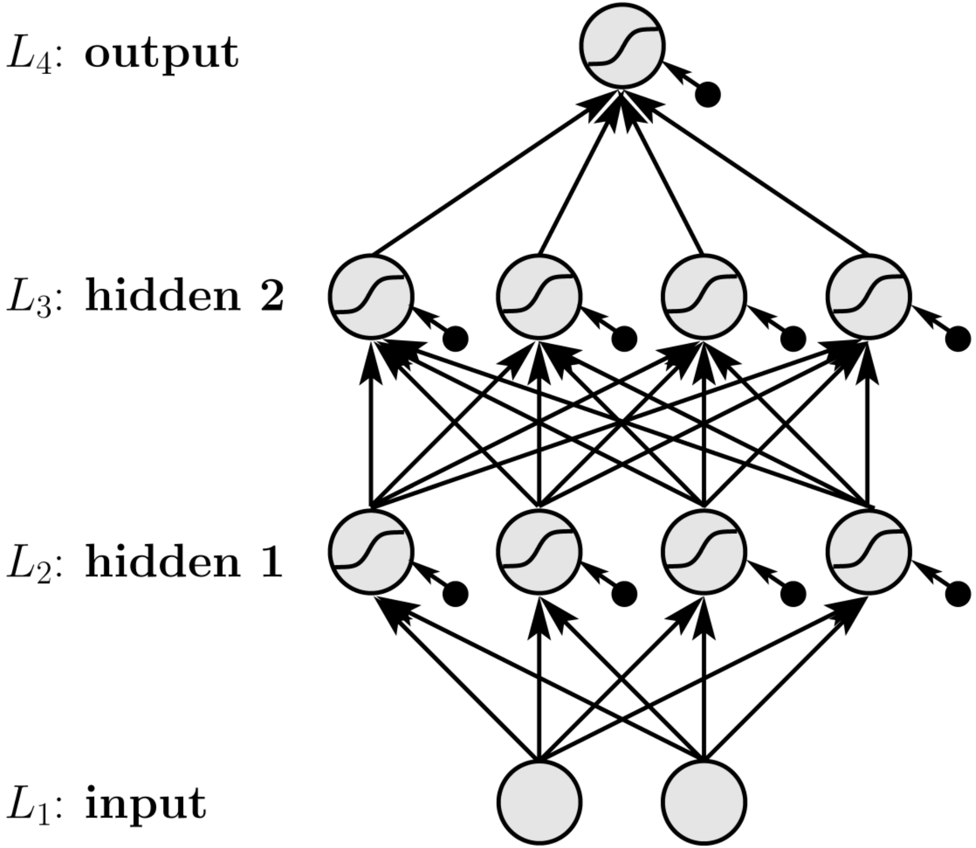
activation function: f



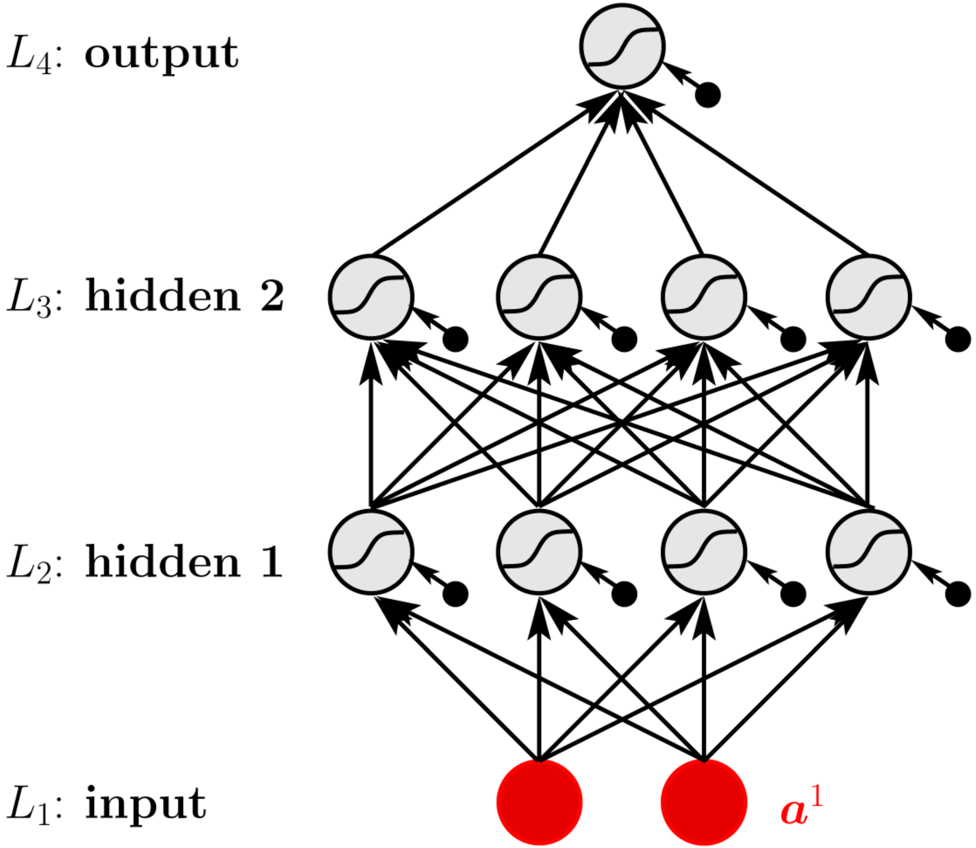
Backpropagation



net input of unit $\boldsymbol{k}$ $\mathrm{net}_k^i \;=\; \sum_n w_{kn}^{i-1} \; a_n^{i-1}$	net input of layer $\boldsymbol{i}$ $\mathbf{net}^i \;=\; \boldsymbol{W}^{i-1} \; \boldsymbol{a}^{i-1}$
activation of unit $\boldsymbol{k}$ $a_k^i \;=\; f(\mathrm{net}_k^i)$	activation of layer $\boldsymbol{i}$ $\boldsymbol{a}^i \;=\; f(\mathbf{net}^i)$



L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**



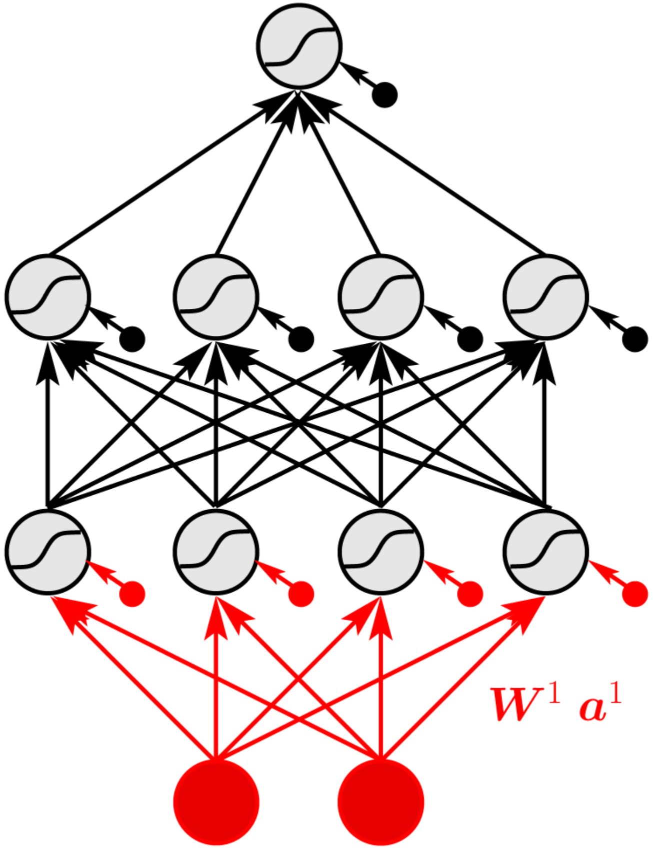
L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**
a^1

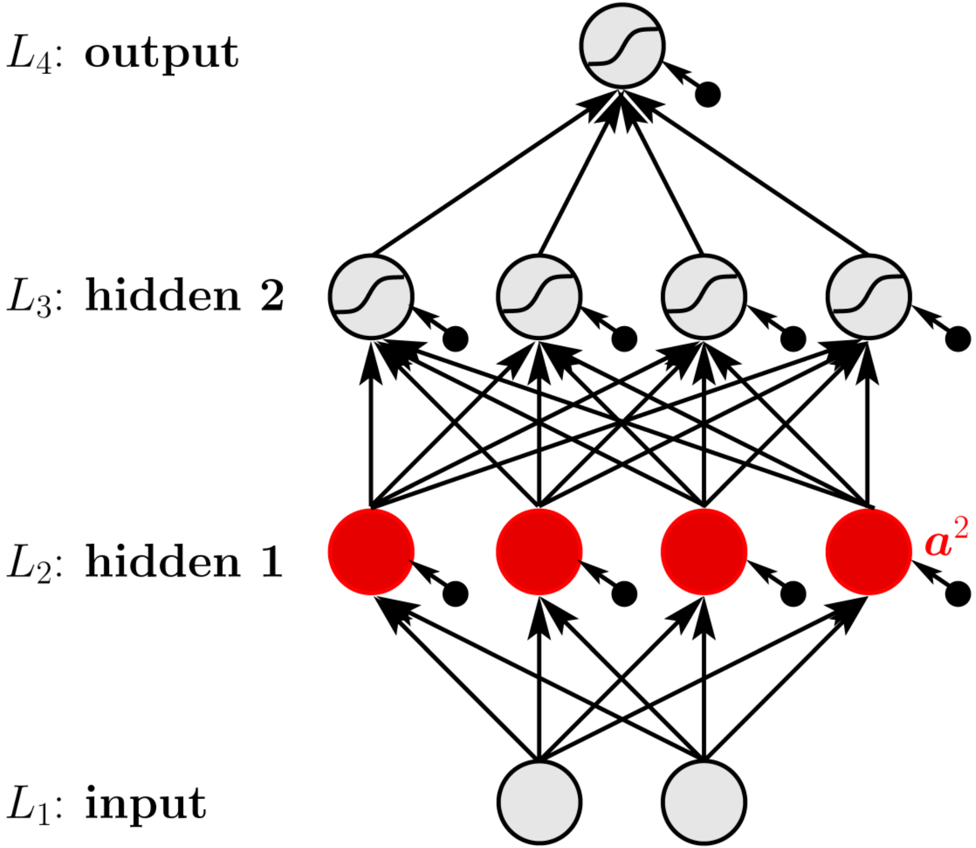
L_4: **output**

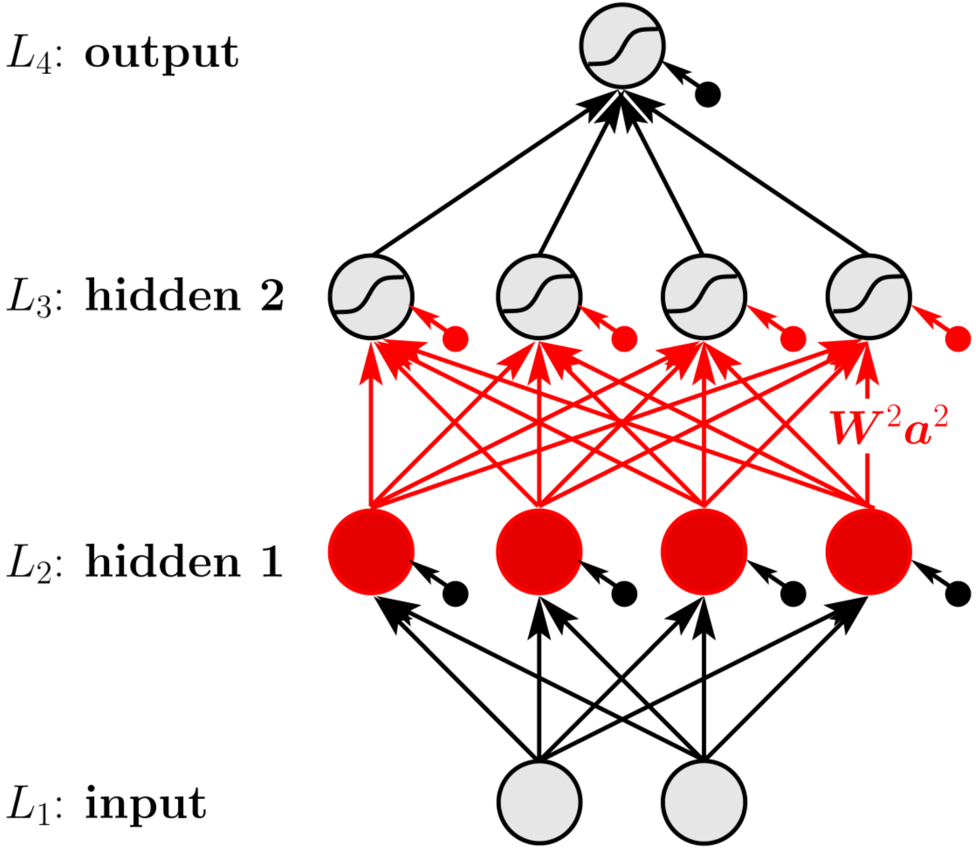
L_3: **hidden 2**

L_2: **hidden 1**

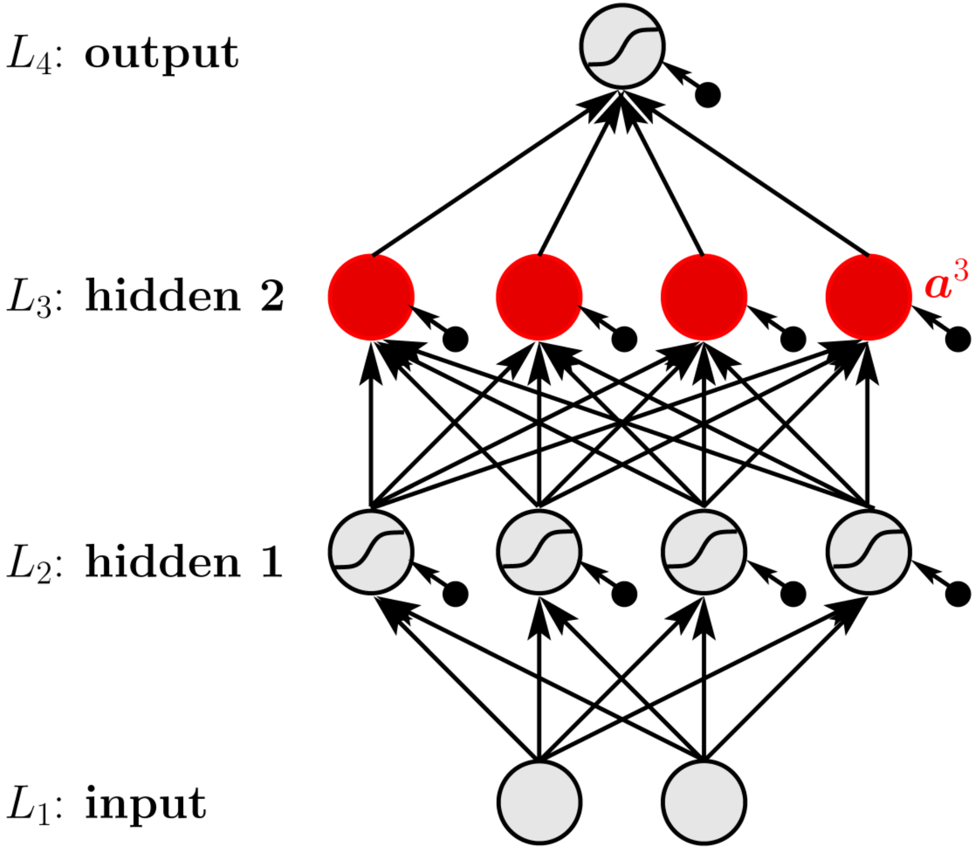
L_1: **input**

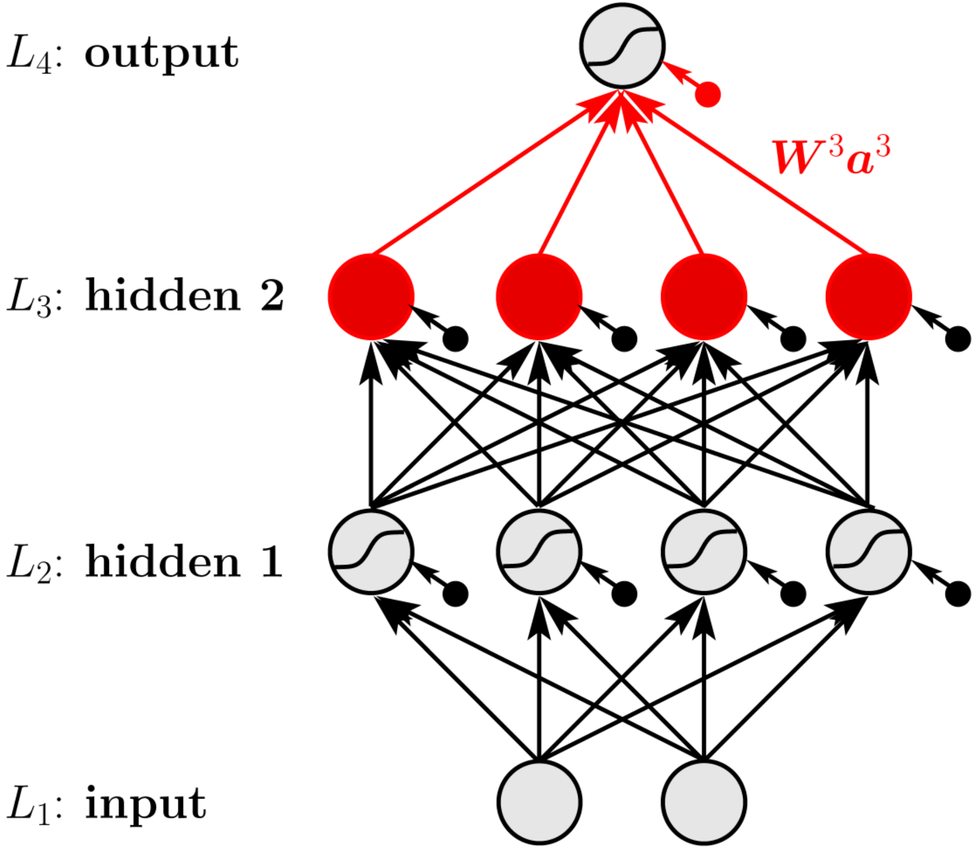


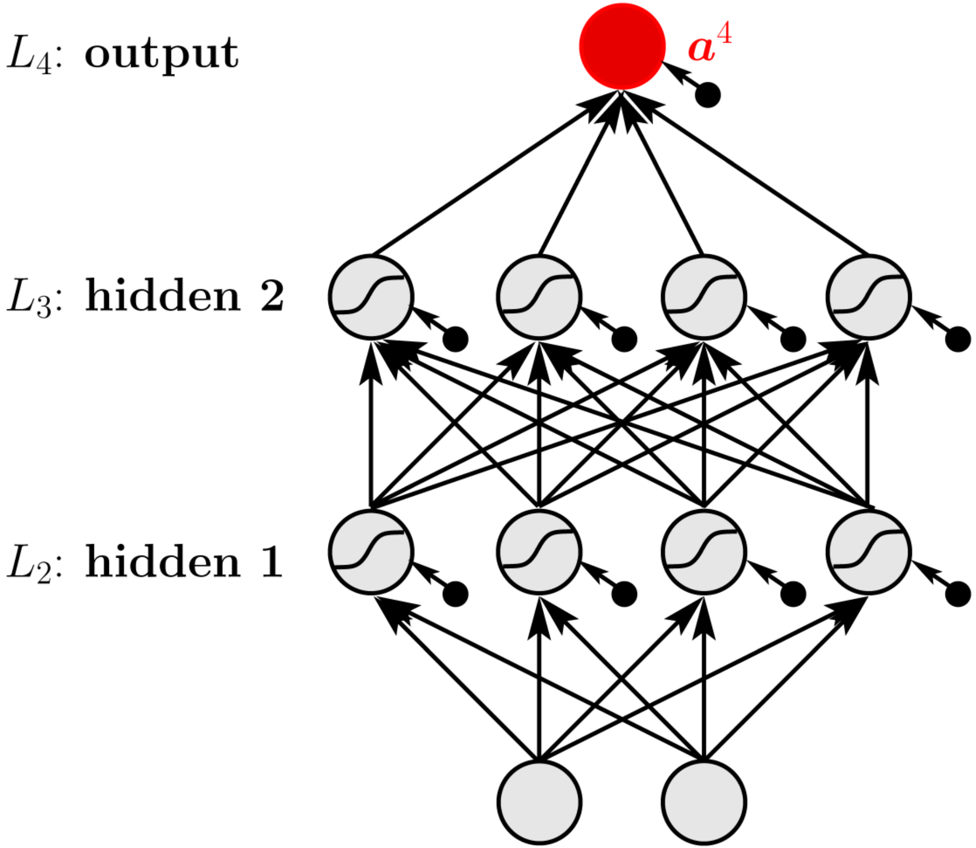

L_4: **output**
L_3: **hidden 2**
a^2
L_2: **hidden 1**
L_1: **input**



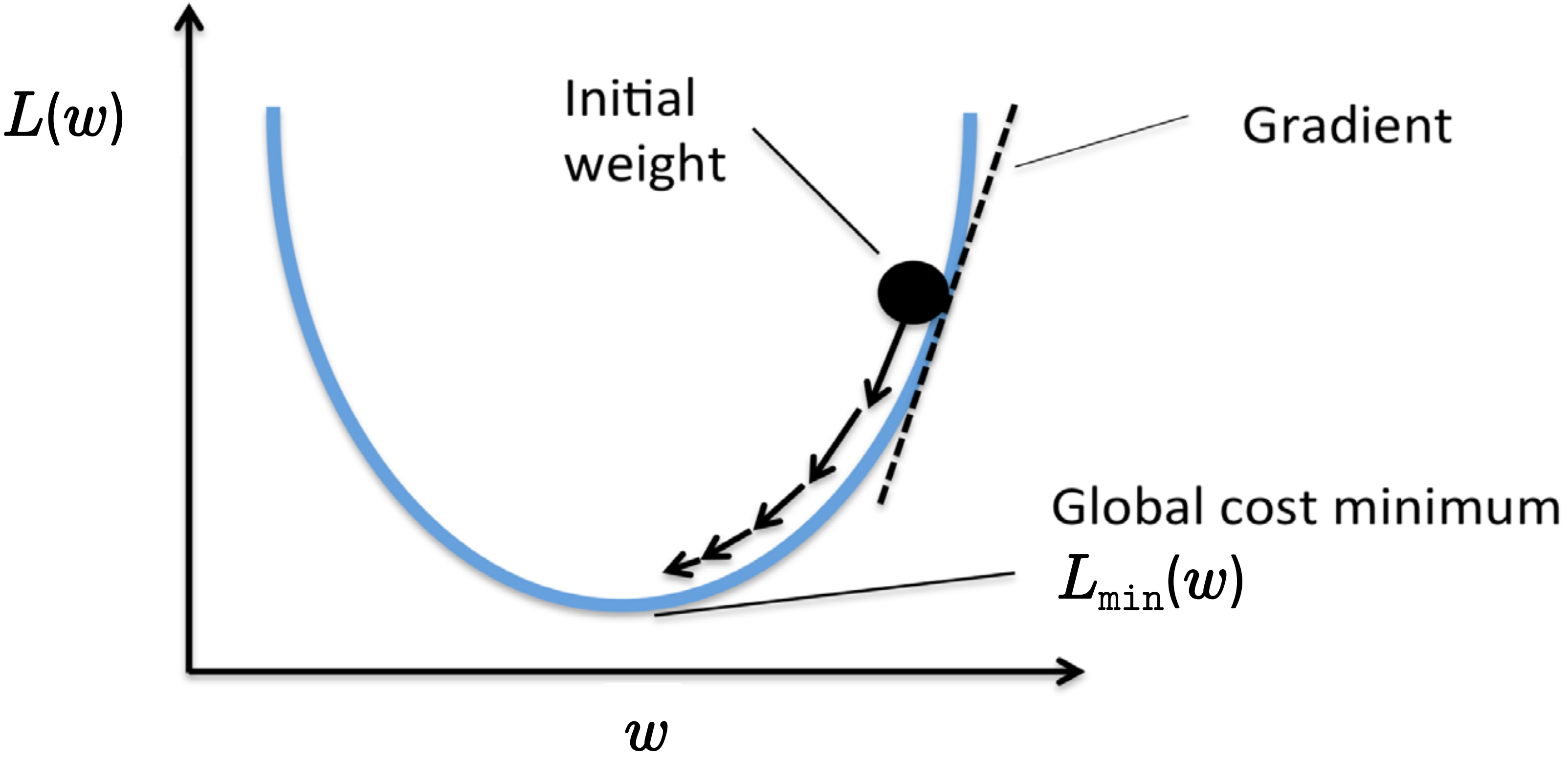
L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**
$\boldsymbol{W}^2\boldsymbol{a}^2$


L_4: **output**
L_3: **hidden 2**
a^3
L_2: **hidden 1**
L_1: **input**


L_4: **output**
$\boldsymbol{W}^3\boldsymbol{a}^3$
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**



L_4: **output**
a^4
L_3: **hidden 2**
L_2: **hidden 1**



L(w)
Initial
weight
Gradient
Global cost minimum
L_min(w)
w

−error at unit $\boldsymbol{k}$

$$\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right) = \frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right) \frac{\partial \mathrm{net}_k}{\partial w_{kl}}$$

$$= \underbrace{\frac{\partial}{\partial \mathrm{net}_k} L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right)}_{\delta_k} a_l$$

backpropagation gradient

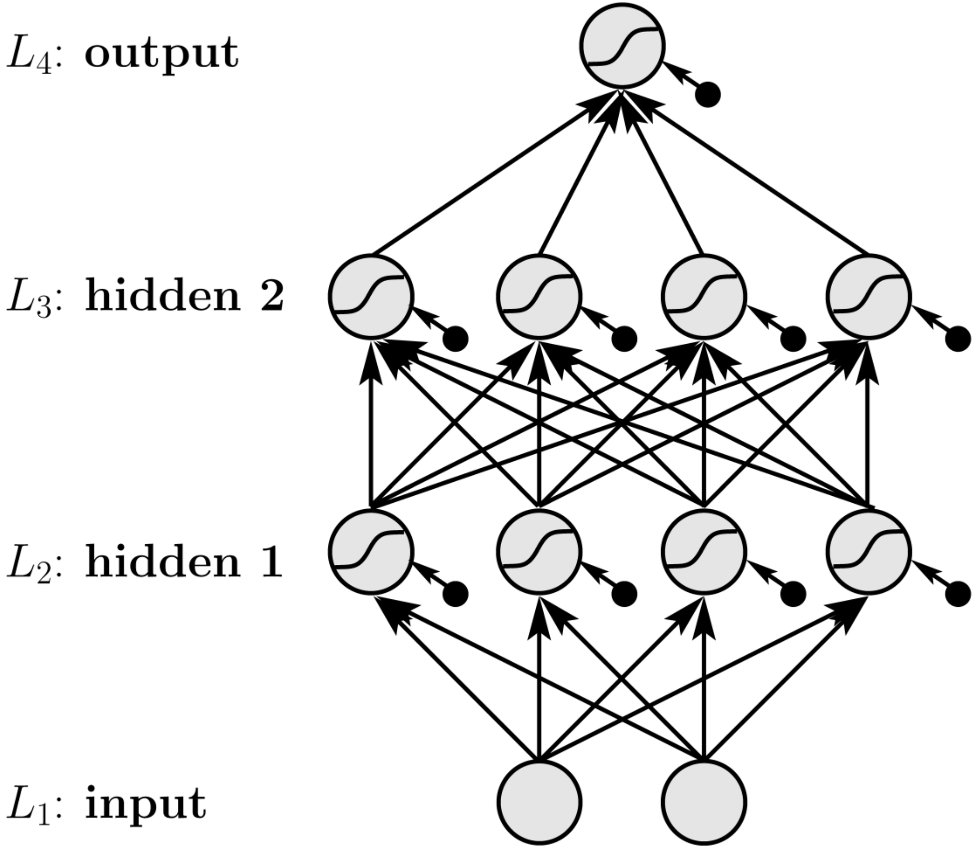
$$\frac{\partial}{\partial w_{kl}} L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right) = \delta_k \; a_l$$

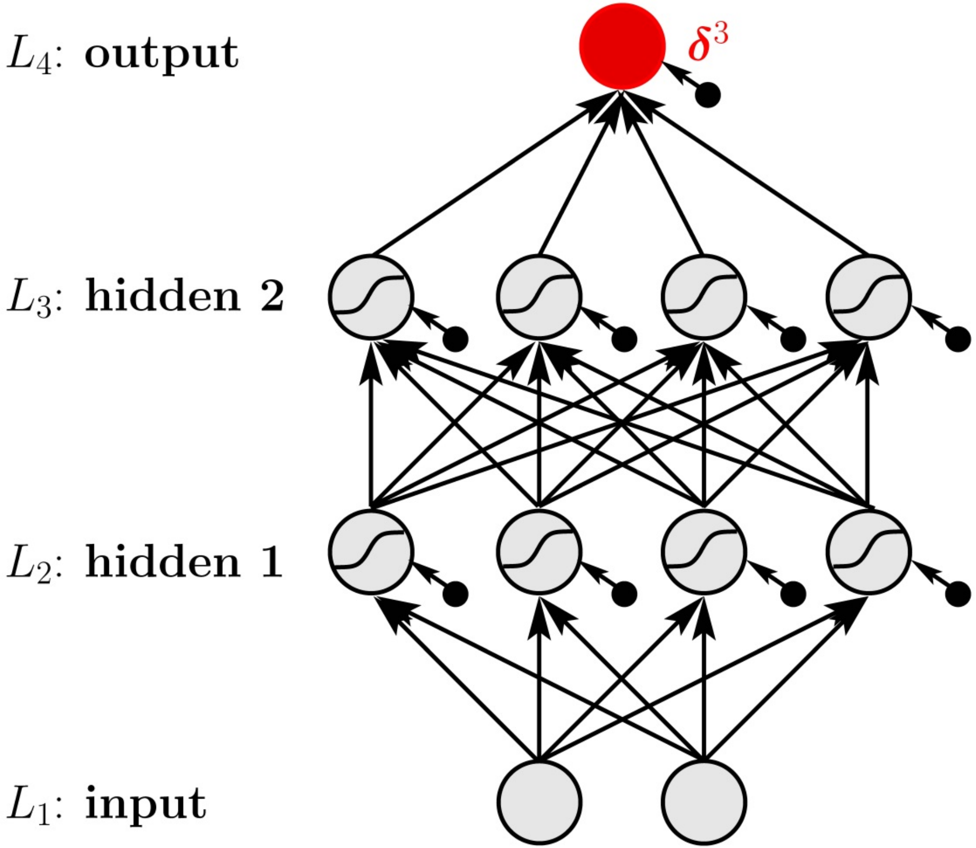
recursion formula

$$\boldsymbol{\delta}^{i-1} \;=\; \frac{\partial}{\partial \mathbf{net}^{i-1}} \; L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right) \;=\; \underbrace{\frac{\partial}{\partial \mathbf{net}^{i}} \; L\left(\boldsymbol{y}, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right)}_{\boldsymbol{\delta}^{i}} \; \underbrace{\frac{\partial \mathbf{net}^{i}}{\partial \mathbf{net}^{i-1}}}_{\boldsymbol{J}^{i}} \;=\; \boldsymbol{\delta}^{i} \; \boldsymbol{J}^{i}$$

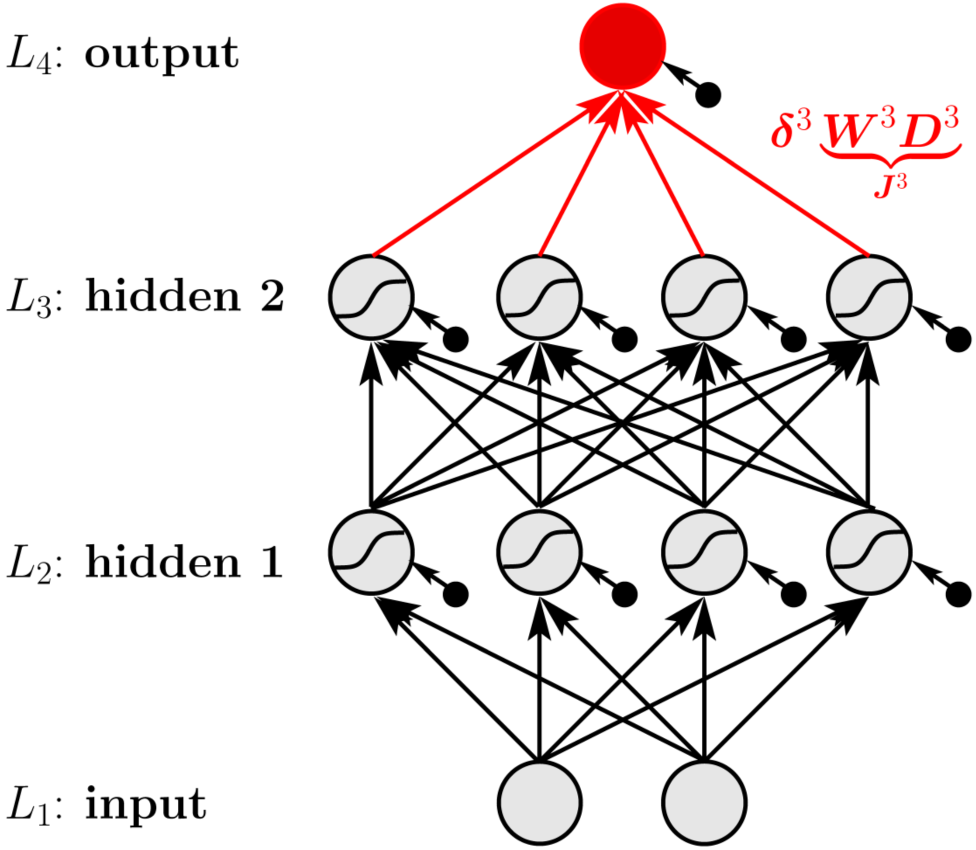
Jacobi matrix

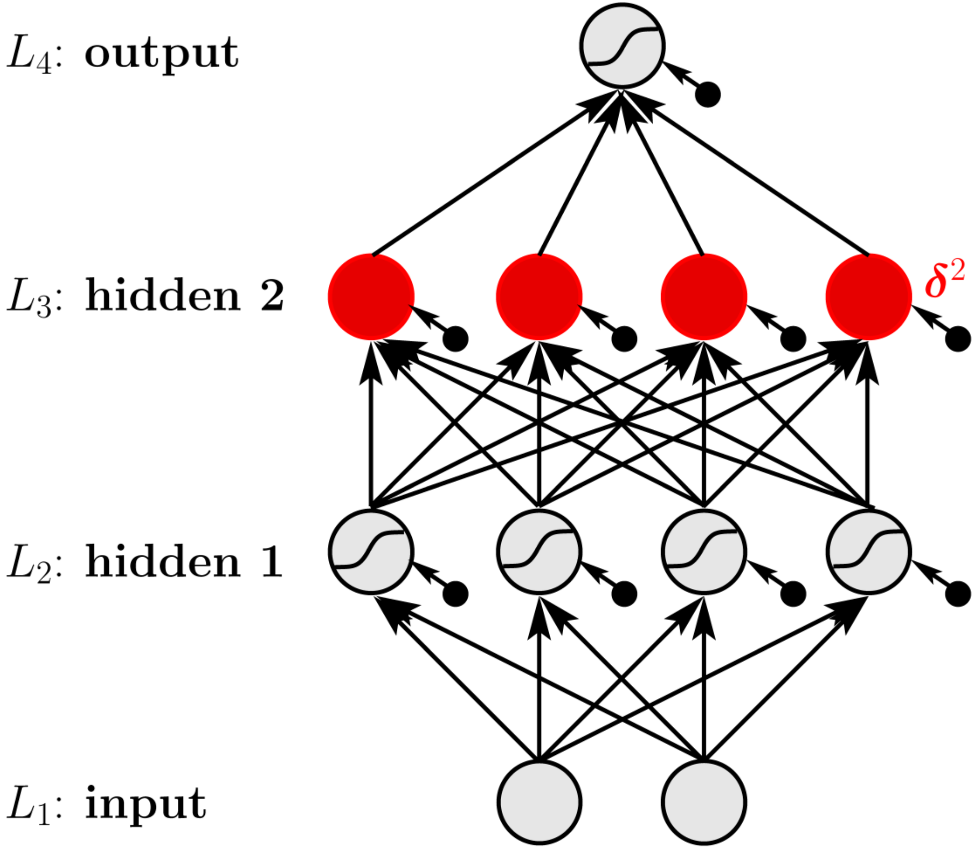
$$\boldsymbol{J}^{i} \;=\; \frac{\partial \mathbf{net}^{i}}{\partial \mathbf{net}^{i-1}} \;=\; \boldsymbol{W}^{i} \; \underbrace{\mathrm{diag}(f'(\mathbf{net}^{i}))}_{\boldsymbol{D}^{i}} \;=\; \boldsymbol{W}^{i} \; \boldsymbol{D}^{i}$$

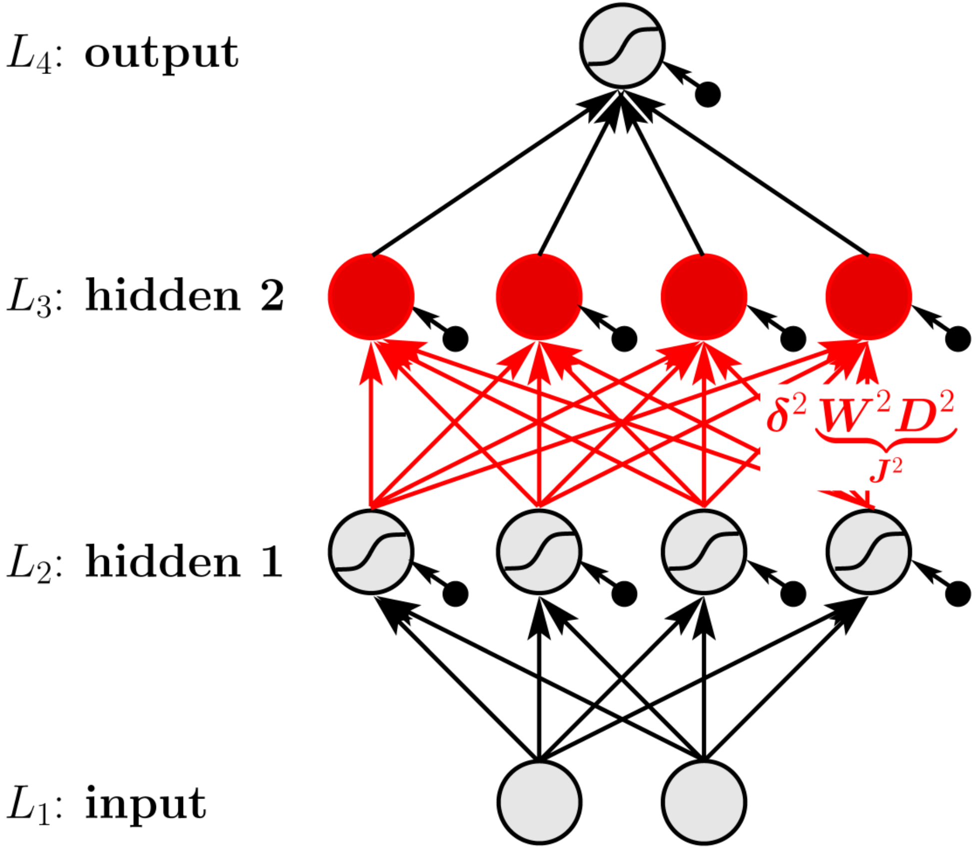

L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**



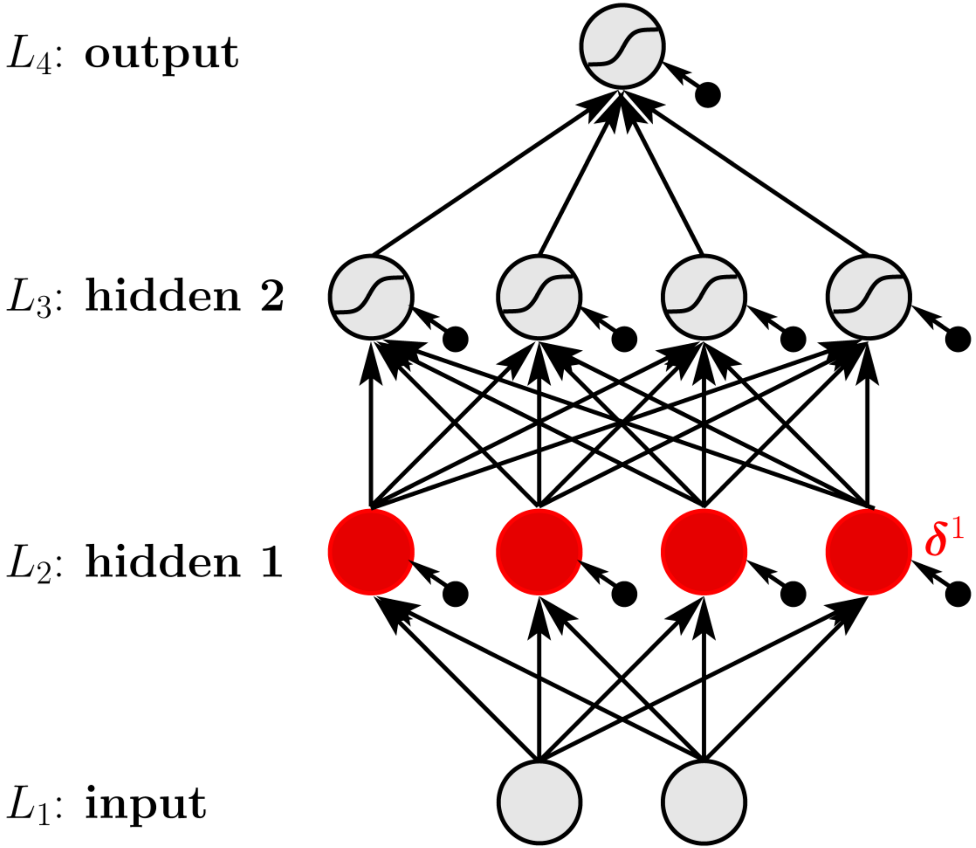
L_4: **output**
δ^3
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**


L_4: output
$\delta^3 \underbrace{W^3 D^3}_{J^3}$
L_3: hidden 2
L_2: hidden 1
L_1: input


L_4: **output**
L_3: **hidden 2**
$\boldsymbol{\delta}^2$
L_2: **hidden 1**
L_1: **input**



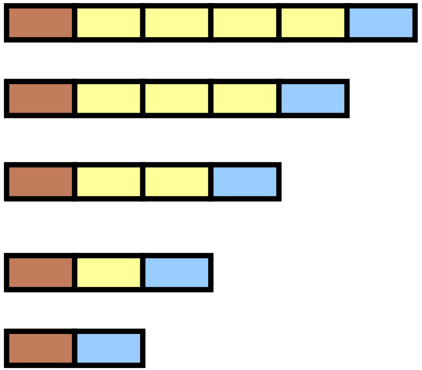
L_4: output
L_3: hidden 2
L_2: hidden 1
L_1: input
$\delta^2 \underbrace{W^2 D^2}_{J^2}$



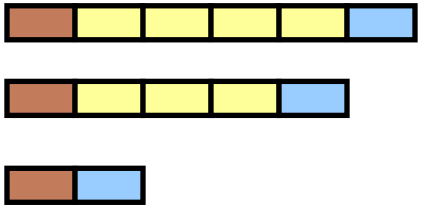
L_4: **output**
L_3: **hidden 2**
L_2: **hidden 1**
L_1: **input**
δ^1

Temporal Generalization

brown-yellow(n)-blue



training set



LSTM learns the rule

Window does not learn the rule

JOHANNES KEPLER UNIVERSITY LINZ

Recurrent networks are Turing complete

- Every computer program can be represented
- All we can do on a computer can be done by RNNs
- RNNs can represent learning algorithms and even neural network models

→ Neural Turing Machine (later in this class)

Recurrent Neural Networks

Feedforward Network

- Classification
- Regression
- Input → output vector
- No loops
- No temp. gen.

Recurrent Network

- Sequence processing
- Loops for storing
- Store past information
- Turing complete
- Temporal generalization

Sepp Hochreiter

Recurrent Neural Networks

A feedforward network is a function $\hat{\boldsymbol{y}} = g(\boldsymbol{x}; \boldsymbol{w})$ that maps an input vector $\boldsymbol{x}$ to an output (or prediction) vector $\hat{\boldsymbol{y}}$ using network parameters $\boldsymbol{w}$.

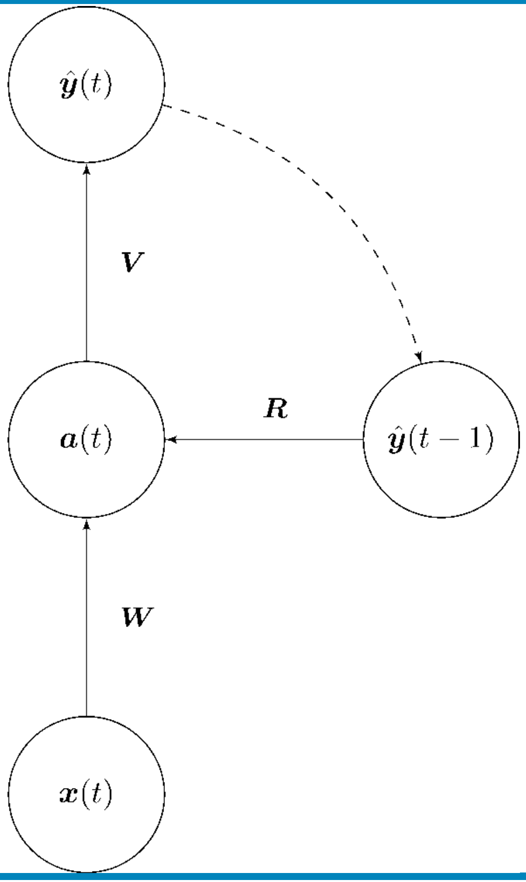
The forward pass activates the network depending on the input variables only and produces output values.

RNNs map an input sequence $(\boldsymbol{x}(t))_{t=1}^{T}$ to an output sequence $(\hat{\boldsymbol{y}}(t))_{t=1}^{T}$ by

$$\hat{\boldsymbol{y}}(t) \;=\; g\left(\boldsymbol{a}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(t); \boldsymbol{w}\right)$$

$\boldsymbol{a}(0)$ is the vector of the initial recurrent activations

Jordan Network



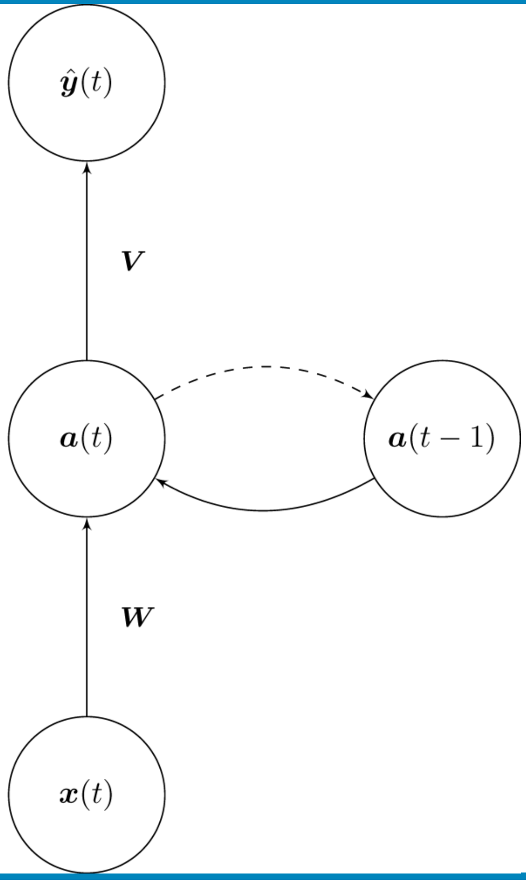
$$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{R}^\top \hat{\boldsymbol{y}}(t-1)$$

$$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$$

$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{V}^\top \boldsymbol{a}(t))$$

$\boldsymbol{W}$:	input weight matrix
$\boldsymbol{R}$:	recurrent weight matrix
$\boldsymbol{V}$:	output weight matrix
f, g:	activation functions / non-linearities
$\boldsymbol{s}(t)$:	pre-activations at time t
$\boldsymbol{a}(t)$:	hidden activations at time t

Elman Network



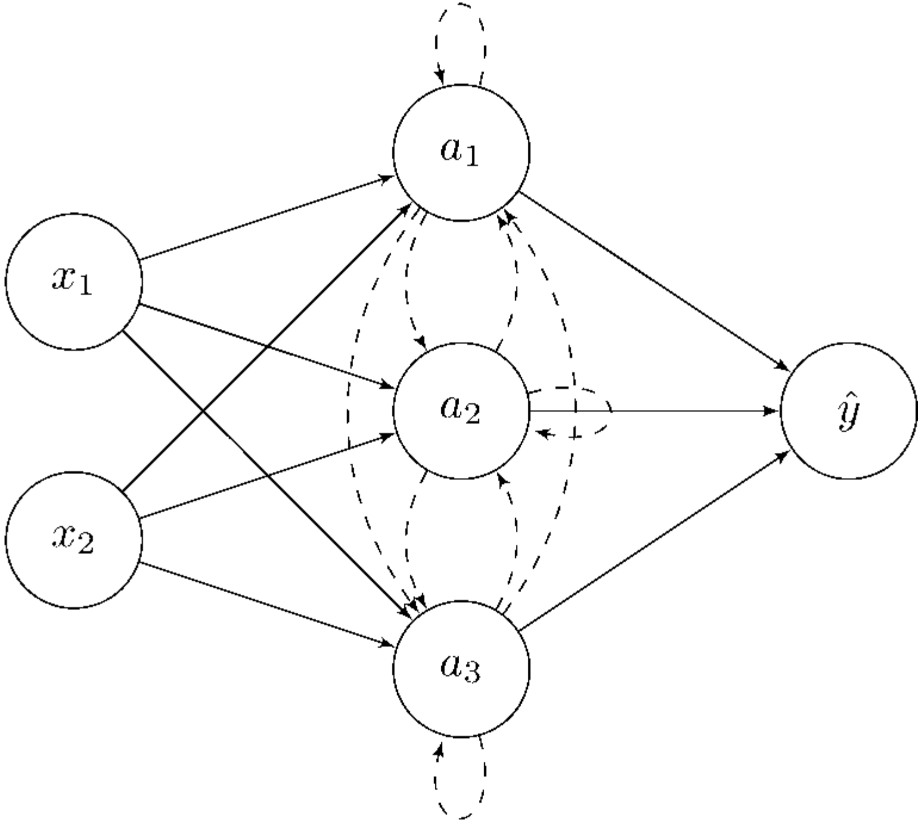
$$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{a}(t-1)$$

$$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$$

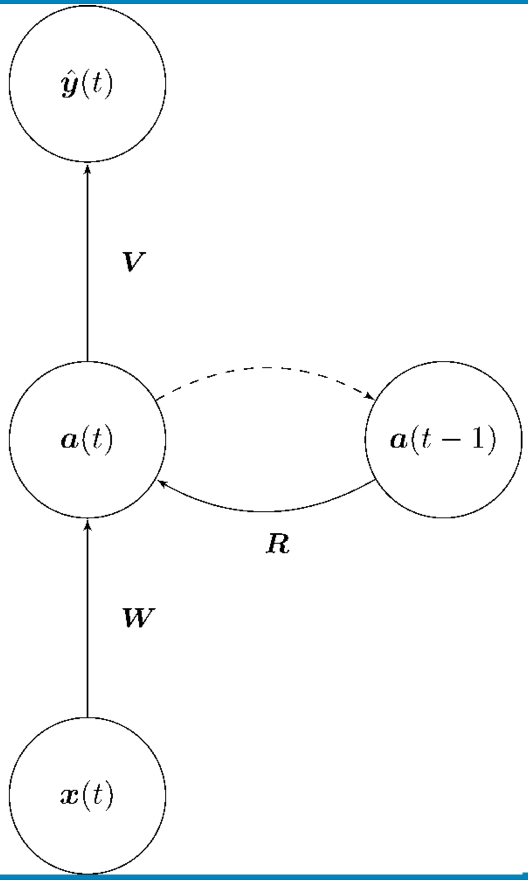
$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{V}^\top \boldsymbol{a}(t))$$

$\boldsymbol{W}$: input weight matrix
$\boldsymbol{V}$: output weight matrix
f, g: activation functions / non-linearities
$\boldsymbol{s}(t)$: pre-activations at time t
$\boldsymbol{a}(t)$: hidden activations at time t

Fully Recurrent Neural Network



Fully Recurrent Neural Network



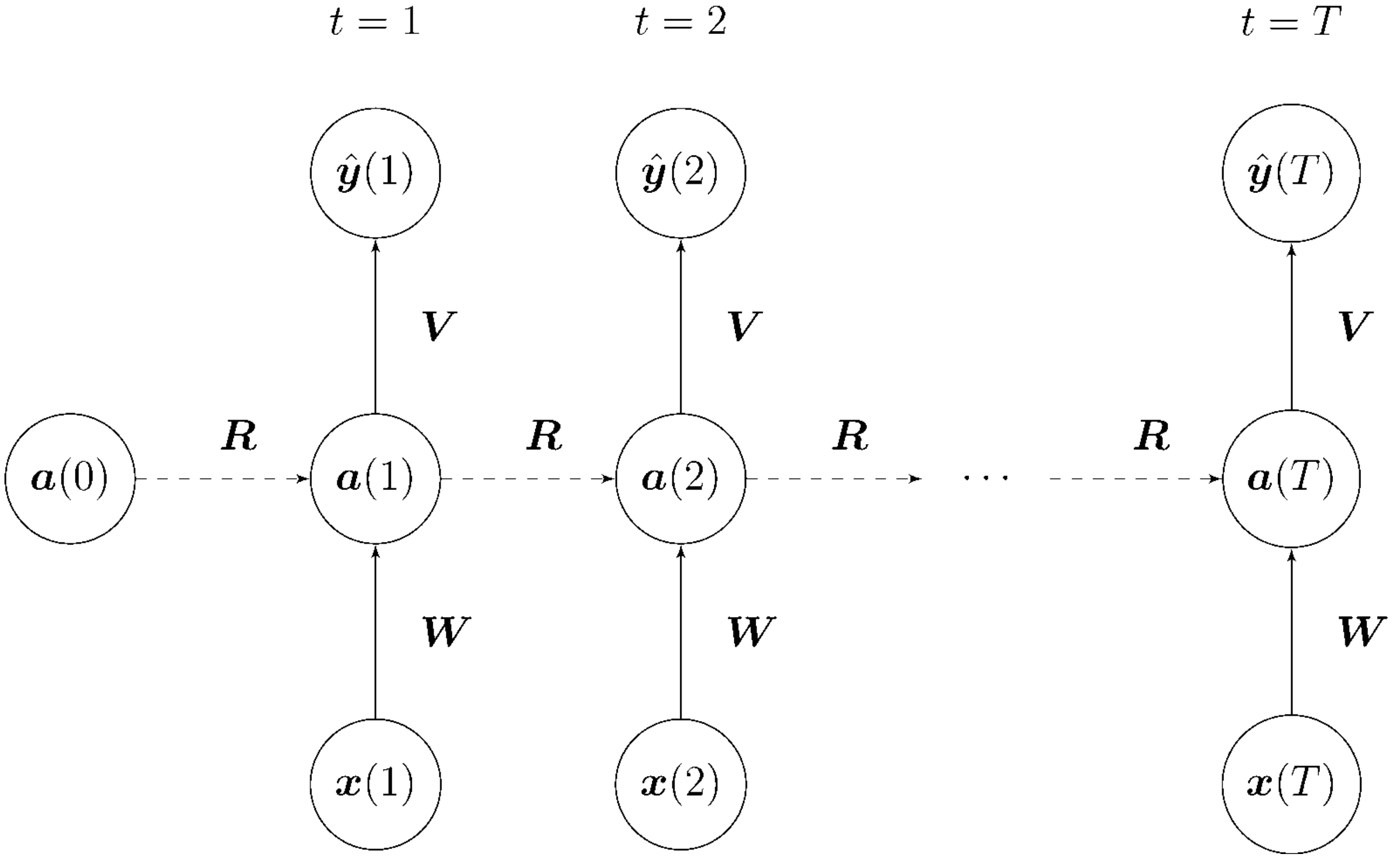
$$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{R}^\top \boldsymbol{a}(t-1)$$

$$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$$

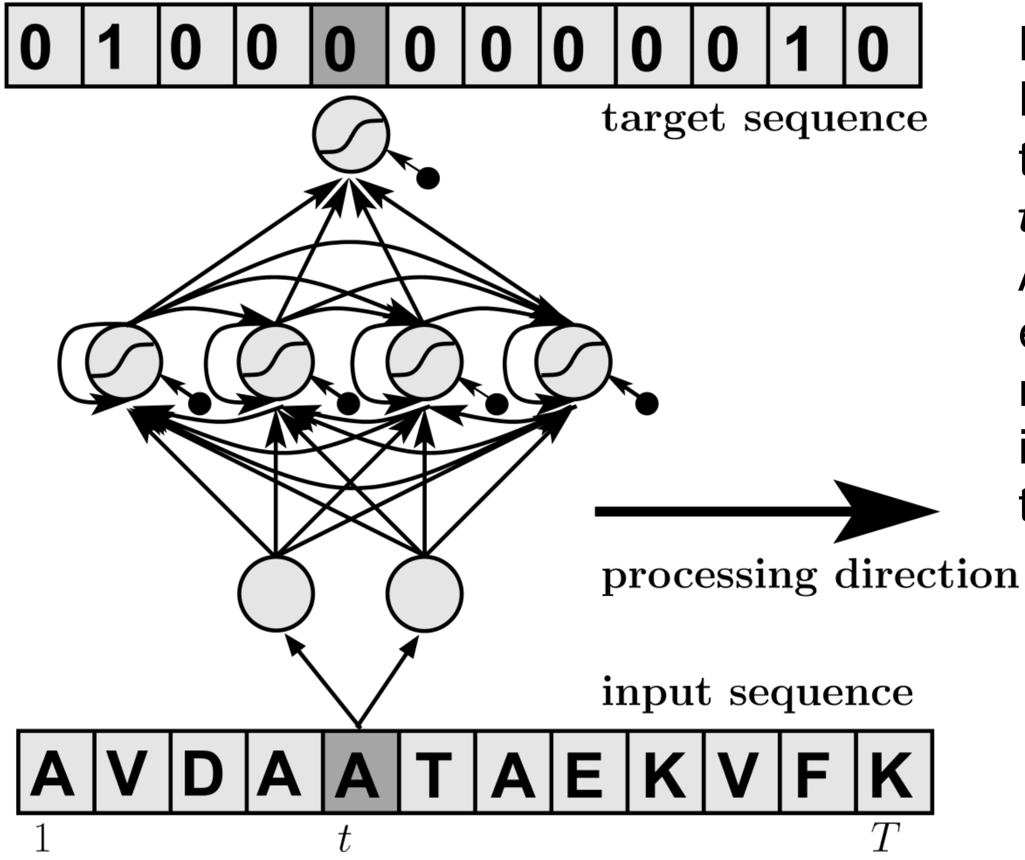
$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{V}^\top \boldsymbol{a}(t))$$

$\boldsymbol{W}$: input weight matrix
$\boldsymbol{R}$: recurrent weight matrix
$\boldsymbol{V}$: output weight matrix
f, g: activation functions / non-linearities
$\boldsymbol{s}(t)$: pre-activations at time t
$\boldsymbol{a}(t)$: hidden activations at time t

Fully Recurrent Neural Network



Fully Recurrent Neural Network



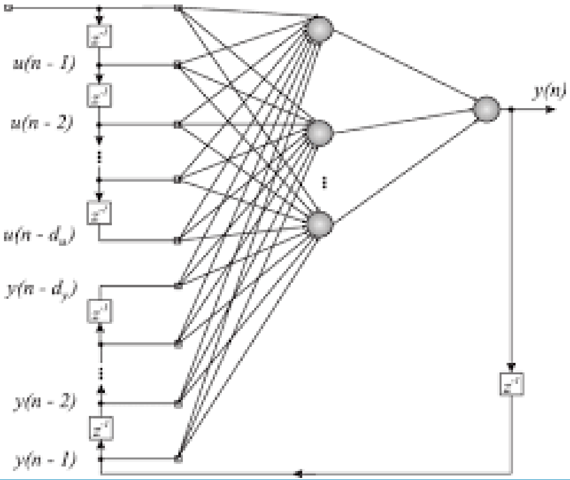
Processing of a sequence with an RNN. sequence starts at time step 1, the current time step is indicated by t, and the end of the sequence is T. At each time step the current input element is fed to the recurrent network. The *weight sharing* can be imagined as sliding the network over the input sequence.

NARX Networks

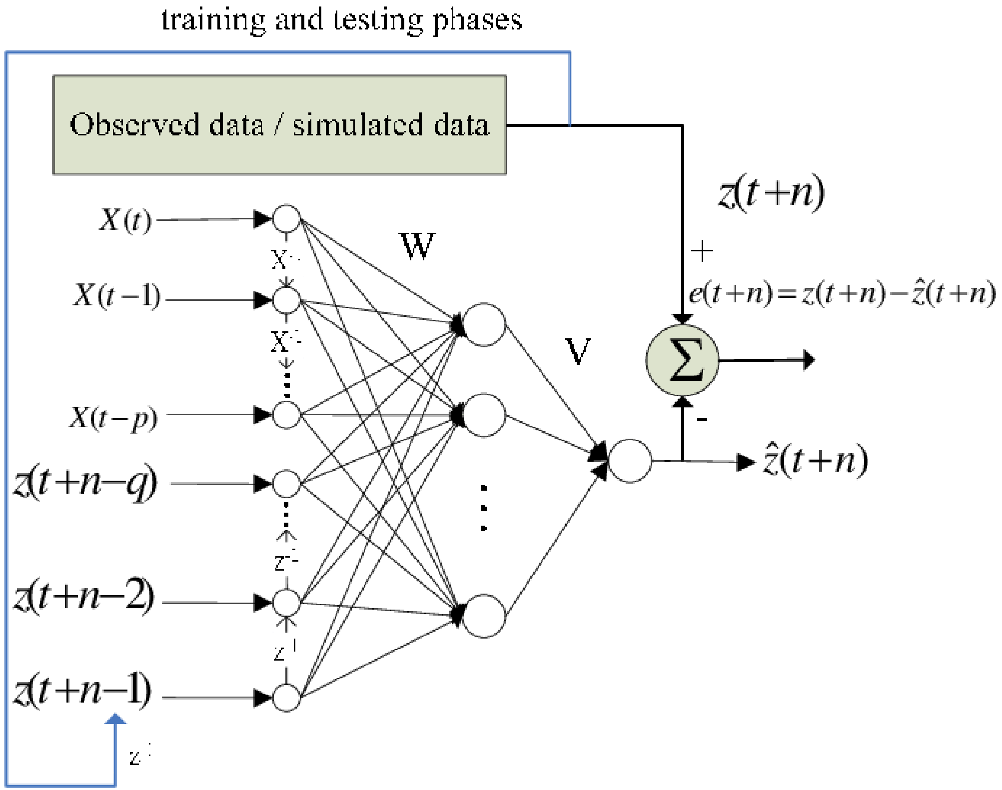
Non-linear auto-regressive exogenous models (NARX) are time series models of the form

$$\hat{\boldsymbol{y}}(t) = \boldsymbol{g}\big(\hat{\boldsymbol{y}}(t-1), \ldots, \hat{\boldsymbol{y}}(t-T_y), \boldsymbol{x}(t), \ldots, \boldsymbol{x}(t-T_x)\big)$$

The Jordan network can be seen as a trivial instance of a NARX recurrent net with T_y=1 and T_x=0.

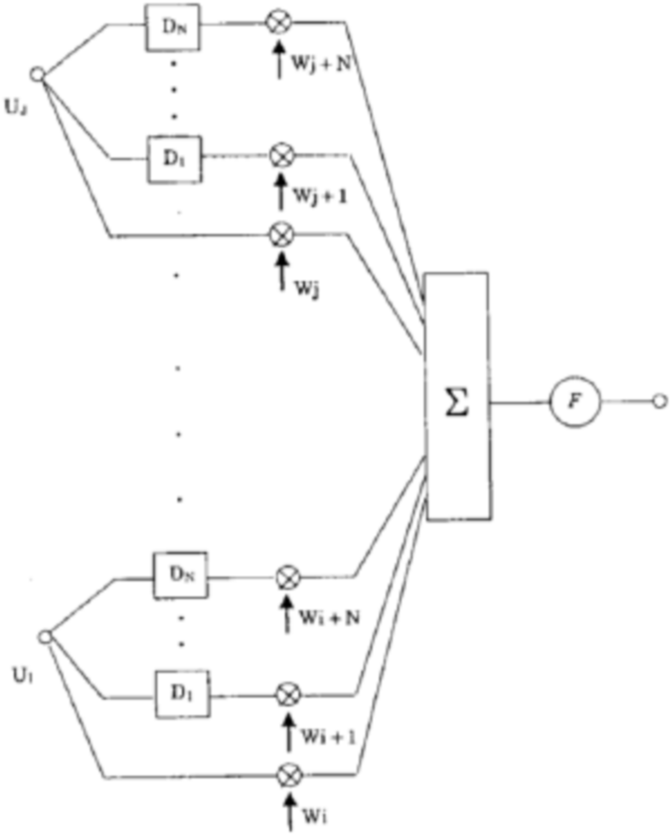


NARX Networks



Time Delay Neural Networks

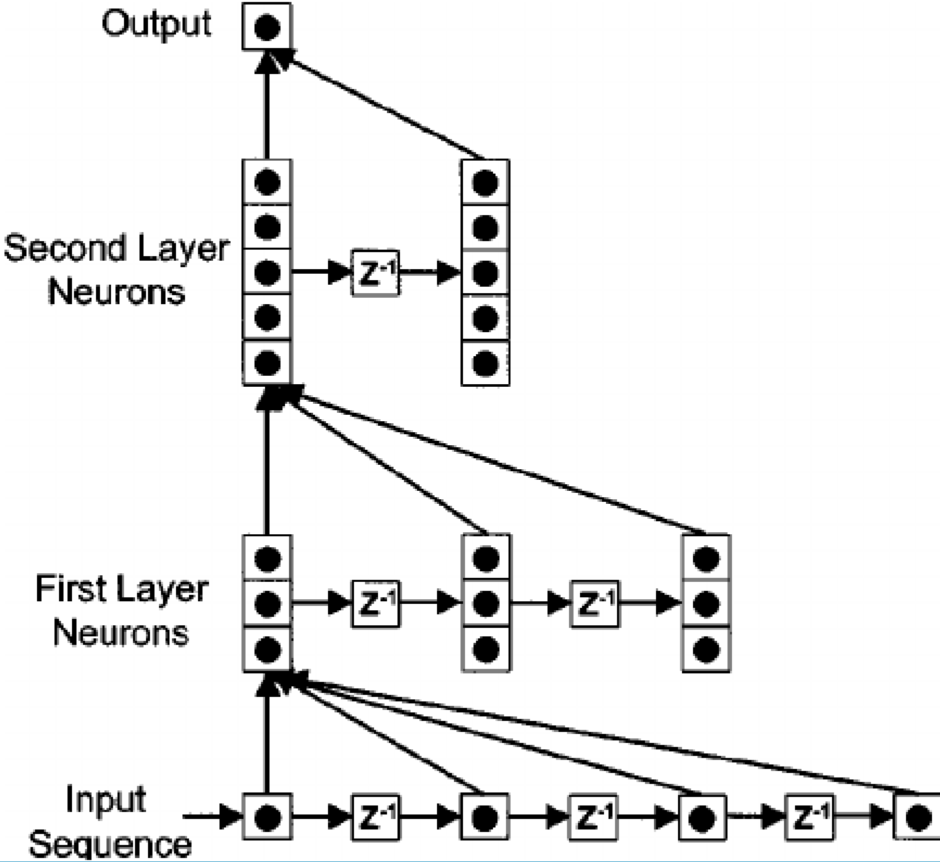
Every connection from one unit i to another unit j has $N+1$ different values for the N delays $(0, D_1, \ldots, D_n, \ldots, D_N)$.



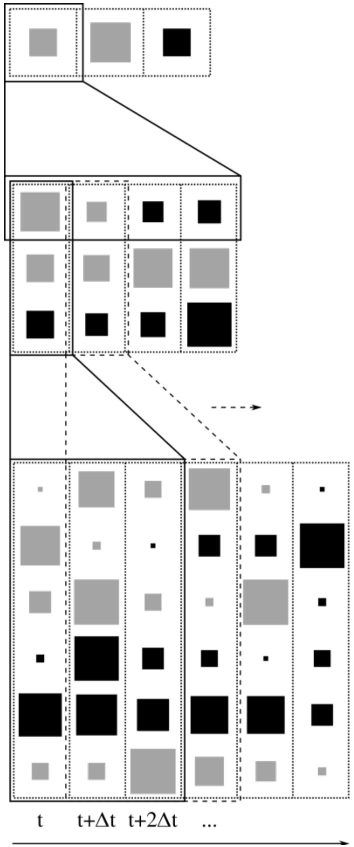
Time Delay Neural Networks

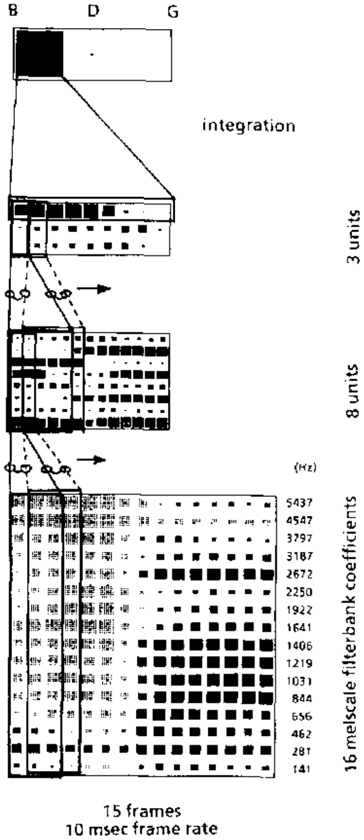
For the special case that $D_n = n$, we have a 1-D convolutional network.

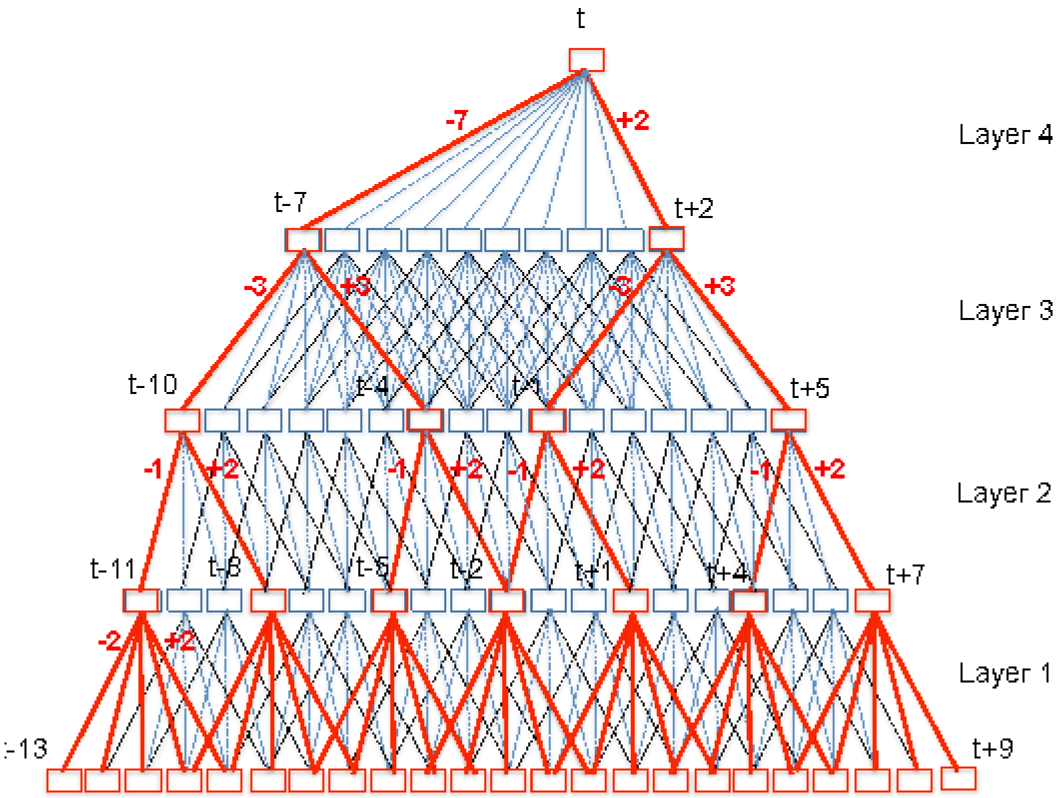
On the other hand each 1-D convolutional network which has window size larger than or equal to the maximal delay can represent the TDNN.



Time Delay Neural Networks







Time Delay Neural Networks

JOHANNES KEPLER UNIVERSITY LINZ

Learning time delays:

- softmax over all delays between 1 and maximal delay
- softmax converges during learning to a one-hot encoding
- selecting one specific delay

As computational intensive as a 1-D convolutional network.

Applications:

- Phoneme recognition
- Online handwriting recognition
- Word recognition
- Speech recognition

Learning Algorithms for RNNs

- Backpropagation through time (BPTT)
- Truncated BPTT
- Real-Time Recurrent Learning (RTRL)
- Focused Backpropagation