Deep Learning for NLP

Lecture 8 – Recurrent Neural Nets

Dr. Steffen Eger
steffen.eger@uni-bielefeld.de

Natural Language Learning Group (NLLG)

Universität Bielefeld

CITEC   NLLG

Previous lectures:

- Introduction (MLPs, loss functions, batch size, activation functions, etc.)

- Embeddings – continuous representations of words, letters, sentences, etc.

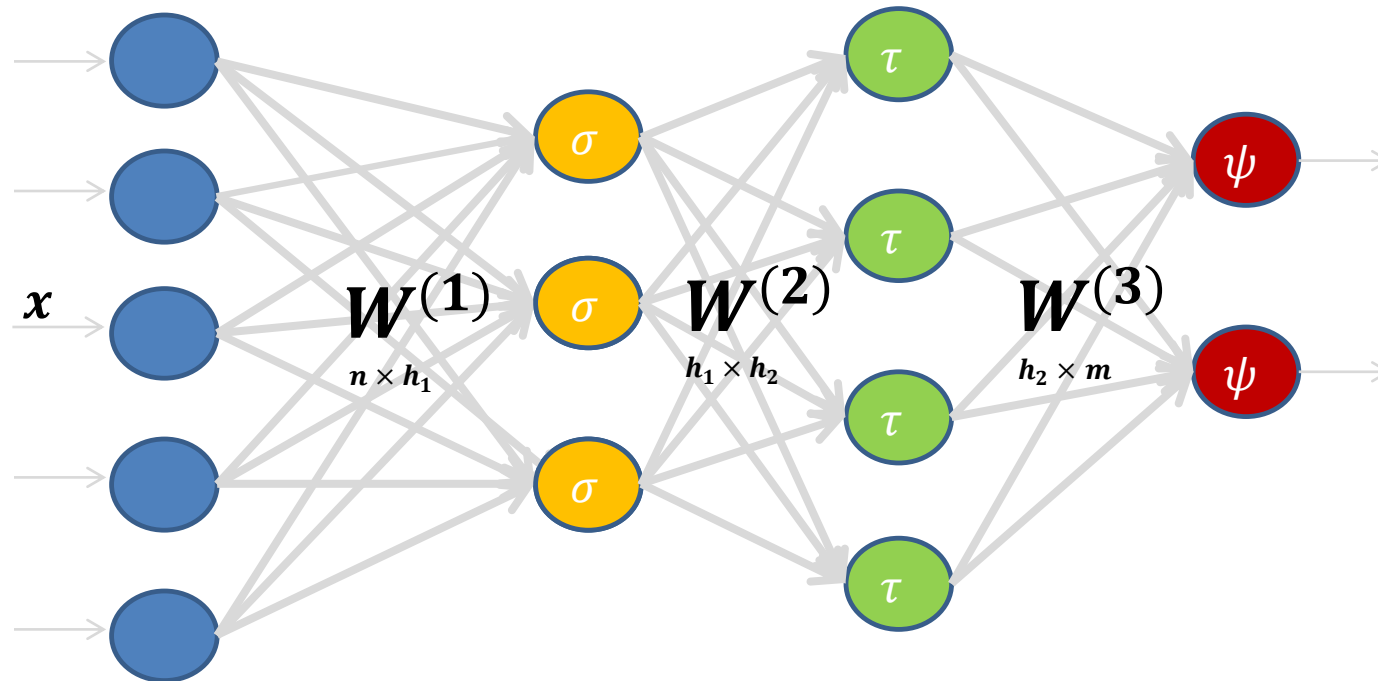- More elaborate architecture: CNN

NLLG

This lecture:

- Recurrent Neural Nets (RNNs)

  - Basic principles

  - Extensions (Bidirectional, etc.)

  - NLP applications: Sequence tagging & sentence classification

- Vanishing gradients

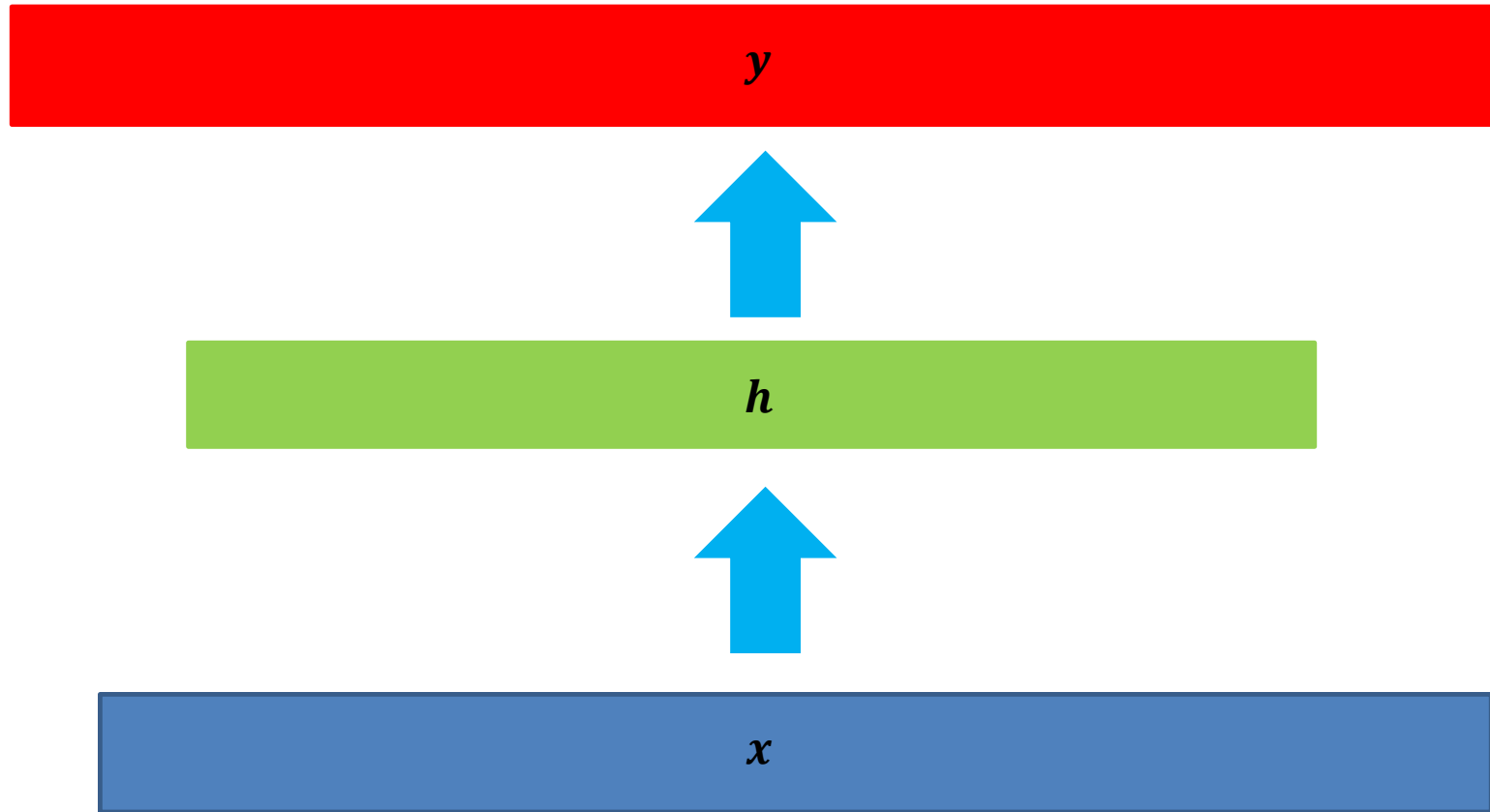 - Simple Remedies
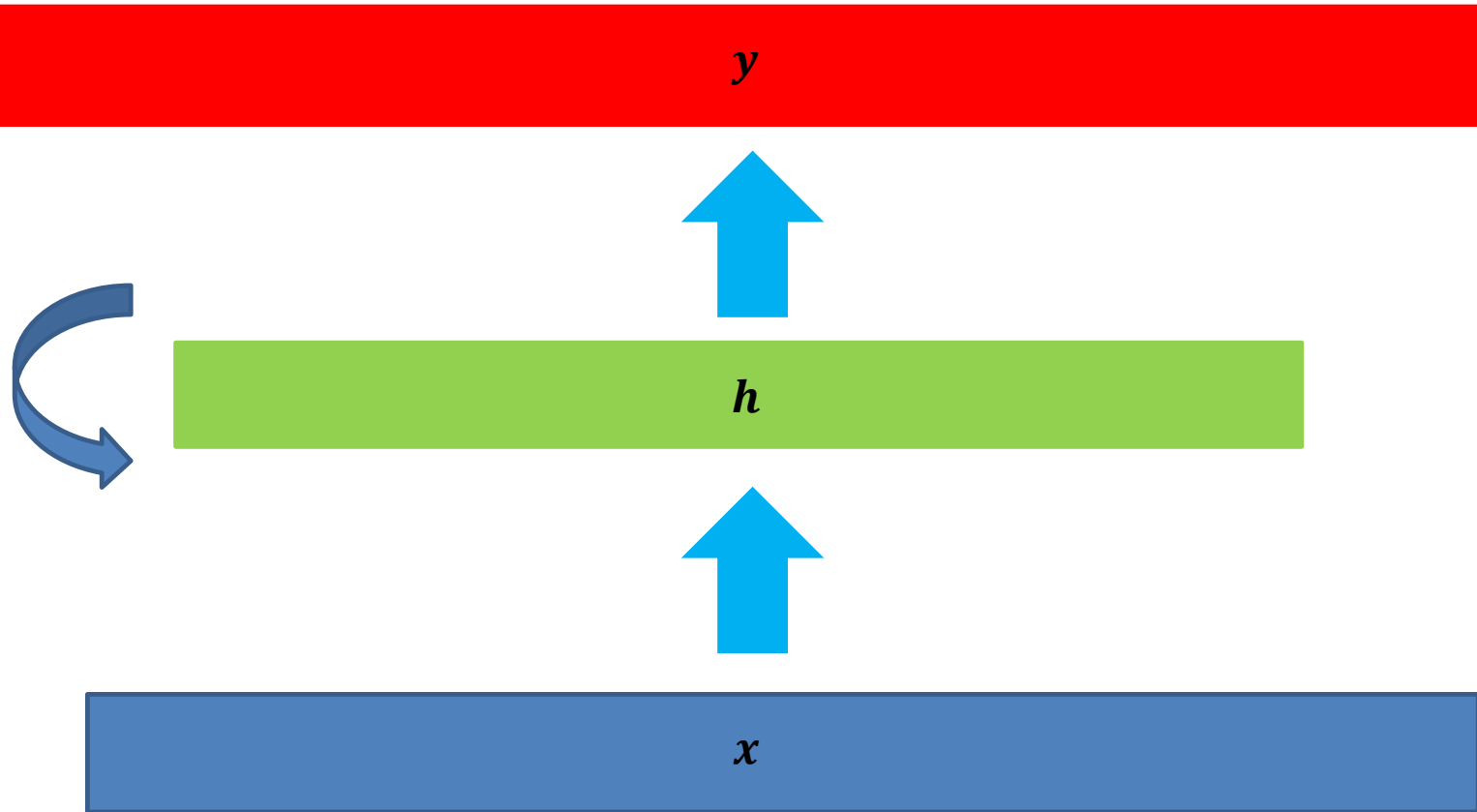
  - GRUs & LSTMs

NLLG

Recurrent Neural Nets
Basic principles

NLLG

Universität Bielefeld

$$x \qquad W^{(1)} \qquad W^{(2)} \qquad W^{(3)}$$

$$n \times h_1 \qquad h_1 \times h_2 \qquad h_2 \times m$$

NLLG

Feedforward / MLP

$$y$$

$$h$$

$$x$$

Recurrent

y

h

x

NLLG

Recurrent

$$y_t$$

$$h_t$$

$$x_t$$

Recurrent

$$y_t$$

Time delay

$$h_t$$

$$x_t$$

NLLG

Recurrent

$$y_t$$

$$V$$

"Memory"

$$W$$

Time delay

$$h_t$$

$$U$$

$$x_t$$

NLLG
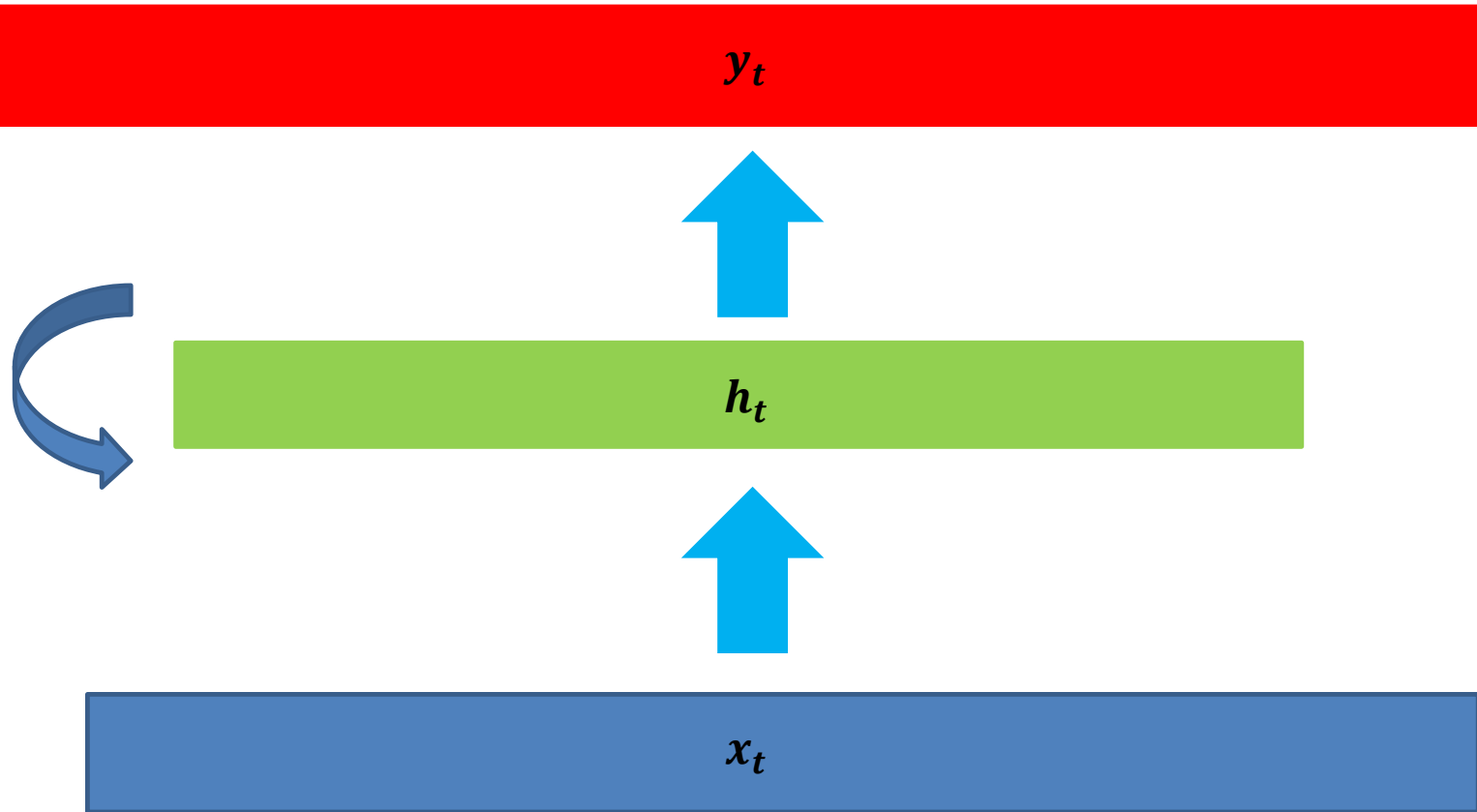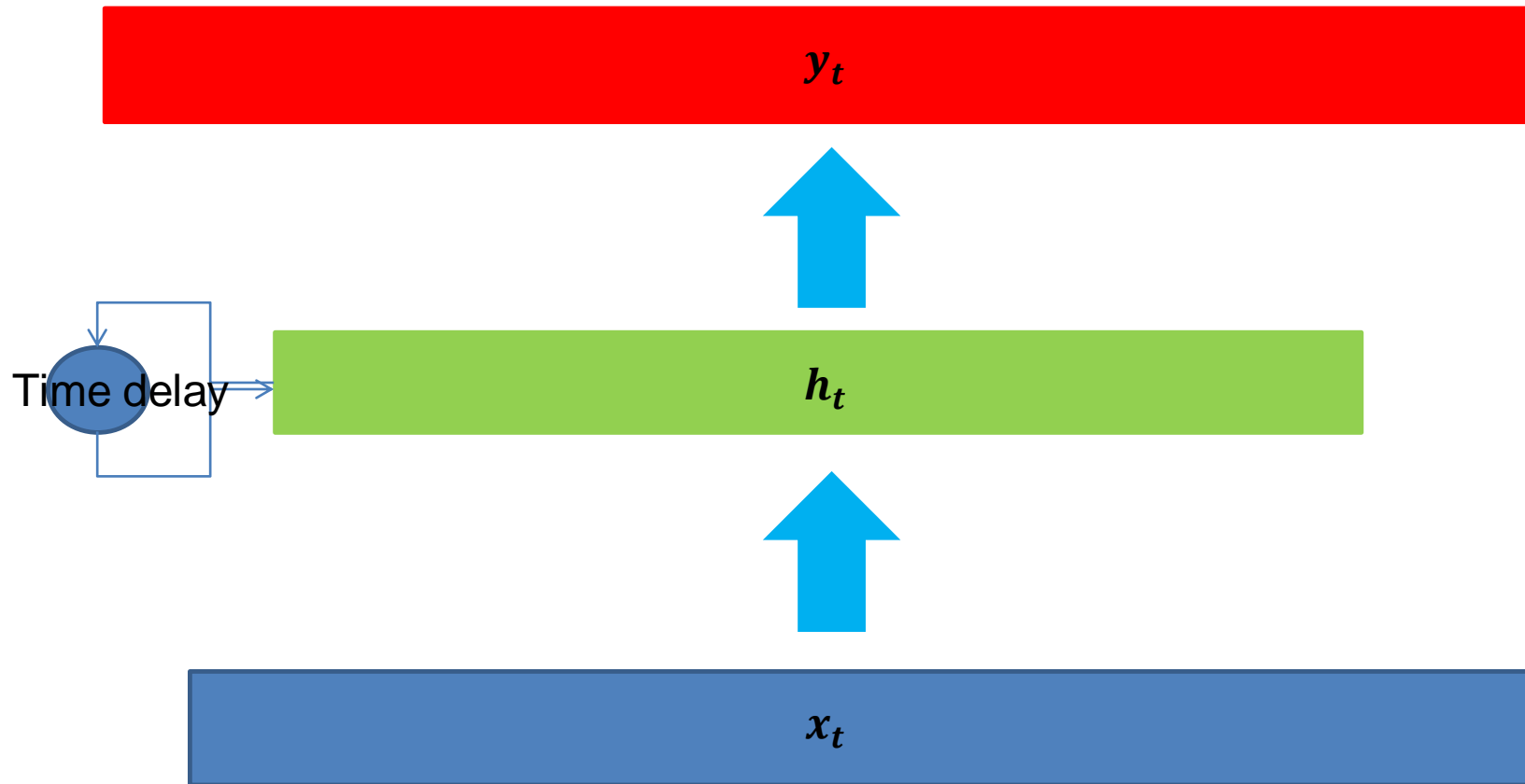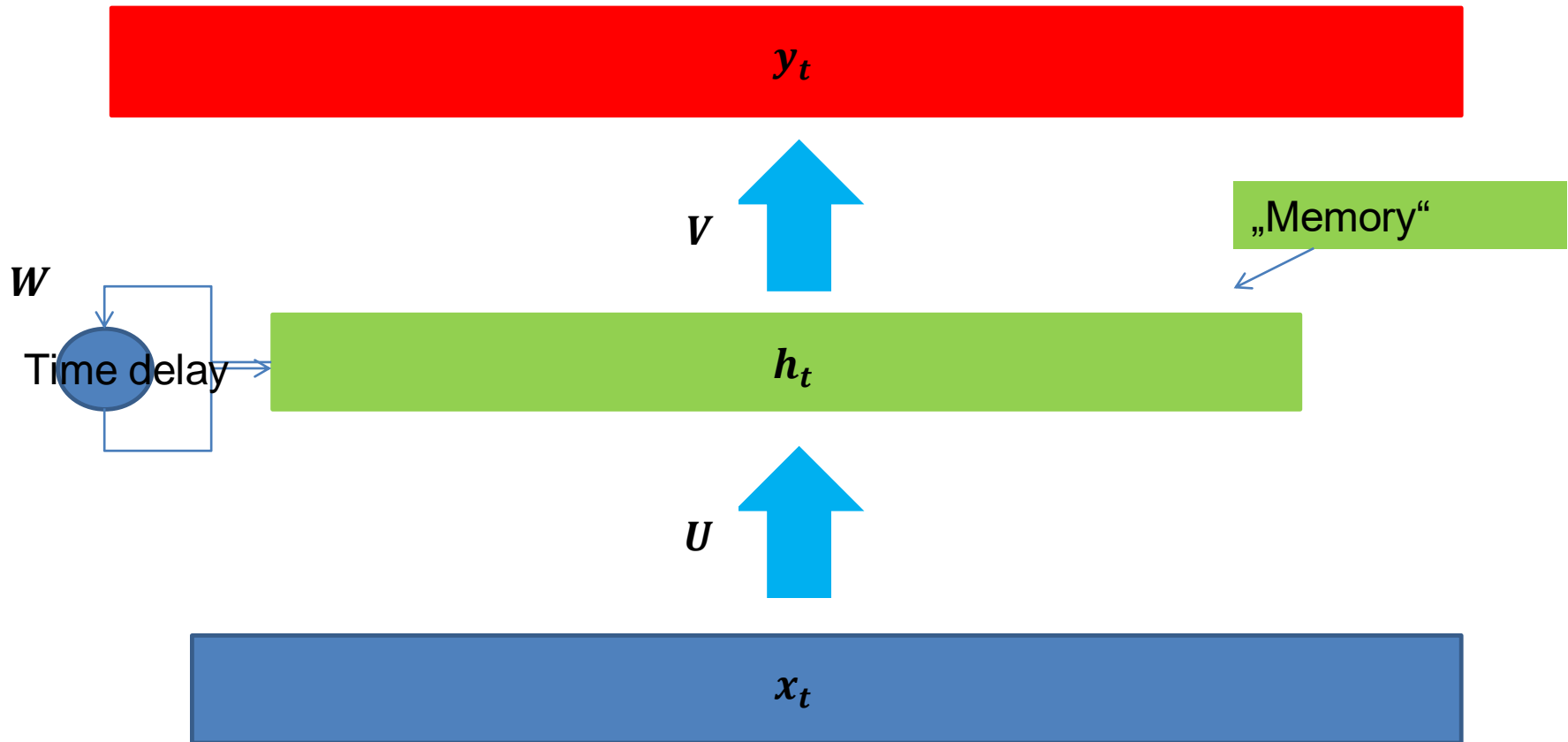
RNN – Formally

- Input vectors $\boldsymbol{x}_t$, $t = 1,2,3, \dots$ lie in $R^{1 \times n}$

- $\boldsymbol{h}_t = \sigma_H(\boldsymbol{x}_t \boldsymbol{U} + \boldsymbol{h}_{t-1} \boldsymbol{W} + \boldsymbol{b})$

  - Where $\boldsymbol{U} \in R^{n \times d}$, $\boldsymbol{W} \in R^{d \times d}$, $\boldsymbol{h}_t \in R^{1 \times d}$
    - d is hidden dimensionality

- $\boldsymbol{y}_t = \sigma_Y(\boldsymbol{h}_t \boldsymbol{V} + \boldsymbol{c})$
  - Where $\boldsymbol{V} \in R^{d \times m}$

NLLG

RNN – Formally

Universität Bielefeld

What we want to optimize is

- Average loss E over individual time losses E_t

    - E.g. $E_t = \text{ce}(\boldsymbol{y}_t, \boldsymbol{t}_t) = -\sum_j t_{t,j} \log y_{t,j}$
    - $E = \frac{1}{T} \sum_t E_t$

NLLG

RNN – Example

- Input: "A rusty can"
- Embeddings: $x_1 = (1,0,0), x_2 = (1,1,2), x_3 = (1,-1,1)$
- Truth: DET,ADJ,NOUN, encoded as 1-hot vectors (in a 4-d label space)
- Activations: ReLU for hidden layer, Softmax for output layer

NLLG

- Initialization:

  - $U = \begin{pmatrix} 1 & 1 \\ 2 & 0 \\ 0.5 & 1 \end{pmatrix}$

  - $W = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

  - $V = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & -1 \end{pmatrix}$
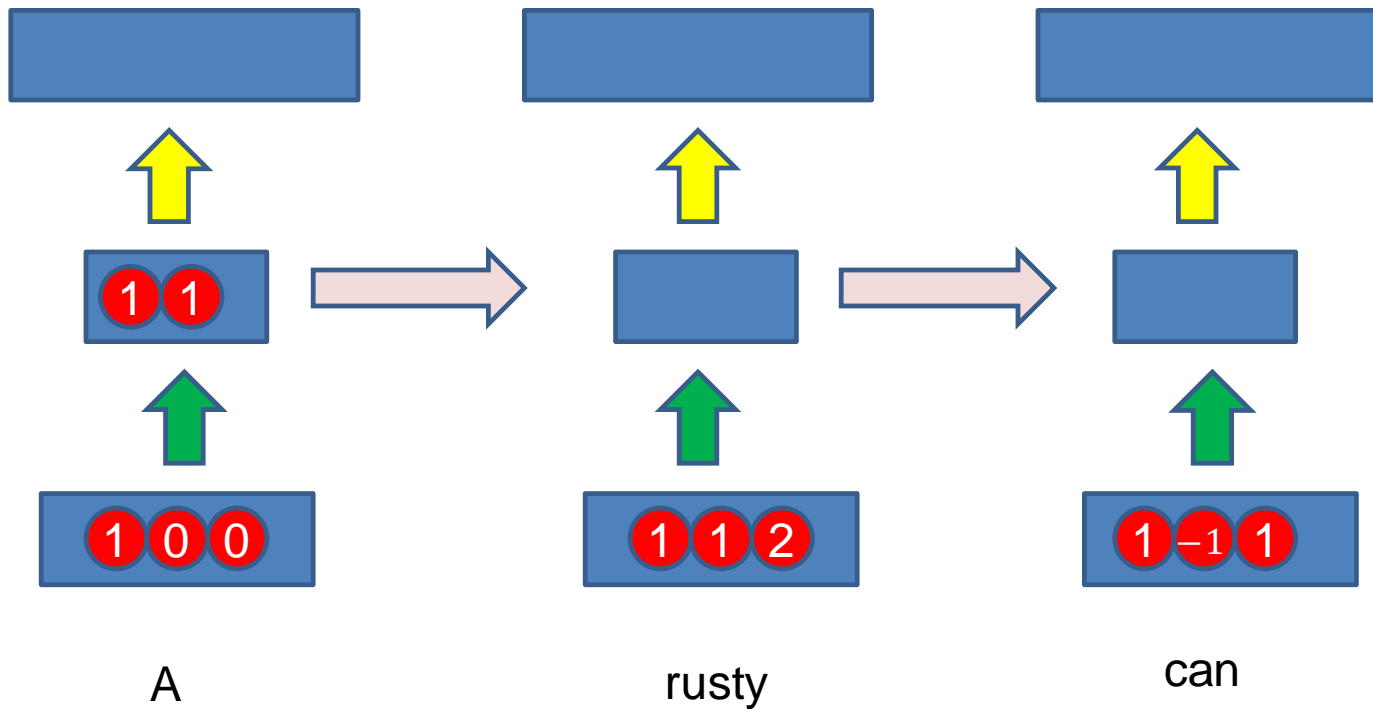
  - $b = c$ = zero-vectors of appropriate size
  - $h_0 = (0,0)$

RNN – Example

$$h_1 = \sigma_H(x_1 U + h_0 W + b)$$
$$h_1 = (1,1)$$

1 1		

| 1 0 0 | 1 1 2 | 1 -1 1 |

A   rusty   can

RNN – Example

$$h_2 = \sigma_H(x_2 U + h_1 W + b)$$
$$h_2 = (5,4)$$

1 1	5 4	

1 0 0	1 1 2	1 −1 1

A　　　　　　　rusty　　　　　　　can
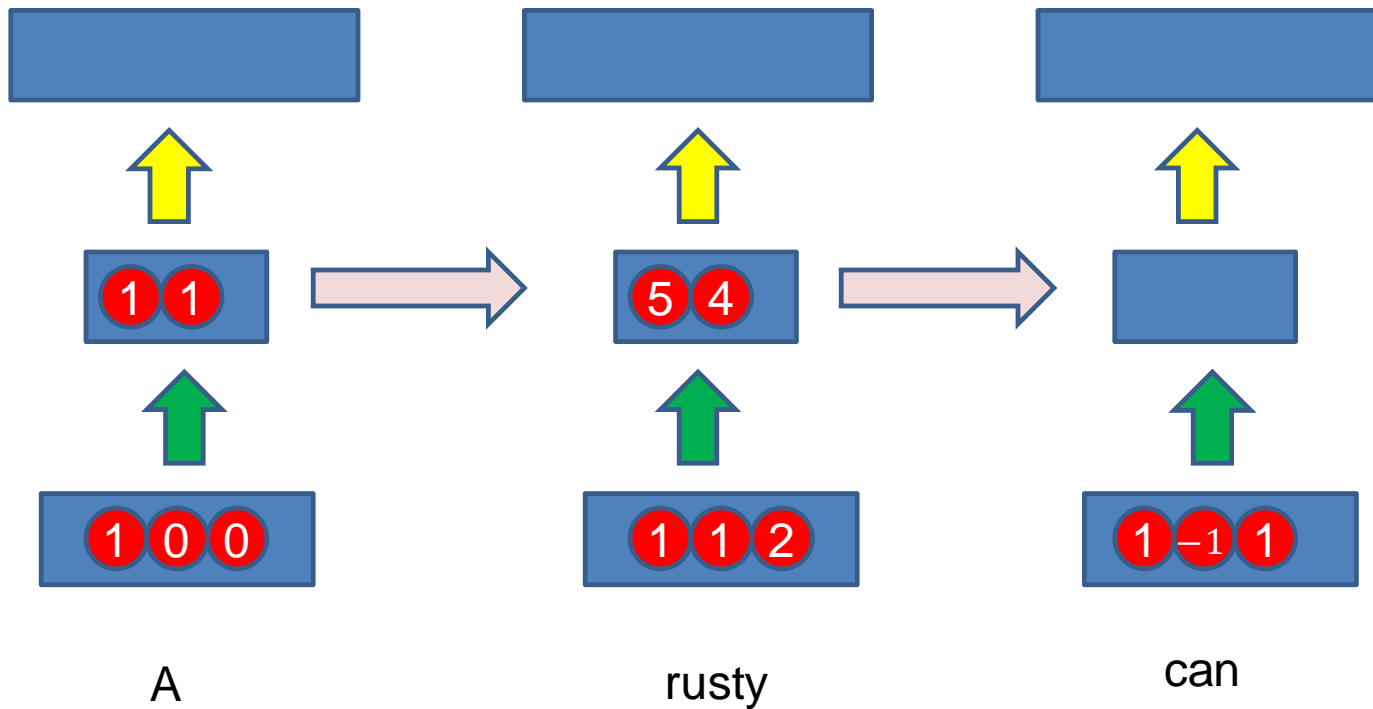
RNN – Example

$$h_3 = \sigma_H(x_3 U + h_2 W + b)$$
$$h_3 = (3.5, 7)$$
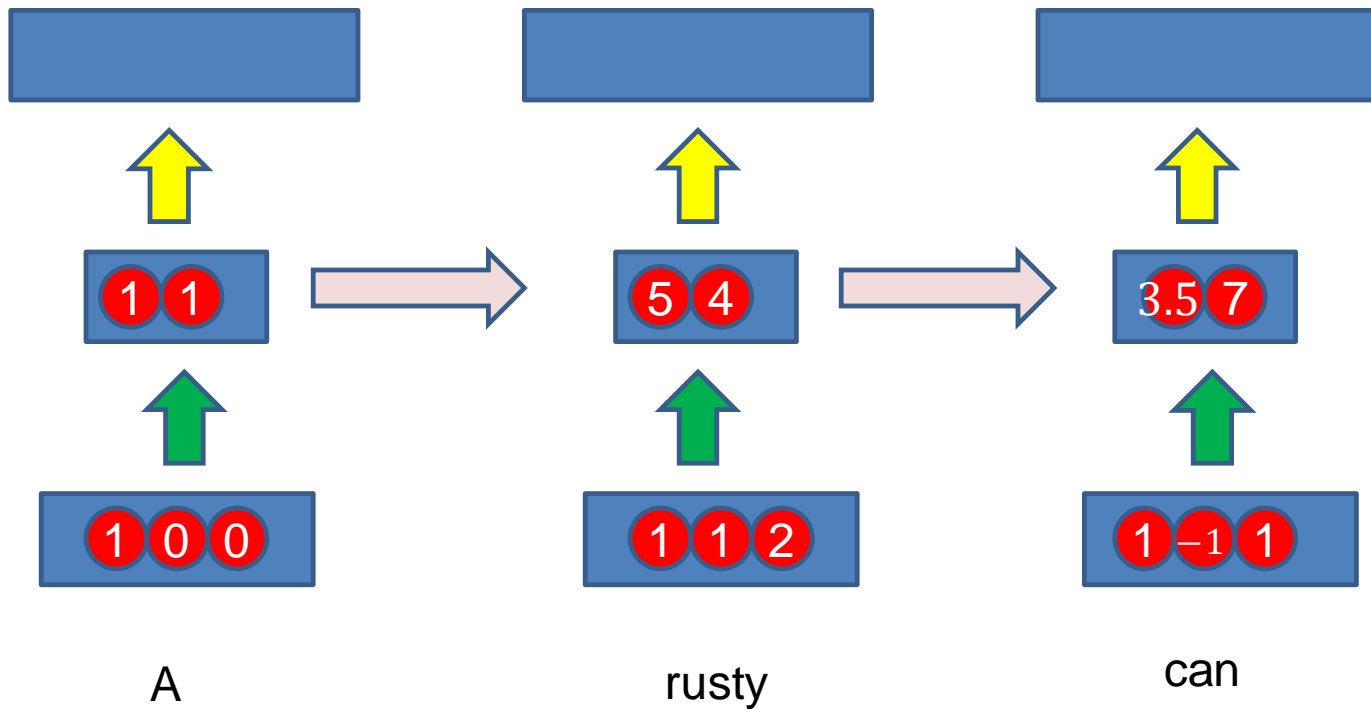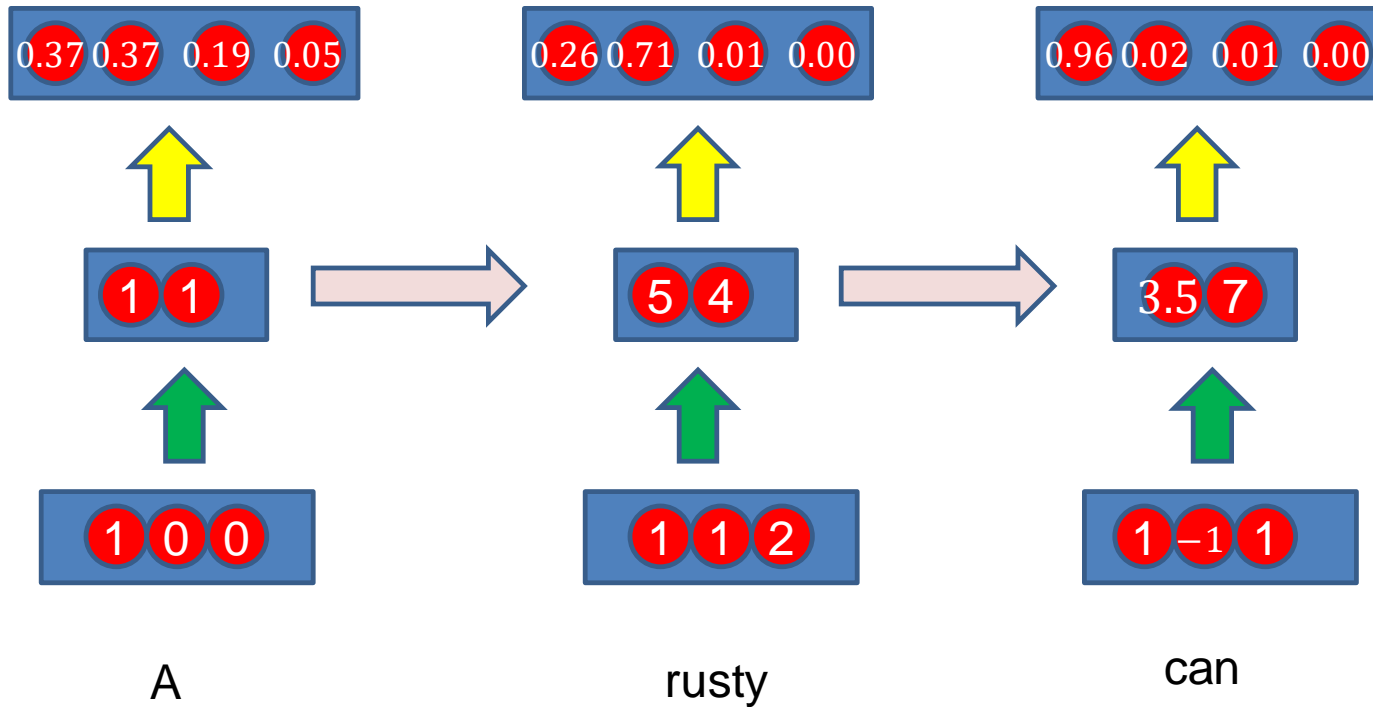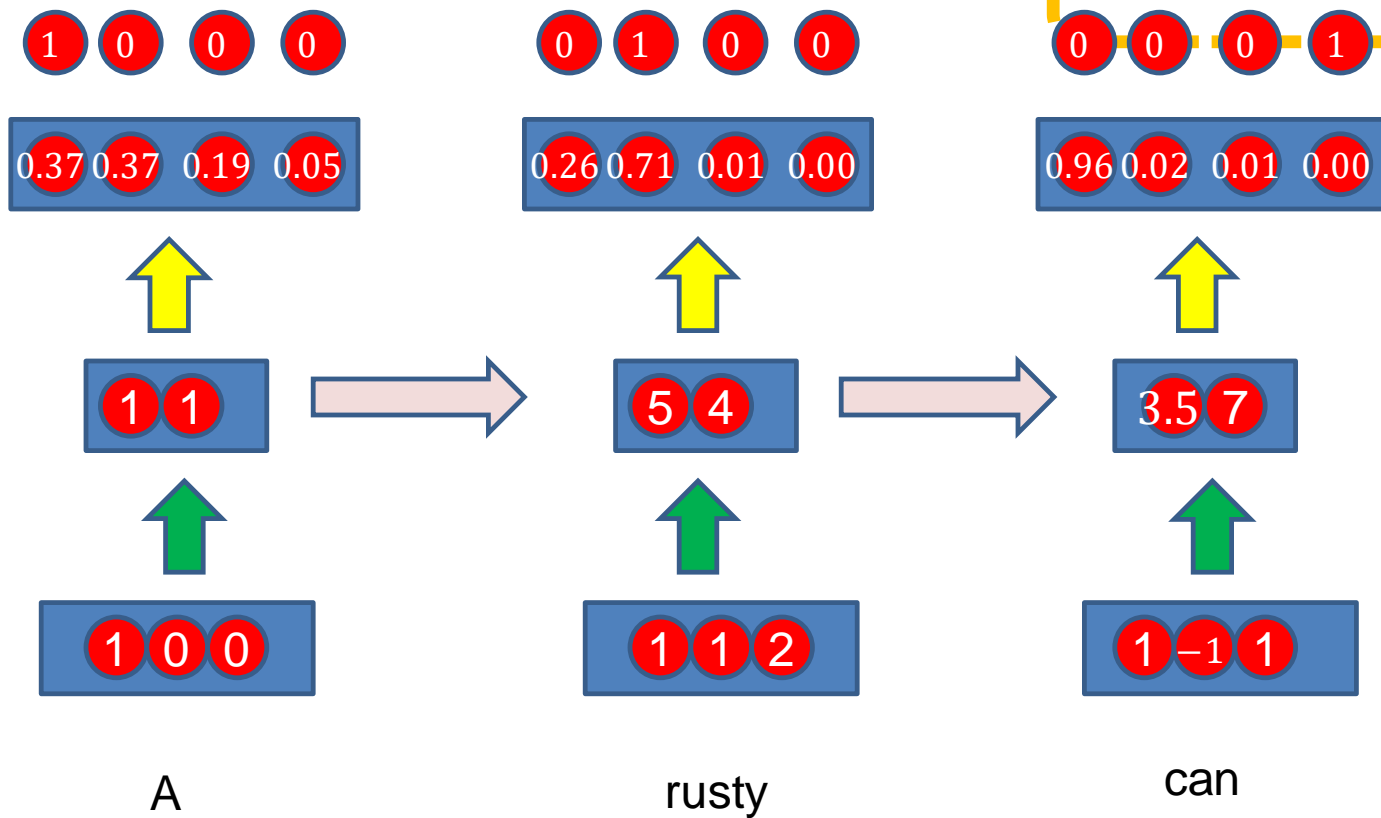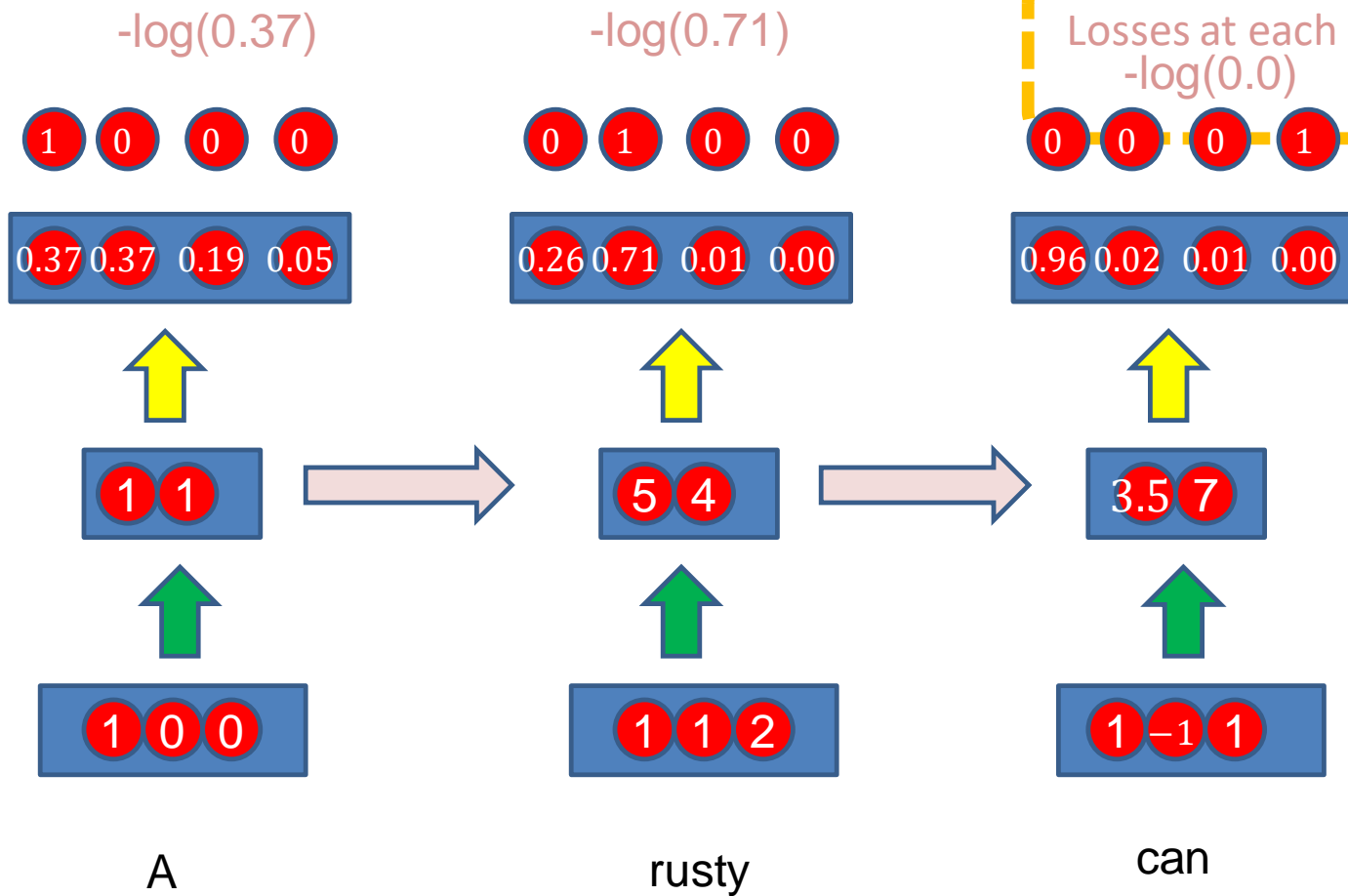
| 1 1 | | 5 4 | | 3.5 7 |

| 1 0 0 | 1 1 2 | 1 −1 1 |

A   rusty   can

NLLG

RNN – Example

$$y_t = \sigma_Y(h_t V + c)$$

| 0.37 | 0.37 | 0.19 | 0.05 |

| 0.26 | 0.71 | 0.01 | 0.00 |

| 0.96 | 0.02 | 0.01 | 0.00 |

| 1 | 1 | | 5 | 4 | | 3.5 | 7 |

| 1 | 0 | 0 | | 1 | 1 | 2 | | 1 | −1 | 1 |

A   rusty   can

NLLG

RNN – Example

-log(0.37) -log(0.71) Losses at each time step -log(0.0)

| 1 | 0 | 0 | 0 |

| 0 | 1 | 0 | 0 |

| 0 | 0 | 0 | 1 |

| 0.37 | 0.37 | 0.19 | 0.05 |

| 0.26 | 0.71 | 0.01 | 0.00 |

| 0.96 | 0.02 | 0.01 | 0.00 |

| 1 | 1 |

| 5 | 4 |

| 3.5 | 7 |

| 1 | 0 | 0 |

| 1 | 1 | 2 |

| 1 | -1 | 1 |

A

rusty

can

NLLG

RNN – Example

$1/3 \cdot (-\log(0.37) - \log(0.71) - \log(0.0))$

-log(0.37)   -log(0.71)   Total loss
                                        -log(0.0)

| 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 1 |

| 0.37 | 0.37 | 0.19 | 0.05 | | 0.26 | 0.71 | 0.01 | 0.00 | | 0.96 | 0.02 | 0.01 | 0.00 |

| 1 | 1 | | 5 | 4 | | 3.5 | 7 |

| 1 | 0 | 0 | | 1 | 1 | 2 | | 1 | –1 | 1 |

A   rusty   can

NLLG
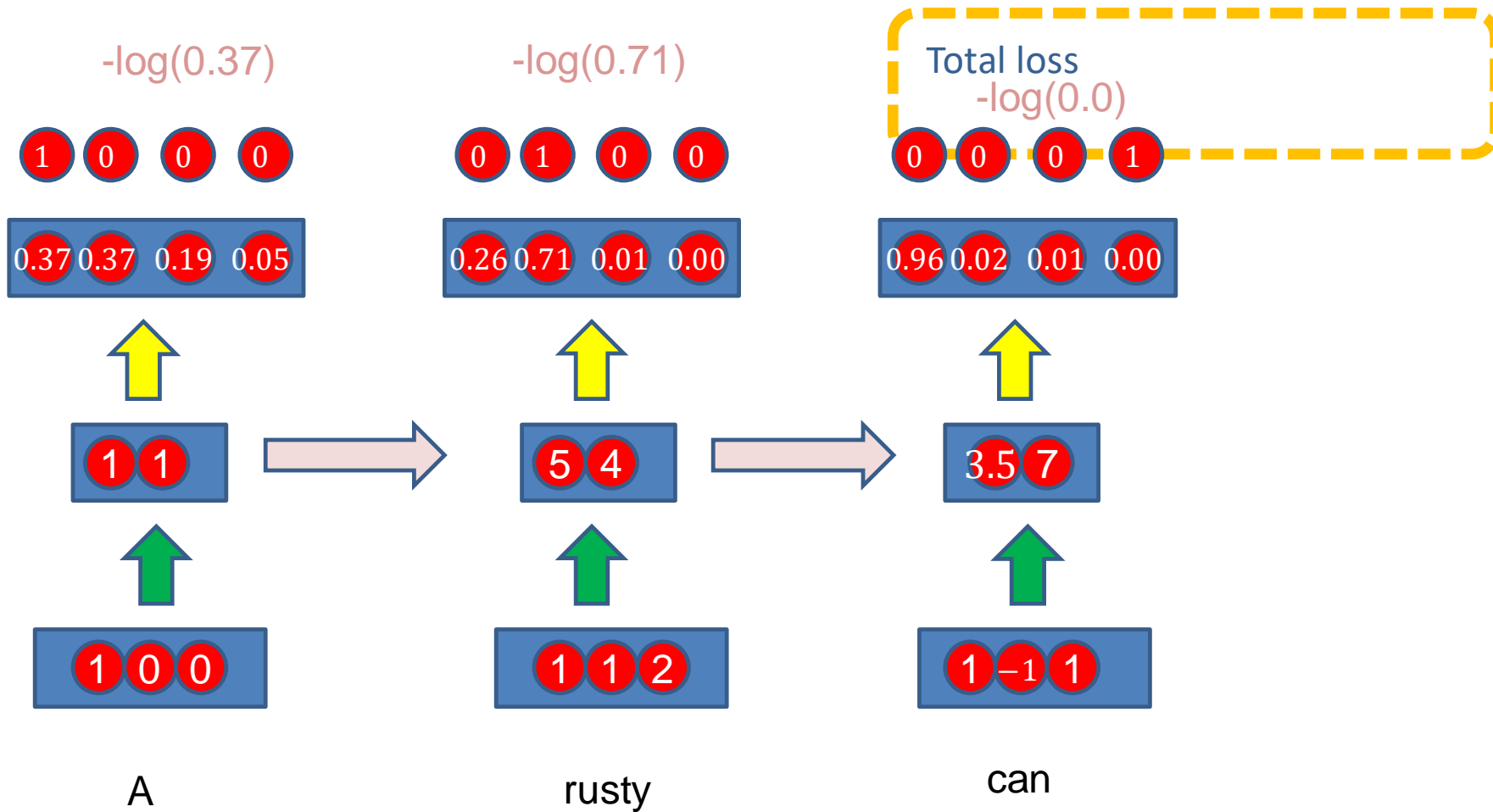
RNN – Example

$$1/3 \cdot (-\log(0.37) - \log(0.71) - \log(0.0))$$
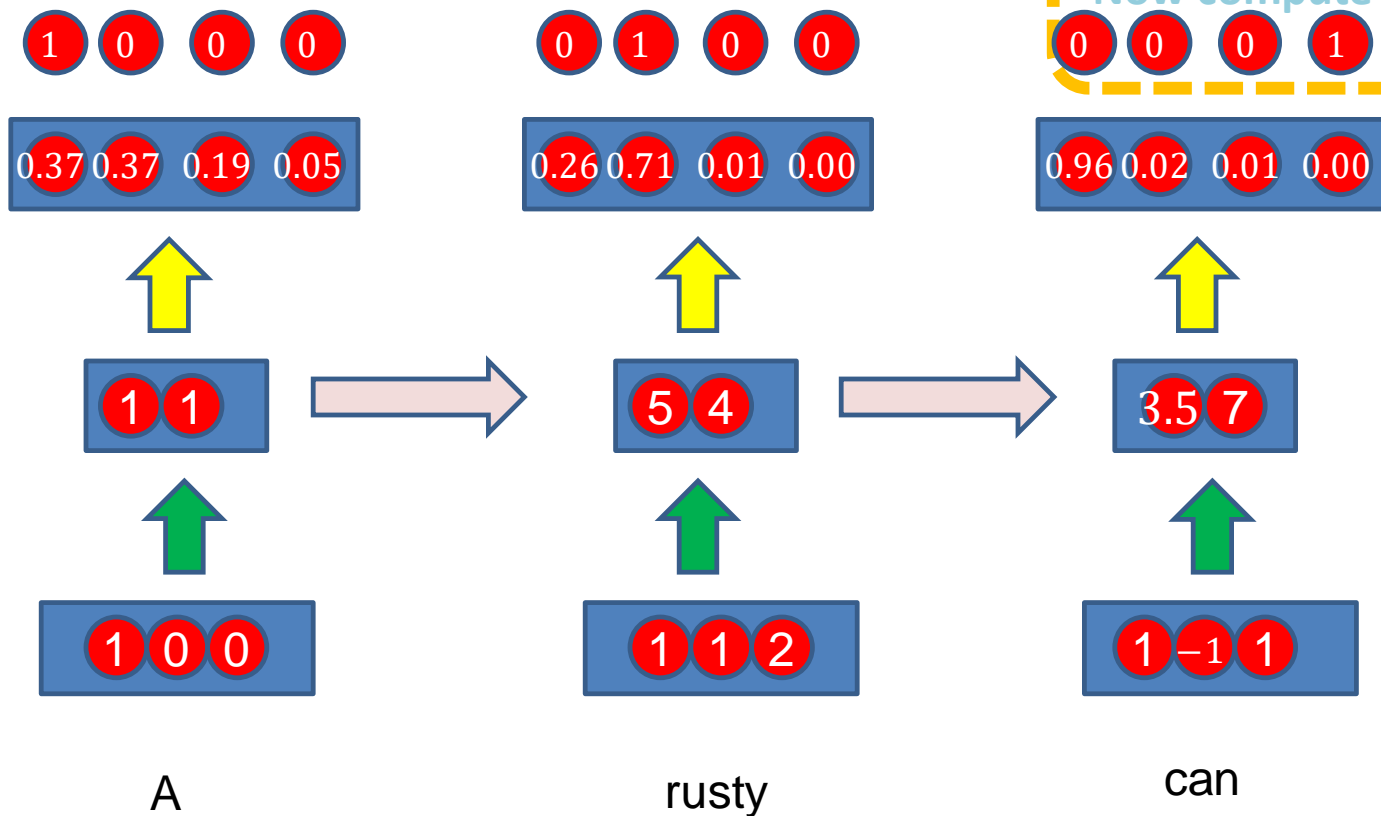
This was the forward pass
Now compute gradients+update weights

| 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 1 |

| 0.37 | 0.37 | 0.19 | 0.05 | | 0.26 | 0.71 | 0.01 | 0.00 | | 0.96 | 0.02 | 0.01 | 0.00 |

| 1 | 1 | | 5 | 4 | | 3.5 | 7 |

| 1 | 0 | 0 | | 1 | 1 | 2 | | 1 | −1 | 1 |

A   rusty   can

22

RNN – Weight update / Gradient computation

Universität Bielefeld

- Computation of gradient is similar as in standard MLP
  - But: Need to keep in mind that several parameters are shared
  - Some people call backprop for RNNs "backpropagation through time" (BPTT)
  - No need to go through, TensorFlow does it for you

- If you want to do it brute-force, can also do it numerically
  - i.e., for each individual weight w, compute $\frac{f(w+h)-f(w)}{h}$
  - Where f is the loss function

- Weight update after gradient computation is $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \, \nabla f$ as usual

NLLG

RNN – Properties

- Infinite window size – "from the left"

  - Memory can (in principle) store everything from the past

- That's good, but we also want to base our decision on future words/tokens

  → Bidirectional RNN:

  - run a second RNN from "right to left"
  - With independent weights
  - Concatenate the forward and backward hidden states
    - $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$
    - Note that V is of dimension $2d \times m$ in this case

NLLG

Recurrent Neural Nets Extensions

NLLG

Bidirectional RNN – Illustration

A   rusty   can

Extensions of simple RNNs
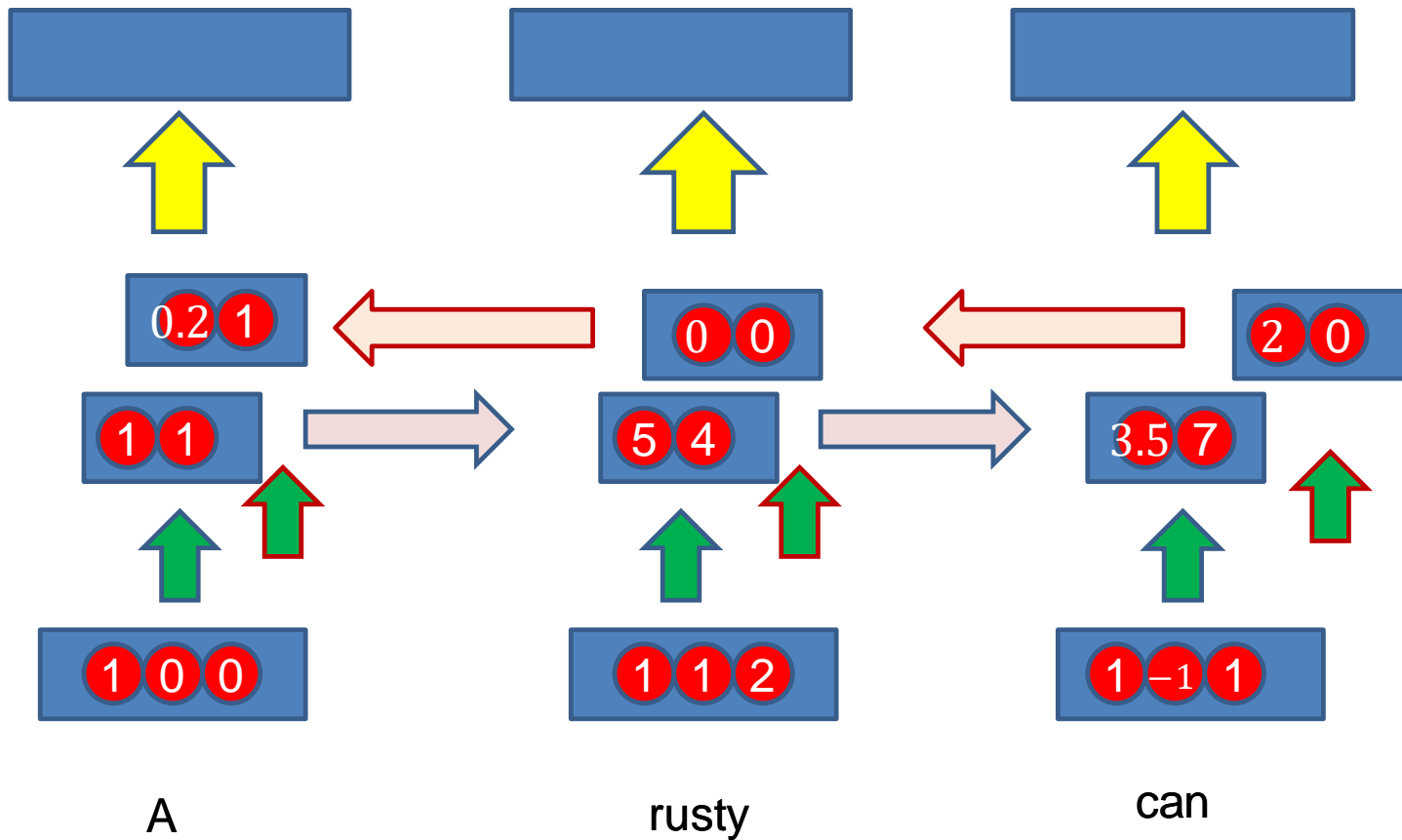
- **Bidirectional RNNs**

  Problem: For classification you want to incorporate information from words both preceding and following
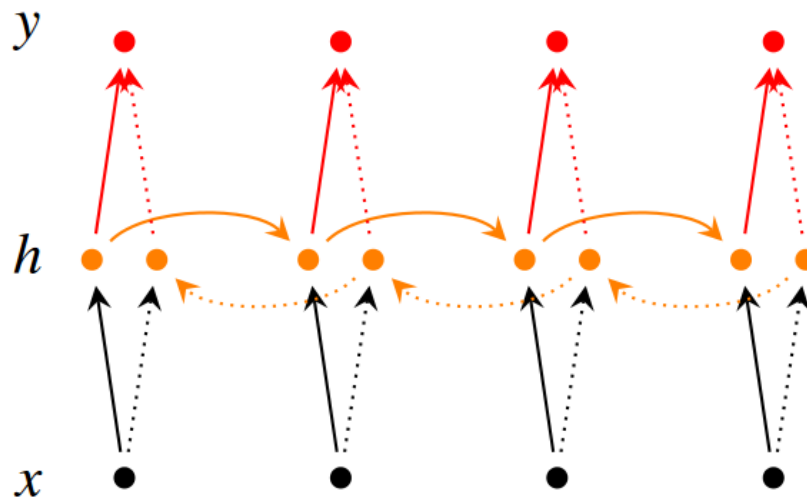
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$
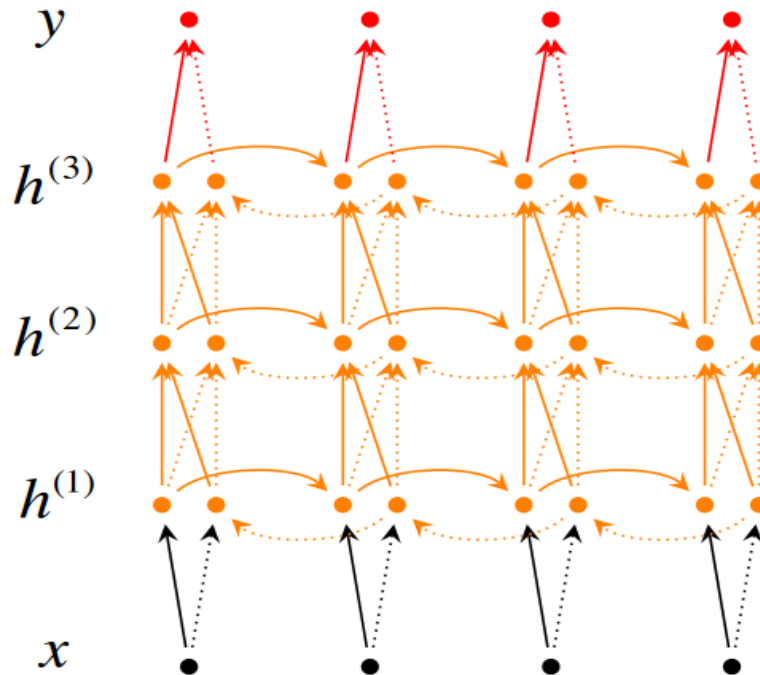
$$y_t = g(U[\vec{h}_t;\overleftarrow{h}_t] + c)$$

$h = [\vec{h};\overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

From: https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Extensions of simple RNNs

- **Deep Bidirectional RNNs**

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$
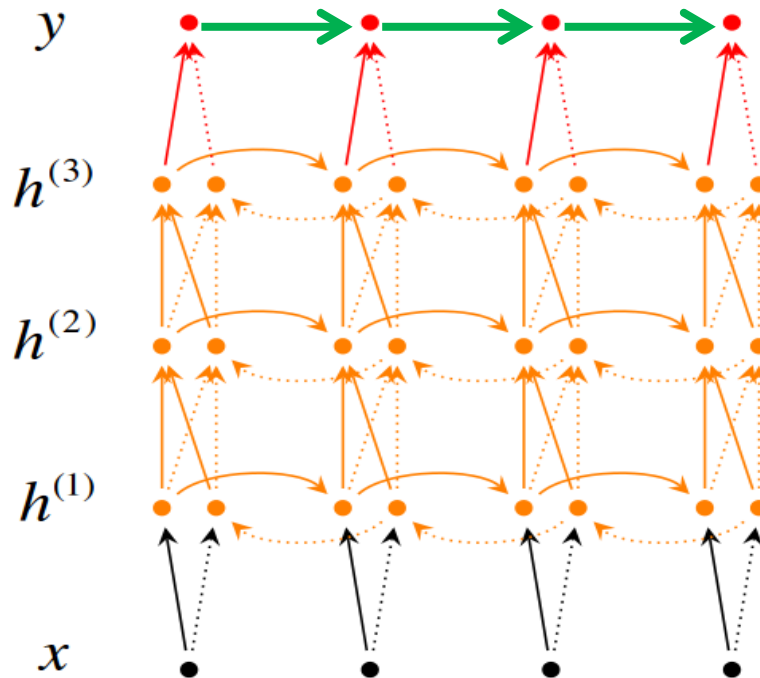
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)} ; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate sequential representation to the next.

From: https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

NLLG

Extensions of simple RNNs

- **RNNs with output connections**

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$
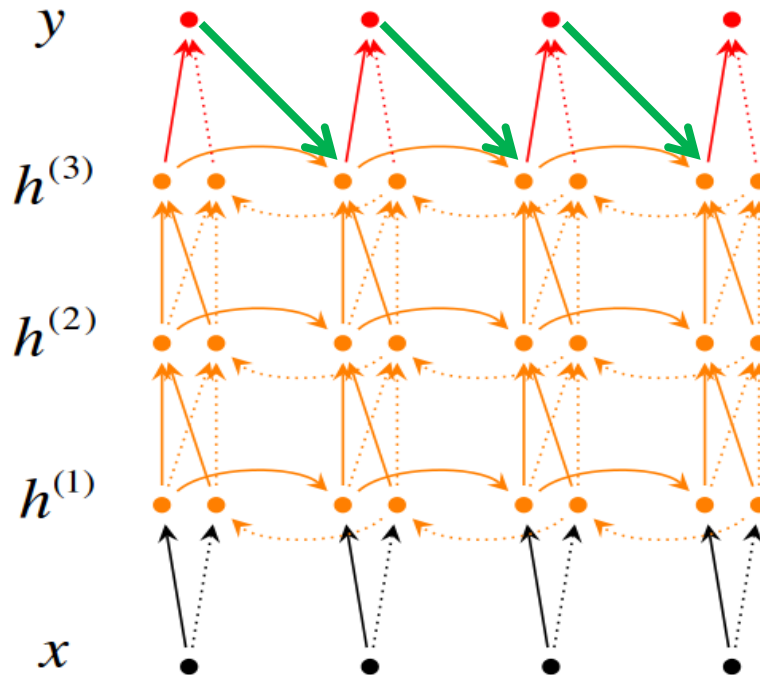
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Equations?

Each memory layer passes an intermediate sequential representation to the next.

From: https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Extensions of simple RNNs

- **RNNs with output connections**

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$
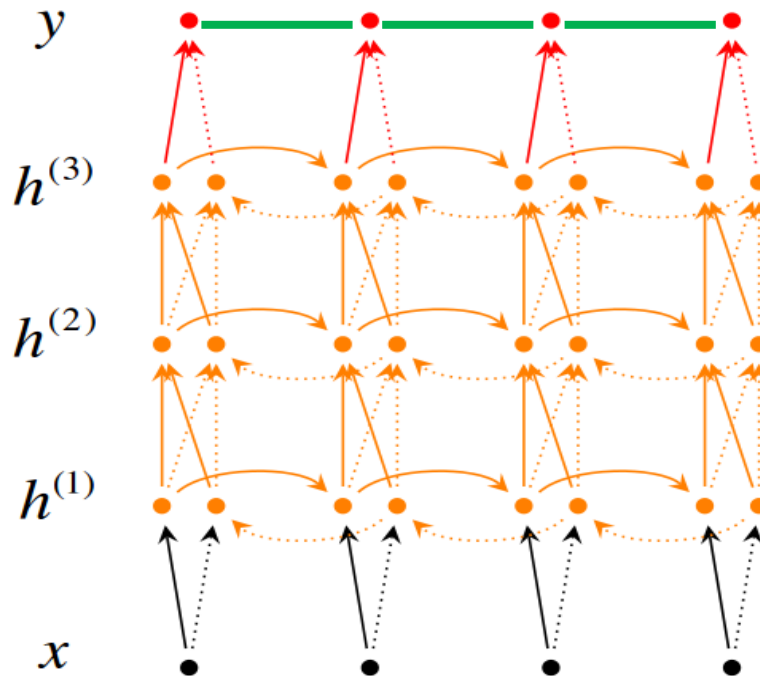
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Equations?

Each memory layer passes an intermediate sequential representation to the next.

From: https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Extensions of simple RNNs

- **RNNs with output connections: CRF instead of forward conn.**

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$
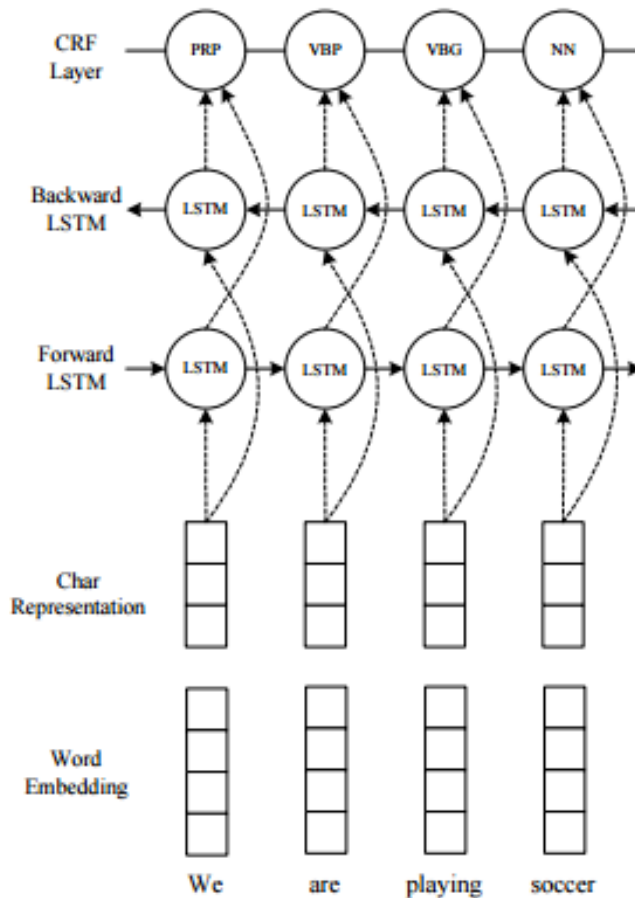
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Equations?

Each memory layer passes an intermediate sequential representation to the next.

Extensions of simple RNNs

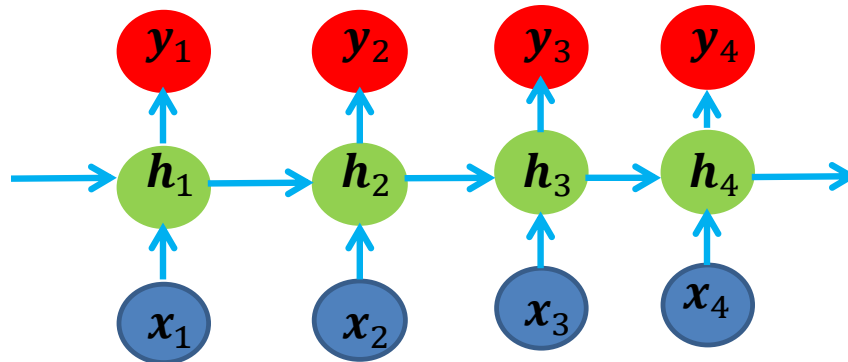- **RNNs with output connections and character information**

Why character information?
Ma and Hovy (2016)
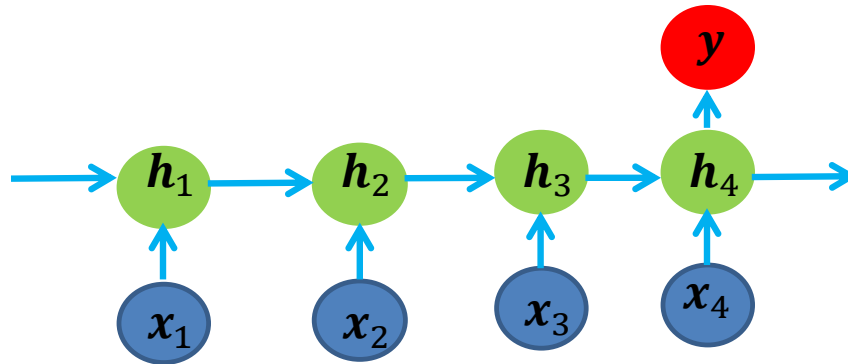Lample et al. (2016)

Recurrent Neural Nets
For sequence tagging & for classification

RNNs for sequence tagging (aka sequence labeling)

RNNs for sentence classification

Code for RNNs

- Many implementations out there

- Nils Reimers has a nice Keras implementation
  - See: https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf
  - He also has one for ELMo embeddings

- We also have a TensorFlow implementation (using Multi-Task Learning, etc.)
  - See: https://github.com/UKPLab/thesis2018-tk_mtl_sequence_tagging

Recurrent Neural Nets
NLP applications

- RNNs are „natural" forms for sequence labeling tasks

  - POS tagging

- RNNs are „natural" forms for sequence labeling tasks

  - POS tagging

x	We	love	cold	beer
y	PRON	V	ADJ	Noun

Label space = y = {PRON,V,DET,ADVERB,…} encoded as 1-hot vectors
Input space = x = natural language words = {I,you,he,she,run,…} encoded as embeddings

- RNNs are „natural" forms for sequence labeling tasks

  - NER

x	Angela	Merkel	loves	New	York
y	B-PER	I-PER	O	B-LOC	I-LOC

Label space = y = {B-PER,I-PER,O,B-LOC,I-LOC,....} encoded as 1-hot vectors
Input space = x = natural language words = {I,you,he,she,run,…} encoded as embeddings

- **RNNs are „natural" forms for sequence labeling tasks**

  - **Grapheme-to-Phoneme Conversion (s c h u h $\rightarrow$ S U:)**

x / y	s	c	h	u	h
x	s	c	h	u	h
y	S	$\emptyset$	$\emptyset$	U:	$\emptyset$

Label space = y = {S,a,a:,$\emptyset$,…} encoded as 1-hot vectors
Input space = x = chars = {a,b,c,…} encoded as char embeddings or 1-hot

- RNNs are „natural" forms for sequence labeling tasks

  - Lemmatization (g e l i e b t → l i e b e n)

x y	g	e	l	i	e	b	t
	Ø	Ø	l	i	e	b	en

Label space = y = {a,b,c,st,en,…}+{Ø} encoded as 1-hot vectors
Input space = x = chars = {a,b,c,…} encoded as char embeddings or 1-hot

- RNNs are „natural" forms for sequence labeling tasks

  - Language Modeling

x y	<SOS>	Here	comes	a	new	year
	Here	comes	a	new	year	<EOS>

Label space = y = {words}+padding encoded as 1-hot vectors
Input space = x = {words}+padding encoded as embeddings

NLLG

Vanishing Gradients
Introduction

- (following mostly the lecture slides of Richard Socher, https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf
- See also de Freitas' video: https://www.youtube.com/watch?v=56TYLaQN4N8

NLLG

The chain rule

- Newton notation: $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$

- Leibniz notation: $\dfrac{df}{dx} = \dfrac{df}{dy} \cdot \dfrac{dy}{dx}$

- In higher dimensions, multiplication becomes scalar product or matrix multiplication

  - And $\dfrac{\partial f}{\partial \boldsymbol{x}}$ is a vector (=gradient) when $\boldsymbol{x}$ is a vector:

    - $\dfrac{\partial f}{\partial \boldsymbol{x}} = (\dfrac{\partial f}{\partial x_1}, \dots, \dfrac{\partial f}{\partial x_n})$

  - And $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is a matrix (=Jacobian) when $\mathbf{y}, \boldsymbol{x}$ are vectors

    - $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = (\dfrac{\partial y_1}{\partial \boldsymbol{x}}, \dots, \dfrac{\partial y_m}{\partial \boldsymbol{x}})$

      vector

NLLG

Back to RNNs

- RNN formulation:
  - $h_t = \sigma_H(x_t U + h_{t-1} W)$
  - $y_t = \sigma_Y(h_t V)$

- Interested in:

$$\frac{\partial h_t}{\partial h_k}$$

NLLG

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{h}_{t-2}} \dots \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k}$$

Each $\frac{\partial \boldsymbol{h}_s}{\partial \boldsymbol{h}_{s-1}}$ is a matrix (called Jacobian)

- Remember:
  - $\boldsymbol{h}_t = \sigma_H(\boldsymbol{x}_t \boldsymbol{U} + \boldsymbol{h}_{t-1} \boldsymbol{W})$

- Hence,
  - $\frac{\partial \boldsymbol{h}_s}{\partial \boldsymbol{h}_{s-1}} = \mathrm{diag}\big(\sigma_H'(\boldsymbol{x}_s \boldsymbol{U} + \boldsymbol{h}_{s-1} \boldsymbol{W})\big) \cdot \boldsymbol{W}$

NLLG

- Definition for diag:

  - Let $\boldsymbol{z} = \boldsymbol{x}_s \boldsymbol{U} + \boldsymbol{h}_{s-1} \boldsymbol{W}$

  - $\text{diag}([z_1 \cdots z_n]) = \begin{bmatrix} z_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & z_n \end{bmatrix}$

NLLG

- Analyzing the norms of the Jacobians yields:

$$\left|\left|\frac{\partial \boldsymbol{h}_s}{\partial \boldsymbol{h}_{s-1}}\right|\right| = \left|\left|\text{diag}\big(\sigma_H'(\boldsymbol{z})\big) \cdot \boldsymbol{W}\right|\right|$$

$$\leq \left|\left|\text{diag}\big(\sigma_H'(\boldsymbol{z})\big)\right|\right| \cdot \left|\left|\boldsymbol{W}\right|\right|$$

- Assume β_H is an upper bound for the norm of diag and β_W is an upper bound for the norm of $\boldsymbol{W}$

- Similarly, assume that the norm of $\boldsymbol{Q} = \text{diag}\big(\sigma_H'(\boldsymbol{z})\big) \cdot \boldsymbol{W}$ is bounded from below by α

NLLG

- Then:

$$\alpha \leq \left|\left|\frac{\partial \boldsymbol{h}_s}{\partial \boldsymbol{h}_{s-1}}\right|\right| \leq \left|\left|\mathrm{diag}(\sigma_H'(\boldsymbol{z}))\right|\right| \cdot ||\boldsymbol{W}|| \leq \beta_H \beta_W$$

NLLG

- Thus

$$\alpha^{t-k} \leq \left\|\left\|\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k}\right\|\right\| = \left\|\left\|\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{h}_{t-2}} \cdots \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k}\right\|\right\| \leq (\beta_H \beta_W)^{(t-k)}$$

- This can become very large **(exploding gradients)** or very small (**vanishing gradients**) quickly (Bengio et al. 1994)

  - If very large:

    - $\frac{\partial E_t}{\partial \boldsymbol{W}} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{W}}$ explodes

  - If very small:

    - $\boldsymbol{h}_k$ (and all that goes into it) has no effect on $\boldsymbol{h}_t$

NLLG

- Vanishing gradient problem for language models/sequence labeling, etc.

    - Time steps far away are not taken into consideration

- „Jane walked into the room. John walked in too. It was late in the day. John said hi to XX"

- „B e r l i n _ (_ t h e _ v e r y _ b e a u t i f u l _ …. _ c a p i t a l _ o f _ XX"

- A note on the term β_H:

  - $\left|\left|\mathrm{diag}\big(\sigma_H'(\boldsymbol{x}_s\boldsymbol{U} + \boldsymbol{h}_{s-1}\boldsymbol{W})\big)\right|\right| \leq \beta_H$

Rule of thumb:
$\sigma' > 1$ exploding gradient
$\sigma' < 1$ vanishing gradient
$\sigma' = 1$ good region

Some Common Activation Functions
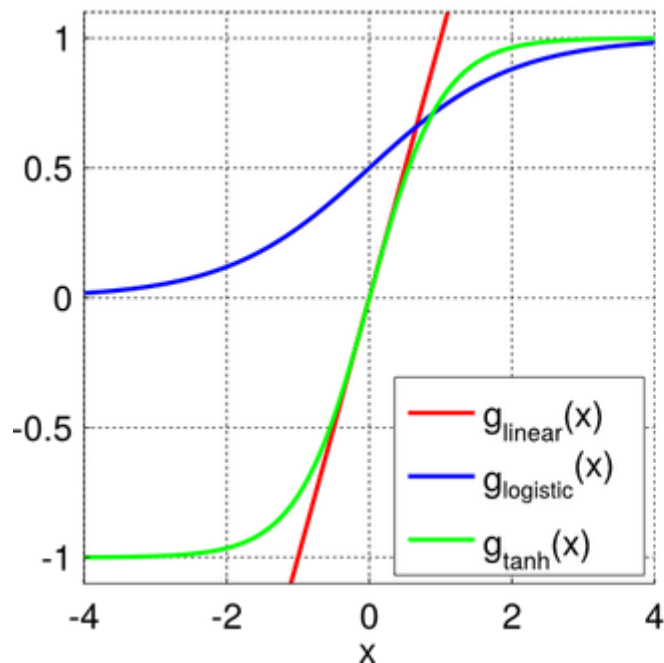
Activation Function Derivatives

NLLG

Vanishing gradients

Simple Remedies

NLLG
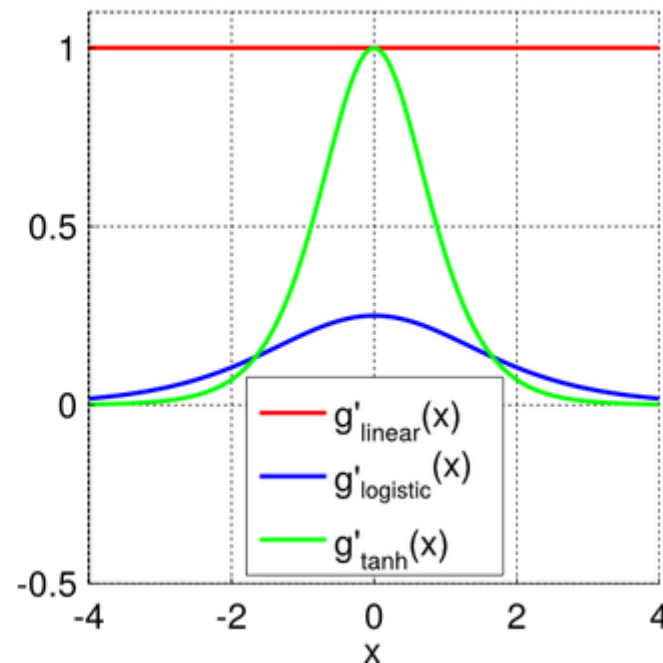
Regularization & Norm clipping

- **Exploding gradients**:

    - L1 or L2 regularization on recurrent weights $\rightarrow$ keeps $\boldsymbol{W}$ small

    - Gradient clipping (first introduced by Mikolov)
        - If error derivative $\frac{\partial E}{\partial w_{ik}}$ is too large, set it to some fixed constant

NLLG

Norm clipping
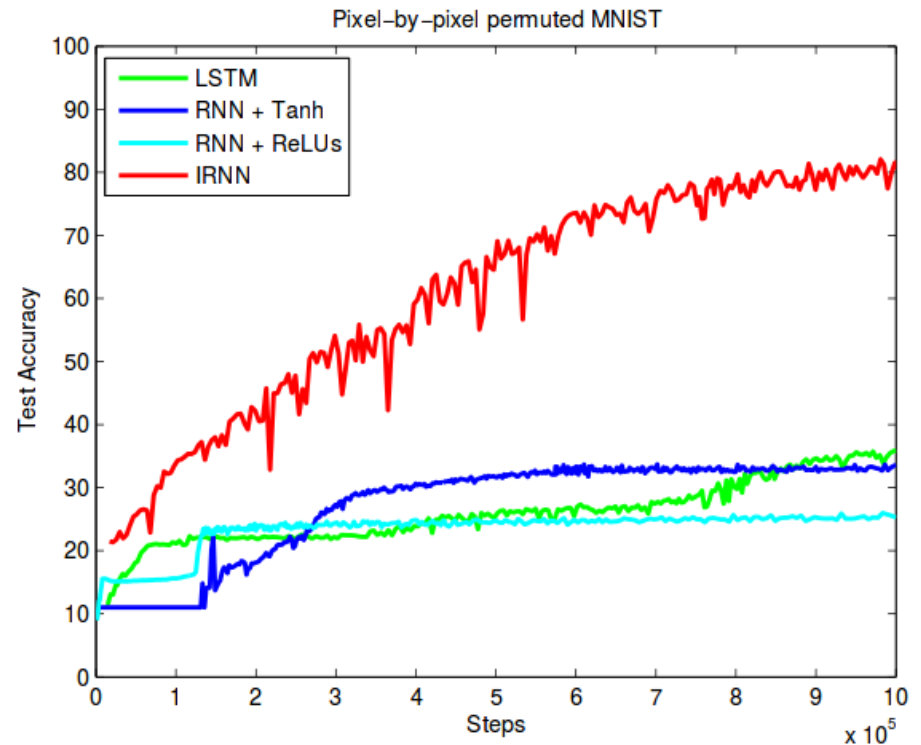
- Gradient clipping intuition:
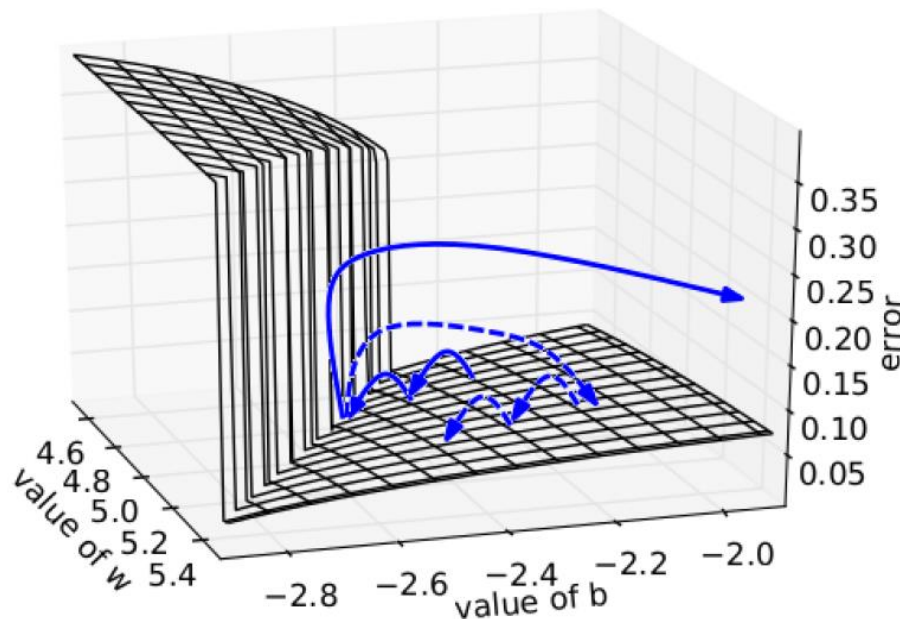
From: On the difficulty of training RNNs, Pascanu et al. 2013

- Solid lines: standard gradient descent trajectories
- Dashed lines: gradients rescaled to fixed size

IRNNs

- **Vanishing (/exploding) gradients**
  - ReLU and initialization, Le et al., 2015
    - Initialize $\boldsymbol{W}$'s to identity matrix $\boldsymbol{I}$
    - $\sigma_H(z) = \max(0, z)$

IRNNs

- **Vanishing (/exploding) gradients**
  - ReLU and initialization, Le et al., 2015
    - Initialize W's to identity matrix I
    - $\sigma_H(z) = \max(0, z)$
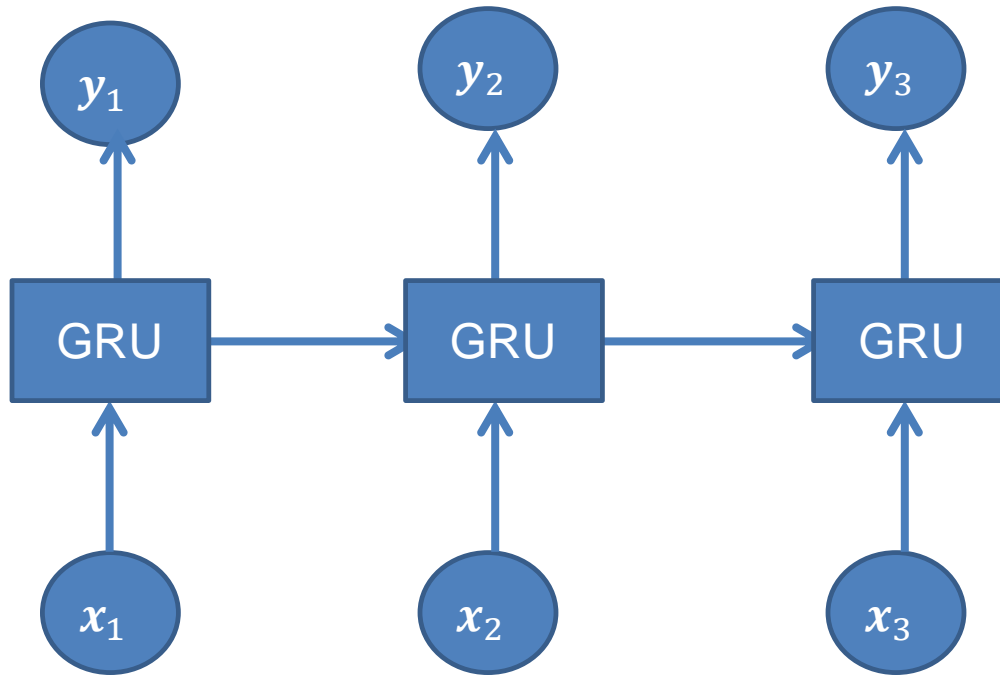   - They call this IRNNs
      (I = identity matrix)

Pixel−by−pixel permuted MNIST

LSTM
RNN + Tanh
RNN + ReLUs
IRNN

Test Accuracy

Steps
x 10^5
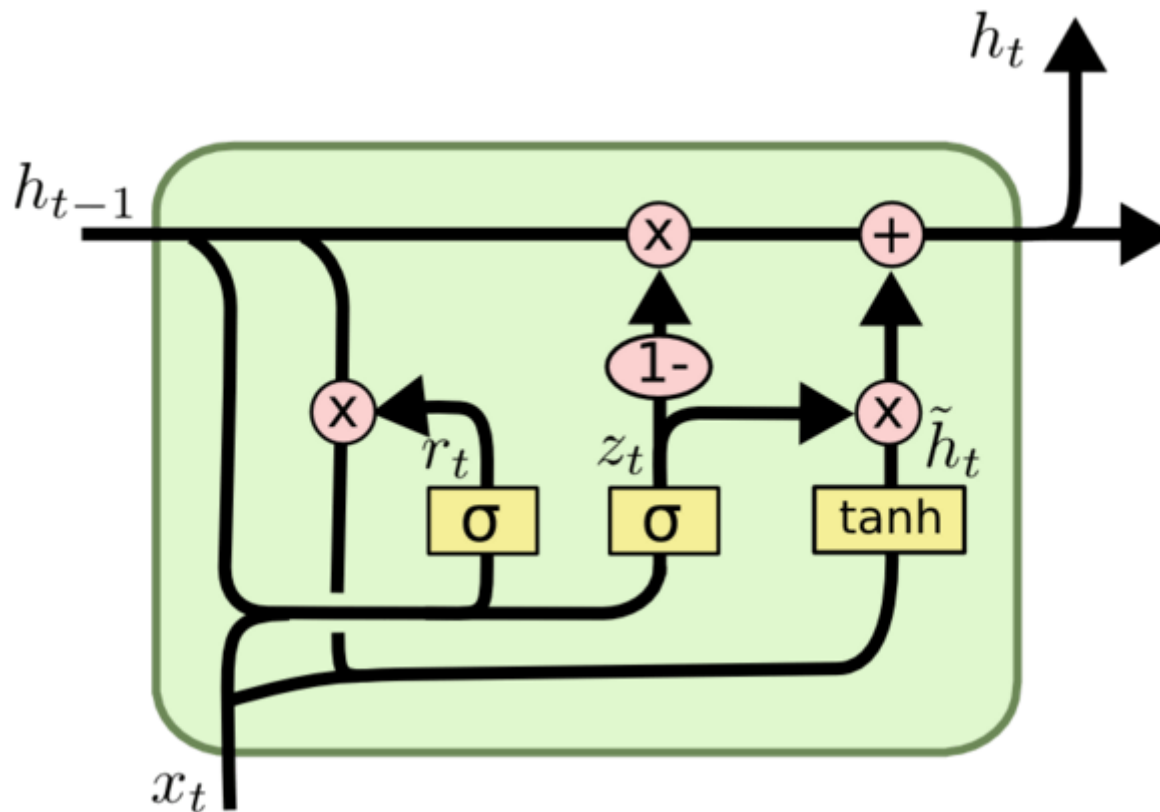
NLLG

Vanishing gradients

GRUs & LSTMs

GRUs

- More complex hidden unit computation in recurrence

- Gated Recurrent Units (GRU) introduced by Cho et al. (2014)

- Main idea:

  - Gates to control the flow of information
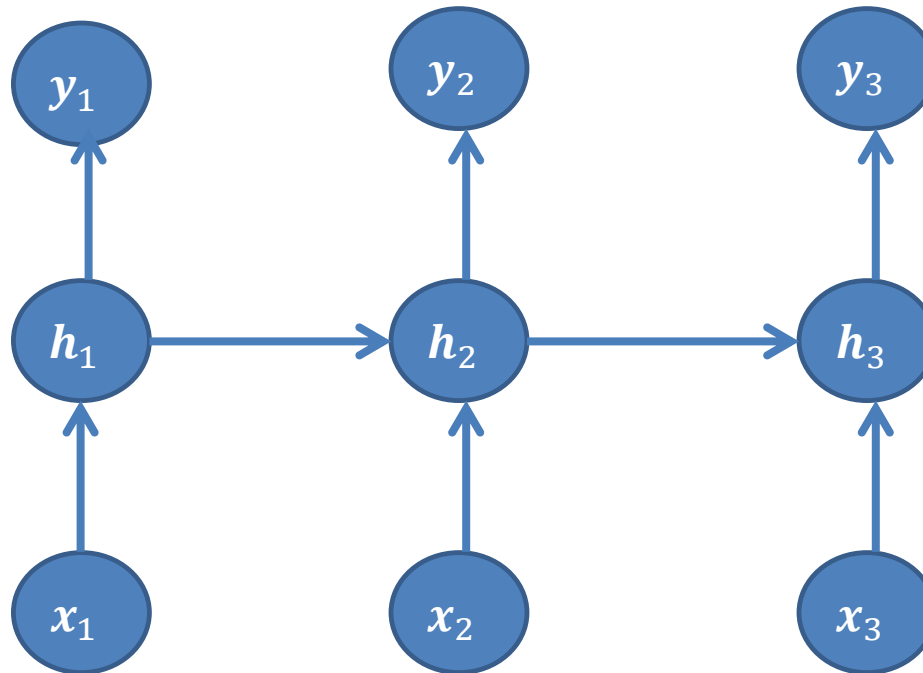
GRUs

GRUs

GRU illustration

NLLG

Some notation

- Conventions for the following slides

  - σ is the sigmoid (=logistic) non-linearity

  - $\odot$ is the *Hadamard* (=point-wise) product

    - $\boldsymbol{a} \odot \boldsymbol{b} = (a_1 \cdot b_1, \dots, a_n \cdot b_n)$

GRU memory unit

- Update gate
  - $z_t = \sigma\left(x_t U^{(z)} + h_{t-1} W^{(z)}\right)$
- Reset gate
  - $r_t = \sigma\left(x_t U^{(r)} + h_{t-1} W^{(r)}\right)$
- New memory content
  - $\widetilde{h}_t = \tanh(x_t U + h_{t-1} W \odot r_t)$
- Final memory at time step combines current and previous time steps
  - $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t$

GRU memory unit - Analysis

- Extreme cases: $z_t \in \{0,1\}, r_t \in \{0,1\}$

    - Note: z_t and r_t are vectors, but we look at individual components here

GRU memory unit - Analysis

- Extreme cases: $z_t \in \{0,1\}, r_t \in \{0,1\}$

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

  - $z_t = 0$:

    - $h_t = h_{t-1}$ → no update → can keep memory from previous time step → no vanishing gradient

GRU memory unit - Analysis

- Extreme cases: $z_t \in \{0,1\}, r_t \in \{0,1\}$

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

- $z_t = 1$:
  - $h_t = \widetilde{h}_t$

GRU memory unit - Analysis

- Extreme cases: $z_t \in \{0,1\}, r_t \in \{0,1\}$

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

  - $z_t = 1$:
    - $h_t = \widetilde{h}_t$
    - $r_t = 0$:
      - $h_t = \tanh(x_t U) \rightarrow$ Forget past
    - $r_t = 1$:
      - $h_t = \tanh(x_t U + h_{t-1} W) \rightarrow$ Standard RNN

$$\widetilde{h}_t = \tanh(x_t U + h_{t-1} W \odot r_t)$$

NLLG

GRU memory unit - Analysis

- Summary:
  - Can store memory at a cell indefinitely
  - Can also forget past memory, and reset everything
  - Can also go back to standard RNN mode, where memory is continuously updated based on past memory and current input

LSTM
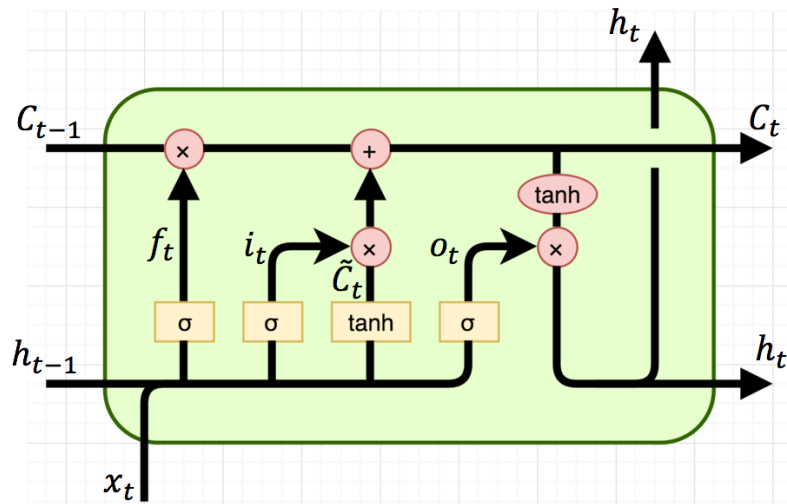
- Can make units even more complex

LSTM (Hochreiter & Schmidhuber, 1997)

$$F(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\theta} = [\boldsymbol{W}, \boldsymbol{U}])$$
$$= \sigma(\boldsymbol{x}\boldsymbol{U} + \boldsymbol{h}\boldsymbol{W})$$

- Input gate (= write gate) $\boldsymbol{i}_t = F(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{\theta}_i)$

- Forget gate (= reset gate) $\boldsymbol{f}_t = F(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{\theta}_f)$

- Output gate (= read gate) $\boldsymbol{o}_t = F(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{\theta}_o)$

- New memory cell
  - $\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{x}_t \boldsymbol{U} + \boldsymbol{h}_{t-1} \boldsymbol{W})$

- Final memory cell
  - $\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$

- Final hidden state
  - $\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$

NLLG

LSTM (Hochreiter & Schmidhuber, 1997)

(a) Long Short-Term Memory
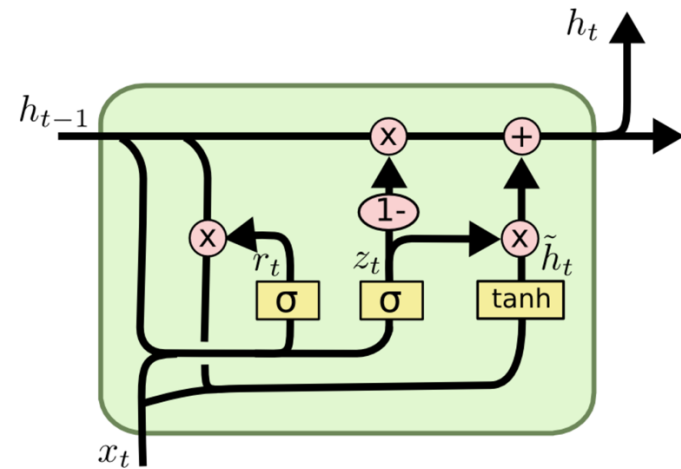
(b) Gated Recurrent Unit
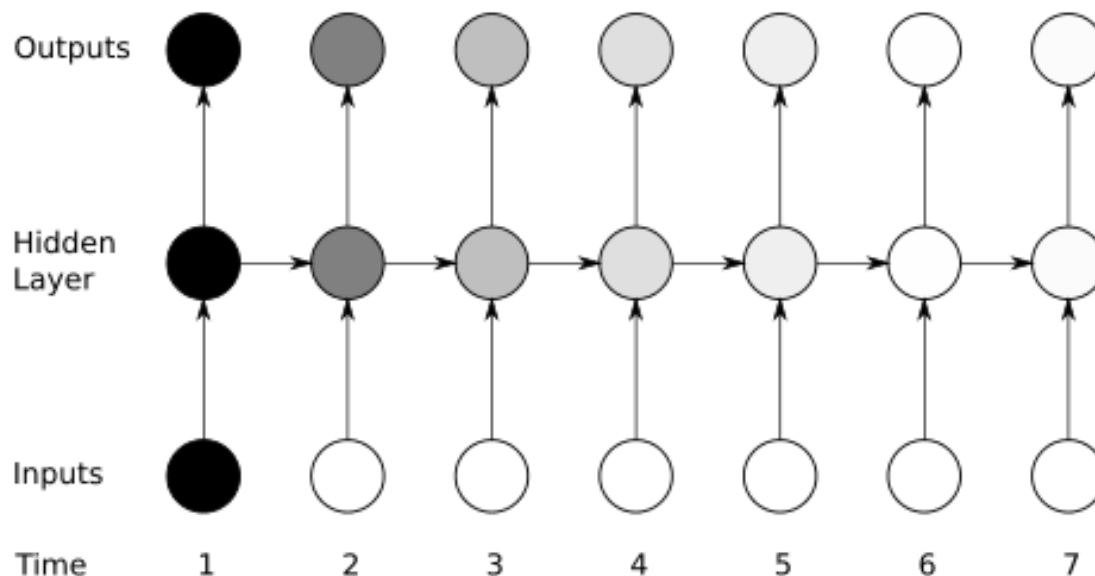
LSTM (Hochreiter & Schmidhuber, 1997)

Figure 4.1: **The vanishing gradient problem for RNNs.** The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Source: Alex Graves, PhD thesis
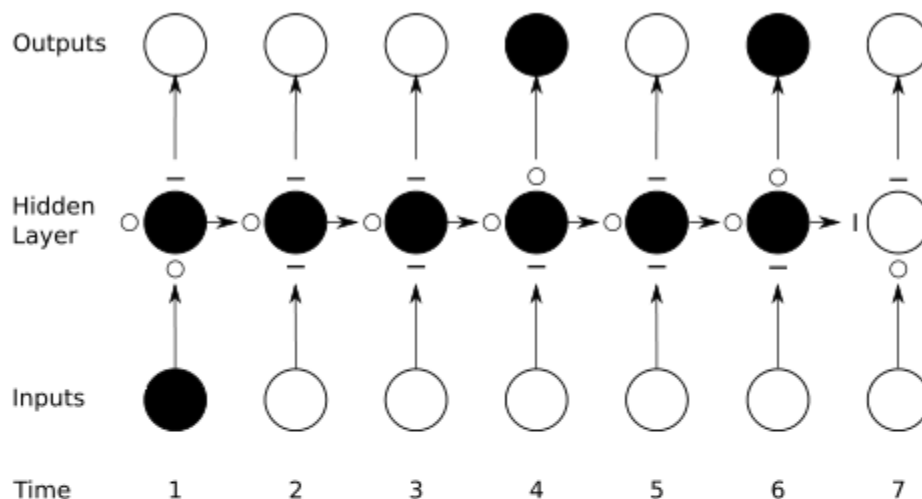
LSTM (Hochreiter & Schmidhuber, 1997)

Figure 4.4: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Source: Alex Graves, PhD thesis

NLLG

GRU vs. LSTM

- LSTM much more popular (a lot has to do with bias)
- But follows same principles of controlling flow of information via gates

Source: Alex Graves, PhD thesis

Summary

- Recurrent Neural Networks are powerful

- Gated Recurrent Units even better

- LSTMs maybe even better

- LSTMs in heavy use until 2018

- Problem: parallelization is difficult
    - Search for more efficient architectures (Transformers)

References

Pascanu, R., Mikolov, T., & Bengio, Y.: On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, 2013

Martens, J. (2010), Deep kearning via Hessian-free optimization.

Martens, J., & Sutskever, I.: Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning,* 2011

Le, Q. V., Jaitly, N., & Hinton, G. E.: A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941,* 2015

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y.: On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Syntax, Semantics and Structure in Statistical Translation*, 2014

Hochreiter, S., & Schmidhuber, J.: Long short-term memory. In *Neural computation*, 1997

Ma and Hovy (2016), End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF

Lample et al. (2016), Neural Architectures for Named Entity Recognition

Sutskever et al. (2013), On the importance of initialization and momentum in deep learning

Presentations and Term Papers

Presentations and Slides

- How to make good presentations
  - Please see the videos on the course github
  - In general:
    - Don't make crowded slides
    - Unfolding is a good idea
    - Adding (many) pictures is a good idea

NLLG

Presentations and Slides

- Further:
  - Know your audience
  - Give illustrations and examples (most important)
  - Explain the setup, give necessary background
  - Detail your experiments and experiment plans
  - Stick to the time limits

Term paper

- How to make good term papers
  - Check the Latex template, it gives good guidelines
  - Check „Best term papers" of previous courses of mine
    - E.g. github.com/steffeneger/dldh/ → 2021

  - Add indication **who did what in the experiments and writing**
    - In the term paper itself
  - Make an illustrating example
  - Make your own tables and figures
  - Write proper English