Machine Intelligence 1

1.6 Recurrent Neural Networks

Prof. Dr. Klaus Obermayer

Fachgebiet Neuronale Informationsverarbeitung (NI)

WS 2017/2018

1.6.1 Time Series

Time series

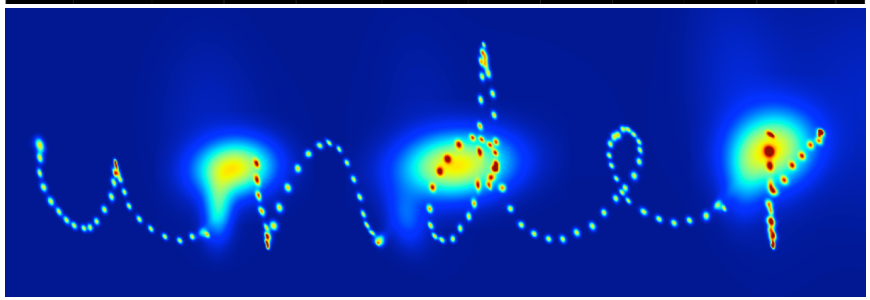
- sequences of samples of arbitrary/varying length n
  - generated by a dynamic system evolving in time

Time series

- sequences of samples of arbitrary/varying length n
  - generated by a dynamic system evolving in time
- time series prediction:
  - predict the next m samples
  - based on the last n samples

Time series

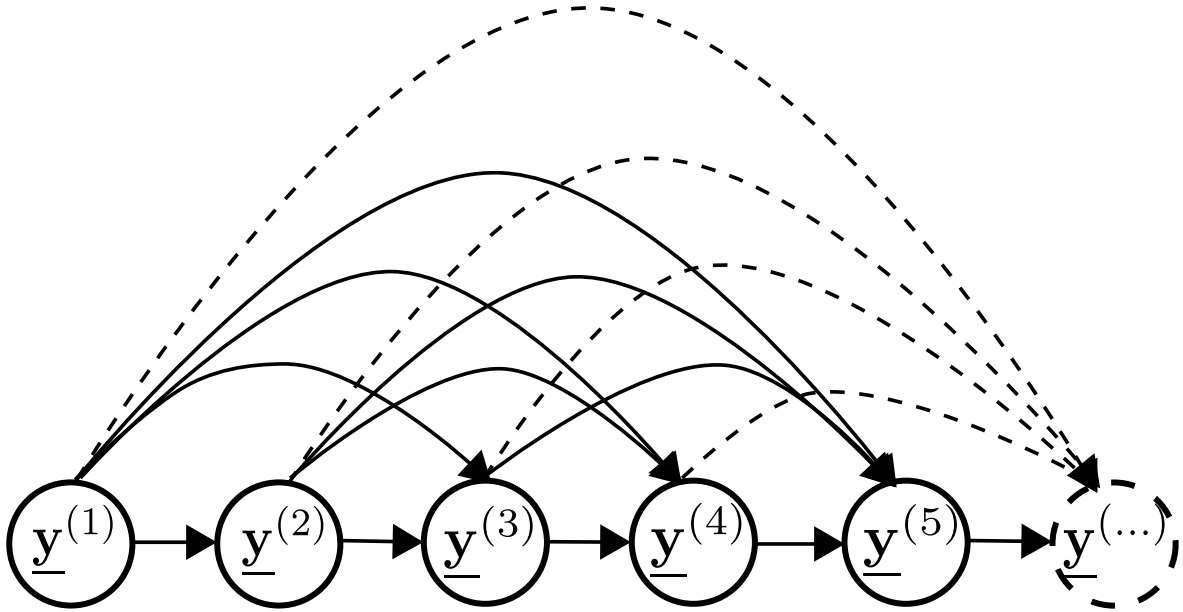
- sequences of samples of arbitrary/varying length n
  - generated by a dynamic system evolving in time
- time series prediction:
  - predict the next m samples
  - based on the last n samples
- time series generation:
  - generate a new time series
  - conditioned on a sequence or image in another format



Time series distributions

- probability at each time step depends on all previous observations

$$P\left(\underline{\mathbf{y}}^{(t)}|\underline{\mathbf{y}}^{(1)},\ldots,\underline{\mathbf{y}}^{(t-1)}\right)$$



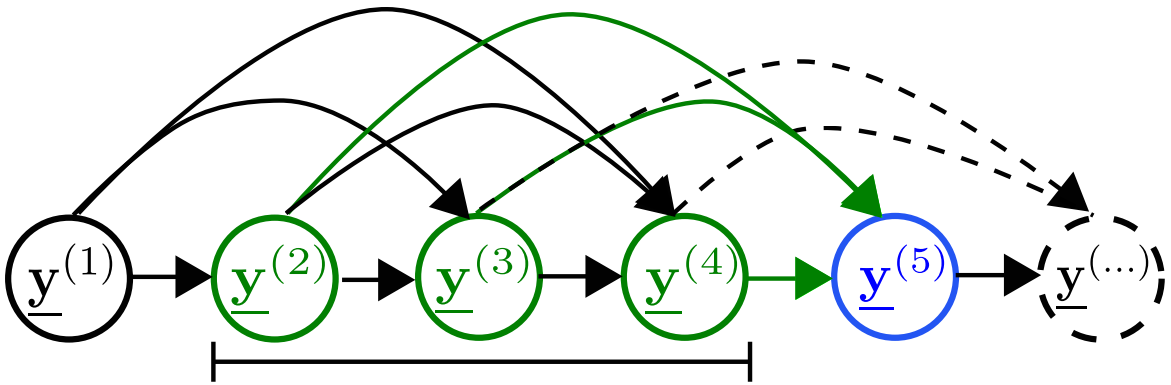
Time series distributions

- probability at each time step depends on all previous observations

$$P\left(\underline{\mathbf{y}}^{(t)}|\underline{\mathbf{y}}^{(1)},\ldots,\underline{\mathbf{y}}^{(t-1)}\right)$$

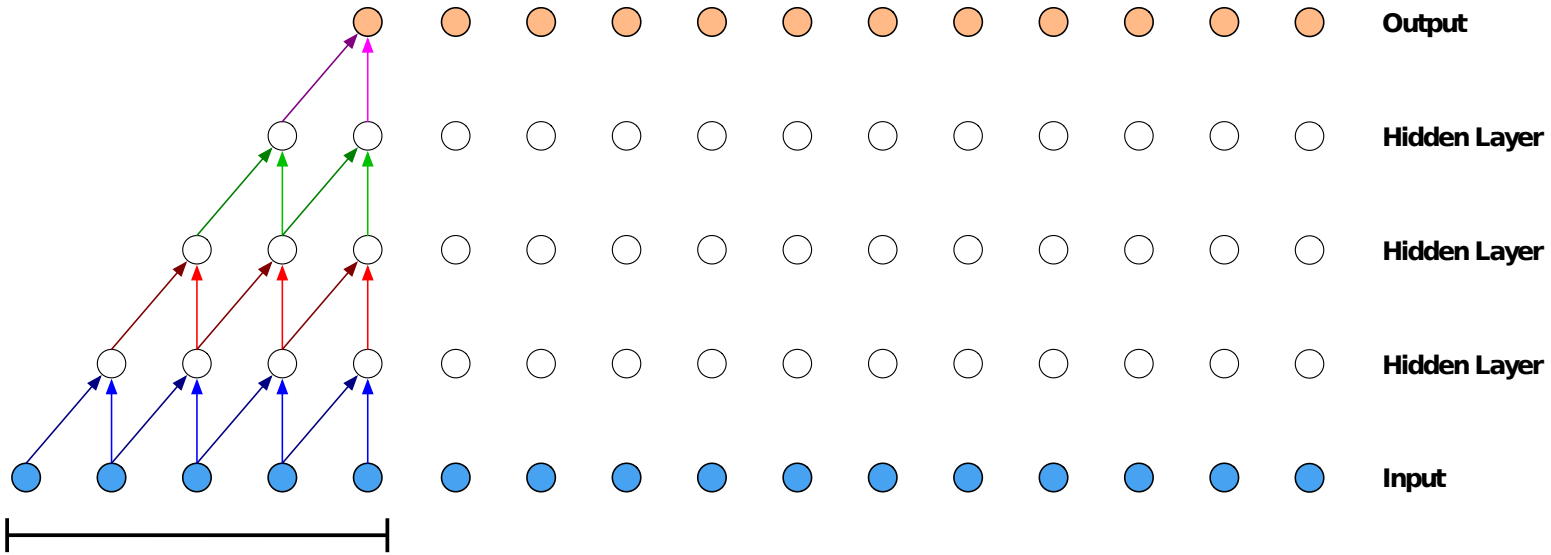
- we assume often the distribution depends on a short time-window

$$P\left(\underline{\mathbf{y}}^{(t)}|\underline{\mathbf{y}}^{(t-m)},\ldots,\underline{\mathbf{y}}^{(t-1)}\right)$$



Example: convolutional network generates time series

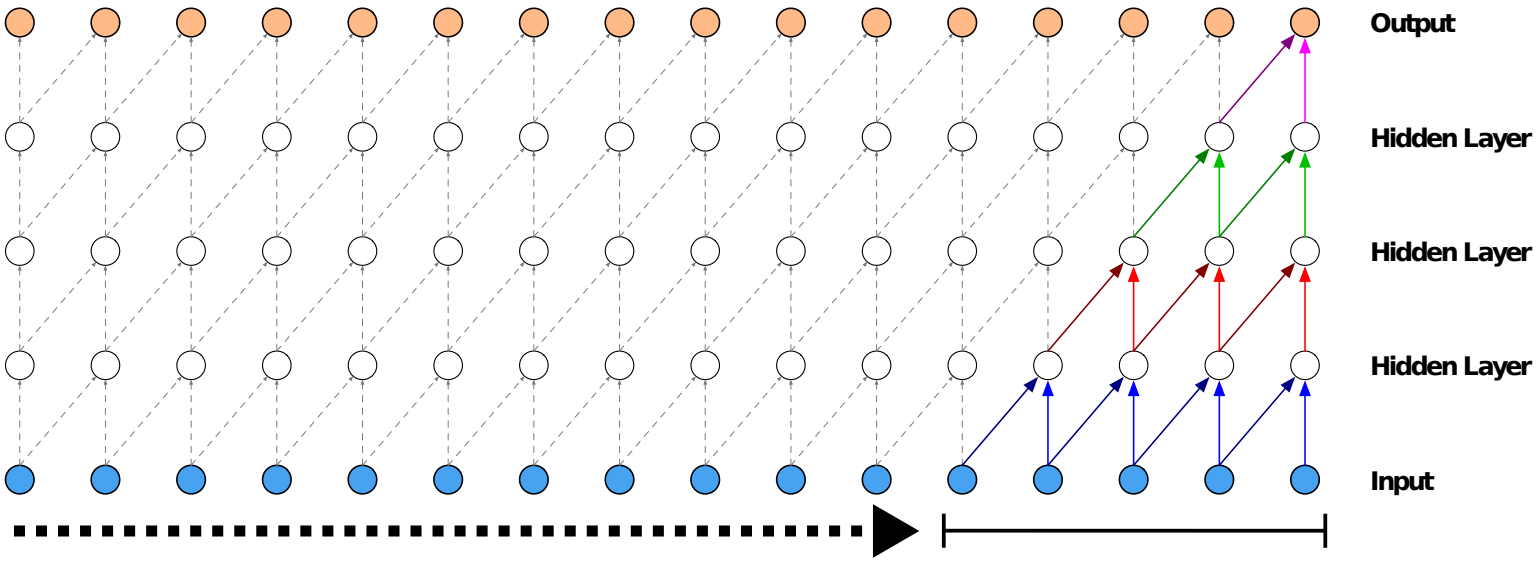
- *causally connected* convolutions over limited *time window*



(Google WaveNet from van den Oord et al., 2016)

Example: convolutional network generates time series

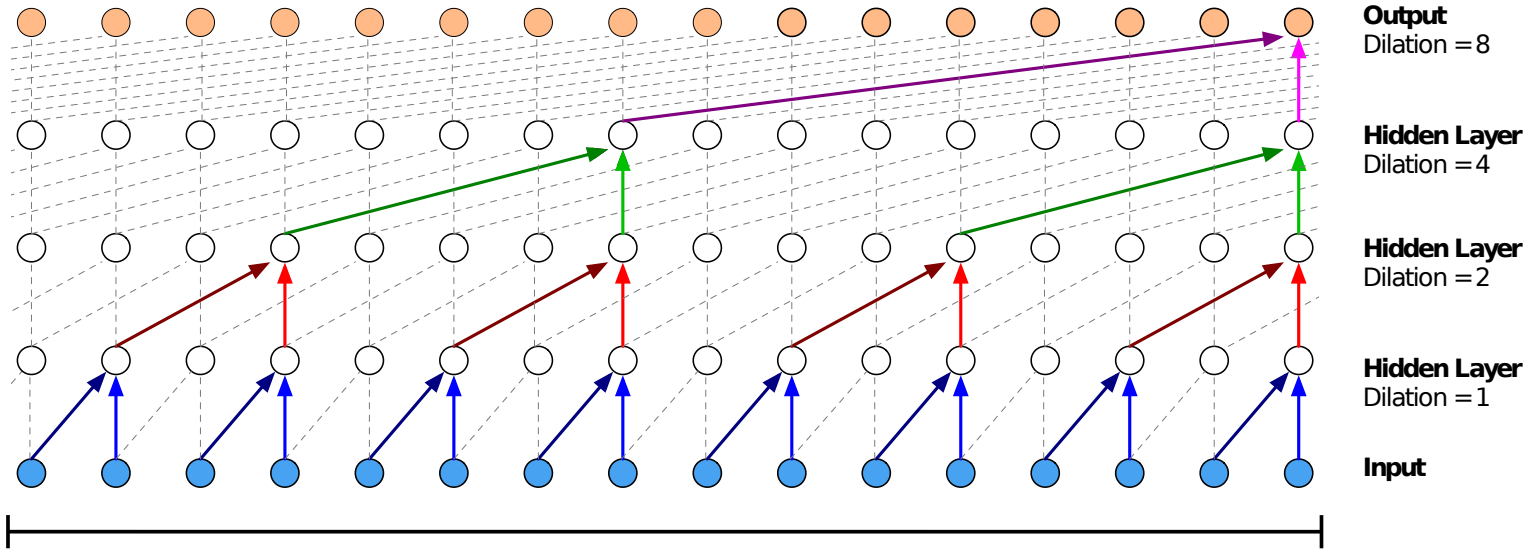
- *causally connected* convolutions over limited *time window*
- shifting the time window generates output time series



(Google WaveNet from van den Oord et al., 2016)

Example: convolutional network generates time series

- *causally connected* convolutions over limited *time window*
- shifting the time window generates output time series
- *time dilation* in each layer $\rightsquigarrow$ large windows

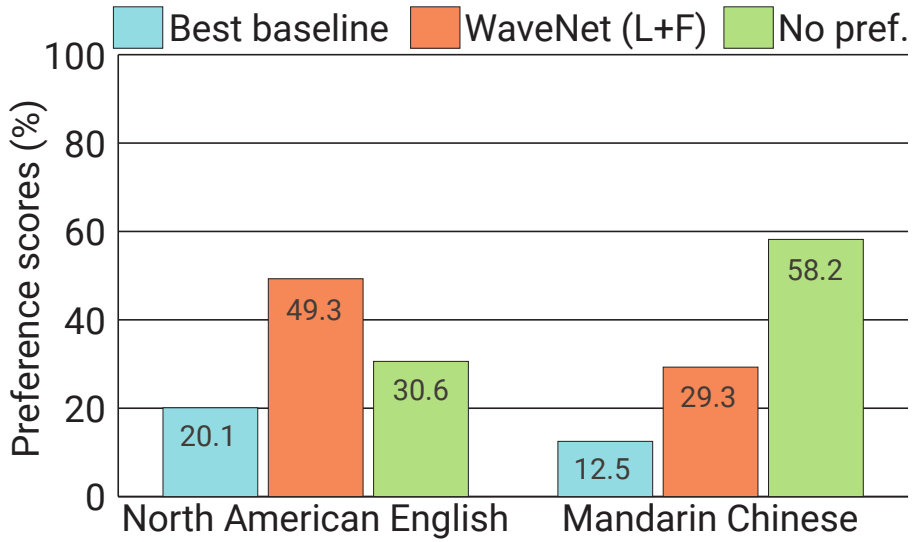


(Google WaveNet from van den Oord et al., 2016)

Example: convolutional network generates time series

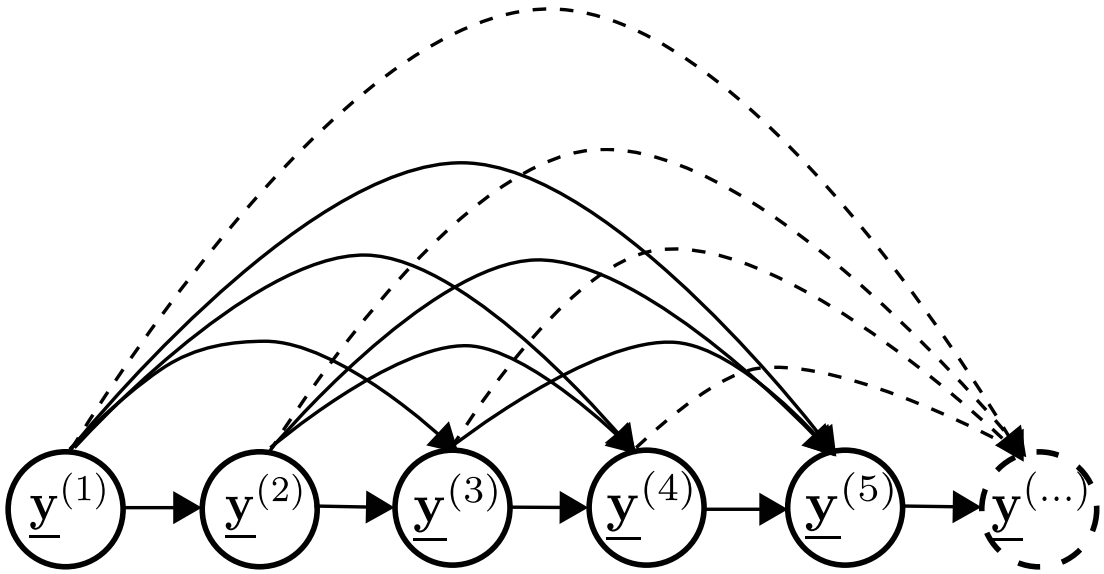
- *causally connected* convolutions over limited *time window*
- shifting the time window generates output time series
- *time dilation* in each layer $\rightsquigarrow$ large windows
- WaveNet generates speech from text that sounds very real

- training with 24/34 hours of spoken text in English/Chinese
- CNN conditioned on linguistic features derived from text
  - 240 ms input window length
  - 16 kHz raw audio output
- validation by subjective preference between two generated sentences

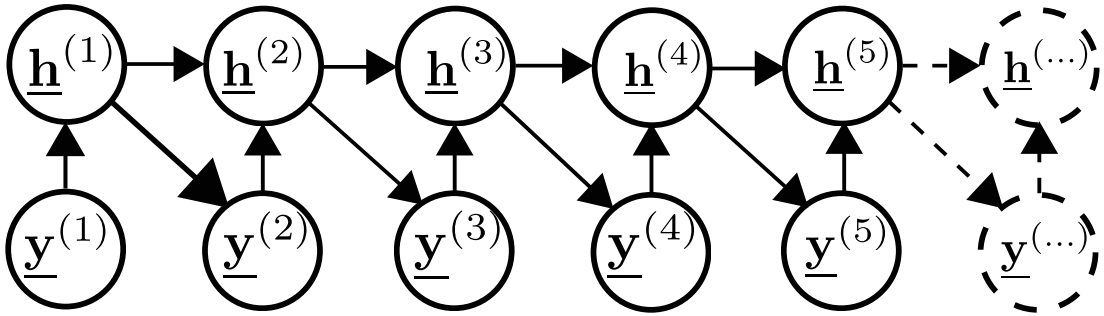


(Google WaveNet from van den Oord et al., 2016)

Modeling time series with hidden states



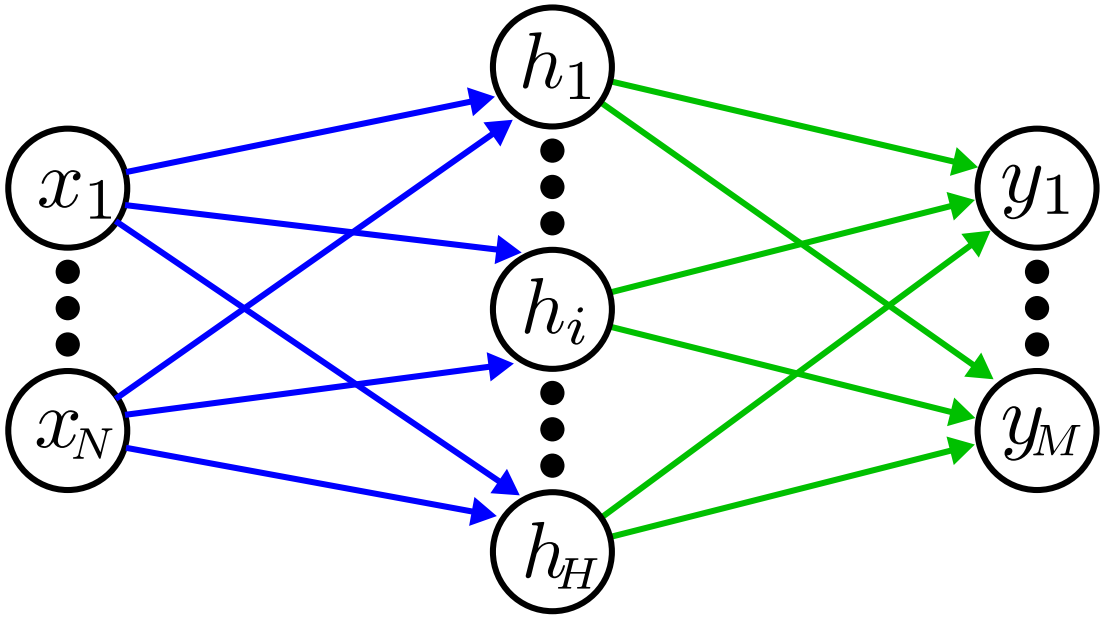
assume: $P\left(\underline{\mathbf{y}}^{(t+1)} | \underline{\mathbf{y}}^{(1)}, \ldots, \underline{\mathbf{y}}^{(t)}\right) \approx P\left(\underline{\mathbf{y}}^{(t+1)} | \underline{\mathbf{h}}^{(t)}\right)$



(Goodfellow et al., 2016, Section 10.2.3)

1.6.2 Recurrent Networks

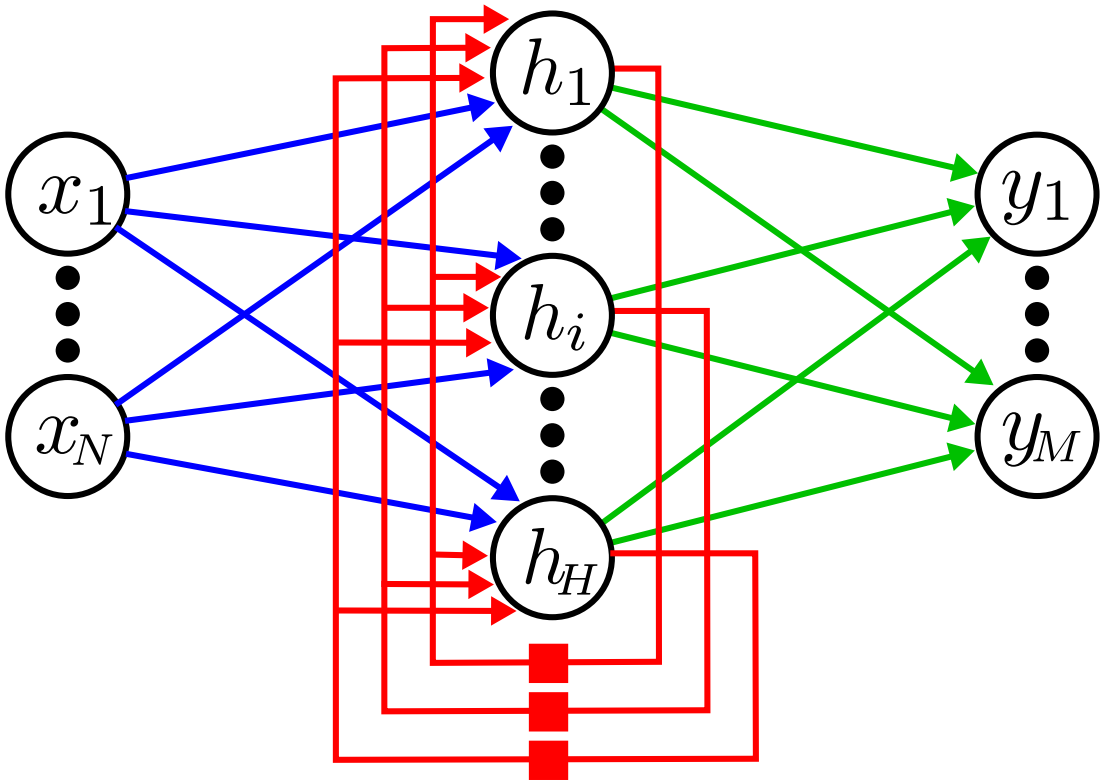
Recurrent neural networks (RNN)



$$h_i^{(t)} = f\Big(\sum_{k=1}^{N} U_{ik}\, x_k^{(t)} + b_i\Big), \qquad \text{e.g. } f(\cdot) = \tanh(\cdot)$$

$$y_k^{(t)} = g\Big(\sum_{j=1}^{H} V_{kj}\, h_j^{(t)} + c_k\Big), \qquad \text{e.g. } g(\cdot) = \text{softmax}(\cdot)$$

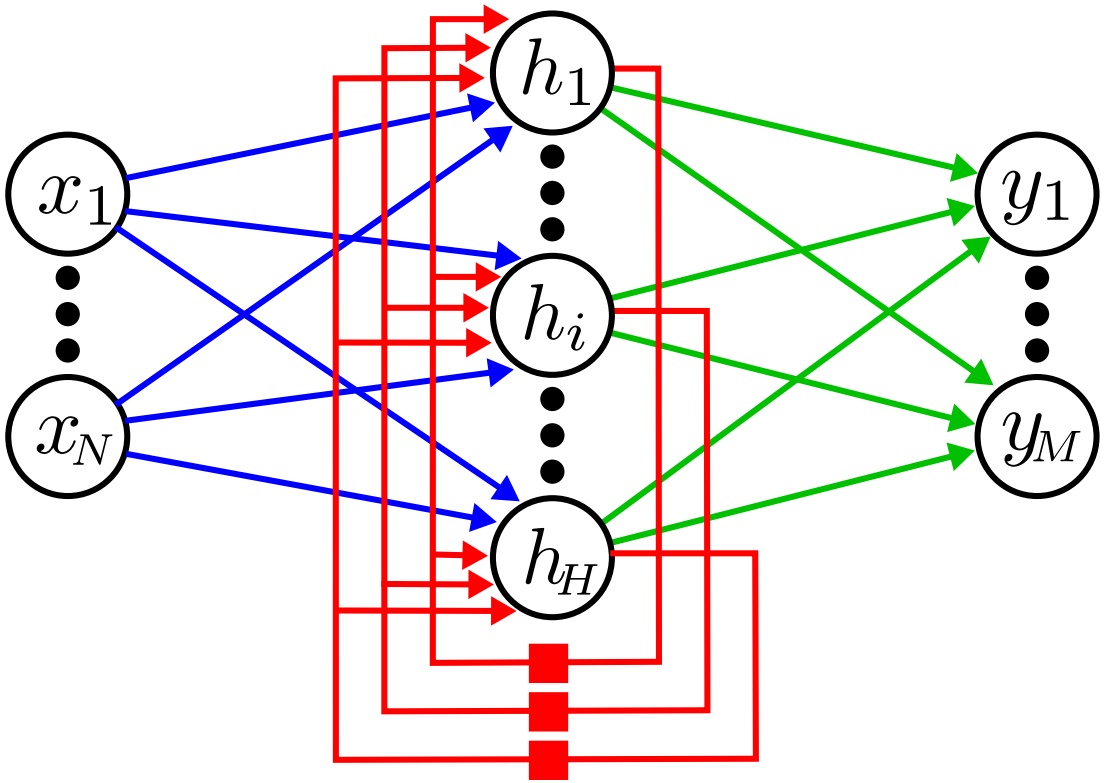
Recurrent neural networks (RNN)

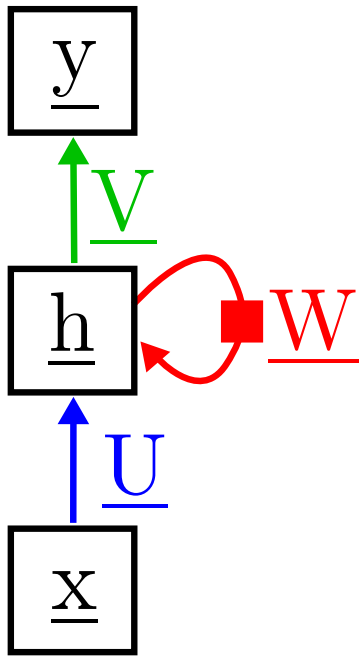


$$h_i^{(t)} = f\Big(\sum_{k=1}^{N} U_{ik}\, x_k^{(t)} + \sum_{j=1}^{H} W_{ij}\, h_j^{(t-1)} + b_i\Big), \qquad \text{e.g. } f(\cdot) = \tanh(\cdot)$$

$$y_k^{(t)} = g\Big(\sum_{j=1}^{H} V_{kj}\, h_j^{(t)} + c_k\Big), \qquad \text{e.g. } g(\cdot) = \text{softmax}(\cdot)$$

Recurrent neural networks (RNN)

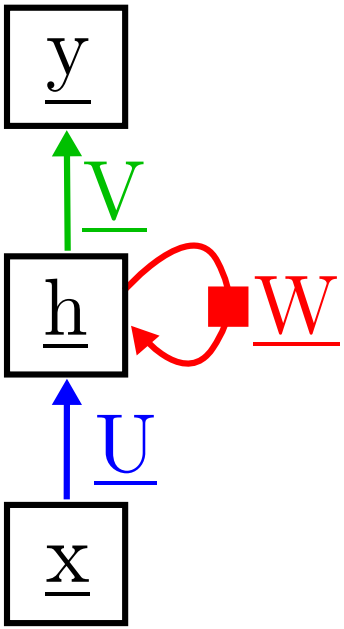




$$h_i^{(t)} = f\Big(\sum_{k=1}^{N} U_{ik}\, x_k^{(t)} + \sum_{j=1}^{H} W_{ij}\, h_j^{(t-1)} + b_i\Big), \qquad \text{e.g. } f(\cdot) = \tanh(\cdot)$$

$$y_k^{(t)} = g\Big(\sum_{j=1}^{H} V_{kj}\, h_j^{(t)} + c_k\Big), \qquad \text{e.g. } g(\cdot) = \text{softmax}(\cdot)$$

Recurrent neural networks (RNN)



- generates an output $\underline{\mathbf{y}}^{(t)}$ at each time step t
- takes an input $\underline{\mathbf{x}}^{(t)}$ at each time step t
  - possibly previous output $\underline{\mathbf{y}}^{(t-1)}$
- summarizes all previous inputs as $\underline{\mathbf{h}}^{(t)}$
- one hidden layer is sufficient
  (hidden layer is universal approximator)

$$h_i^{(t)} = f\Big(\sum_{k=1}^{N} U_{ik}\, x_k^{(t)} + \sum_{j=1}^{H} W_{ij}\, h_j^{(t-1)} + b_i\Big), \qquad \text{e.g. } f(\cdot) = \tanh(\cdot)$$

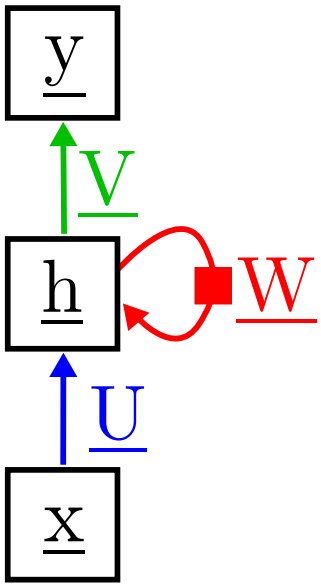
$$y_k^{(t)} = g\Big(\sum_{j=1}^{H} V_{kj}\, h_j^{(t)} + c_k\Big), \qquad \text{e.g. } g(\cdot) = \text{softmax}(\cdot)$$

RNN are Turing machines

- Turing machines (TM) are theoretical computing machines
  - TM apply a finite set of operations between a number of *stacks*
  - TM can compute any recursively computable function $\phi : \mathbb{N} \rightarrow \mathbb{N}$

RNN are universal Turing Machines (Siegelmann and Sontag, 1991)

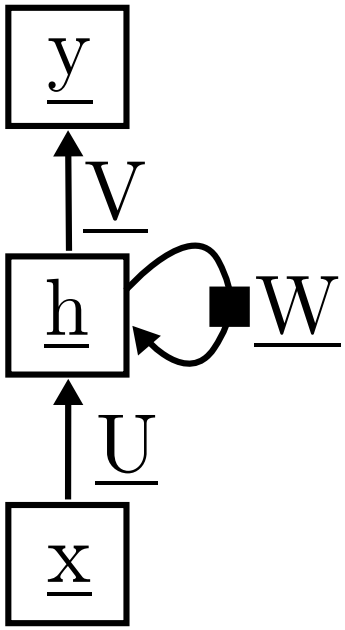
Any function computable by a TM can be computed by a RNN with constant size.



- input sequence $x_1^{(t)}$ contains function ϕ and parameter n
- transfer functions encode the binary size of stacks
  $f(\cdot) = g(\cdot) = \min\left(\max(\cdot, 0), 1\right)$
  - binary $n = 1011...$ are represented $h_i^{(t)} = 0.1011...$
- all operations of a TM can be encoded by weights
- output sequence $y_1^{(t)}$ contains answer $m = \phi(n)$

$$y_1^{(\ldots)} = 0 \ldots 0 \overbrace{1 \ldots 1}^{m \text{ times}} 0 \ldots$$

Training a RNN

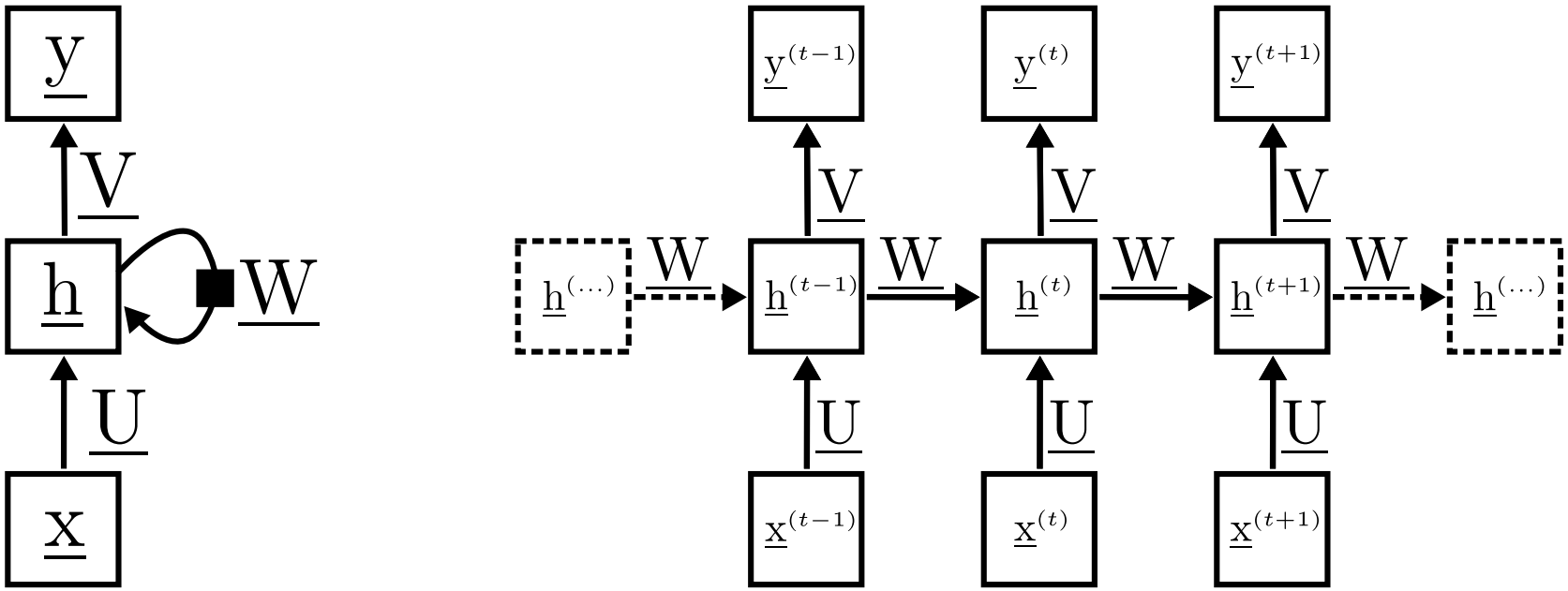


- cost function $e^{(\alpha,t)}$ for time step t of training sequence α

$$E^T = \frac{1}{p}\sum_{\alpha=1}^{p}\frac{1}{n_\alpha}\sum_{t=1}^{n} e^{(\alpha,t)}, \quad \text{for training set} \quad \left\{\{\underline{\mathbf{x}}^{(\alpha,t)}, \underline{\mathbf{y}}^{(\alpha,t)}\}_{t=1}^{n_\alpha}\right\}_{\alpha=1}^{p}$$

(for details see Williams and Zipser, 1995)

Training a RNN



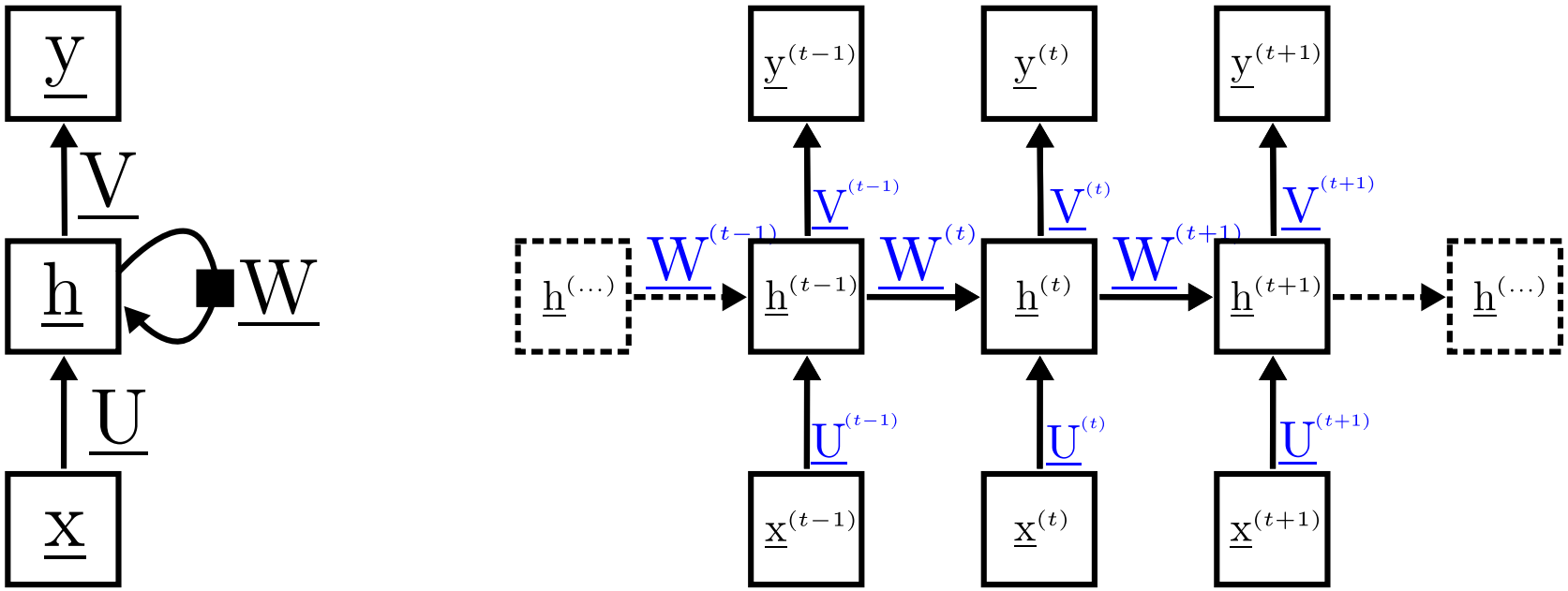
- cost function $e^{(\alpha,t)}$ for time step t of training sequence α

$$E^T = \frac{1}{p}\sum_{\alpha=1}^{p}\frac{1}{n_\alpha}\sum_{t=1}^{n} e^{(\alpha,t)}, \quad \text{for training set} \quad \left\{\{\underline{\mathbf{x}}^{(\alpha,t)}, \underline{\mathbf{y}}^{(\alpha,t)}\}_{t=1}^{n_\alpha}\right\}_{\alpha=1}^{p}$$

- compute gradient of a sequence by unfolding the network in time
- online gradient computation for sequence α has complexity $\mathcal{O}(n_\alpha^2)$

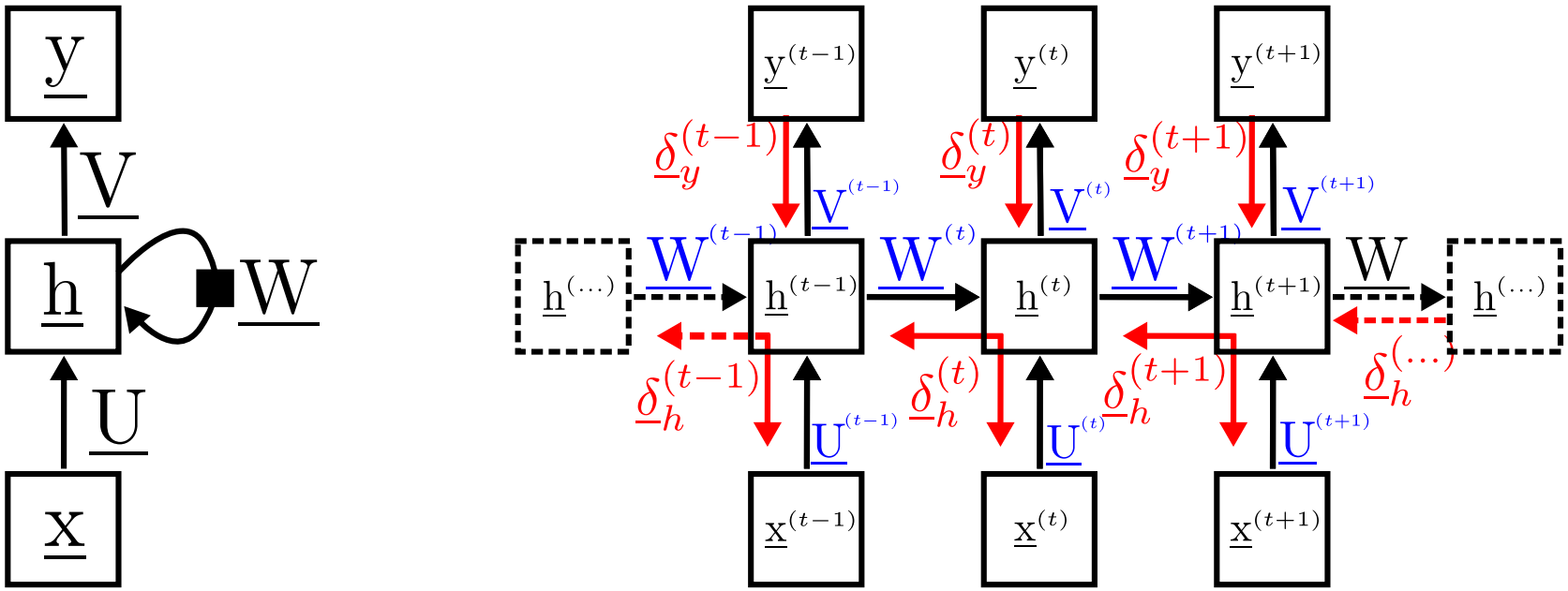
(for details see Williams and Zipser, 1995)

Backpropagation through time (BPTT)



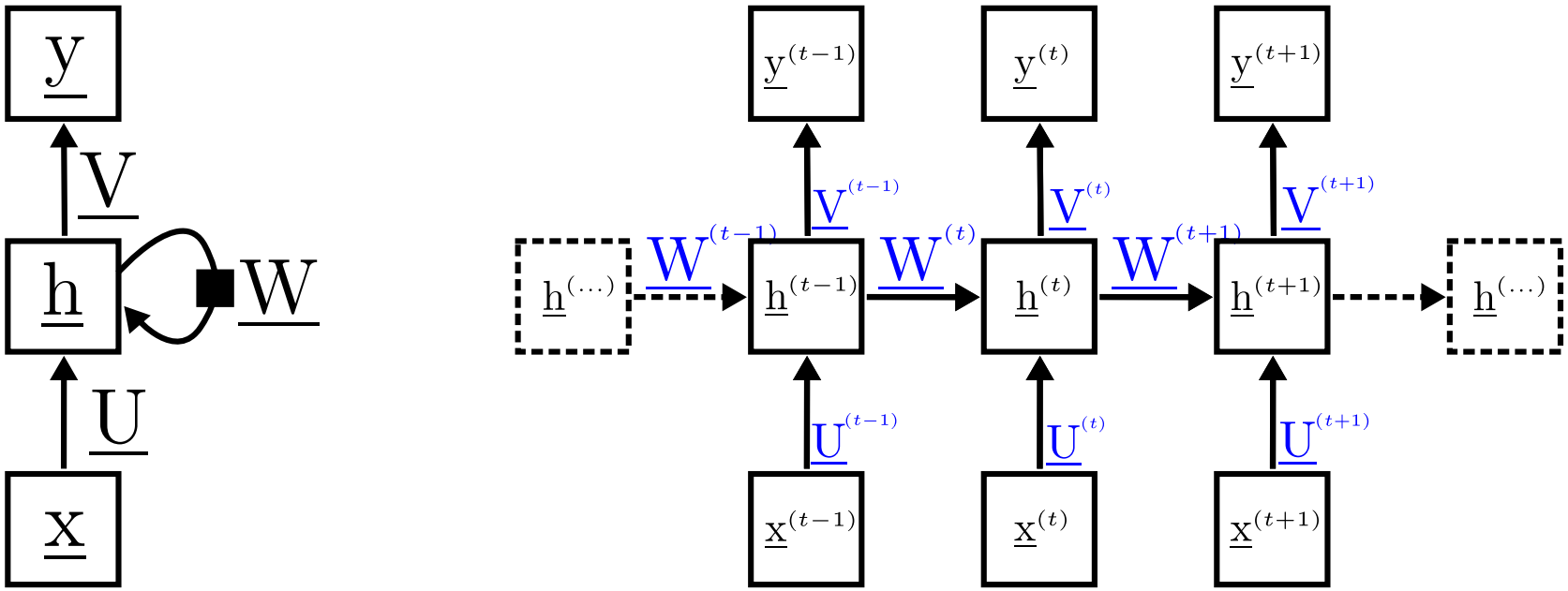
- assume all unfolded weights, e.g. $\mathbf{W}^{(t)}$, are independent

Backpropagation through time (BPTT)



- assume all unfolded weights, e.g. $\underline{\mathbf{W}}^{(t)}$, are independent
- compute gradients of deep MLP with backpropagation ($\rightsquigarrow \mathcal{O}(n_\alpha)$)

Backpropagation through time (BPTT)



- assume all unfolded weights, e.g. $\underline{\mathbf{W}}^{(t)}$, are independent
- compute gradients of deep MLP with backpropagation ($\rightsquigarrow \mathcal{O}(n_\alpha)$)
- average all computed gradients $\frac{\partial e^{(\alpha,t)}}{\partial \underline{\mathbf{W}}^{(\alpha,\tau)}}$ for weight update

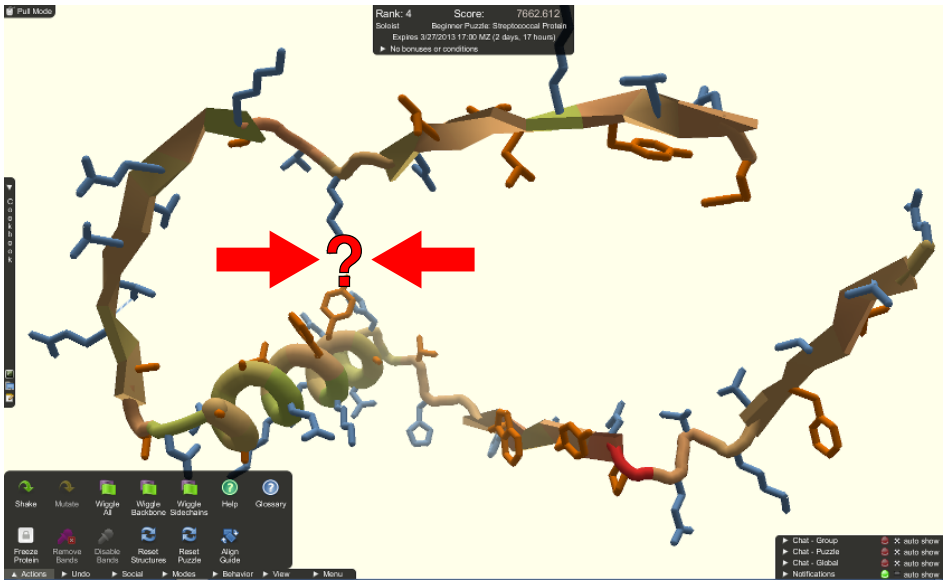
$$\Delta \underline{\mathbf{W}} = -\eta \frac{\partial E^T}{\partial \underline{\mathbf{W}}} = -\frac{\eta}{pn} \sum_{\alpha=1}^{p} \sum_{t=1}^{n} \frac{\partial e^{(\alpha,t)}}{\partial \underline{\mathbf{W}}} \approx \frac{\eta}{pn^2} \sum_{\alpha=1}^{p} \sum_{t=1}^{n} \sum_{\tau=1}^{n} \frac{\partial e^{(\alpha,t)}}{\partial \underline{\mathbf{W}}^{(\alpha,\tau)}}$$

1.6.3 Long-term Dependencies

Long-term dependencies

- long sequences require often a long memory

Example: protein folding



Example: text understanding

When you are old and grey and full of sleep,
And nodding by the fire, take down this book,
And slowly read, and dream of the soft look
Your eyes had once, and of their shadows deep;
How many loved your moments of glad grace,
And loved your beauty with love false or true,
But **one man** loved the pilgrim Soul in you,
And loved the sorrows of your changing face;
And bending down beside the glowing bars,
Murmur, a little sadly, how Love fled
And paced upon the mountains overhead
And hid **his face** amid a crowd of stars.

William Butler Yeats

Exponential forgetting

- a RNN with no inputs and a linear transfer function

$$\underline{\mathbf{h}}^{(t)} = \underline{\mathbf{W}}\,\underline{\mathbf{h}}^{(t-1)} = (\underline{\mathbf{W}})^t\,\underline{\mathbf{h}}^{(0)}$$

Exponential forgetting

- a RNN with no inputs and a linear transfer function

$$\underline{\mathbf{h}}^{(t)} \quad = \quad \underline{\mathbf{W}}\,\underline{\mathbf{h}}^{(t-1)} \quad = \quad (\underline{\mathbf{W}})^t\,\underline{\mathbf{h}}^{(0)}$$

- eigenvalue decomposition: $\underline{\mathbf{W}} = \underline{\mathbf{U}}\,\underline{\mathbf{\Lambda}}\,\underline{\mathbf{U}}^\top$

$$\underline{\mathbf{h}}^{(t)} \quad = \quad \underbrace{\underline{\mathbf{U}}}_{\text{rotate back}}\ \overbrace{(\underline{\mathbf{\Lambda}})^t}^{\text{exponential scaling}}\ \underbrace{\underline{\mathbf{U}}^\top \underline{\mathbf{h}}^{(0)}}_{\text{rotation}}$$

Exponential forgetting

- a RNN with no inputs and a linear transfer function

$$\underline{\mathbf{h}}^{(t)} \quad = \quad \underline{\mathbf{W}}\,\underline{\mathbf{h}}^{(t-1)} \quad = \quad (\underline{\mathbf{W}})^t\,\underline{\mathbf{h}}^{(0)}$$

- eigenvalue decomposition: $\underline{\mathbf{W}} = \underline{\mathbf{U}}\,\underline{\mathbf{\Lambda}}\,\underline{\mathbf{U}}^\top$

$$\underline{\mathbf{h}}^{(t)} \quad = \quad \underbrace{\underline{\mathbf{U}}}_{\text{rotate back}}\ \overbrace{(\underline{\mathbf{\Lambda}})^t}^{\text{exponential scaling}}\ \underbrace{\underline{\mathbf{U}}^\top\underline{\mathbf{h}}^{(0)}}_{\text{rotation}}$$

- long-term activity dominated by the largest $|\lambda_1|$
  - $|\lambda_1| > 1 \quad \rightsquigarrow \quad \lim_{t\to\infty} h_i^{(t)} = \pm\infty\,, \forall i$ (exploding activity)
  - $|\lambda_1| < 1 \quad \rightsquigarrow \quad \lim_{t\to\infty} h_i^{(t)} = 0\,, \forall i$ (vanishing activity)

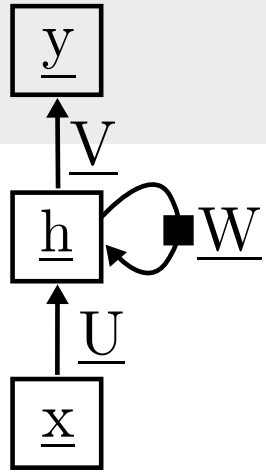
Vanishing gradients

- local error terms $\underline{\delta}^{(t)}$ in backpropagation through time
  - for transfer function $f(x) = \tanh(x)$, i.e. $f'(x) = \left(1 - f^2(x)\right)$

$$\delta_i^{(t)} = \frac{\partial E^T}{\partial h_i^{(t)}} = \underbrace{\left(1 - \left(h_i^{(t)}\right)^2\right)}_{\text{derivative} < 1 \text{ i.e. contracting}} \cdot \underbrace{\left(\sum_{j=1}^{H} W_{ij}^{\top} \delta_j^{(t+1)}\right)}_{\text{potentially exploding/vanishing}}$$

- over many time steps the local errors $\delta_i^{(t)}$ often vanish
- small short-term errors overshadow large long-term errors

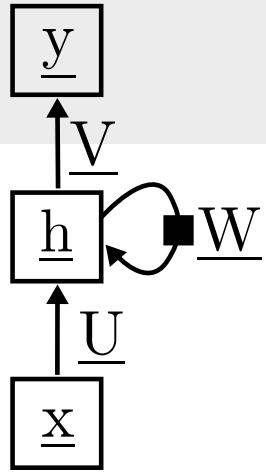
Solution 1: echo-state networks (1)



- system dynamics determine the input representation
  - non-linear preprocessing step with memory
- address vanishing gradients by keeping $\mathbf{W}$ and $\mathbf{U}$ fixed
  - output weights $\mathbf{V}$ can be trained by e.g. linear regression
  - hidden "reservoir" maintains inputs/errors over time
- system dynamics determine how fast inputs/errors are forgotten

(Jaeger and Haas, 2004; Goodfellow et al., 2016, Section 10.8)

Solution 1: echo-state networks (2)



- eigenvalues λ_i of the system's *Jacobian* $\underline{\mathbf{J}}$

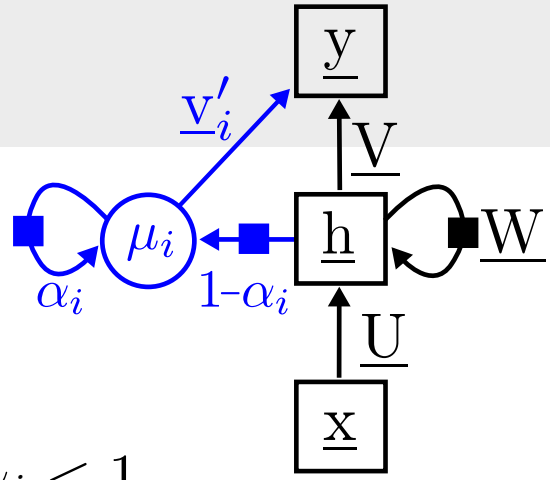
$$J_{ij}^{(t)} = \frac{\partial h_i^{(t)}}{\partial h_j^{(t-1)}} = f'(h_i^{(t)}) \cdot W_{ij}$$

- eigenvectors with $|\lambda_i| \approx 1$ vanish/explode *very slowly*
  - inputs/errors are maintained for a long time

- generate random $\underline{\mathbf{W}}$, and $\underline{\mathbf{U}}$ where all $|\lambda_i| \approx r$
  - $f'(h_i^{(t)}) < 1$ is a contraction $\rightsquigarrow$ $r > 1$
  - values between $r = 1.2$ and $r = 3$ are used in practice
  - n hidden neurons can maintain inputs/errors for up to n time steps

(Jaeger and Haas, 2004; Goodfellow et al., 2016, Section 10.8)

Solution 2: leaky units (1)



- introduce linear "leaky units" μ_i

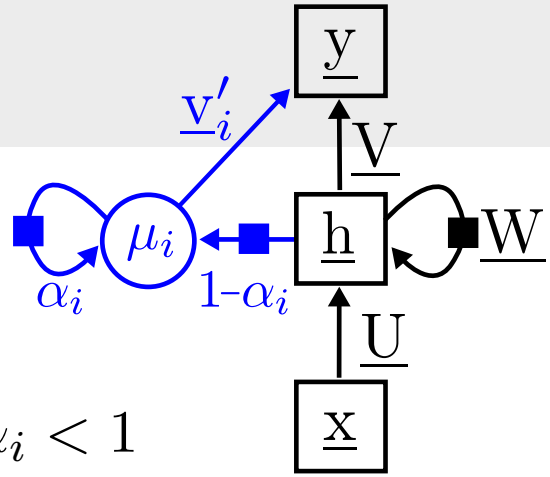
$$\mu_i^{(t)} \quad = \quad \alpha_i \, \mu_i^{(t-1)} + (1 - \alpha_i) \, h_i^{(t-1)} \,, \qquad 0 < \alpha_i < 1$$

- μ_i estimates the average activity of h_i with time constant α_i
  - averages maintain errors/activity over longer time periods
  - small α_i average over short periods
  - large α_i average over long periods

- leaky unit's have different time constants α_i

(Goodfellow et al., 2016, Section 10.9)

Solution 2: leaky units (2)



- introduce linear "leaky units" μ_i

$$\mu_i^{(t)} \quad = \quad \alpha_i \, \mu_i^{(t-1)} + (1 - \alpha_i) \, h_i^{(t-1)} \,, \qquad 0 < \alpha_i < 1$$

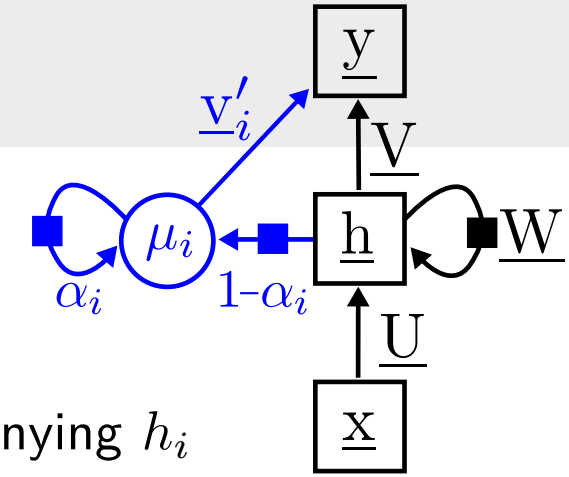
- the extended system's Jacobian $\underline{\mathbf{J}}_{h\mu}$:

$$\underline{\mathbf{J}}_{h\mu} \quad = \quad \begin{bmatrix} \partial \underline{\mathbf{h}}^{(t)} / \partial \underline{\mathbf{h}}^{(t-1)}, & \partial \underline{\mathbf{h}}^{(t)} / \partial \underline{\mu}^{(t-1)} \\ \partial \underline{\mu}^{(t)} / \partial \underline{\mathbf{h}}^{(t-1)}, & \partial \underline{\mu}^{(t)} / \partial \underline{\mu}^{(t-1)} \end{bmatrix} \quad = \quad \begin{bmatrix} \underline{\mathbf{J}}_h \,, & \underline{\mathbf{0}} \\ \mathrm{diag}(1 - \underline{\alpha}) \,, & \mathrm{diag}(\underline{\alpha}) \end{bmatrix}$$

- errors/inputs in μ_i vanish with eigenvalue α_i
  - pairs h_i / μ_i with small α_i specialize in *short-term memory*
  - pairs h_i / μ_i with large α_i specialize in *long-term memory*

(Goodfellow et al., 2016, Section 10.9)

Leaky units with indefinite memory?



- unit μ_i with $\alpha_i = 1$
  - μ_i would maintain memories indefinitely
  - μ_i can no longer be overwritten by the accompanying h_i

- BPTT sends local errors $\delta_i'^{(t)}$ in an endless loop

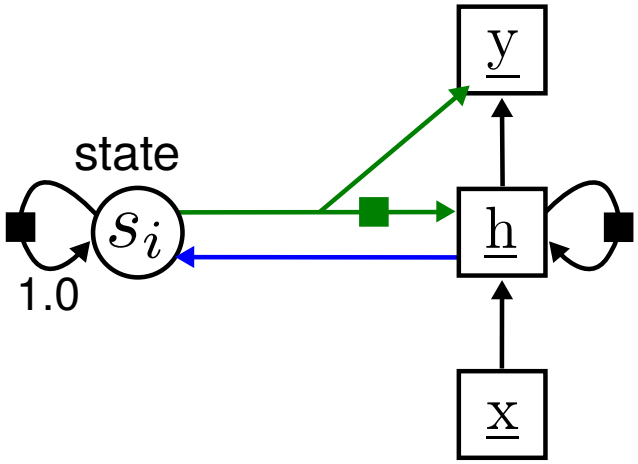
$$\delta_i'^{(t)} := \frac{\partial E^T}{\partial \mu_i^{(t)}} = \delta_i'^{(t+1)} + \sum_{k=1}^{M} \frac{\partial E^T}{\partial y_k^{(t)}} \cdot V'_{ki} \cdot f'\Big(\sum_{j=1}^{H} V_{kj}\, h_j^{(t)} + \sum_{l=1}^{L} V'_{kl}\, \mu_l^{(t)}\Big)$$

  - local errors from the output nodes $y_k^{(t)}$ accumulate in $\mu_i^{(t)}$ over time
  - the accumulated error $\delta_i^{(t)}$ never leaves μ_i, due to $1 - \alpha_i = 0$

- memory units require a gating mechanism
  - for reading, writing and error handling

Solution 3: long short-term memory (LSTM)

- leaky unit s_i with $\alpha_i = 1$
- time delayed feedback to hidden layer
- transfer function $\sigma(x) = \left(1 + e^{-x}\right)^{-1}$

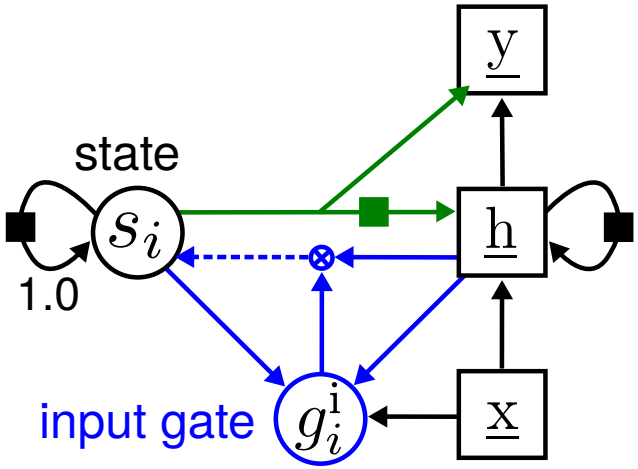


$$
\begin{aligned}
s_i^{(t)} &= s_i^{(t-1)} + \sigma\left(\sum_{j=1}^{H} W_{ij}^{s} h_j^{(t)}\right) \\
y_k^{(t)} &= f\left(\sum_{j=1}^{H} V_{kj}^{y}\, h_j^{(t)} + b_k + \sum_{i=1}^{L} U_{ki}^{y}\; s_i^{(t)}\right), \qquad \text{e.g. } f(\cdot) = \text{softmax}(\cdot) \\
h_j^{(t)} &= \tanh\left(\sum_{k=1}^{N} U_{jk}^{h}\, x_k^{(t)} + \sum_{l=1}^{H} W_{jl}^{h}\, h_l^{(t-1)} + b_j^{h} + \sum_{i=1}^{L} V_{ji}^{h}\; s_i^{(t-1)}\right)
\end{aligned}
$$

(Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2016, Section 10.10)

Solution 3: long short-term memory (LSTM)

- introducing an input ("write") gate g_i^{i}
  - gate transfer function $\sigma(x) = \left(1 + e^{-x}\right)^{-1}$
- state input is multiplied ($\otimes$) with gate
- state s_i only changes when $g_i^{\mathrm{i}(t)} \gg 0$
- local errors $\delta_i^{(t)}$ still accumulate in s_i



$$s_i^{(t)} = s_i^{(t-1)} + \sigma\left(\sum_{j=1}^{H} W_{ij}^{s} h_j^{(t)}\right) \cdot g_j^{\mathrm{i}(t)}$$

$$y_k^{(t)} = f\left(\sum_{j=1}^{H} V_{kj}^{y}\, h_j^{(t)} + b_k + \sum_{i=1}^{L} U_{ki}^{y}\; s_i^{(t)}\right), \qquad \text{e.g. } f(\cdot) = \mathrm{softmax}(\cdot)$$

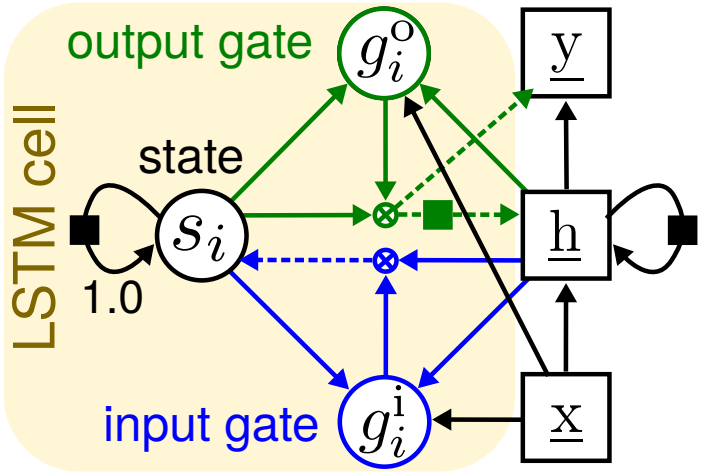
$$h_j^{(t)} = \tanh\left(\sum_{k=1}^{N} U_{jk}^{h}\, x_k^{(t)} + \sum_{l=1}^{H} W_{jl}^{h}\, h_l^{(t-1)} + b_j^{h} + \sum_{i=1}^{L} V_{ji}^{h}\; s_i^{(t-1)}\right)$$

$$g_i^{\mathrm{i}(t)} = \sigma\left(\sum_{k=1}^{N} U_{ik}^{\mathrm{i}}\, x_k^{(t)} + \sum_{l=1}^{L} V_{il}^{\mathrm{i}}\, s_l^{(t)} + \sum_{j=1}^{H} W_{ij}^{\mathrm{i}}\, h_j^{(t)} + b_i^{\mathrm{i}}\right)$$

(Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2016, Section 10.10)

Solution 3: long short-term memory (LSTM)

- introducing an output ("read") gate g_i^{i}
  - gate transfer function $\sigma(x) = \left(1 + e^{-x}\right)^{-1}$
- state output is multiplied ($\otimes$) with gate
- error $\delta_i^{(t)}$ only changes when $g_i^{\mathrm{o}(t)} \gg 0$
- both gates and the state form a LSTM cell



$$s_i^{(t)} = s_i^{(t-1)} + \sigma\left(\sum_{j=1}^{H} W_{ij}^{s} h_j^{(t)}\right) \cdot g_j^{\mathrm{i}(t)}$$

$$y_k^{(t)} = f\left(\sum_{j=1}^{H} V_{kj}^{y}\, h_j^{(t)} + b_k + \sum_{i=1}^{L} U_{ki}^{y}\left(s_i^{(t)} \cdot g_i^{\mathrm{o}(t)}\right)\right), \qquad \text{e.g. } f(\cdot) = \mathrm{softmax}(\cdot)$$

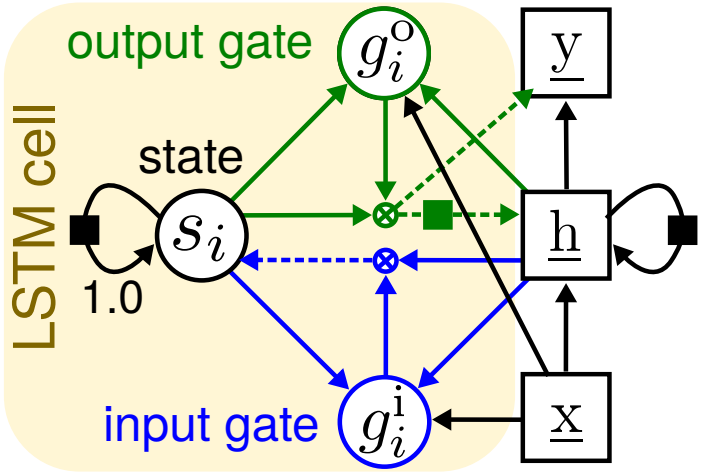
$$h_j^{(t)} = \tanh\left(\sum_{k=1}^{N} U_{jk}^{h}\, x_k^{(t)} + \sum_{l=1}^{H} W_{jl}^{h}\, h_l^{(t-1)} + b_j^{h} + \sum_{i=1}^{L} V_{ji}^{h}\left(s_i^{(t-1)} \cdot g_i^{\mathrm{o}(t)}\right)\right)$$

$$g_i^{\mathrm{i}(t)} = \sigma\left(\sum_{k=1}^{N} U_{ik}^{\mathrm{i}}\, x_k^{(t)} + \sum_{l=1}^{L} V_{il}^{\mathrm{i}}\, s_l^{(t)} + \sum_{j=1}^{H} W_{ij}^{\mathrm{i}}\, h_j^{(t)} + b_i^{\mathrm{i}}\right)$$

$$g_i^{\mathrm{o}(t)} = \sigma\left(\sum_{k=1}^{N} U_{ik}^{\mathrm{o}}\, x_k^{(t)} + \sum_{l=1}^{L} V_{il}^{\mathrm{o}}\, s_l^{(t)} + \sum_{j=1}^{H} W_{ij}^{\mathrm{o}}\, h_j^{(t)} + b_i^{\mathrm{o}}\right)$$

Solution 3: long short-term memory (LSTM)

- gates determine access to the state s_i
  - g_i^{i} learns what to remember
  - g_i^{o} learns when use the memory
- gates regulate the flow of activity and error
  - g_i^{i} regulates the forward-pass
  - g_i^{o} regulates the backward-pass



$$s_i^{(t)} = s_i^{(t-1)} + \sigma\Big(\sum_{j=1}^{H} W_{ij}^{s} h_j^{(t)}\Big) \cdot g_j^{\mathrm{i}(t)}$$

$$y_k^{(t)} = f\Big(\sum_{j=1}^{H} V_{kj}^{y}\, h_j^{(t)} + b_k + \sum_{i=1}^{L} U_{ki}^{y}\big(s_i^{(t)} \cdot g_i^{\mathrm{o}(t)}\big)\Big), \qquad \text{e.g. } f(\cdot) = \mathrm{softmax}(\cdot)$$

$$h_j^{(t)} = \tanh\Big(\sum_{k=1}^{N} U_{jk}^{h}\, x_k^{(t)} + \sum_{l=1}^{H} W_{jl}^{h}\, h_l^{(t-1)} + b_j^{h} + \sum_{i=1}^{L} V_{ji}^{h}\big(s_i^{(t-1)} \cdot g_i^{\mathrm{o}(t)}\big)\Big)$$

$$g_i^{\mathrm{i}(t)} = \sigma\Big(\sum_{k=1}^{N} U_{ik}^{\mathrm{i}}\, x_k^{(t)} + \sum_{l=1}^{L} V_{il}^{\mathrm{i}}\, s_l^{(t)} + \sum_{j=1}^{H} W_{ij}^{\mathrm{i}}\, h_j^{(t)} + b_i^{\mathrm{i}}\Big)$$

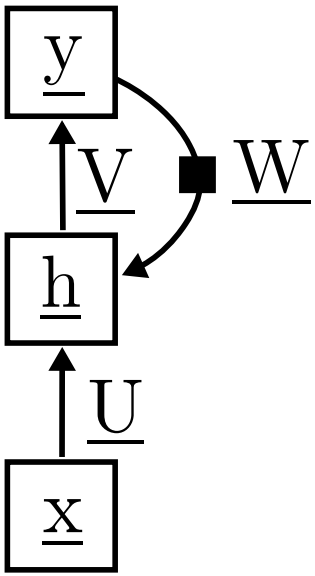
$$g_i^{\mathrm{o}(t)} = \sigma\Big(\sum_{k=1}^{N} U_{ik}^{\mathrm{o}}\, x_k^{(t)} + \sum_{l=1}^{L} V_{il}^{\mathrm{o}}\, s_l^{(t)} + \sum_{j=1}^{H} W_{ij}^{\mathrm{o}}\, h_j^{(t)} + b_i^{\mathrm{o}}\Big)$$

Training long time series

- RNN extended with LSTM cells are the state-of-the-art
  - LSTM cells often have an additional "forget" gate for states s_i
  - forget gates determine at each step the "self-recurrence" α_i
- BPTT can train LSTM networks efficiently
  - see e.g. Williams and Zipser (1995) for details
- LSTM cells can also be inserted in deep feed forward architectures
  - e.g. in deep reinforcement learning (Mnih et al., 2016)

1.6.4 Recurrent Architectures

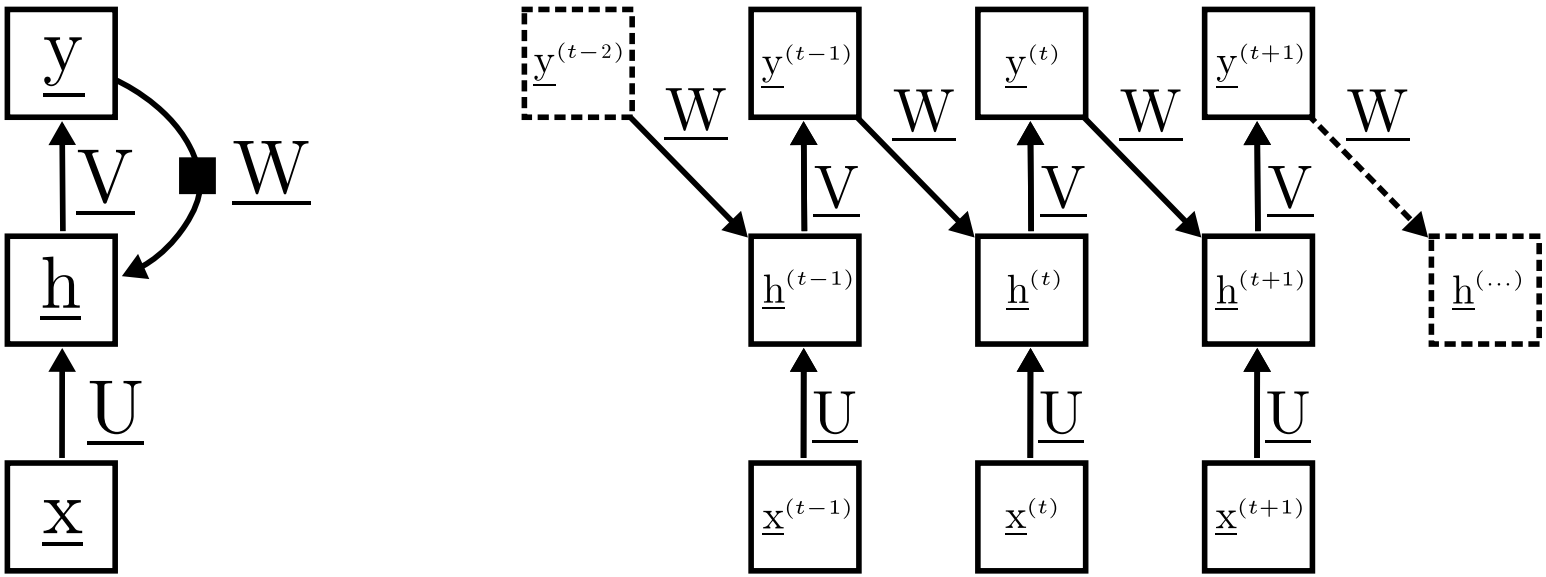
Output recurrence



- The activity of the hidden neurons depends on the previous output

(Goodfellow et al., 2016, Section 10.2.1)

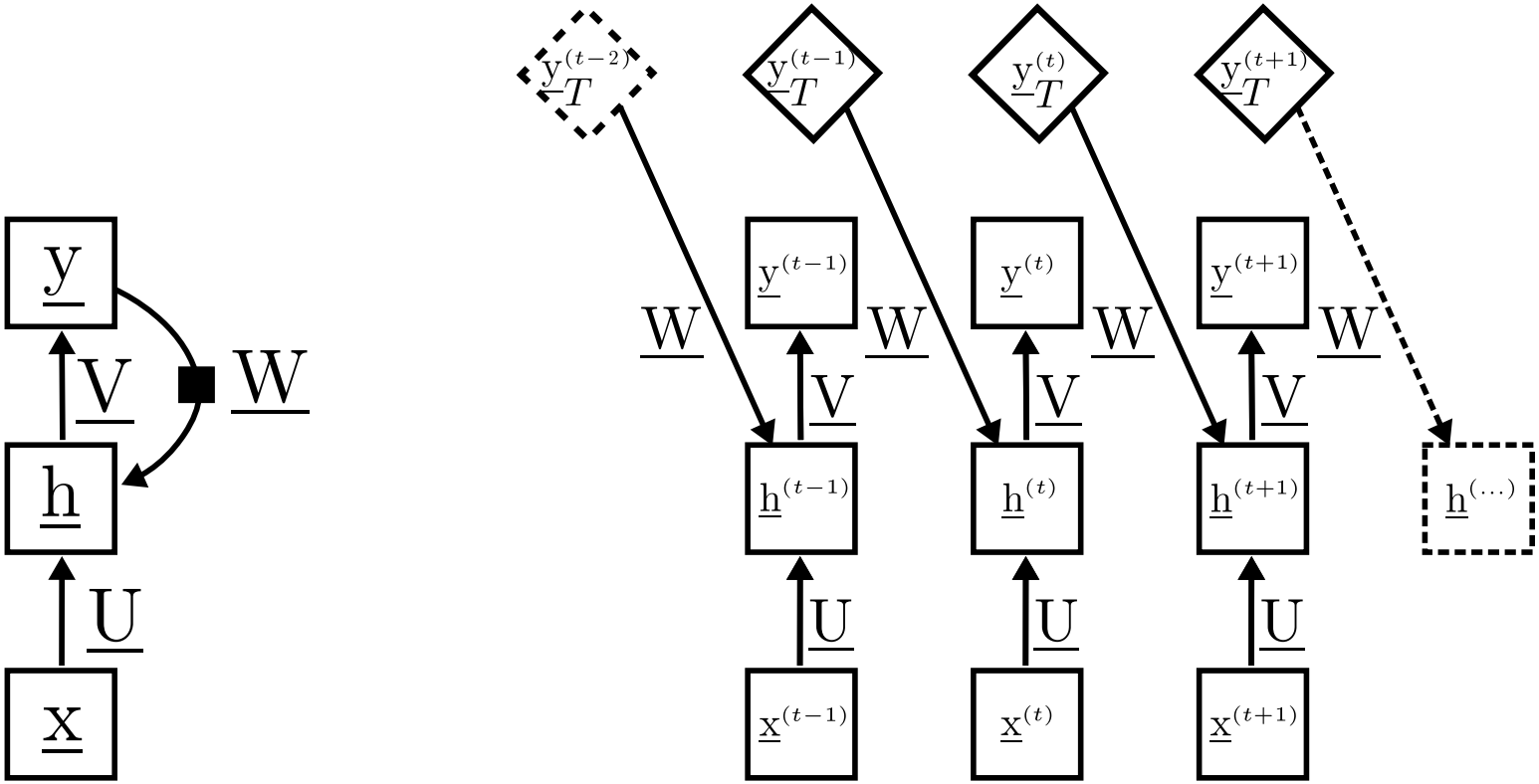
Output recurrence



- The activity of the hidden neurons depends on the previous output
- less powerful function class than general RNN
  - but faster training procedure

(Goodfellow et al., 2016, Section 10.2.1)

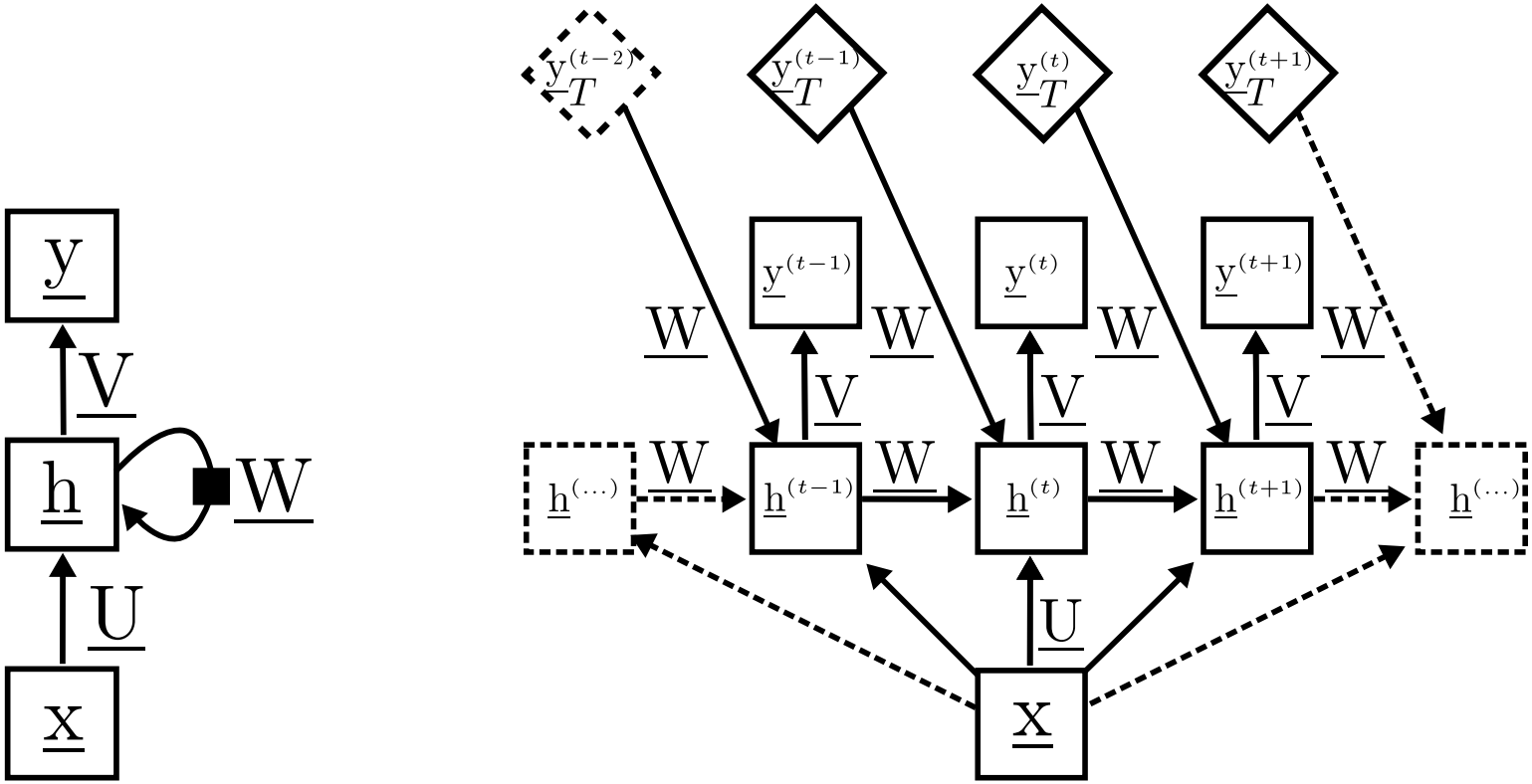
Teacher forcing



- replace $\underline{\mathbf{y}}^{(t-1)}$ with $\underline{\mathbf{y}}_T^{(t-1)}$ during training
  - each time step is independent and can be sampled i.i.d.
- training becomes robust when real outputs and labels are mixed

(Goodfellow et al., 2016, Section 10.2.1)

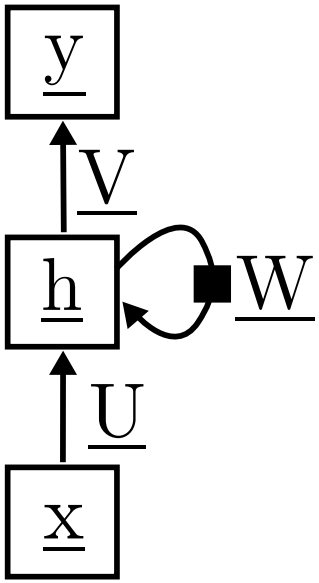
Conditional sequences

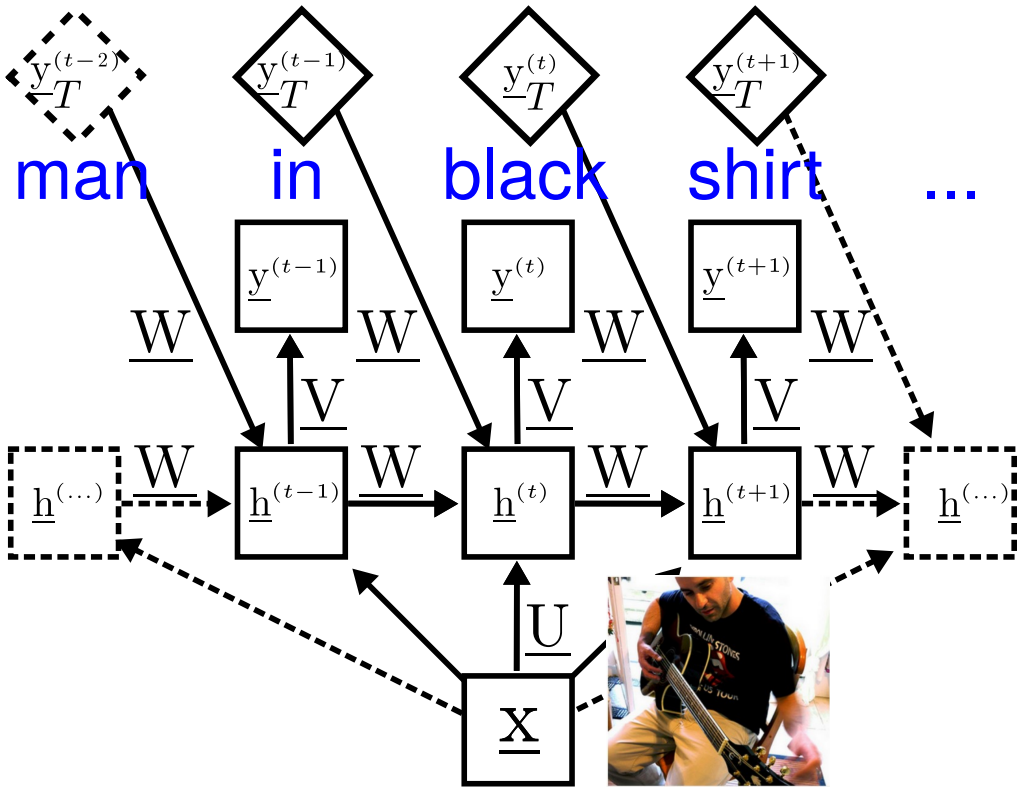


- all teacher forced time steps are *conditioned* on the same input $\underline{\mathbf{x}}$

(Goodfellow et al., 2016, Section 10.2.4)

Conditional sequences





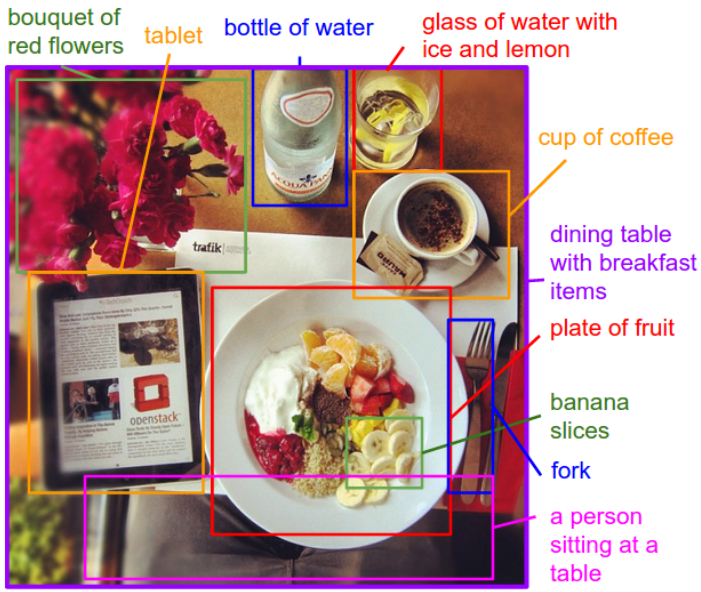
- all teacher forced time steps are *conditioned* on the same input $\underline{\mathbf{x}}$
- the generated sequence can e.g. describe the input $\underline{\mathbf{x}}$

(Goodfellow et al., 2016, Section 10.2.4)

Example: image description (1)

- GOAL: generate a sentence that describes a given image

1. use existing technique to predict bounding boxes of objects in a scene
   - trained on approx. $1,200,000$ samples from 200 classes
   - cuts out bounding boxes of the 19 most likely objects in the image



(bounding box prediction by Girshick et al., 2014), (image description by Karpathy and Fei-Fei, 2017)

Example: image description (1)

- GOAL: generate a sentence that describes a given image

1. use existing technique to predict bounding boxes of objects in a scene
   - trained on approx. $1,200,000$ samples from 200 classes
   - cuts out bounding boxes of the 19 most likely objects in the image

2. CNN transforms each bounding box into a common representation
   - CNN down-samples selected region to 4096 dim. representation
   - CNN has approximately $60,000,000$ parameters

(bounding box prediction by Girshick et al., 2014), (image description by Karpathy and Fei-Fei, 2017)

Example: image description (1)

- GOAL: generate a sentence that describes a given image

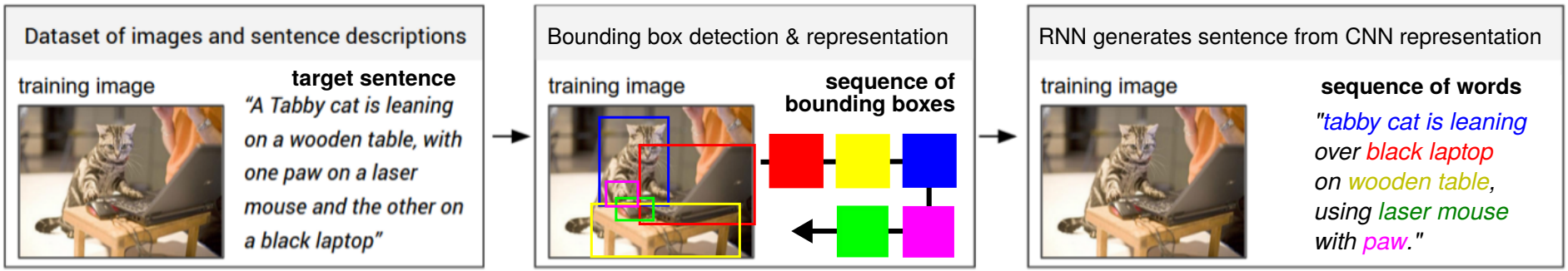
1. use existing technique to predict bounding boxes of objects in a scene
   - trained on approx. $1,200,000$ samples from 200 classes
   - cuts out bounding boxes of the 19 most likely objects in the image

2. CNN transforms each bounding box into a common representation
   - CNN down-samples selected region to 4096 dim. representation
   - CNN has approximately $60,000,000$ parameters

3. learn RNN to generate sentences describing the image
   - conditioned on CNN's representation of all found boxes
   - maximize the likelihood of the next word in the training-sentence
   - RNN gradient is propagated back to CNN

(bounding box prediction by Girshick et al., 2014), (image description by Karpathy and Fei-Fei, 2017)

Example: image description (2)



- RNN generates sentences conditioned on CNN representation

man in black shirt is playing guitar.

construction worker in orange safety vest is working on road.

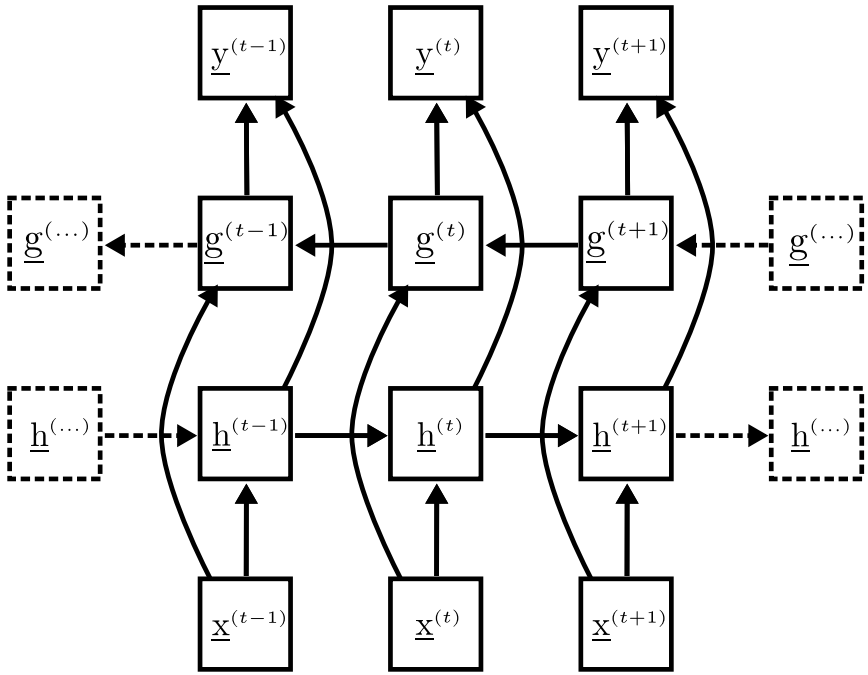
two young girls are playing with lego toy.

boy is doing backflip on wakeboard.

(figures adapted from Karpathy and Fei-Fei, 2017)

Bidirectional RNN

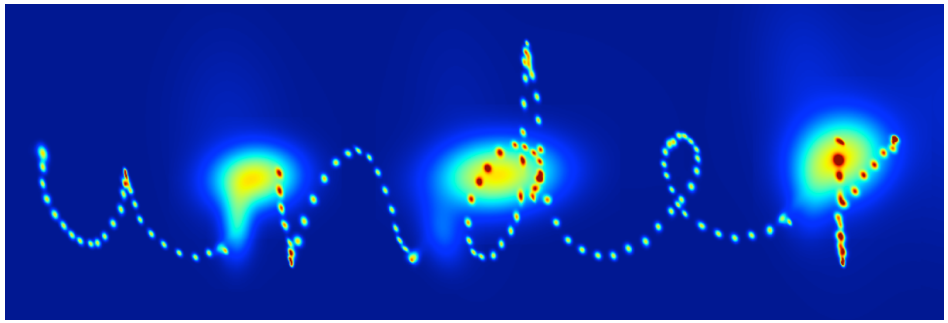
- tasks like speech recognition and translation are *context sensitive*
  - the end of a sentence may be as important as the beginning
- RNN evolve in one direction
- bidirectional RNN propagate information in *both* directions
- training as before with BPTT



(Schuster and Paliwal, 1997; Goodfellow et al., 2016, Section 10.3)

Example: generate handwriting (1)

- hand-written sentences consist of $n_\alpha \approx 700$ pen-positions
  - training set contains $p = 10,741$ written sentences
- bidirectional RNN outputs the probability for the next pen-position
  - RNN output is a *mixture of* 20 *Gaussian distributions*
  - the RNN is conditioned on the current ASCII character

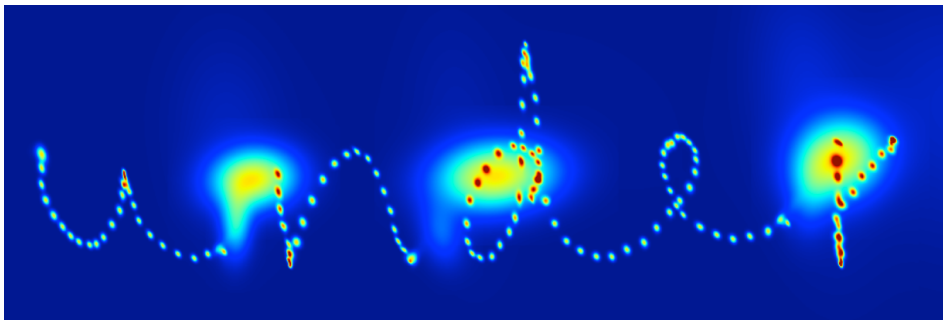


(accumulated probabilities conditioned on the word `under`)

(Graves, 2013)

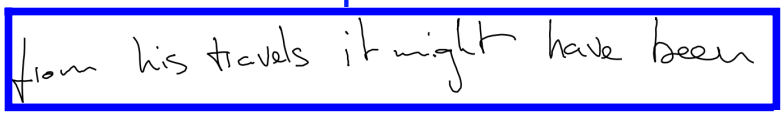
Example: generate handwriting (2)

- the next pen-position is drawn from the predicted distribution
  - BPTT maximizes the likelihood for the training sequences
- bidirectional RNN generates sequence of pen positions
- conditioned on ASCII text

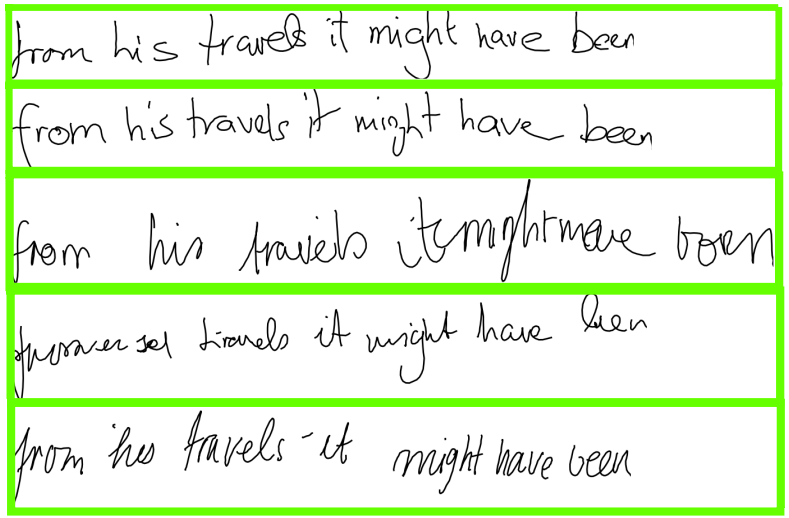


(accumulated probabilities conditioned on the word `under`)

validation-sample



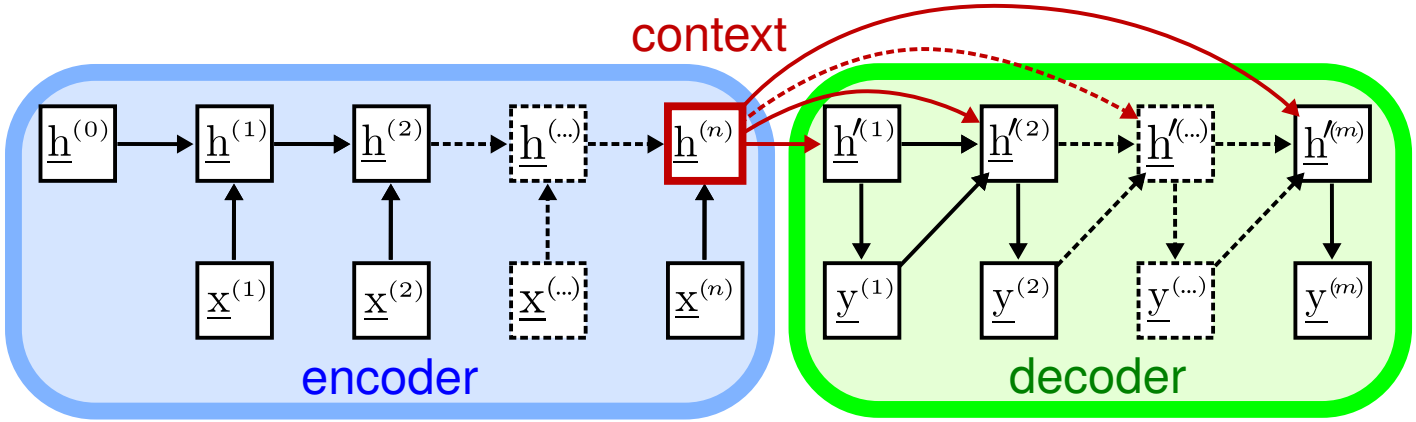
generated by the RNN



(results from Graves, 2013)

Sequence-to-sequence architectures

- some tasks must be conditioned on sequences
  - e.g. machine translation, synthesized speech
- generate sequence of length m from sequence of length $n \neq m$
- final hidden neurons $\underline{\mathbf{h}}^{(n)}$ of the decoder are called context
  - the context "summarizes" the content of the input sequence
- the decoder generates a sequence from the context

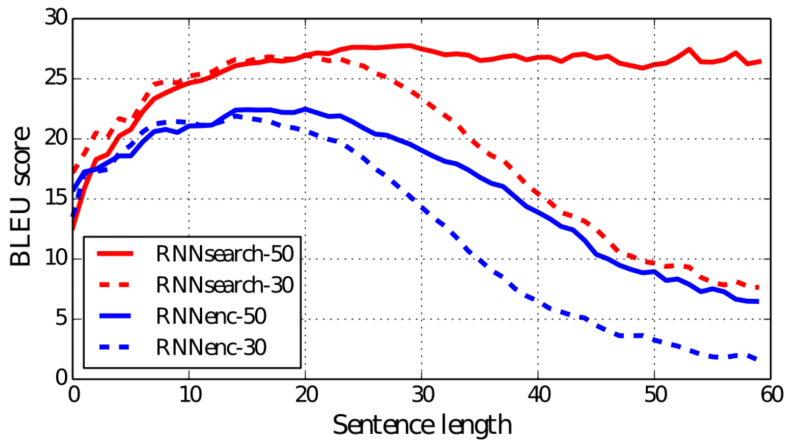


(Goodfellow et al., 2016, Section 10.4)

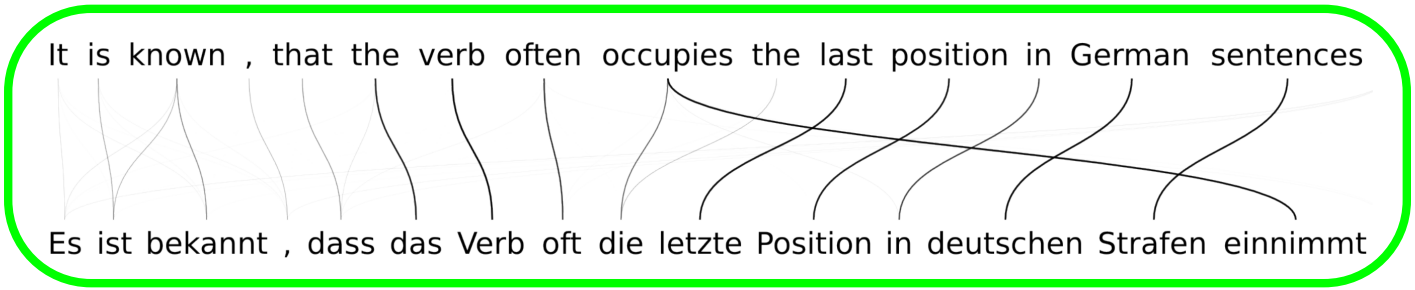
Example: machine translation

- translation of full sentences into another language
  - English to French training set contains $348,000,000$ words
- context must have a sufficient size to contain a sentence
  - encoder and decoder networks have 1000 hidden units each
- encoder-decoders often fail to remember long sentences

  (numbers refer to the length of training sentences)



- dynamic selection of context
  - context is gated sum of all $\underline{\mathbf{h}}^{(t)}$
  - softmax-gates are also learned



(context-gate-strength at each time step of the translation)

(for details on the alignment search see Bahdanau et al., 2015)

End of Section 1.6

the following slides contain

OPTIONAL MATERIAL

References I

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2015. URL http://arxiv.org/abs/1409.0473. Accepted at ICLR 2015 as oral presentation.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. URL http://science.sciencemag.org/content/304/5667/78.full.pdf.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):664–676, 2017.

References II

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, pages 1928–1937, 2016. URL http://jmlr.org/proceedings/papers/v48/mniha16.html.

M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77 – 80, 1991.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, 2016. URL https://arxiv.org/pdf/1609.03499.pdf.

Ronald J. Williams and David Zipser. Backpropagation. chapter Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pages 433–486. L. Erlbaum Associates Inc., 1995.